Modelling Escalation of Attacks in Federated Identity Management

Sean Joshua Mallory Simpson

Supervisors: Thomas Groß
Aad van Moorsel
Charles Morisset

School of Computer Science
Newcastle University

This dissertation is submitted for the degree of
Doctor of Philosophy

April 2021

Abstract

Federated Identity Management (FIM) is an increasingly prevalent method for authenticating users online. FIM offloads the authentication burden from a Service Provider (SP) to an Identity Provider (IdP) that the SP trusts. The different entities involved in the FIM process are referred to as stakeholders. The benefits of FIM to stakeholders are clear, such as the ability for users to use Single Sign-On. However, the security of FIM also has to be evaluated. Attacks on one point in a FIM system can lead to other attacks being possible, and detecting those attacks can be hard just from modelling the functionality of the FIM system. Attacks in which the effect of one attack can become the cause for another attack are referred to in this thesis as escalating attacks. The overall research question this thesis revolves around: how can we model escalating attacks to detect attacks which are possible through an adversary first launching another attack, and present causality of attacks to the FIM stakeholders involved?

This thesis performs a survey of existing attacks in FIM. We categorise attacks on FIM using a taxonomy of our own design. This survey is the first attempt at categorising attacks that target FIM using a taxonomy. Some attacks can have an effect that causes another attack to be possible in ways that are difficult to predict. We consider a case study involving OAuth 2.0 (provided by existing literature), as a basis for modelling attack escalation.

We then seek to present a language for modelling FIM systems and attacker manipulations on those systems. We find that FIM systems can be generalised for the purpose of a programmatic logical analysis. In addition, attacker manipulations on a system can be broken down using an existing conceptual framework called Malicious and Accidental Fault Tolerance (MAFTIA). Using a generalised FIM system model and MAFTIA, we can express a complex interlinking of attacks informed by case studies in FIM security analysis. This is the first attempt to model FIM systems generally and apply logical analysis to that model.

Finally, we show how causality of attacks can be analysed using attack trees. We find that any solutions to an escalating attack can be expressed using a tree model which conforms to existing research on attack trees. Our approach is the first attempt of modelling attacks on FIM systems through the use of attack trees. We consider stakeholder attribution and cost analysis as concrete methods for analysing attack trees.

Invictus

Out of the night that covers me,
    Black as the pit from pole to pole,
I thank whatever gods may be
    For my unconquerable soul.

In the fell clutch of circumstance
    I have not winced nor cried aloud.
Under the bludgeonings of chance
    My head is bloody, but unbowed.

Beyond this place of wrath and tears
    Looms but the Horror of the shade,
And yet the menace of the years
    Finds and shall find me unafraid.

It matters not how strait the gate,
    How charged with punishments the scroll,
I am the master of my fate,
    I am the captain of my soul.

Henley, William Ernest (1849–1903)

Dedication

I dedicate this thesis to all my family, friends, and colleagues (these categories not being mutually exclusive) that supported me in ways that can never be repaid if I were to live a thousand lifetimes.

Mum and Dad: thank you both for the sacrifices you made for me over the years. It took me too long to fully realise that. I hope my efforts have made you proud.

Jack, Imogen, Alex, Darcie, and Alan: over the past few years I've neglected my duty as brother especially in the face of such overwhelming loss. I promise to make it up to you, and we will enjoy some time together soon! Alan, I grieve that you are not here to see me cross the line, but I can just as well imagine you saying "well done ma' son!" all the same.

Rest of the Clan: too many times have I declined events because of work and failed to see you all. I hope to greatly improve on this. Grampa, I wish we all had a few more years with you, I'm sure that seeing us all grow tall would have made you happy.

Team Mass Debate: it must seem like an eternity to you as it does to me. How long have I been stranded up here? You all kept me propped upright when the winds were roughest. Truly, English fails here and only the ancient language can come close: Hiraeth. Here's to brighter days ahead!

Assorted Newcastle Lot: Will, Dalley, Phil, Jack, Ed, Joe, Dave, and so many more. You have all made my time in Newcastle truly special and we have shared memories that I'll carry with me forever. Thanks, and let's have many more!

Office Support Squad: Peter, Luca, John, Ehsan, and so many others. Always making the days brighter (and louder). Thank you for making the PhD a much less of a lonely experience.

The Iron: never thought I'd owe so much to a collection of inanimate heavy objects and the people around them that encouraged me to lift them. Thank you for helping me escape for a few hours almost every day.

Lastly, but certainly not least, Caitlin: I'm not sure I would have made it without you and your family, Caitlin. The brightest light in the blackest dark. All words fail to describe the endless support you provided and how you always made it all look like nothing, even when it wasn't. Thank you.

Acknowledgements

I would like to acknowledge my supervisors for their patience and assistance with my research. Thomas Groß, who also collaborated with me on my publication regarding attack analysis in FIM and who truly demonstrates what the word "expert" really means. Also to Charles Morisset, whose unwavering patience, teaching, and knowledge I will forever be thankful for. I acknowledge any other lecturers and staff at Newcastle University that have helped me over the years, especially Nigel Thomas. Thanks to both of my examiners Sadegh Soudjani and Shujun Li who were instrumental in getting this thesis up to par, and offered much interesting discussion. And finally, The British Government for helping fund this research.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 80,000 words including footnotes, tables and equations, but excluding the front matter, appendices, and references. This dissertation has fewer than 150 figures.

Sean Joshua Mallory Simpson
April 2021

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

1.1. Problem Description

Cyber attacks on systems are common and impactful. According to the UK Government (2019) about a third of UK businesses report being affected by cyber attacks in the last 12 months. The average cost of an incident is rising with the cost of an attack in 2019 being £4,180 up from £3,160 in 2018 and £2,450 in 2017. In addition, we have devastating cyber attacks such as WannaCry, which according to the BBC (2017), affected 75,000 computers including 48 National Health Service (NHS) trusts.

One specific type of system that this thesis focuses on is Federated Identity Management (FIM), which was described in-depth by Chadwick (2009). FIM is three-party authentication where the burden of authentication for a user to a Service Provider (SP) is provided by an Identity Provider (IdP). Facebook serves as a specific FIM system as users can login to other services through their Facebook account according to Wang et al. (2012). Isaac and Frenkel (2018) report a particularly severe security breach where 50 million accounts were compromised. Since Facebook serves as an IdP, security breaches that involve Facebook can also affect the SPs linked to the IdP account which is a risk of FIM systems in general as pointed out by Kormann and Rubin (2000). The importance of detecting attacks on FIM systems is clear when we consider that stopping an attack on a FIM system can save a user from data being accessed, used, disclosed, disrupted, modified, or destroyed across many SPs.

There are three main kinds of attack detection methods. The first is intrusion detection which focuses on detecting attacks in real-time. For example, Denning (1987) provides a general model of intrusion detection based on monitoring a system's audit records to detect anomalies. Working intrusion detection systems are also provided by the literature like Snort, introduced by Roesch (1999), which detects attacks over a network by sniffing the packets and pattern matching the packets to known attack methods (like buffer overflow). The limitation of intrusion detection is that typically a live system is needed in a high risk environment in order to perform the intrusion detection. In addition, FIM is distributed by nature so it might be difficult to understand an overall picture of FIM in a live environment in order to conduct intrusion detection. Also of note is that in this thesis a survey of attacks is provided in chapter 3, and little evidence is found of intrusion detection being used there. This is not to say that intrusion detection is useless for FIM, as there is potential for classifiers to be built from a dataset similar

to what is presented by Le et al. (2017). However, this thesis does not focus on attack detection from the intrusion detection perspective.

The second general method in attack detection is ethical hacking. Ethical hacking involves taking the role of the attacker and probing a system for possible security flaws from the point of view of the attacker. Many resources are available on how to do ethical hacking in general, such as the book provided by Scambray et al. (2000). Ethical hacking has been used on FIM systems in the past, for example in the work by Sun and Beznosov (2012), which investigates OAuth 2.0 implementations for attacks. The limitation of ethical hacking is that typically a live system is needed so the environment can involve a high degree of risk. However, the outcomes gleaned from ethical hacking approaches can be used in other attack detection approaches. For instance, in this thesis the security issues uncovered by Sun and Beznosov inform the attack detection process.

The third general method of attack detection is through modelling the design of a system and analysing the model of that system for possible attacks. Sheyner et al. (2002) provides a method for modelling networks and constructing attack graphs (reachability of some undesirable state from another state) via a model checker. Vigo et al. (2014) describes a process algebra approach for generating attack trees automatically for a system. Chen et al. (2004) describes a process through which one-million lines of code are analysed via a model checker on real UNIX applications and over a dozen security weaknesses are found. Modelling allows for a low risk environment as no live system is required, replacing the live system with a model of that system that can detect attacks before they happen. Modelling is an approach which has been performed on FIM systems to detect attacks, for example the work done by Armando et al. (2008), who provides a formal model and analysis of the Security Assertion Markup Language (SAML).

However, modelling FIM has mainly focused on the functionality of the FIM system, which has been flawed in the past. Chari et al. (2011) models the functionality of the OAuth 2.0 protocol and finds the protocol to be secure under certain assumptions. Pai et al. (2011) formally verifies OAuth 2.0 to confirm the existence of already known vulnerabilities but does not find any completely new exploits. Sun and Beznosov (2012) then proves that OAuth 2.0 is not secure at the implementation level through an ethical hacking approach. What if we could model the situation described by Sun and Beznosov? Doing this would allow for a different approach to attack detection where a modelling approach is informed through ethical hacking, which detects attacks without modelling a detailed protocol specification.

Complications occur with the work described by Sun and Beznosov (2012) as there is not just one attack that compromises the OAuth 2.0 implementations. There is an interlinking of attacks where some attack's effect becomes the cause for another attack. For example, a Cross-Site Request Forgery attack (CSRF) could be used as a staging point for a Cross-Site Scripting attack (XSS). This thesis refers to the phenomena of an attack's effect becoming the cause for another attack as *attack escalation*. So, how do we model attack escalation on a system? Modelling

attack escalation on a system could detect attacks that are possible through an attacker first launching another attack (such as the attacks described by Sun and Beznosov), which could escape detection by modelling through other means (such as Chari et al. (2011)).

An interesting aspect to attack detection research regarding FIM is that there are a number of stakeholders to report attack information to. Authors declare specific stakeholders in FIM. Stakeholders are mentioned briefly by Chadwick (2009) when referring to entities that benefit from a FIM federation. The idea is to present attack possibilities to stakeholders for FIM in a way that the causality of an attack can be clearly attributed to the stakeholders being considered. Subsequently, tailored solutions can be offered to this group of stakeholders. Sun and Beznosov (2012) isolate vulnerabilities to SPs and IdPs and therefore considers the SPs and IdPs to be the stakeholders in the FIM system. However, further inspection of the work by Sun and Beznosov indicates that the browser could also be involved in attacks on the FIM system. For example, Cross-Site Scripting (XSS) attacks play a role in the numerous attacks described. It is clear from the literature that the browser can be at fault for XSS attacks. Chiu and Chan (2006); Gupta and Gupta (2015), both describe solutions for browsers to adopt in order to stop XSS attacks. If only SPs and IdPs are considered as stakeholders, then possible fixes for attacks considered at other stakeholders are ignored, like the browser. Some authors do consider the browser as a stakeholder, such as Jøsang et al. (2005) who consider trust requirements on the browser in FIM systems. A more holistic view of FIM can be taken to provide as much information to stakeholders as possible from an attack on FIM. This approach is useful when building on existing literature. Additional solutions to the problem can be considered when additional stakeholders are included, such as protecting the browser against XSS attacks in the case of the work presented by Sun and Beznosov.

The overall problem of this thesis is how can attacks be modelled on a FIM system that captures how attacks on one stakeholder can cause possible attacks on other stakeholders, possibly detecting attacks that could be missed by other means. Causality of those attacks needs to be modelled in such a way that it is clear how stakeholders are impacted so that solutions relevant to the interested stakeholders can be presented. Through modelling escalating attacks on a system model, the situation of attacks being possible through an attacker first successfully executing another attack is captured. Application of this process could allow for detection of attacks before they happen. The solution to the attack should be presentable with respect to a set of considered stakeholders using appropriate analysis tools.

1.2. Thesis Outline

In this chapter (1) an introduction to the general problem we are dealing with is presented through the problem description. This section outlines how the following chapters, written in the order they appear, contribute to solving that problem. In the following subsection the overall research question and aims is presented.

This thesis has a chapter (2) describing the background of this problem area. Broadly, we consider how this thesis fits into different aspects of Computer Science literature. In addition, we summarise the work that this thesis builds on. In particular, we highlight key influences on this work, such as Malicious and Accidental Fault Tolerance (MAFTIA) and logical programming. Related work is mostly considered alongside the chapters we introduce rather than the thesis as a whole.

The Survey into Attack Analysis chapter (3) gathers knowledge about attacks in FIM. The original contribution for this chapter is a lay of the land of attack analysis in FIM. To the best of our knowledge, this is the most comprehensive survey into attacks on FIM. To understand attacks as they are presented in the literature, we use MAFTIA as presented by Powell and Stroud (2003). MAFTIA allows for a systematic way of viewing attacks. We have a peer reviewed paper concerning this which can be found in the reference Simpson and Groß (2016). The entire paper was almost entirely written by the author of this thesis with Groß providing primarily consultation and oversight. In terms of the overall story about modelling escalating attacks, we are interested in the categorisation of attacks to identify causality. In this chapter we introduce a case study that is used throughout this thesis which revolves around the work presented by Sun and Beznosov (2012). In particular, we encode knowledge of escalating attacks which is used in the subsequent chapter.

The Security Modelling in FIM chapter (4) introduces the system model. The idea is to detect attacks on a model of a FIM system using logical analysis. This logical analysis is informed by a knowledge encoding introduced in the survey chapter. The various stakeholders in FIM are described as non-logical symbols. Attacks on these stakeholders in terms of the concepts outlined by MAFTIA are also described as non-logical symbols. We can then see what attacks are possible on the FIM system automatically by using logical rules which can be described as an attack trace. To the best of our knowledge, this is the first logical analysis approach applied to attack detection on FIM.

The Attack Tree Assisted Analysis chapter (5) focuses on analysing attacks on FIM systems using attack trees. The attack traces identified in the security modelling chapter are formulated as an attack tree. Since the traces are generated automatically, the attack tree can also be generated automatically. We provide a mapping from the attack tree nodes to the actual stakeholders for a FIM system. Solutions to security problems can then be considered for different stakeholders of FIM systems. We demonstrate stakeholder attribution usefulness by considering some cost analysis techniques. To the best of our knowledge, this is the first automatic attack tree generation and subsequent analysis performed on FIM systems.

The conclusion chapter (6) outlines an overall summary and future work that could build on this thesis.

1.3. Research Question and Aim & Objectives

Overall Research Question: *How can we model escalation in FIM to detect attacks which are possible through an attacker first launching another attack and present the causality to the stakeholders involved?*

Aim: *Model escalating attacks in Federated Identity Management.*

Objectives and Overall Contributions:

1. Gather knowledge about the landscape of attacks in FIM using a taxonomy based approach. How can we encode knowledge of escalating attacks?
2. Provide a model that captures attacker manipulations on a FIM system. How can attacks be detected on this model?
3. Formulate attacks on the FIM system as attack trees. How can we analyse this model and present relevant information to FIM stakeholders?

Chapter 2. Background

2.1. General Information Security Background

2.1.1. Information Security and Authentication

This thesis is concerned with *information security*. The definition of information security as provided by NIST (n.d.) is, "the protection of information and information systems from unauthorised access, use, disclosure, disruption, modification, or destruction in order to provide confidentiality, integrity, and availability". We use this definition because FIM is a system that is concerned with distributing identity information across security domains as according to Maler and Reed (2008). Security domains is loosely defined as the different entities in FIM (e.g. Service Providers and Browsers) interacting with one another. There are other general security terms that might be applicable, such as cyber security, which is defined by NIST (n.d.) as "the ability to protect or defend the use of cyberspace from cyber attacks". This definition immediately begs the question of what cyberspace is. Cyberspace has three different definitions according to NIST (n.d.), one of which includes the people using the system as part of the cyberspace. It seems that what cyber security and cyberspace means is continuing to evolve which brings us back to information security which is a more established area of research. That is not to say that terms like cyber security cannot be used in the context of FIM, but for simplicity and clarity we talk about security in the context of FIM as information security.

When we talk about information security the primary thing we are concerned with is the *confidentiality*, *integrity*, and *availability* of information or information systems. These concepts are often referred to collectively using the acronym CIA. Let us present these three concepts which we source from Andress (2014).

- **Confidentiality:** refers to the ability to protect information from those that are not authorised to view it. Andress makes the comment that confidentiality can be implemented in many levels in a process. If we consider a concrete example of how confidentiality is implemented in a process, passwords come to mind. When passwords are implemented for a web service, there is normally an effort to obscure the password at the client-side from a shoulder surfing (where an attacker observes the user's screen) type of attack. Then the password would be expected to be hashed in transit and compared to a hashed version of the actual password so that not even the web service knows the password. Confidentiality can be compromised, for example if a user password is somehow disclosed. A user

password could perhaps be disclosed through a keylogger or incorrect implementations of confidentiality mechanisms. In regards to FIM, confidentiality can be quite complex because the purpose of FIM is to share identity information. For example, Alrodhan and Mitchell (2009) describes a situation in Microsoft Cardspace where the trustworthiness of web services needs to be considered. If web services are not trustworthy, then the sharing of user information could be considered a confidentiality breach.

- **Integrity:** refers to the ability to protect information from unauthorised or undesirable change. The obvious approach to protecting information integrity is to prevent the information from being changed in the first place. According to Andress, the other important approach to information integrity is to revert changes that do occur, even if those changes were authorised. Andress considers the example of how information integrity is addressed in modern operating systems. For instance, operating systems use a permission based system to prevent unauthorised changes to information. Additionally, operating systems and many applications offer ways to rollback changes once they have occurred. In the context of FIM, we do see attacks on the integrity of user information. For example, Sovis et al. (2010) describes how users of OpenID can have the integrity of their information compromised through an attacker injecting information into the parameters of the message.
- **Availability:** refers to the ability to access information when it is needed. Andress states that there are many possible places the availability of information to users can break down. These breakdowns can occur from the power cord on the user's PC to the correct functioning of the web service the user is trying to access information on. We note here that rarely would it be solely upon a single organisation to provide complete availability. The web service could not reasonably be expected to ensure that the user's PC is working correctly, for instance. Regarding FIM and availability, there are particular problems. For instance, early on in FIM's emergence it was noted that a centralised infrastructure immediately lends itself to complete availability compromise by authors such as Kormann and Rubin (2000).

The information stored across a FIM system is typically associated to an account. When an account is compromised, the confidentiality, integrity, and availability can suffer. For instance, user information could be viewed compromising confidentiality. User information could be changed violating integrity. Or the user account could be entirely deleted denying availability. When talking about FIM in terms of CIA often times we talk in terms of account compromise because the entire CIA triad is compromised.

Authentication: one particular method of achieving CIA. Andress (2014) states that authentication is the process of verifying the claim on an identity. Authentication takes place by a factor of authentication. There are three different factors of authentication:- something you *know* (like a password), something you *are* (fingerprint), and something you *have* (key).

Through authentication, confidentiality of a user's information can be provided by ensuring that the user is the owner of that information. Integrity similarly can be provided by ensuring that the user is the owner of information and can therefore modify it. Availability to the authenticating service also needs to be provided so that the user can access their information. Authentication provides the capability for a service to distinguish one user from another over the internet where almost anything can be talking to almost anything else. Thereby, a user's CIA can be protected against malicious and accidental security incidents.

Authorisation: the process of determining whether a user has the right to access or change information in any way and follows authentication. This is another important method to information security as it is often the case that information does need to be shared, for example, confidential information to a law firm. Authorisation is an important part of information security, but we focus primarily on authenticating systems in this thesis.

Access Control: according to Sandhu and Samarati (1994) has the purpose of limiting authenticated users via authorisation. This thesis views access control as another security mechanism to enhance the CIA properties for a system by limiting authenticated users. An example of access control is when a university computer system allows students to authenticate, but those students should not be authorised to access objects such as the exam paper for one of their modules. While access control is an important part of the information security landscape, it is not the focus of this thesis.

Many websites have some kind of authentication option. Users authenticate with most services online so that the service can tailor itself to the user. A common form of credentials for a user authentication is a username/password combination which in terms of factors of authentication is 'something you know'. Passwords are easy to implement for services because the main thing that is required is the user's memory and no special equipment is needed.

A common problem with passwords according to Ives et al. (2004), Jenkins et al. (2014), Gaw and Felten (2006) is that the same username/password combination can, and often are used across different online services. The issue is that if the user's credentials are stolen then an attacker can compromise the user's account and potentially any other accounts that the user accesses with the same credentials. The alternative to using mostly the same password is using a unique password for every service but according to Herley and Van Oorschot (2012) it is frustrating for a user to manage many passwords.

Users vary in how securely they use username/password combinations, where some users will only use one username/password combination for every account, and other users will employ different username/password combinations for every online account, while the rest of the users will fall somewhere in-between these two extremes. On the one hand, users who do not use different username/password combinations risk having large sectors of their online presence

compromised in the event of stolen credentials, on the other hand, users who have many different username/password combinations risk losing access to their account in the event of forgotten credentials.

The way authentication is handled currently on the World Wide Web has negative implications on information security for users of computer systems around the world. This thesis is generally focused around the authentication topic area. Specifically, we focus on Federated Identity Management (FIM) as an alternative to passwords. First, let us consider more deeply the current state of passwords and other authentication solutions.

2.1.2. Passwords and Other Authentication Solutions

This thesis focuses on FIM. However, here we provide a general background into other authentication methods as having some overview here shows how FIM is necessary. Passwords are convenient methods of authentication in terms of their simplicity to both implement and to be understood by a user. There is general academic consensus on the shortcomings of passwords. There is still much debate on how to move on from passwords, if at all. The motivation for FIM becomes clear when we understand passwords and their shortcomings. We will provide a summary on a number of academic works on passwords. Consider tab. 2.1 for a summary of prominent academic work on passwords from Security & Privacy (S&P), or Computer and Communications Security (CCS), or Communications of the ACM. Although there are certainly other top conferences, since this thesis does not focus heavily on passwords in general, we constrain the scope of the papers selected.

Paper	Brief Summary
Herley and Van Oorschot (2012)	Highlights common issues in organisation password policies and suggests that no definitive systematic method of evaluating different security policies exists. Comments on the issues of restrictive security policies that include excessive password length requirements and how these policies do little to protect against threats like keyloggers.
Kelley et al. (2012)	Experiments with password-cracking algorithms against common password policies.
Adams and Sasse (1999)	There needs to be better communication between users and security teams. Help users understand security issues, and do not inflict upon the user unreasonable password policies.
Yan et al. (2004)	Three different control groups generate passwords in different ways. The security of the passwords generated by these control groups is examined through cracking algorithms.
Juels and Rivest (2013)	Offers a technical improvement on passwords called honeywords. Using honeywords can detect if a password database is leaked without adding any new vectors of attack.

Table 2.1 A limited overview of academic discourse on passwords.

Password Alternatives and Improvements: The specific shortcoming we highlight is that a user should use different passwords for all the different websites they have accounts for to prevent leaks of passwords compromising their entire online presence. However, doing so raises serious usability concerns as the user has to remember many different passwords for all the different websites they use. As a result of this problem, there have been numerous attempts to overthrow the dominion of passwords. Bonneau et al. (2012) perform a comprehensive survey to the alternatives and improvements to password. Alternatives include password managers, graphical passwords, cognitive authentication, mobile-phone based authentication, and biometrics. It should also be noted that FIM and any of these other solutions are not necessarily mutually exclusive. FIM could be accessed as part of a password manager. FIM could use graphical passwords to authenticate. However, it is still useful to consider these methods for FIM in order to realise the specific problem that FIM tries to solve. For a thorough understanding of password alternatives, consider Bonneau et al. (2012).

A Case for FIM: After reviewing passwords and other authentication solutions the case for FIM becomes clear. Firstly, there is a trust relationship between an IdP and SP. The user authenticates to that IdP in whatever way the IdP chooses to implement and then the user can access tailored services on any SP. It should be noted that at least some scholars have advocated that FIM uses a standard browser (also referred to as zero-footprint), for instance Pfitzmann and Waidner (2003b). This is because many users seem unwilling to download certain software or purchase specific hardware. This would mean that certain other authentication solutions (such as biometrics) would not fit into this model. Assuming a zero-footprint approach, the motivation for FIM is that users can authenticate to many SPs with one set of credentials overcoming the shortcomings of passwords without having to resort to specific software or hardware on the user's side that many other authentication solutions require.

2.2. Federated Identity Management (FIM)

2.2.1. General

Identity Management is the basis for FIM. We consider a simplified definition from ITU-T (2009b), who outlines that Identity Management as a set of functions and capabilities (e.g. authentication) and is used for:

- Assurance of identity information (e.g. credentials).
- Assurance of the identity of an entity (e.g. users).
- Enabling business and security applications.

FIM is an authentication method extending from Identity Management. ITU-T (2009a) outlines a federation as an association between SPs and IdPs. According to Chadwick (2009), there is an implicit trust relationship between these SPs and IdPs such that messages can be exchanged

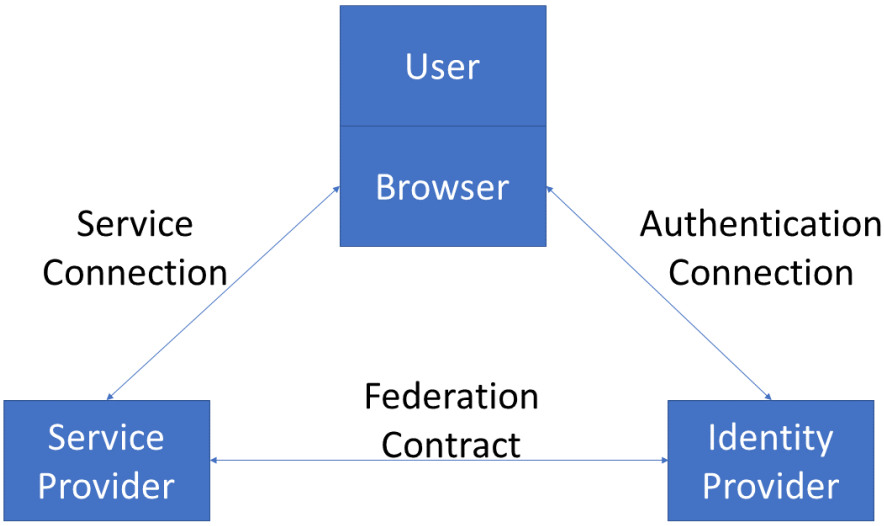

Figure 2.1 General Overview of FIM

between them. Chadwick concludes that when these messages exchange authentication and authorisation information such that users from one system can access resources in a federation, FIM occurs. In addition, we note that once authenticated, the FIM system can provide access control mechanisms based on the authorisation information. A user aims to authenticate themselves to a SP in order to access tailored services on that SP. The SP has some kind of federation contract between a Service Provider (SP) and Identity Provider (IdP). The user provides authentication credentials to the IdP and provided these credentials are correct the IdP validates the user's identity to the SP confirming the user's identity.

A number of individual different FIM implementations exist. Typically, a user through a Browser, which is often referred to as a user-agent in the literature, attempts to access services tailored towards the user through authentication at a SP. We call the connection that exists between the Browser and the SP a *service connection* and any messages between the browser and the SP occurs over this connection. That SP requests that the user authenticates with an IdP. We call the connection that exists between the Browser and the IdP an *authentication connection* and any messages transmitted from the browser to the IdP occurs over this connection. The IdP must be an IdP that the SP trusts. The connection that exists between the SP and the IdP we call a *federation contract* and any messages exchanged between the SP and the IdP occur over this connection. The user provides their credentials to the IdP, and provided that those credentials are correct, the IdP asserts identity of the user to the SP. This general overview of FIM can be seen in fig. 2.1.

In the remaining subsections in this section we summarise a prominent FIM system, OAuth, as it relates to a case study we examine in this thesis. In addition, we briefly cover other FIM systems, which will be considered as part of a security survey regarding FIM systems in the next chapter of this thesis. Finally, we consider the general paradigms of FIM systems.

Terminology: Irrespective of the terminology employed by other authors in talking about FIM, we consistently use the same terminology when talking about the core participants in FIM. We use the term *user* when talking about the individual requesting authentication through FIM. The user accesses the FIM system through a *browser*. We use the term *Service Provider* (SP) when referring to the website offloading the authentication burden to an authenticating website. We use the term *Identity Provider* (IdP) when referring to the authenticating server.

2.2.2. OAuth

In this thesis OAuth plays a significant role because we refer to a case study on the OAuth protocol suite. According to Leiba (2012), OAuth started as a community effort between several companies that provide internet services. This community approached the Internet Engineering Task Force (IETF) and eventually the OAuth working group was formed which developed the first standard for OAuth. At the time Leiba wrote that paper, an open standard for OAuth 2.0 was already close and large internet services started to adopt OAuth such as Google and Facebook. Facebook (n.d.) refers to documentation available for software developers which makes use of OAuth. Google (n.d.b) refers to how Google uses OpenID Connect and OAuth for authentication and authorisation. Given the ubiquitous nature of both of these corporations, it is clear to see that OAuth currently plays an important part in the ecosystem of World Wide Web technologies.

The use case for OAuth as described by Leiba (2012) is for one service to interact with another on behalf of the user without divulging secret information about the user such as passwords. The example given is when a user has many contacts on Gmail and joins Facebook and wants all of their contacts imported to Facebook automatically. Instead of giving user credentials to Facebook, OAuth provides access to user resources for a website through an access token in which the context for the usage of the access token and time limit for the token are specified. This way, the user can automatically import their contact information to Facebook without divulging secret information.

The terminology used for OAuth differs between sources. The security analysis community generally uses the term relying party (RP) to refer to the service that is requesting an access token to a resource owned by the resource owner (or user). The service which holds the resources are referred to as the Identity Provider (IdP). Examples of this kind of terminology within the security analysis community can be seen in Sun and Beznosov (2012); Li and Mitchell (2014). However, the specification for OAuth (which is provided by Hardt (2012)) does use some different terminology. The client is referred to as the service that is requesting the access token and the service which holds the resources is referred to as the resource server. There is also the additional consideration by the specification that in practice the authorisation server could be different to the resource server. Some within the security analysis community can be seen using this terminology like Yang and Manoharan (2013). This general discrepancy between the security analysis community and the OAuth specification could be because in practice OAuth is used on several large services that hold identifying information about users

(Facebook, Google) and so it is easy to think of them as identity providers. Furthermore, the security analysis community typically do not often separate the authorisation server and the resource server, perhaps because these two servers often belong to the same company. There are also several different modes of operation for OAuth to suit a few different purposes. For instance, Sun and Beznosov (2012) considers both the client and server flow modes of operation. In general in this thesis, we refer to the client/RP as the SP, the resource server as the IdP (which is also presumed to be the authorisation server), and the resource owner as the user who uses a browser to interact with the OAuth system. This is to be consistent with our general FIM terminology.

Consider a general overview of how OAuth works, adapted from Anicas (2014), which can be seen in fig. 2.2 and is described below. We make several adaptions to terminology from Anicas (2014), and ultimately consider the authorisation server and resource server to be the IdP.

- 1. The SP requests access by consulting the user.
- 2. If the user agrees, the server gets an authorisation grant.
- 3. The SP presents the authorisation grant which proves the user has approved the request.
- 4. If the Service Provider is authentic, and the authorisation grant is valid, the Authorisation Server provides an access token to the Service Provider.
- 5. The Service Provider requests the resource by presenting the access token to the Resource Server.
- 6. If the access token is valid, the Resource Server provides the resource to the Service Provider.

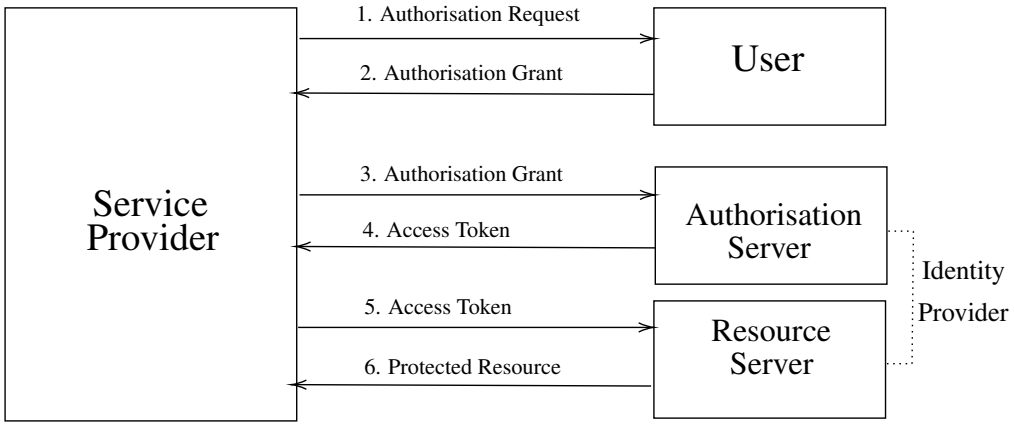

Figure 2.2 Abstract Overview of OAuth adapted from Anicas (2014)

2.2.3. Other FIM Technologies

Although this thesis mostly focuses on OAuth, other FIM technologies are covered in this thesis at the very least in a survey capacity. The FIM technologies are based off other surveys in the area, notably Pfitzmann and Waidner (2003b). However, we also based what we were concerned

with regarding FIM as what we noted as the most active areas of research from an analysis point of view. For instance, although Pfitzmann and Waidner do not cover OpenID in their survey, OpenID is clearly the subject of a lot of security analysis. To name a couple: Sovis et al. (2010) and Sun et al. (2012) perform security analysis of OpenID. However, not every FIM technology from Pfitzmann and Waidner is going to be a subject of study. For instance, browser-based attribute exchange (BBAE) has not received significant security analysis, and so will not be a subject.

One thing that is interesting about FIM technologies is whether or not the technology is more of a standard or an implementation. Let us consider tab. 2.2 which is a brief summary of each of these FIM technologies and specification of whether or not this technology is a standard of an implementation.

FIM Technology	Standard or Implementation	Brief Summary
OpenID	Standard	OpenID is a standard claiming to eliminate the need for websites to implement their own login systems by providing a standard which allows a user to provide a URL with various attributes associated to that user. The most recent standard is OpenID connect and can be found here: OpenID (2014).
SAML	Standard	The Security Assertion Markup Language (SAML) uses the Extensible Markup Language (XML) to perform identity federation. According to Groß (2003), one of the most important features of SAML is that it is zero-footprint, meaning no protocol specific software needs to be installed for SAML to function. A technical overview for SAML version 2.0 can be found here: Oasis (2008).
Liberty Alliance	Standard	An extension of SAML, and according to Pfitzmann and Waidner (2003a) provides four different authentication protocol specifications. Intends to operate using separate federations in circles of trust. Full details of a version (v1.2) of the specification can be found in Cantor et al. (2003).
WS-Federation	Standard	According to Goodner et al. (2007), WS-Federation builds on the WS-Trust protocol and the intended operation is to perform FIM for web services and browser based applications. A technical overview for version 1.2 can be found here: Oasis (2009).
Microsoft Technologies	Implementation	There have been several attempts staged by Microsoft to perform FIM over the years. This thesis references specifically Microsoft Passport and Microsoft Cardspace. This is because significant research effort has been spent on the analysis of these systems. Notably, Microsoft Passport was the first FIM technology to receive security analysis by Kormann and Rubin (2000).
Facebook	Implementation	Fairly ubiquitous at the time of writing, login with Facebook now appears as an authentication option for many web services. However, the analysis of this protocol has proven to be tricky in the past, since to the best of our knowledge there is no public protocol specification for at least some versions of the authentication. This is noted by Miculan and Urban (2011). Currently, login with Facebook appears to operate using OAuth 2.0 which can be ascertained by studying the Facebook developer page here: Facebook (n.d.).
Google	Implementation	Similar to Facebook in terms of prevalence. Google FIM technologies have gone through several changes over the years. Notable analysis has been performed by Armando et al. (2008) on a SAML implementation of Google. These days, like Facebook, login with google operates over OAuth 2.0 according to Google's own developer integration website: Google (n.d.a)
Shibboleth	Implementation	Notably used by universities to access academic articles and journals. According to Groß (2003), Shibboleth is based on SAML. We consider Shibboleth to be an implementation of SAML. Shibboleth documentation can be found here: Shibboleth (n.d.)

Table 2.2 Brief Summary of FIM technologies Considered in this Thesis and whether they are a standard or implementation.

2.2.4. *General Paradigms of FIM*

This thesis does primarily focus on the security of FIM. It is of note that security is not the only challenge that faces FIM. A survey by Jensen (2012) explores these different challenge domains which we briefly cover here:

- Investment Cost: Smith (2008) describes how FIM projects require leadership and sponsorship across organisations. This investment needs to be driven not just by technology experts, but business experts also, and this collaboration could be a daunting task if the rewards are not clearly understood. There could be technical difficulties in establishing a common identity across an internal organisation such as when Google first implemented a common identity across all of their systems. Then making that identity usable to external systems is an additional challenge that would require even more investment.

- Liability: According to Jensen, liability is the state of being legally obliged and responsible. Speltens and Patterson (2007) points out that liability in FIM is complicated in that FIM systems can cross legal jurisdictions which have different laws on aspects relevant to FIM like privacy and disclosure. Contractual agreements about who is responsible in different situations when things go wrong need to be in place. We deduce that this demonstrates how FIM is more than just a technological problem as law professionals might also be required in order to implement FIM.

- Assurance: Jensen gathers that identity assurance is the process of ensuring that identity management is under appropriate control. Ultimately, people control this process and the people who control identity assurance are put there by vetting and enrollment. Vetting and enrollment are also dependent on people doing the right things. According to Baldwin et al. (2010), identity management can cut across different organisations with different vetting and enrollment standards due to varying risk appetites. This could lead to trust issues and a possible end to a FIM project.

- Trust: Although many definitions of trust exist, the one Jensen adopts is one by Mayer et al. (1995). Trust is the willingness of a party to be vulnerable to the actions of another party based on the expectations that the other will perform a particular action important to the trustor, irrespective of the ability to monitor or control that other party. Smith (2008) points out that trust occurs across many different areas of a FIM system from users to organisations and to achieve trust at all these different levels might be a hard task. From a security point of view, we perceive that trust can be abused. For example, a user's trust in an organisation storing their identifiable information could be abused by secretly profiling the user.

- Knowledge: Smith (2008) argues that a lack of knowledge about FIM systems might be stopping FIM from reaching its potential. Users are generally familiar with FIM. For

example we see ubiquitous deployments of Google ID which many users are familiar with. However, Smith states that at the inter-organisation level there is still a lack of knowledge. We observe that the paper by Smith is relatively old for a paper reporting on the state of a particular technology. Since then, we have seen increased adoption of FIM systems, for example the UK now has a centralised login gateway UK Government (2021). Although it is not clear if FIM has reached its full potential, it is certainly better than when Smith first took a snapshot of the state of FIM.

- Data Synchronisation and Consistency: Hoellrigl et al. (2010) argues that despite FIM being a solution to centralise information for users, data still needs to duplicated for fault tolerance. The study performed by Hoellrigl et al. points out concerns on synchronising data in the event changes occur. The claim put forth is that data inconsistency can lead to erroneous system behaviour such as unauthorised access.

- IdP Discovery: When an authentication is provoked by a SP, that SP needs to find an appropriate IdP for the user to authenticate with. This is known as IdP discovery. The typical way IdP discovery is handled is for the SP to list a number of IdPs that the SP has established a federation with and have the user choose from this list. Rieger (2009) mentions an issue where if a user is part of multiple federations this could cause usability concerns. We consider an example where the user has authenticated to a particular SP before, but has not used that SP for a long time and forgets which IdP was chosen initially. The user could then choose a new IdP and be confused when the SP does not remember any information about the user.

- Interoperability: FIM protocols will work if everyone is using the same standard. However, Wolf et al. (2009) points out that this might not always be possible. The complexity of the process is increased if different standards are used. We observe that there are many different versions and modes within one protocol suite. For example, OAuth has two versions and many different modes.

- Revocation: Jensen states that FIM revocation is when identity data, which is often represented in security tokens, is made invalid so that they can no longer be used for authorisation or identification. Nordbotten (2009) states that current FIM systems lack sufficient revocation processes. A common way that this issue is handled is by limiting the lifetime of security tokens. We note that the lifetime of a token is an integral part of the OAuth specification (Hardt (2012)) but whether or not this issue has completely been addressed over time is not clear.

- Privacy: Naturally, sharing an identity across many different locations raises privacy concerns. Maler and Reed (2008) makes the point that one of the purposes of FIM is to share user information. Furthermore, a possible solution to the privacy issue is for users to control their own identities. According to Maler and Reed this is an ongoing problem. We perceive that in current times this is still an open problem with Facebook still being under

scrutiny of more recent studies. For example, Bartsch and Dienlin (2016) performs a user study on Facebook privacy exploring what factors make users more privacy literate.

2.2.5. Security Issues in FIM

The above issues are important problems in FIM systems. However, we choose to focus on security above these issues. So let us look in more detail about the security of FIM systems. Specifically, let us start by examining inherent risks that exist in FIM. From the conception of FIM there has been immediately identifiable risks. Identifying these fundamental risks is important because the security of FIM revolves around these risks.

Kormann and Rubin (2000) lists several practical risks for Microsoft Passport which could cause some security issues in FIM. Even though Passport no longer exists, we believe that these risks are still valid risks for current or fledgling FIM systems to consider.

- User Interface: The point is made that the user interface can be problematic for FIM. The specific point is made about how it could be difficult for a user to understand exactly what logging out of a service means. For instance, does logging out of the Passport IdP log the user out of the IdP or the FIM system? Reasonable mistakes by users regarding the use of FIM systems could put the user's information security at risk.

- Key Management: Passport relied on every merchant having a secure way of transferring data to the Passport IdP. This was a non-trivial challenge as keys could have been intercepted on route. We note that these days, key management is less of a consideration for FIM system designers, as typically this is handled by SSL/TLS protocols. However, careful consideration still needs to be applied for how access tokens are sent, as without SSL/TLS protection they can be intercepted according to OAuth (n.d.).

- Central Point of Attack: Consider if there are many SPs that are dependent on just one entity, the IdP. If this IdP is compromised, then the affect on user's information security could be severe. An additional consideration are DoS attacks, for which there is a clear target for bringing the whole system down.

- Cookies and Javascript: Cookies store encrypted credential information on the browser. Javascript is used to make the protocol more efficient with less redirects. Kormann and Rubin state that a careless user could forget to logout and allow for an attacker to steal the credential information. We also note, that even if a user is careful, there could still be problems. For instance, Sun and Beznosov (2012) show how Cross-Site Scripting attacks can be utilised by attackers to steal access tokens. Kormann and Rubin also highlight how cookies stored persistently are a problem, as all that is needed to prove knowledge of the cookie is to present the cookie, which could be used to impersonate a user.

- Automatic Credential Assignment: All hotmail accounts were automatically transferred to passport credentials. Kormann and Rubin also highlight that hotmail has been fraught

with security issues and as a result of this, every other SP that the user authenticates with could also be at risk. The message we take here is that FIM designers need to carefully consider what happens to the overall FIM system if a part of that system is compromised.

Kormann and Rubin specifically go into risks of Passport in their seminal work. Other papers also go into risks of using FIM but are generally narrower in scope. For example, Gajek et al. (2009) considers how the DNS system could undermine the security of FIM. The DNS is also considered by Korminn and Rubin for an attack. DNS flaws could also be leveraged in SAML as is pointed out by Groß (2003). The DNS system needs to be considered a risk for FIM systems as the literature supports that the DNS has been leveraged for attacks in the past.

Others consider the security issues with FIM systems by thinking directly about the attacks that can compromise those systems. Sun et al. (2012) considers several attacks for OpenID FIM system that are documented in the OpenID specification itself. OpenID systems are said to be susceptible to phishing attacks, man-in-the-middle attacks, replay attacks, and DoS. Sun et al. then goes on to propose several more novel attacks such as a cross-site request forgery attack. This kind of approach, focusing on attacks rather than risk, is repeated throughout the literature. Sovis et al. (2010), Feld and Pohlmann (2011), Krolo et al. (2009) all focus on attacks in OpenID. Armando et al. (2008), Pfitzmann and Waidner (2003a), Mayer et al. (2014) focus on attacks in SAML. This reservoir of attacks needs to be collated and studied which is the subject of chapter 3 in this thesis. Chapter 3 is our primary reference point for security issues in FIM.

Attack Escalation: One risk to FIM systems is attack escalation. The idea of attack escalation in this thesis is introduced in chapter 1. Attack escalation occurs when the effect of one attack leads to the cause of another attack. Potentially, attack escalation can undermine the security of the entire FIM system and can be hard to detect. For example, Chari et al. (2011) considers OAuth secure using the universally composable framework. But Sun and Beznosov (2012) shows how implementation errors and vulnerability interplays can interact to compromise FIM security. Attack escalation is the focus of this work and we aim to detect attacks by modelling how attacks can build on each other.

2.3. Malicious and Accidental Fault Tolerance (MAFTIA)

We are concerned with modelling escalating attacks in FIM. MAFTIA is aimed towards handling problems both accidental and malicious in systems. The main conceptual foundations for MAFTIA are explained by Powell and Stroud (2003). MAFTIA exists as a branch of the general dependability area. In part of handling problems, MAFTIA separates problems into divisible aspects rather than a monolithic problem. This is valuable, because as part of modelling escalation it is essential for attacks to be broken down past just one problem. Specifically in MAFTIA, we have a framework which is a robust research project to describe malicious attacks in stages. A reasonable approach is to adopt the concepts introduced by MAFTIA in order to model the attacks we observe in FIM as divisible aspects rather than monolithic entities.

2.3.1. Fundamental Concepts of Dependability

Malicious and Accidental Fault Tolerance introduces a set of concepts that can be used to consider security issues. In addition, MAFTIA concepts are especially suited for understanding security issues involving many different systems. Let us first address the roots of MAFTIA and how MAFTIA fits into the wider field of computer science. The story of MAFTIA begins within the dependability community. There exists a number of works that summarise the core concepts of dependability. Laprie (1992) is the first of such attempts and first introduces the core concepts informally. Avizienis et al. (2001) builds on this and introduces the idea of security to dependability. Powell and Stroud (2003), who sets the conceptual foundation of MAFTIA, focuses on the security aspect of dependability, but does also provide a chapter summarising work in dependability. We use concepts and ideas that originate from this area of research. This subsection is not intended as an entire summary of dependability, just the core concepts and definitions we use throughout this thesis.

Laprie (1992) provides a definition of a system that we will use. A system is an entity having interacted with, interacting, or able to interact with another system. Since we use a systems of systems approach, we also refer to a component. Laprie (1992) states that a component is another system, which is part of the considered system. This method of viewing systems is recursive, and can be applied until a system cannot be broken down any further or breaking the system down any further is not of interest. Any system that is a component of another system is said to be inside the system's system boundary. The system boundary is the frontier between a system and other systems.

- **System** An entity having interacted with, interacting, or able to interact with another system.
- **Component** Another system, which is part of the considered system.
- **System Boundary** The frontier between a system and other systems.

Avizienis et al. (2001) introduce formalised notions of dependability. Dependability is concerned with a systems ability to deliver correct service with respect to the system's requirements. This conceptual way of looking at systems is useful, because when we consider escalating attacks we consider what components in a system fail and how. The system states including and leading up to a failure and are categorised in terms of *faults*, *errors*, and *failures*.

- **Fault:** A fault is the cause of an error.
- **Error:** An error is the part of a system state which is not performing correctly that may lead to a failure.
- **Failure:** When the system is adjudged to not be offering correct service.

As a result of a system of systems approach, failures of the components of a system can be perceived as faults to that system. This is useful because we can begin to conceptualise how failures can be compartmentalised, and allows us to model the failure of systems in stages rather than one monolithic entity.

2.3.2. Adversarial Aspect of MAFTIA

Powell and Stroud (2003) refine the ideas described in dependability by further exploring the dependability of systems in the presence of a malicious attacker. Specifically, MAFTIA is interested in applying what is defined as *Intrusion Tolerance* to intrusions. These MAFTIA concepts can be used to breakdown attacks on a system to different stages. We introduce a number of key terms from MAFTIA which can be used to understand attacks.

- **Attacker:** Malicious person or organisations at the origin of attacks.
- **Vulnerability:** A fault that is created during the development or operation of the system that if exploited, causes an intrusion.
- **Attack:** A malicious interaction fault that attempts to exploit a vulnerability. Can be thought of as an intrusion attempt. It is important to note that sometimes we use the term *Attack Event* to refer to an attack when being specific to the attack stage in MAFTIA. This is because literature in general tends to refer to an attack as the entire sequence of events that takes place when an adversary attempts to exploit a system. Sometimes there can be some confusion between an attack in the general sense and an attack being the specific attack stage in MAFTIA so in these cases the term attack event is used.
- **Intrusion:** An attacker-introduced fault. An intrusion is created as the product from an attack successfully exploiting a vulnerability by an attacker.

2.3.3. Security Policies, Goals, Rules

Security policies relate to an organisations general approach in dealing with the confidentiality of information, integrity of information, and availability of a system. There are several well known security policies, which can also be based on security policy models. Well known security policy models include the Chinese Wall, as detailed by Brewer and Nash (1989), for dealing with the confidentiality of a business who may receive services from a firm which also provides services from a different business which they are competing against. Another well known security policy model is Bell-Lapadula, as detailed by Bell and La Padula (1976), which is a model for dealing with the confidentiality and integrity of information by designating a clearance classification to information, for example, unclassified, confidential, secret, and top-secret.

MAFTIA, according to Powell and Stroud (2003), views security policies as a composition of security goals and rules over several layers of abstraction. Goals capture the high-level security requirements for a system. A violation of a security goal corresponds to a security failure. Rules

are lower level constraints (such as minimum length of passwords) which are intended to imply the goals. A violation of a security rule corresponds to an error in a system.

Security goals in MAFTIA are stated to be in terms of CIA. However, the subject of the CIA terms can be different. We can talk about the integrity of system data, or the integrity of user data as two different things but are still stated in terms of CIA. MAFTIA considers how many different security properties can be expressed using CIA such as privacy, authenticity, and non-repudation. However, CIA is not the only way of expressing security properties. For instance, Cremers and Mauw (2012) considers secrecy and authentication as security properties. We generally consider attacks with the MAFTIA model in mind and as a result of this we use MAFTIA security properties which are CIA.

Security rules are more specific and should ideally imply the goals. The goals of a security policy should be guaranteed if all the rules are complied with. A security rule for passwords in a FIM system could be that all passwords are not shared online. If a user shares their password online, a security goal like the user account information is confidential is not immediately broken, but it becomes much more likely.

2.3.4. Intrusion Propagation through A Systems of Systems approach

Powell and Stroud (2003) describe an approach where failures on one system can be seen as intrusions on another system. An example is given, shown in fig. 2.3, of an authentication and authorisation system which is part of a larger system. We can see through this example that failures on one system can be seen as vulnerabilities, attacks, and intrusions on other systems. This is a conceptual model since MAFTIA aims express that when a failure occurs on one system another failure might be possible at the next system. In this thesis, we focus on how intrusions can be harnessed by an attacker to cause further attacks and this is the key difference of focus for this thesis.

We aim to model exactly what the specific security issue is that allows further attacks to be possible. In that sense, rather than failures on one system appearing as intrusions on another system to become more errors and failures, these intrusions can become possible conditions for other attacks. For instance, a malicious script intrusion placed on a server could be used to stage other attacks. Sometimes, the intrusions caused can also become new systems which is an idea unexplored by MAFTIA. As an example of systems created through attacker intrusions consider how redirects could be caused by an attacker. The redirect could be modelled as a connection system between two other systems. Or, consider how an attacker could prompt a login by a user. Intrusion propogation is an important concept for this thesis, but it works differently to MAFTIA in that intrusions are leveraged to cause other attacks rather than just becoming a sequence of errors and failures leading to more intrusions.

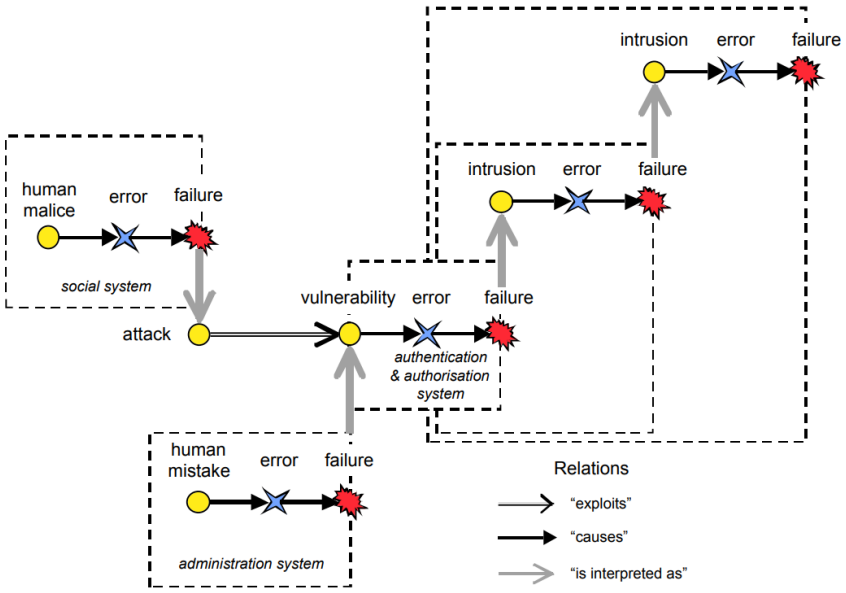

Figure 2.3 MAFTIA Intrusion Propagation according to Powell and Stroud (2003)

2.4. Logical Programming

The field of logical programming is vast in applications ranging from automatic mathematical theorem proving to natural language parsing. In security logical programming has been used to automatically generate attack graphs as seen in Ou et al. (2006). Additional uses include code safety guarantees through Proof-Carrying Code, which guarantees the safety of executing certain code as is proposed by Necula (1997). There is also some evidence for the use of logical programming in FIM already, with Gomi (2010) showing how logical programming can be used to establish the trustworthiness of a user's identity as propagated through multiple entities. In this section we focus on the aspects of logical programming that we use for modelling FIM systems.

Specifically we use the programming language Prolog as an implementation of a logical programming approach. According to Lloyd (2012) (page 2), one of the main ideas of logical programming is to separate the *what* of the problem from the *how* which is useful for our approach of modelling escalating attacks in FIM. We can specify the non-logical symbols representing the *what* aspects of the FIM system such as the different kinds of attacks in FIM and Prolog handles the logical *how* we can establish those attacks as true. In other words, the different attacks in FIM are specified, and Prolog decides how we get there.

The purpose of this section is not a full theoretical background to Prolog. The purpose of this section is to describe what Prolog does in the context of this thesis. For a more comprehensive theoretical overview of logical programming we recommend Lloyd (2012), which we use to boil down the field of logical programming into what is useful for this thesis. Lloyd describes logical programming languages in general. We will tailor what is described by Lloyd to be applicable to Prolog, which is what we use for the logical analysis of FIM systems later in this thesis.

⟨*constant*⟩ ::= ⟨*lcLetter*⟩ | ⟨*digit*⟩ | ⟨*constant*⟩ ⟨*letter*⟩ | ⟨*constant*⟩ ⟨*digit*⟩

⟨*lcLetter*⟩ ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

⟨*variable*⟩ ::= ⟨*ucLetter*⟩ | ⟨*variable*⟩ ⟨*letter*⟩ | ⟨*variable*⟩ ⟨*digit*⟩

⟨*ucLetter*⟩ ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

⟨*term*⟩ ::= ⟨*constant*⟩ | ⟨*variable*⟩ | ⟨*function*⟩

⟨*terms*⟩ ::= ⟨*term*⟩ | ⟨*terms*⟩ , ⟨*term*⟩

⟨*function*⟩ ::= ⟨*constant*⟩ (⟨*terms*⟩)

Figure 2.4 BNF of a constant, variable, function, and term in Prolog

Furthermore, we do not describe the full specification of Prolog. In particular, the syntax we describe is not a perfect representation of valid syntax for Prolog. Instead we provide a background to the syntax which will be used throughout this thesis. Specifically, we specify a non-logical signature as a formal language which operates on the logical basis of Prolog.

2.4.1. Prolog

Prolog has clear basis in first-order logic however Prolog does many things differently to first-order logic both in terms of notation and in expression. Note that this this thesis does not focus on first-order logic, and first-order logic is not important in this thesis beyond stating that Prolog has its roots in first-order logic. Prolog is not strictly completely within first-order logic. For example, in Prolog we can use predicates within predicates which is not defined in first-order logic. $P(x, Q(y))$ is not a valid expression in first-order logic. The notation of Prolog can confuse those who are more familiar with first-order logic (for example, predicates in first-order logic are expressed with an uppercase, in Prolog with a lowercase). The aim of this section is to clarify the notation and basic concepts as they are actually used in Prolog.

2.4.2. Terms

Terms are made up of constants, variables, and n-arity functions. Constants describe elementary aspects of a logical problem, such as a qualitative measure of the temperature or a name. Variables are a placeholder for another term in logic. N-arity functions are used to describe relationships between terms. A function could be used to describe a paternal relationship between two people, or an internet connection between two servers. Therefore, a term is an inductive structure that can be a constant, a variable, or a function containing other constants, variables, or even other functions. We present a BNF description of terms in fig. 2.4. Note the general assumed definitions in BNF such as letters and digits.

Terms form the backbone of specifying programming structures in Prolog. Next we will introduce facts and rules. Facts establish a base of what is true at the axiom level. Rules seek to present solutions through examining the facts and other rules within a program.

⟨*fact*⟩ ::= ⟨*NoVariableTerm*⟩

⟨*noVariableTerm*⟩ ::= ⟨*constant*⟩ | ⟨*functionNovariable*⟩

⟨*functionNovariable*⟩ ::= ⟨*constant*⟩ (⟨*termsNoVariable*⟩)

⟨*noVariableTerms*⟩ ::= ⟨*constant*⟩ | ⟨*functionNovariable*⟩

⟨*head*⟩ ::= ⟨*term*⟩ :-

⟨*body*⟩ ::= ⟨*term*⟩ | ⟨*body*⟩ , ⟨*body*⟩ | ⟨*body*⟩ ; ⟨*body*⟩

⟨*rule*⟩ ::= ⟨*head*⟩ ⟨*body*⟩

⟨*line*⟩ ::= ⟨*fact*⟩ | ⟨*rule*⟩ .

⟨*program*⟩ ::= ⟨*line*⟩ | ⟨*program*⟩ ⟨*line*⟩

Figure 2.5 BNF of a rule, line, and a program in Prolog

2.4.3. Facts, Rules, Lines, and Prolog Programs

A fact is simply a term which is asserted as true in Prolog. A fact can be viewed as an axiom in a specific problem area. Since a fact is asserted as being true, there can be no variables in a fact as then the fact would be ambiguous.

A rule is used to describe if a term is true by evaluating the truth of other terms. A rule consists of a head, which is a term that the rule is attempting to prove the truth of, and a body which is a number of terms which can be combined by logical connectives. At the end of the head of a rule is a colon followed by a hyphen to indicate the end of the head of the rule and the start of the body.

A formula consists of logical connectives connecting terms. We consider logical and, and logical or. Logical and is denoted "," in Prolog. Note that the comma "," is also used within function definitions to separate each parameter in the function. Logical or is denoted ";" in Prolog.

A line of code in a Prolog program consists of a term followed by a full stop or a rule followed by a full stop. A Prolog program is made up of a sequence of lines. The overall syntax for rules, lines, and programs can be seen in fig. 2.5.

Generally we refer to rules in a Left-Hand Side (LHS) and Right-Hand Side (RHS) format. A represents the terms that form the head of a rule. B represents the body of the rule. A rule therefore takes the form $A :- B$ in Prolog. We also use the notation $A \leftarrow B$ when referring to rules in general throughout this thesis.

2.4.4. Unification, Goals, and Backtracking

A term can be unified to another term. We use the concise information provided by Robertson (1998) to outline the process of unification in Prolog. The idea behind unification is that we aim to see if two terms a and b can be matched.

1. If a and b are constants and $a = b$ then unify. Else do not unify.
2. If a is a variable then instantiate a such that $a = b$.
3. If b is a variable then instantiate b such that $b = a$.
4. If a and b are functions with the same arity, find the top-level functor of a and b. If the top-level functor is the same, then take the tuple of arguments $(a_1, ..., a_n)$ of a and the tuple of arguments $(b_1, ..., b_n)$ of b. Then $\forall a_i$ in a and $\forall b_i$ in ba_i must unify with b_i.
5. Else do not unify.

A Prolog program is run by inputting a goal. A goal is a term that Prolog will attempt to prove the truth of by evaluating the goal for every line which is a term in a Prolog program. Prolog finds all possible ways a goal can be found to be true through instantiations through a process known as backtracking. The process of evaluating a goal g against a line l in a Prolog program is described roughly through induction as follows.

1. if l is a rule then unify g with the head of the rule. Treat the terms in a body of the rule as a set of goals $\{g_1, ..., g_n\}$. Attempt to evaluate all goals $\{g_1, ..., g_n\}$ and store the results of successful unifications as partial solutions.
2. if l is a term and g can unify with l and that is a solution. Output the solution and all partial solutions that were needed to find the solution.

2.5. Attack Trees

2.5.1. General Background of Attack Trees

Notions of attack trees were first discussed by Schneier (1999). The idea behind an attack tree is an intuitive way of modeling risk for a particular system which can be traced back even further to fault trees, which are a method for analysing how a failure to a system could occur using a systems of systems approach according to Vesely et al. (1981). Intuitively, the idea is that we consider a failure of a system as a fault to a larger containing system. The essence of an attack tree is in refining attack steps from a high level goal to specifics on how we achieve that goal. The nodes that specify how a particular attack step is reached are said to be children to that attack step. All attack trees have a root node which represents the ultimate goal for an attack. If a node has no children, then it is a leaf node.

For instance, Schneier introduces a physical safe example and the various ways in which this safe could be opened maliciously. We can assign the open safe root node with more specific attacks (children) of opening that safe like pick the lock of the safe, learn the safe combination, cut the safe open, or we could install the safe improperly. These attacks could in themselves be broken down into further children that would be required in order to achieve the goal. For example, if we wanted to learn the combination, we could try and find the combination written

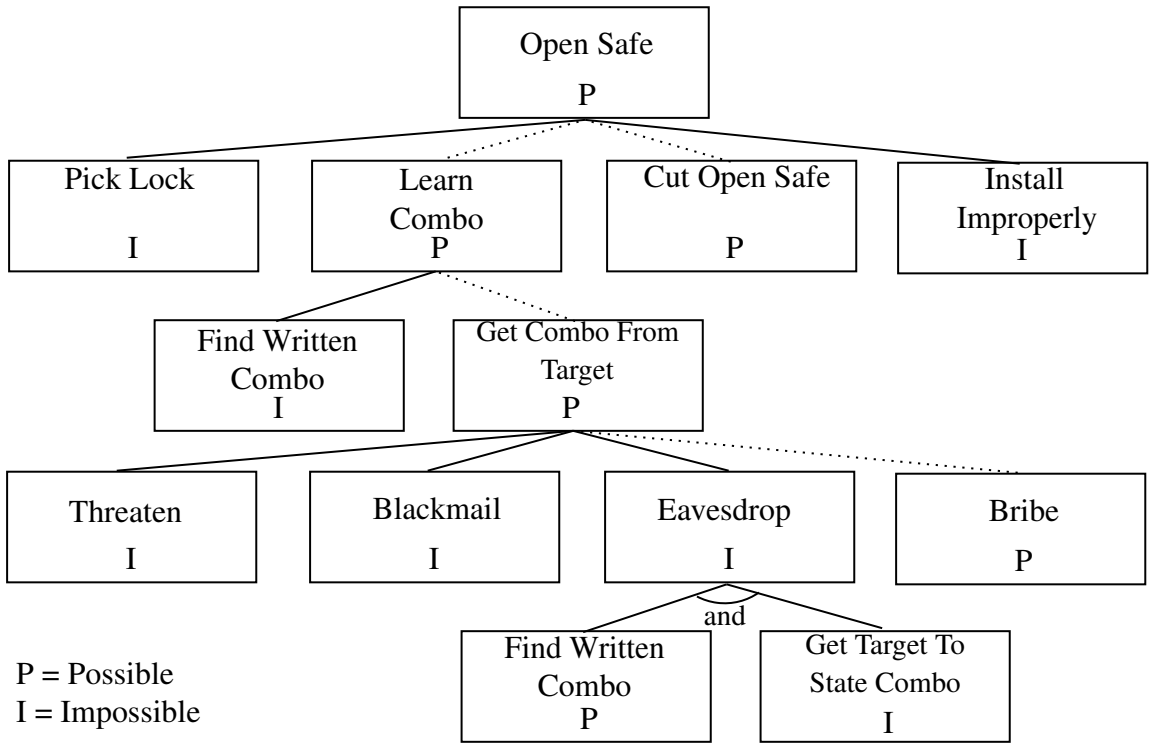

Figure 2.6 Attack Trees with Assigned Possible or Impossible Values According to Schneier (1999)

down somewhere or find out the combination from a person who knows the combination. These attacks could be broken down more, and more, and more until we reach a point in which we choose to not break the attack down anymore.

One benefit to using attack trees is the resulting analysis that can be performed using the attack tree. Schneier describes how values can be assigned to nodes in the tree in order to perform calculations on the risk of a particular system. In particular we refer to how Schneier describes some attacks being possible and other attacks being impossible. We might decide that since the safe has already been installed, it is no longer possible to install the safe improperly. We might be unsure on if we can learn the combination, to try and calculate if learning the combination is possible we can investigate the children for learn combo until we reach the leaf nodes in the tree and decide whether those leaf nodes are possible or not. Any non-leaf OR gated node in the tree is said to be possible if any of its children are possible. Any non-leaf AND gated node in the tree is said to be possible if all of its children are possible. This full example provided by Schneier can be seen in fig. 2.6.

Another possible way of analysing attack trees that Schneier introduced is cost analysis. In this thesis, cost analysis is used to demonstrate application for our model. First, some kind of cost metric is assigned to leaf nodes. Schneier assigns a monetary cost for the attacker to execute the attack. The idea is that by propagating these values up the attack tree the cheapest option for the attacker can be considered. For OR nodes, the cheapest node is considered. For AND nodes, the sum of their children is considered. The cheapest attack for the attacker can then be seen as per the example in fig. 2.7.

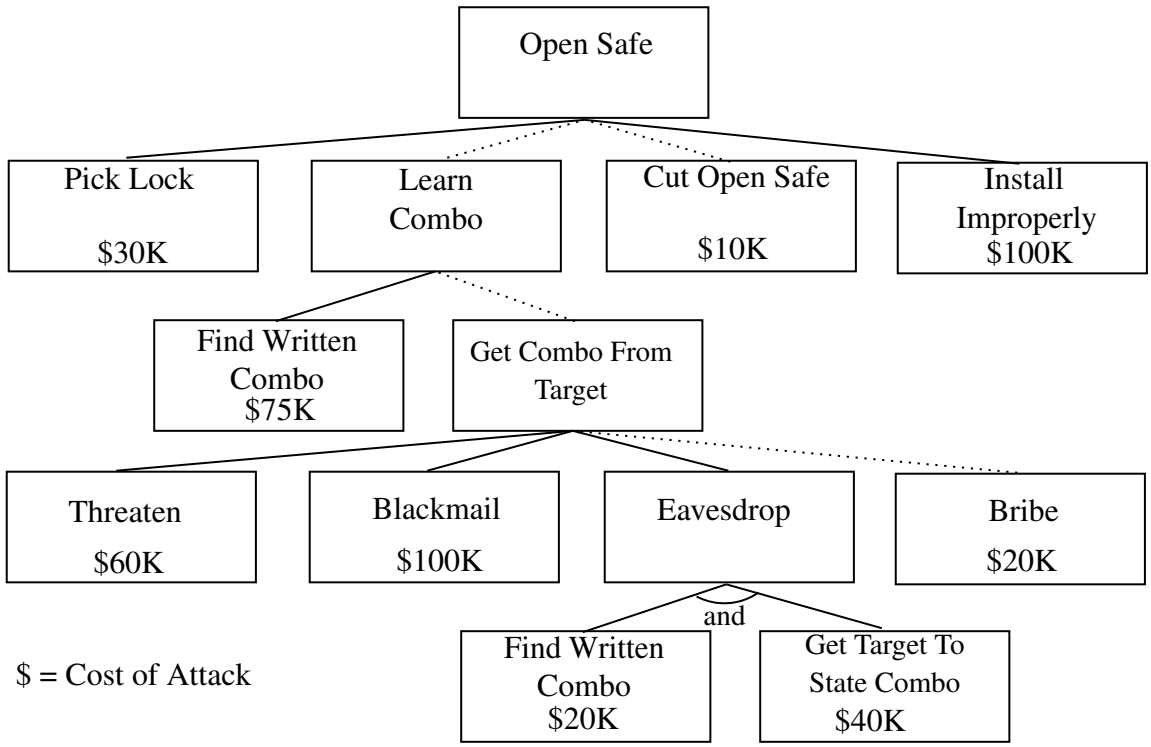

Figure 2.7 Attack Trees with Appended Cost and Cheapest Attack According to Schneier (1999)

The overall idea of attack trees is so intuitive that the approach described by Schneier was not formalised until 2005. Mauw and Oostdijk (2005) describe attack trees formally. Since then, extensions to the formalism have been considered such as attack-defence trees by Kordy et al. (2010).

Bundle: One important concept from the approach by Mauw and Oostdijk is the idea of a bundle. A bundle defines a a set of children nodes which must be true for a parent node to be true. A node can have any number of bundles, where bundles consisting of more than one node indicate conjunction relationships between the nodes within that bundle. Representation of disjunction in the tree is made by considering the different bundles that exist for a node. Each bundle represents a possible option for an attack to progress to the parent node. More formally, consider a universe of all possible attacks U. Every node n in an attack tree is in the set U such that $n \in U$. A given node n has any number of bundles (zero if that node is a leaf node) where each bundle is a set of attacks A_i such that $A_i \in U$. Each individual attack $a_i \in A$ must be true for n to be true where A is a bundle for n. Only one A_i in the bundles for n needs to be true for n to be true.

Attack Trees in FIM: The idea of using attack trees in FIM is unexplored to the best of our knowledge. We observe that attack trees intuitively make sense to capture escalation of attacks in FIM because attack trees provide an abstraction of attacks leading to the possibility of other attacks. The question becomes: how do we synthesise attack trees for FIM?

2.5.2. *Attack Tree Synthesis*

Attack trees have been manually synthesised in the past. For example, Fraile et al. (2016) uses a taxonomy on attacks regarding ATM machines. Attacks trees are synthesised manually from this taxonomy for ATMs in general. The problem with this is the general attack tree does not relate to a specific ATM. There could be specific vulnerabilities for that ATM that make the ATM more vulnerable to a specific type of attacks, for instance the ATM could lack physical security. Manual synthesis could be considered for each ATM that requires security analysis. However, the manual synthesis of attack trees for every possible ATM seems like a time consuming process.

The automation of graphical attack models is a large area of research. Related to attack tree generation is attack graphs which also graphically model security threats similar to attack trees. Automated generation of attack graphs is something that has been studied by Sheyner et al. (2002), Ingols et al. (2006), Ou et al. (2006). Both attack graphs and attack trees are valid ways of determining threats to a system and are similar in function. Attack graphs are an inductive method of reasoning about attacks, we start at some initial state, and we consider the reachability of some adversary goal. Attack trees are a deductive method of reasoning about attacks, we start at some goal and then we define attack states that could lead to that goal recursively until we get to what we consider an atomic state that does not require further investigation (a leaf node). Sheyner et al. specifically refer to the synthesis of attack graphs being tedious, error-prone, and impractical especially in reference to the more complex the attack graph gets.

The idea of attack graph synthesis being error-prone and impractical is also echoed in attack tree synthesis by Vigo et al. (2014), who considers the automatic synthesis of attack trees using a process algebra approach. Birkholz et al. (2010) synthesises attack trees from a vulnerability database. Gadyatskaya et al. (2017) considers the automatic synthesis of attack trees where meaningful levels of abstraction exist in intermediary nodes, which are claimed to be lost in most automatic attack tree generation. We can see that automatic synthesis of attack trees is an active field of research, and we aim to apply this approach to FIM to analyse escalating attacks in FIM.

2.5.3. *Attack Tree and Trace Formal Definition*

We now introduce a more formal definition of attack trees which is sourced from the literature. We provide a formal definition to clearly define our approach.

Notably, Mauw and Oostdijk (2005) provides a rigorous definition for attack trees. This definition is weighty and comprehensive to encapsulate theoretical problems with attack trees, like how structurally different attack trees can semantically mean the same thing. This thesis does not focus on these theoretical problems, so having this strong formal definition is perhaps not the most appropriate definition for this thesis.

Gadyatskaya et al. (2017) also focuses on a theoretical problem regarding attack trees. The problem being addressed is how to generate refinement-aware attack trees. Trees which are

refinement-aware are said to be semantically appropriate at all levels in the tree, a trait which is useful for attack trees but is often lost when attack trees are automatically generated. While this thesis does not focus on this formal refinement-awareness property, the definition for attack trees being provided is useful.

The definition provided by Gadyatskaya et al. (2017) focuses on a logical SAND operator. While this is useful for consideration of the refinement-awareness problem, it is not necessary for this thesis so we will modify the definition to not consider this logical SAND operator.

Formally, $\mathbb{B}$ denotes a set of actions. Logical $\{\text{AND}, \text{OR}\}$ are two unranked associative and communicative operators. An attack tree t is then expressed as a signature $\mathbb{B} \cup \{\text{AND}, \text{OR}\}$ over the grammar below for every b in $\mathbb{B}$:

$$t ::= b \mid b \triangleleft \text{OR}\ (t, ...t) \mid b \triangleleft \text{AND}\ (t, ...t)$$

Each action describes an attack, for instance we could describe a sub-tree of fig. 2.6 (sans the cost of an attack) with the root node set as "Get Combo From Target":

$$GetCombo \triangleleft \text{OR}\ (Blackmail, Eavesdrop \triangleleft \text{AND}\ (Listen, State))$$

Also of note in this thesis is an attack trace, which is used as an intermediary data structure. Referring to the formal definition of an attack tree, then an attack trace is simply a modification of the grammar:

$$t ::= b \mid b \triangleleft \text{AND}\ (t, ...t)$$

This syntax is not necessarily used to describe attack trees and related structures used in this thesis because we do not focus on the formal properties of attack trees or attack traces. However, this syntax is useful to make clear exactly what these two data structures are.

Chapter 3. A Survey into Security Analysis in FIM

Abstract

We conduct a survey of security analysis in Federated Identity Management (FIM). A taxonomy based approach is applied to attacks in FIM inspired primarily by the Malicious and Accidental Fault Tolerance framework (MAFTIA). When security issues are categorised, we can paint a picture of the landscape of problems that have been studied in FIM. We outline the security issues that are happening across FIM protocols and present solutions to those security issues as proposed by others.

3.1. Introduction

Protocols in FIM have been analysed by others in an attempt to find security problems. The issue is that there is a lot of different information on the analysis of security protocols in FIM that remains uncompiled. Our goal is to create a survey for security analysis in FIM. We review existing peer reviewed academic publications that perform security analysis on FIM protocols and implementations to establish a common ground and collect knowledge. In addition, we aim to create a unified way of looking at security issues in FIM and offer a framework to do that. We do this to provide insight on attacks that are seen on multiple protocol suites and state the solutions to security issues as provided by the surveyed papers.

Once we have categorised security issues in FIM, we investigate how to turn these categorised issues into knowledge encodings, which are more specific representations of data used in logical analysis. We shall pick a few of these categorised issues that occur within OAuth 2.0 and show how they can be encoded as knowledge to be used for the modelling the escalation of attacks in FIM.

Publication: The broad categorisation of security issues in FIM is beneficial to the field of computer science as a whole, and has been published as part of the post proceedings in IFIP Summer School 2016. We extend on that paper here and detail changes made. The specific reference to that paper is Simpson and Groß (2016).

3.1.1. Contribution to the field

RQ1: Understanding the Security Landscape in FIM. What is the landscape of the security analysis in FIM? We investigate to what extent FIM faults, errors and failures can be

modelled systematically in a taxonomy inspired by fault-tolerance to understand security issues in FIM.

Also considered are two sub-research questions which assist RQ1 but are narrower in scope:-

RQ1.1: Common Attack Classes and Occurrence. What are the common attack classes and do these attack classes occur across protocol suites?

RQ1.2: Solutions. What solutions are proposed to attack classes?

3.2. Fundamentals

3.2.1. Malicious and Accidental Fault Tolerance (MAFTIA)

We considered Malicious and Accidental Fault Tolerance (MAFTIA) principles as summarised by Powell and Stroud (2003). MAFTIA has already been covered in the background chapter section 2.3.2. Here we mainly state the motivation for using MAFTIA especially in terms of categorising attacks which is the focus of this chapter.

MAFTIA is built upon the foundation of dependability. Consider Avizienis et al. (2001) for a background on dependability in general. The dependability area is concerned with understanding what happens when a system fails in order to apply fault tolerance to avoid a failure. While dependability focuses on accidental failures, MAFTIA adapts the founding notions of dependability for use in understanding how systems fail under the influence of a malicious attacker. Specifically, MAFTIA is interested in applying what they define as Intrusion Tolerance to intrusions. The concepts founded by the dependability community and MAFTIA can be used to understand the different aspects of an attack.

The complete set of terms that are used in MAFTIA (vulnerability, attack, intrusion, etc.) can seem unnecessary and it may be difficult to distinguish the semantics between one term and a different term. The need for these concepts arises when attempting to understand attacks on systems formally, which is required for a taxonomy on these attacks. For the taxonomy, these concepts allow for understanding of the different aspects of an attack that contribute to an overall failure.

3.2.2. Good Taxonomies

We base the core properties of the taxonomy we introduce with properties established by Lindqvist and Jonsson (1997), who also considered what a good taxonomy for computer security issues would look like. We collect observations made about other literature which have been observed by Lindqvist and Jonsson. For instance, the philosophy noted by Landwehr et al. (1994) explains that a taxonomy is not just a collection of data but a system for identifying specimens which are alike and which specimens are not alike. This notion is important because sometimes an attack might not be identical to another attack, but it is suitably close to be

categorised as the same general issue. This helps us avoid getting too tangled up with details and having too many categories. Onto the properties Lindqvist and Jonsson identifies:-

- Categories should be mutually exclusive. Specimens should only fit into at most one category.
- Every category should have clear descriptions of what belongs in that category.
- Categories should be useful not only to experts but users and administrators also.
- Terminology used for categories should conform with existing security terminology.

Good Taxonomy: We aim to compare our taxonomy and specimens within the taxonomy against the properties introduced by Lindqvist and Jonsson (1997).

In addition, Lindqvist and Jonsson (1997) considers several properties introduced by Amoroso (1994). Completeness, which is that the taxonomy should be able to describe the complete scope of attacks for the system considered. Appropriateness, the taxonomy should consider constraints relevant to the system; for instance, in FIM, the protocol suite should be considered as that is highly relevant to the issue. Finally, external vs internal threats. Our taxonomy does not directly deal with external vs internal threats. This is because sometimes this can be a gray area. For instance, in a session swapping issue described by Sun and Beznosov (2012) an attacker controlled account is already required as part of the attack. Alternatively, in the MITM attack described by Pfitzmann and Waidner (2003a) where a dishonest SP is considered; the dishonest SP does not necessarily need to be in the federation but the attack would be easier if it was. Also consider that most of the time the IdP is considered as internal but Mayer et al. (2014) considers FIM explicitly when the IdP is the external threat. In this case, when categorising attacks the line of thinking would be viewing the IdP as an external threat while normally an IdP is considered as internal. The aim is to avoid inconsistencies as much as possible and considering external vs internal threats could interfere with this.

3.2.3. Differences from Published Work

The fundamental difference between the work we have published in Simpson and Groß (2016), and what is presented here, is that in the published work we were mainly concerned with the broad categorisation of threats in FIM. In this thesis, we are concerned with the escalation of attacks in FIM which requires a specific attack knowledge encoding. This difference in thinking has necessitated extensions to the original work. This extension is described in 3.9. This extension forms the basis of subsequent chapters because we use the extension as a logical basis for modelling escalating attacks in FIM.

There have also been numerous other corrections made in the survey itself. For instance, in going over the survey it has become clear that some issues observed actually warrant an

additional category. As an example, Yang et al. (2016) considers a dual-role IdP as a vulnerability. Previously, this was described as a vulnerable IdP, but on further consideration this is too general and fails to categorise an interesting vulnerability that is suitably differentiated from other vulnerabilities. Other corrections include correcting outright mistakes, such as, in the original survey all DoS attacks are attributed to DDoS attacks. This is clearly wrong as a DDoS is a specific type of DoS attack. The hope is that these corrections make for overall higher quality research.

In addition, we also expand on the original work because we are not under such a strict page limit and can go into more detail about the categorised security issues. In this expansion, we can be much more descriptive of the process for categorising each individual specimen.

3.3. Related Work

3.3.1. FIM Surveys

Van Delft and Oostdijk (2010) presented a collection of security issues that exist in OpenID. There is an additional need to examine security issues across FIM which is what we aim to do. There have been attempts to survey FIM in general. For instance, Ghazizadeh et al. (2012) survey issues within OAuth, OpenID and SAML. We recognise limitations with the survey provided by Ghazizadeh et al. and attempt to address these limitations with our own survey:-

- The survey has no systematic paper selection method. The method in which Ghazizadeh et al. selected papers is not clear, so it is difficult to gauge the thoroughness of the survey.
- There is no systematic presentation of the findings. With our survey we aim to address this by introducing a taxonomy so that the security issues in FIM can be more easily analysed and understood.

3.3.2. FIM Taxonomies

Arias-Cabarcos et al. (2012) considers a taxonomy for risk on FIM. This taxonomy is concerned with measuring risk on FIM. Interestingly, Arias-Cabarcos et al. consider two different phases of FIM. The phase when the federation is being setup, also known as the bootstrapping phase, and the post-federation phase when the federation is actually serving as an authentication system. The main difference that the taxonomy introduced in this thesis is that we attempt to break down attacks into stages using MAFTIA. Arias-Cabarcos et al. focuses on high-level security properties without reference to the causality. Also, in this work we apply the survey to an extensive selection of academic papers of FIM analysis, while Arias-Cabarcos et al. is limited to a specific application scenario introduced.

3.3.3. *Attack Categorisation*

We aim to categorise attacks so that we can paint an overall picture of threats in FIM. We have already discussed what makes a good taxonomy in subsection 3.2.2. This subsection is more specific in that we are going to be looking at related work in attack categorisation and seeing how we can directly apply their taxonomies to be used for attacks on FIM. Since FIM is a type of computer network, we are mostly interested in attack categorisation that can be applied to computer networks.

Surprisingly, there has been little direct attack categorisation for network attacks in recent years. Lindqvist and Jonsson (1997) describe a taxonomy for detailing intrusions for computer systems in general which has already been discussed in subsection 3.2.2, as several notions on what makes a good taxonomy is introduced. Howard and Longstaff (1998) describe a common language for talking about security issues which includes concepts for considering issues in different stages, like the vulnerability involved and the action used to exploit that vulnerability. The issue is, is that this common language is not suitable for FIM systems and is more general. Bishop (1995) describe a taxonomy for categorising vulnerabilities on UNIX systems. The taxonomy has six dimensions which are referred to as an axis but not all of these dimensions are suited to considering FIM. For example, the exploitation domain refers to specific UNIX file structures.

More recently, Cambiaso et al. (2013) introduces a categorisation for DoS style attacks on cloud computing solutions, and we consider cloud computing solutions as a type of computer network, but it is limited in that only DoS attacks are considered and we are looking for a more generalised solution than the solution presented by Cambiaso et al.

A general network attack classification is presented by Hansman and Hunt (2005). The taxonomy consists of four dimensions. The first dimension covers what the attack behaviour is. The second dimension covers the target of the attack. The third dimension covers what vulnerabilities are exploited. The fourth dimension covers the payload for an attack. As part of the taxonomy, a case study is included which applies a number of well known attacks to the taxonomy. This is done primarily as a way of validating that the taxonomy is suited to purpose, but also serves as an example for understanding how the taxonomy works. Hansman and Hunt provide useful direction, particularly because the fourth dimension can consider attack escalation as the payload could contain another attack.

Looking more broadly at attack categorisation, we can also consider the Common Weakness Enumeration (CWE) framework and the Common Vulnerabilities and Exposures (CVE) framework. CWE is used to assess critical programming errors that can lead to vulnerabilities. An example for applying CWE is Martin et al. (2011), who builds a list of 25 software weaknesses including missing encryption of sensitive data and missing authentication for critical function. Christey and Martin (2007) conducts a study using CVE to understand research trends using reported vulnerabilities. One benefit of using the CVE framework is that layered onto this

TITLE-ABS-KEY((Analysis OR Evaluation OR Examine OR Proof OR Attack OR Intrusion OR Vulnerability OR Risk) AND Security AND Identity AND (OAuth OR OpenID OR "Liberty Alliance" OR SAML OR "Security Assertion Markup Language" OR WS-Federation OR "Microsoft Passport" OR (Passport AND Protocol) OR Cardspace OR "Facebook Connect" OR "Google Accounts" OR Shibboleth)) AND (LIMIT-TO(SUBJAREA, "COMP"))

Figure 3.1 Overall search term used in Scopus, modulo plural forms.

Protocol Security Identity Analysis OR Evaluation OR Examine OR Proof OR Attack OR Intrusion OR Vulnerability OR Risk

Figure 3.2 Overall search term in Google Scholar, modulo plural forms.

is the Common Vulnerability Scoring System (CVSS) which can quantify security impacts. For example, Gallon and Bascou (2011) outlines modifying attack graphs to include CVSS score. The problem with using these frameworks for security analysis in FIM is that researchers analysing FIM do not typically associate a named CVE/CWE to the security issue, making the application impractical for this work. In the future, FIM researchers could link uncovered security issues with these well known broad attack categorisation frameworks.

3.4. Method

3.4.1. Search Methodology

We base our search methodology on a method sourced from Kitchenham (2004). We are specifically interested in the inclusion/exclusion criteria as this provides a way of systematically selecting a sample of papers to be considered in our survey. We used two search engines to perform the literature search: Scopus and Google Scholar. An initial scoping study was performed, which is suggested by Kitchenham, to determine appropriate sources. We found that Scopus and Scholar were fruitful in the studies they contained, with other sources, such as IEEE, bringing up studies that were already found using Scopus and Scholar. Fig. 3.1 contains the overall search term, in the Scopus format.

We observed a number of key words being used and used synonyms. We based the protocol searched for on a survey performed by Pfitzmann and Waidner (2003b) which considers FIM protocols in general. We needed a similarly systematic search in Google Scholar. In fig. 3.2 is a part of the resulting search term. We executed the search term with the "search by relevance" radio option selected and the "where the words appear anywhere in the article" radio button selected, so that the entire article content is considered which turns up more results. No limitations on year range were specified. *Protocol* as can be seen in the template search in fig. 3.2 is substituted with the twelve FIM protocol terms that can be seen in the Scopus search. We note here how not all of these protocol terms are strictly just protocols, but are also standards (like SAML), or also organisations (like Liberty Alliance). For simplicity, we just refer to them as protocols. Also note that these protocols are covered generally in section 2.2 but especially in subsection 2.2.3.

3.4.2. Exclusion and Inclusion

Exclusion: We excluded papers returned from the search based on the following criteria:-

- Excluded papers not in subject area of computer science.
- Excluded papers that do not refer to FIM protocols. Papers that do not cover FIM protocols which somehow appear in the search are immediately discarded.
- Excluded papers not part of a peer-reviewed venue (i.e., workshop, conference or journal).
- Exclude duplicates: some papers are published somewhere, and then a very similar version of that paper can appear from the same authors at other venues.
- Exclude papers not written in the English Language.

Inclusion: After the exclusion phase, we included papers based on the following criteria:

- Secondary sources are not included (hence, constrain the review to original research).
- Only papers that offer a combination of vulnerabilities and anticipated exploits thereof are included. Claiming a vulnerability is not sufficient.
- We do not include hypothetical analyses, which modify a standardised FIM protocol to conduct a "what-if" analysis. We aim to collect security issues for real FIM systems and not for proposed extensions which may or may not be acted upon.
- We do not include security issues that are put forward for FIM in general. The security issue has to be specific to a certain FIM protocol. The reason for this is to ensure that we get a representation of security issues possible on real FIM systems.

3.4.3. Our Taxonomy

After reviewing the various attack classification systems already available in 3.3.3, we decided on requirements for our own taxonomy. More on this in 3.6. We also assessed the papers we selected to form the basis of the taxonomy, iterating on the taxonomy as needed. The development of the taxonomy is informed by the papers selected. Further on in this sub-section we can consider how the requirements are met by the taxonomy. First of all we state that our taxonomy should follow the properties of a good taxonomy outlined in 3.2.2. Our taxonomy be able to capture the:-

- Sequence of steps an attacker takes in compromising a system.
- Vulnerability in a system that allows an attack to occur.
- Impact an attack has on a user.
- Target FIM protocol the attack occurs on.

- Essence of whether a FIM protocol or implementation itself has been attacked.
- Proposed solution of an attack, if any.

Hansman and Hunt's taxonomy provides a useful direction for attack classification. At first glance, this taxonomy seems good for modelling escalation of attacks, namely because payloads for attacks can actually be further attacks by themselves. However, there is a problem in doing this which is described by Hansman and Hunt in the conclusion of the paper. When attacks are described in the literature, the lines are blurry between what could actually be considered a sub-attack in and of itself as part of a larger attack. In essence, it is hard to ascertain from the literature on attacks on FIM all of the steps in an attack start to finish due to the varying levels of detail provided from authors on attacks. For example, Sun and Beznosov (2012) specifies how a force-login CSRF attack is possible. The different ways in which this attack is possible is through first injecting malicious code into an email, hosted on a malicious website, or injected onto benign websites through XSS or SQL injections. However, these options are troublesome when we apply it directly to Hansman and Hunt's taxonomy because we would have all of these different attack entries where the payload is the CSRF force-login. Instead, we decide to be more broad so that the general idea of the attack is captured in one entry in the taxonomy.

We also took direction from MAFTIA concepts. We use the notions founded by the dependability community to understand and thereby categorise security issues in FIM. While MAFTIA terms help us understand attacks, they are too low level to be used for categorisation. This is for a very similar reason as is described above. Authors often do not provide enough detail to use the MAFTIA terms directly. It is not always clear what the attacker does at every stage of the attack and how this impacts the system. Another problem occurs with categorisation in MAFTIA, where other authors describe attacks in so much detail that categorisation would be difficult as so many MAFTIA concepts are used to describe a single attack. For example, if an author describes the action of logging in to a server with a user's credentials, this is technically an attack event as the attacker is acting on the system. Other authors however, might not actually describe the process of an attacker harvesting user credentials and using them to login to a system, although it is reasonable to assume that this would happen. We view the disparity between authors in the level of detail provided in describing attacks as the biggest challenge of data collection and attack categorisation.

MAFTIA ultimately helps us code attacks by logically separating an attack into vulnerabilities, attack events, intrusions, errors, and failures. When an author describes an attack, we can interpret the language they use in MAFTIA terms. This allows us to interpret, conceptualise, and finally categorise attacks that are investigated across different kinds of attacks in FIM.

Hansman and Hunt's and MAFTIA both have some kind of concept for what a vulnerability is in an attack. MAFTIA has the notion that a vulnerability is a the broad location of a weakness or flaw in a system that is the root cause of an attack. In a systems of systems approach, it is the lowest sub-system which an attack (or series of attacks) occur. Hansman and Hunt's taxonomy

focuses on specific CVE named vulnerabilities. When this is not possible, the general vulnerabilities are associated to configuration, implementation, and design vulnerabilities. Generally speaking, Hansman and Hunt perform their case study for their taxonomy on well known attacks which have typically been associated to a named vulnerability in the CVE database.

We prefer the MAFTIA approach to naming vulnerabilities. This is because the attacks we observe in our survey are mostly not attributed to specific CVE vulnerabilites. The alternative would be to generally capture the vulnerability to the implementation, configuration, or design like Hansman and Hunt do but this approach would be too general as vulnerabilities in the categorisation system could be significantly different to the vulnerabilities in practice.

The philosophy of Hansman and Hunt's taxonomy is slightly different to what we aim for in terms of impact of attacks. We aim for our categorisation system to make it clear when a user is impacted. If there is no CIA impact on the user, then the attack cannot be categorised in our taxonomy. Hansman and Hunt's taxonomy is concerned with how attacks impact a system through the fourth dimension, which is a payload. This payload does not contain any additional information about how a user can be impacted. MAFTIA associates failures with how attacks impact a user's CIA. Since the majority of literature in attacks on FIM are user orientated, it makes sense to also adopt this approach.

Both of these frameworks also do not fully take into account one requirement that we are interested in. We are interested in what FIM protocol the attack targets. This can to some extent be covered by Hansman and Hunt's taxonomy through the second dimension. The second dimension covers the target of the attack, which could be a protocol. However, another one of our requirements is how attacks impact a user. The impact the attack has on a user could also be covered by the second dimension, but it makes sense to keep separate these different requirements so that it is clear what protocol is targeted and the impact the attack has on the user. We call this separation of the protocol 'protocol targeted' .

The attacks we have observed are divided into theoretical attacks that target a FIM protocol, or an attack that targets an implementation the FIM protocol is operating on. Some also discover vulnerabilities with the protocol, and then attempt to exploit the vulnerabilities on implementations. Hunt and Hansman account for the target of an attack via the second dimension. The target of an attack could be the protocol or/and aspects of the implementation. Hunt and Hansman's taxonomy's second dimension allows for categorisation of targets to be incredibly detailed, going all the way down to cabling, peripherals, and specific implementations of Microsoft Word. To get a lay of the land of attacks on FIM, we are primarily concerned if protocols or/and implementations are targeted. We call this an 'issue type'.

Hansman and Hunt in particular suggest a 'solution proposed' dimension as an extension, but do not include the solution proposed dimension in their formal taxonomy. It is interesting to

analyse the extent of which solutions are presented in the literature. We therefore adopt Hansman and Hunt's proposal for extension.

We therefore introduce our taxonomy for use in categorising security issues in FIM. The taxonomy has six dimensions *Vulnerability , Attack Class, CIA Failure, Target Protocol, issue Type,* and *Solution Presented*. The introduced terms have evolved to capture six different dimensions in a surveyed security issue: What does the attacker do to attack the system (*Attack Class*)? What is the weakness in the system (*Vulnerability*)? How does the security issue impact the user (*CIA Failure*)? What FIM protocol is the subject of the attack (*Target Protocol*)? Is the issue due to an implementation or design flaw (*Issue Type*)? Was a solution put forward by the author (*Solution Presented*)? We use all of these terms to describe surveyed security issues in FIM in subsequent sections. See tab. 3.1 for a summary of the taxonomy.

Taxonomy Dimension	**Description**
Attack Class	A collection of attack events, intrusions, and resultant errors in a system that form a causal chain from vulnerabilities to a security failure if the errors are not dealt with. Considering a causal MAFTIA chain for vulnerabilities, attacks, intrusions, errors and ultimately security failures, the attack class contains the middle of the chain. An attack class is an abstraction of of the attacker event attempting to exploit a vulnerability up until the point of failure. We capture the essence of what an attacker does and abstract away from unimportant details; for example, trivial attack events like the attacker entering a user password. The order of events can vary in specific instances of attack classes. There might be slight variation in the attack events, intrusions, and errors actually occur. However, there is always a critical attack event which becomes the namesake of the attack class.
Vulnerability	A fault that is created during the development or operation of the system that if exploited causes an attack to be possible. The primary cause that makes an attack class possible.
CIA Failure	We judge a system to have failed when the confidentiality, integrity, or availability of a service is violated for a user. We aim to know how a user is impacted by an attack. Also of note is that an account can be compromised by an attack, in this case, the CIA of the service can be affected as an attacker has potential control over the CIA of the user's account.
Target Protocol	The FIM protocol which is being analysed.
Issue Type	Whether the issue is found to be at the protocol or implementation level. If an issue is found to be possible at the implementation level, evidence should be given for this. Otherwise, the issue is a protocol issue introduced at a more high level.
Solution Presented	To what extent a solution was proposed as part of the security analysis. There must be a clear solution presented, or we assume that there is none presented by the paper.

Table 3.1 Taxonomy for Categorising Security Issues in FIM

3.4.4. Data Collection

Scopus allows for some filters to be applied in the search term such that some exclusion can be applied already. For example, using Scopus we can filter by 'Computer Science' which

significantly reduces the number of search results and therefore the amount of papers to be considered for inclusion. We considered 107 papers from Scopus.

Google Scholar returns a very large amount (in the order of thousands) of results for each search term and is more limited in automatic exclusion methods (no subject area for example). We only considered the first 20 results (or two pages) of results for each search term. We considered 14 search terms (one for each Protocol we considered) and therefore 280 papers were considered in total from Google Scholar.

We applied the exclusion criteria manually to Google Scholar. Notably, there were many duplicate papers found from Scopus and Google Scholar. After exclusion was applied, we were left with 145 papers. After applying the inclusion criteria to the remaining we have a sample of 31 papers. These 31 papers can be found in appendix A. From those papers, we manually code parts that describe security issues and weaknesses (e.g., vulnerabilities or attack events). The challenge in data collection and categorisation for attacks in FIM is being able to consistently interpret a wide variety of authors description of attacks. This is the purpose of the taxonomy described in 3.1.

One paper that did not appear in the searches applied is the work by Kormann and Rubin (2000). We added this paper as part of the survey as an other source. To the best of our knowledge, this is the first academic paper critically analysing the security of FIM in the context of relevant risks and attacks on that FIM system. We consider this paper seminal work in the field of FIM security analysis and felt it necessary to include it as an other source. Fig. 3.3 summarises the exclusion and inclusion process.

Coding vulnerabilities proved an interesting challenge, which we will cover here. There are different approaches in which the coding of vulnerabilities can be considered:-

- **Most Clear Vulnerability:** Code the obvious vulnerability which the object of study names. If the object of study is somewhat vague on the vulnerability, consider the general consensus in the literature of what vulnerability is typically attributed to the attack class being described. For instance, objects of study typically do not attribute vulnerabilities for bogus merchant attacks. Kormann and Rubin (2000) first considers bogus merchant attacks for FIM systems and a bogus merchant involves tricking a user into entering fake credentials for a fake website controlled by the attacker impersonating a legitimate website. The vulnerability here is difficult to identify because as Kormann and Rubin discusses, the fundamental issue with the bogus merchant is that the user trusts the web and all of the many components that make the web work such as certificate authorities and domain name servers. To not conflate the vulnerability dimension too much, we can simplify this vulnerability to a general lack of trust infrastructure.
- **Enumerate Explicit Vulnerabilities:** An alternative is to enumerate every vulnerability specifically considered by the object of study. So in the case of the bogus merchant attack

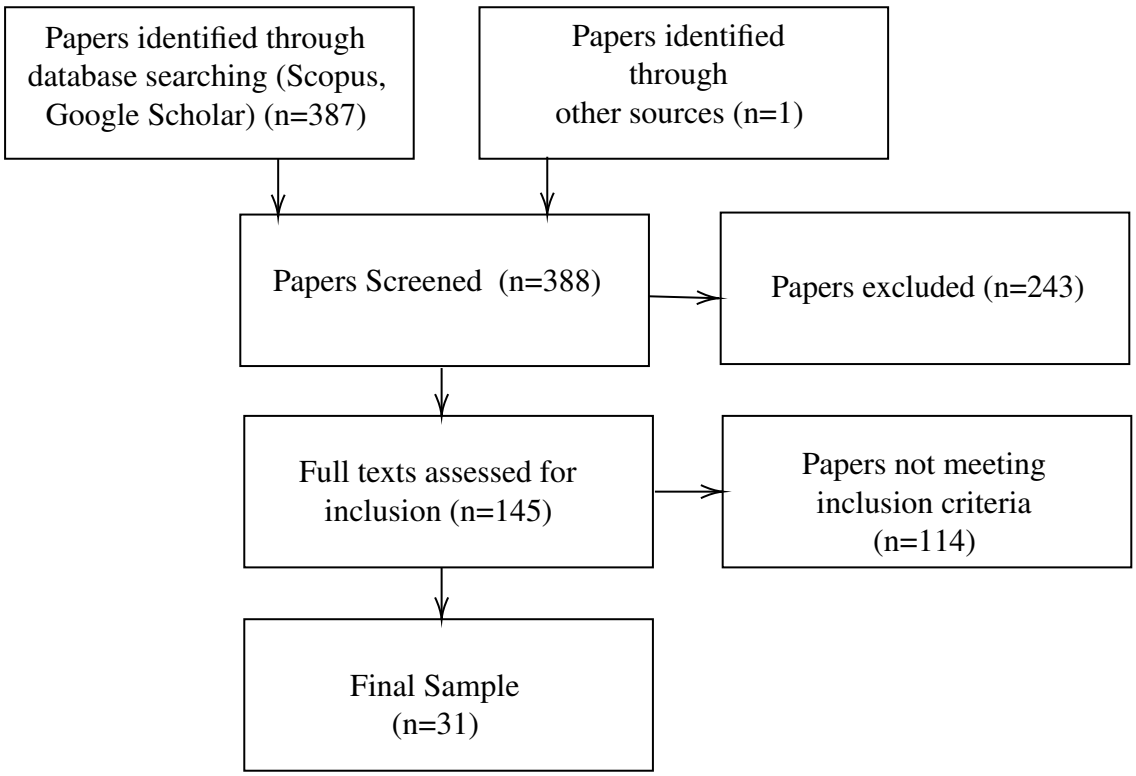

Figure 3.3 Application of Exclusion and Inclusion Criteria

referenced by Kormann and Rubin (2000), we could consider multiple vulnerabilities like the user's inherent trust in the web, weak certificate authorities, and weak domain name servers as these are all made explicit by Kormann and Rubin.

- **Enumerate Implicit Vulnerabilities:** Going beyond the explicit descriptions of vulnerabilities, we can consider all possible vulnerabilities known to cause an attack class to be possible. For instance, even though Kormann and Rubin (2000) does not explicitly consider it, a malicious email could be used by an attacker to get a user to navigate to the bogus merchant. Therefore, we could also attribute possible vulnerabilities to email services also.

There are benefits and drawbacks to each of these approaches. The *Most Clear Vulnerability* approach allows for simplification of a large pool of specific vulnerabilities to a general vulnerability theme. When considering bogus merchants, that theme can be a general lack of trust infrastructure on the web. The main drawback of this approach is that information can be lost. For instance, if two completely different vulnerability themes are considered, we would have to choose only one to remain consistent.

The *Enumerate Explicit Vulnerabilities* captures all of the information that the object of study intends to convey. However, the cost of this approach is that we can have a long list of specific vulnerabilities which could conflate the vulnerability dimension. Furthermore, this could pose a problem for objects of study who focus less on vulnerabilities as results would be heavily influenced by work which is thorough with vulnerability specification.

The *Enumerate Implicit Vulnerabilities* captures all of the information that we know to be possible for a specific attack class. The benefit of this approach is that the most amount of information for a particular attack class. The drawback is that the vulnerability dimension could become conflated. In addition, we could be wrong about the possible vulnerabilities that cause the attack. It also seems likely that we introduce our own biases for including vulnerabilities as we may know more about the causality for certain attacks over other attacks.

For this survey, we employ the *Most Clear Vulnerability* approach. This allows us to generalise vulnerability themes for a specific issue. We accept that sometimes information is lost when more than one distinct vulnerabilities are involved. As we conduct the survey, we consider cases where multiple vulnerabilities can be involved and provide a rationale as to why one vulnerability is chosen over another. We also note in section 3.2.2 the idea that each specimen should fit at most one category is introduced.

3.5. Attack Categorisation

In subsection 3.4.3 we present our taxonomy. We now present a summary of the different values that can appear in each of these dimensions. These values were not pre-determined before we did the survey. We established the different values as the survey progressed. If we found something that was different enough from any existing categories, we established a new category.

3.5.1. Attack Classes

Attack classes provide the signature property of any attack and summarises the actions an attacker takes into one general value. Here we go over the different values an attack class can have as according to our categorisation.

Denial of Service (DoS): A DoS attack class involves the attacker somehow making a FIM service unavailable to a user. In the context of FIM, this type of attack class is described as a particular risk because of the centralised nature of FIM authentication. In the context of FIM, this attack class is described by Kormann and Rubin (2000).

Bogus Merchant: A bogus merchant attack class involves an attacker somehow tricking a user into visiting a website under the attacker's control. This attack class is more broad than a phishing attack in that a bogus merchant could be used to stage other attacks, such as XSS, while a phishing attack is specific to getting a user to voluntarily enter valid user credentials as specified by Fette et al. (2007). This attack class emerged in our categorisation system as a result of the analysis of the work by Kormann and Rubin (2000), which describes the bogus merchant. Since this attack is closely related to phishing, we include phishing attacks in this category.

Man-in-the-Middle (MITM) This attack involves an attacker existing between the communications of users such that the users do not realise that they are actually communicating with the attacker. The attacker is then sending messages on from the user in an attempt to

impersonate the user. In the context of FIM, an example of this attack class is described by Armando et al. (2008). This attack class is relatively common though, and is seen in other works such as Pfitzmann and Waidner (2003a) on analysis of the Liberty protocol.

Domain Name Server (DNS) Poisoning: This attack class involves somehow subverting the DNS so that the entries in the DNS point to incorrect hosts. Typically, this attack is leveraged by getting a DNS to point to a malicious host. Kormann and Rubin (2000) applies this attack class to Microsoft Passport. It should be noted that this attack class is underrepresented in this survey because DNS poisoning tends to assist another, more notable/prominent attack class and so the attack class is described as something else. One of the properties of a good taxonomy of the survey is that a specimen fits exactly one category as described in 3.2.2. For example, Groß (2003) describes a MITM attack which is made possible through also poisoning the DNS, so the overall attack is described as a MITM attack rather than DNS poisoning.

Replay Attack: This attack involves resending an already sent message in a particular protocol exchange. Typically, this is used by an attacker to resend a message containing authentication information in order to get access to a particular resource. This attack can thereby overcome encryption since the attacker does not necessarily need to be able to read the message, the message is simply resent. In the context of FIM, Groß (2003) describes a replay attack for SAML.

Malicious Provider: The idea behind this attack is that there exists a trusted IdP who is somehow able to leverage their position to do something malicious to the user. For example, Mainka et al. (2016) bases attacks found on OpenID on a malicious provider being the cornerstone of attacks. This attack class can sometimes also be used to describe malicious SPs. Oftentimes, the malicious SP leverages their position for MITM attacks, which means the issue would overall be described as MITM. But in some cases a malicious SP does leverage their position to directly compromise a user, and in this case the attack class would be attributed to a malicious provider. For instance, Yang et al. (2016) describes how malicious SPs can compromise a user's account.

Packet Sniffing: This attack class involves the attacker reading information directly from messages and using that information for something malicious. Typically, this attack involves reading messages in transit for a token used to access a user's account. This attack class is described by Sun and Beznosov (2012).

Cross-Site-Scripting (XSS) Attack: The attacker somehow implants a malicious script into a server. This is typically done when a website has a HTML element of which the contents are user controlled and there is not sufficient server side validation to ensure that no malicious code can be inserted. A typical example of such user controlled elements are website guest books where users can leave comments. In the context of FIM, how an XSS can be used is described by Sun and Beznosov (2012).

Message Modification: This attack involves the manipulation of a message by an attacker in some way. This could be to violate the integrity of a protocol run, for example. This is an attack class we realised when we generalise some of the attacks involved in FIM analysis. For instance, Sovis et al. (2010) describes a parameter forgery attack. The kind of parameters described are specific to OpenID, but to make this attack class applicable to other protocols also we generalise it to message modification. For example, Sun and Beznosov (2012) also describes a process where a user can be impersonated through modifying messages.

Session Swapping: This attack class involves the attacker somehow getting the user to switch accounts. This attack class is described in detail in at least two different protocols: OAuth as described by Sun and Beznosov (2012), and OpenID as described by Li and Mitchell (2016). For both of these papers, the session swapping attack is based on the attacker getting the user to login to the attacker's account and hoping that the user performs an action that violates their security. For example, the user might enter credit card information, thinking that they are using their own account when actually, the attacker account is being used. We have not observed a research paper reporting a session swapping attack where an attacker is somehow able to swap to a user session. We do note though, that if this were to occur, it would still fall under this category.

Cross-Site-Request-Forgery (CSRF): The attacker is able to somehow force the user to send a malicious link to a website they are interacting with. This could be done through an email or a website under the attacker's control. For example, a link could be constructed to send funds from a bank account the user has control over to an attacker controlled account. A typical restriction of this attack is that the user must be logged in, as the user would only have control over the account if they were logged in. This attack is possible when malicious actions can be constructed entirely using a link. This attack class is described in the context of FIM by Sun and Beznosov (2012).

Brute Force: For this attack we mean an attempt to brute force login credentials for users. This is terminology we have used to describe any kind of repeated attempt to access a user account by guessing. This includes dictionary attacks, as described by Grzonkowski et al. (2011), which is one of the papers that references a brute force attack.

Account Binding: Unlike many attacks which are possible on other kind of systems this attack class is specific to FIM. The attack involves binding an account on a SP to an IdP account owned by the attacker. This attack is described in detail by Li and Mitchell (2014) who describes a problem with the logic of the protocol. In principle though, we note that a protocol logic flaw does not necessarily need to be exploited for this attack to work. For instance, if there exists some functionality to bind an account on a SP to an IdP, an attacker could build a backdoor to the user's account by linking the SP account to the attacker IdP account. The attacker could then at a later time login to the SP account by using the IdP account. This kind of account linking functionality is present in popular FIM solutions like Facebook and Google so we highlight the risk of this attack class in FIM here.

AssertionConsumerService (ACS) Spoofing: Similar to account binding, this is an attack that is specific to FIM. However, unlike account binding, this is a specific attack class unique to just one paper Mayer et al. (2014). We were unable to generalise this attack class to something that is applicable to different protocols so this is a very specific case of an attack class. The attack is described in detail by Mayer et al. (2014), but in brief, a logic flaw is exploited in the protocol where a valid token is directed to the attacker instead of the IdP.

User Interface (UI) Redressing: This attack involves changing a website to deceptively click on some malicious action but overlay that malicious action with a seemingly innocent action from another website. Similar to ACS Spoofing this attack class is reported on just one paper (the same paper as ACS Spoofing) Mayer et al. (2014). However, unlike ACS Spoofing, UI redressing is a general security issue commonly known as Clickjacking, which is described in detail generally by Portswigger (n.d.). We prefer the terminology of UI Redressing simply because in the context of FIM we only have one sample of this attack occurring and the authors that describe the issue use the terminology UI Redressing.

3.5.2. Vulnerabilities

Vulnerabilities are the identified features in an attack that allow for an attack class to be possible. In this subsection we go over the different values vulnerabilities can take in our categorisation.

Centralised Infrastructure: This vulnerability describes how often with FIM systems an IdP can serve many SPs. The entire FIM system becomes dependent on the IdP and this can cause availability concerns as Kormann and Rubin (2000) point out. This vulnerability can also raise security and privacy concerns though, as a centralised infrastructure could lead to an IdP monitoring individual users as considered by Urueña et al. (2014).

Insufficient Trust Infrastructure: This vulnerability exists in FIM when the user's trust in the FIM system is such that it can be leveraged by an attacker to the user's peril. For example, when a user is lured to a bogus merchant to phish the credentials of the user. This attack works because the user trusts the authentic SP. We observe that this vulnerability is beyond the reach of just FIM designers to fix. It also requires better browsers that are more able to identify fraudulent sites. In practice we use this vulnerability for security issues like bogus merchants to capture the complex order of things that go wrong for a user to be tricked into interacting with bogus merchants, for example described by Kormann and Rubin (2000).

Weak DNS: When the DNS system can be modified so that domains can point to IP addresses that are not the legitimate IP addresses we say that a weak DNS vulnerability exists. This vulnerability can be used to do DNS poisoning, where the DNS entries are modified by an attacker to a malicious website as described by Kormann and Rubin (2000). This same vulnerability can also be used to conduct more intricate MITM attacks as described by Groß (2003)

Insecure Communications: When the protocol messages of a FIM system can be read we say that there exists an insecure communications vulnerability. Often this vulnerability is used to packet sniff as is described by Sun and Beznosov (2012).

Automatic Authorisation: Automatic authorisation occurs when a certain SP has been granted access privileges for a resource previously, and when those same access privileges are asked for again, they are granted without any explicit consent from the user. This vulnerability can be exploited by an attacker by prompting a resource access from a user and then harvesting the result as described through an XSS attack reported by Sun and Beznosov (2012).

Lack of Binding: When a message is not correctly linked to the context it is intended for, we say that there is a lack of binding vulnerability at play. This vulnerability is described by Sun and Beznosov (2012) when pointing out situations in which OAuth access tokens are not tied to the session in which they are generated. This can be exploited in numerous ways, for example, session swapping.

Weak User Credentials: In the event users use simple or guessable credentials for a FIM service, a weak user credential vulnerability exists. This is typically exploited through a a brute force password guessing attack, for example described by Alotaibi and Mahmmod (2015).

Vulnerable SP: This is a quite general vulnerability to state when the SP has somehow failed in providing services securely. The specifics of these failings are not relevant to our categorisation as we are only concerned with the fact that an SP is vulnerable to certain attack classes that rely on the SP doing something incorrectly. An example of a specific failing of an SP could be that an SP does not check time stamps for access tokens granted leading to possible replay attacks, as is described by Mainka et al. (2014). We do not consider the details of how the SP is vulnerable, just that the SP is vulnerable.

Message Formatting: Similar to the insecure communications vulnerability, but is more specific in that a particular parameter is targeted in the message. This vulnerability is common in OpenID, and is demonstrated by Sovis et al. (2010). The vulnerability does also appear elsewhere though, for instance in Facebook a particular message is formatted so that cookie values are sent together with a query as demonstrated by Wang et al. (2012).

Vulnerable IdP: Similar to the vulnerable SP in principle but instead applied to the IdP. For instance, Mayer et al. (2014) centers security issues on the IdP being compromised. The argument being made is that IdPs do not prevent the website from being used in a HTTP x-frame so that a UI redressing attack is possible.

Dual-Role IdP: This is a unique vulnerability which we found to warrant its own vulnerability category. Yang et al. (2016) considers the possibility of an IdP also existing as a SP to some other IdP. Why this can be considered as a vulnerability is discussed at length by Yang et al. (2016). We note that in particular this vulnerability can amplify other problems relating to CSRF attacks.

3.5.3. *Confidentiality, Integrity, Availability Failure*

The CIA failure is the goal of an attacker conducting an attack.

Confidentiality: When an attack is explicitly trying to gain access to the information of a user. This is associated to sensitive information that is normally accessible through a user account. For example, confidentiality could be affected if the attacker is actually a malicious provider secretly harvesting information on the user as considered by Alrodhan and Mitchell (2009).

Integrity: If an attacker is able to take an unauthorised action on behalf of the user, or change information belonging to a user that is said to be a breach in in integrity. Sovis et al. (2010) shows how messages can be modified to invalidate the integrity requirement.

Availability: When an attacker manages to stop a system from delivering its service, an availability failure is said to occur. A DoS attack is a common means to achieving an availability failure and is discussed by Kormann and Rubin (2000).

Account Compromised: The compromise of a user account can have a profound impact on the user's CIA. The attacker can compromise the confidentiality of user data on the user account. The attacker can act in the name of the user thus violating the user's Integrity. Finally, the attacker could deny the user's availability to the account by deleting the account or locking the user out of the account by changing credentials. Most of the attacks in the literature in FIM are geared towards compromising accounts, which means that most attacks lead to compromises of CIA in general. Sun and Beznosov (2012) demonstrates various ways in which user accounts can be compromised.

3.5.4. *Target Protocol*

The value of this dimension describes the FIM protocol that the attack is found to be possible on. The values are therefore the FIM protocols we consider attacks on: Microsoft, OAuth, OpenID, SAML, Liberty Alliance, Facebook Connect, and Shibboleth.

3.5.5. *Issue Type*

There are two types of issue we consider. The first issue type occurs when an author examines a protocol specification, and comes up with a hypotheses for an attack on a FIM system. We say that the issue type for this is 'Protocol'. The other kind of issue is 'Implementation'. This is when the author identifies an attack is possible, (which could also be informed by a protocol implementation) and then tests and documents this attack on an actual implementation. This does not necessarily mean that the implementation issue type is stronger than the protocol issue type as typically protocol issue types could in theory be applied in general across many FIM systems.

3.5.6. Solution Presented

The purpose of this dimension is simply to state whether or not a solution is presented by the authors, not what that solution is. The solutions to an attack could be the subject for its own taxonomy. We could consider things such as what aspect of a FIM system needs to implement a solution, whether or not a FIM protocol change is needed, whether or not there are human error elements at play, and more. To avoid the complexities involved in a solution, we simply say if a solution is presented by the authors or not. However, the manner in which a solution is proposed is not as straight forward as it might first seem. For instance, some authors go a step further and announce that they have notified the relevant institutions about the problem. Other authors even state that they worked with those institutions to provide a fix to the issue. Therefore, the possible solution proposed values are 'Implemented', 'Notified', 'Suggested', and 'None'.

3.6. Surveyed Papers

In this section we outline the landscape of what we have seen in a taxonomy sorted by targeted FIM protocols that we have considered in the survey. The various protocols considered can have a number of different versions; to keep the survey into the protocols simple, we ignore this complication. The full list of papers considered for this final sample can be found in appendix A.

We will provide evidence as to why we categorise the attacks the way we did. In addition, we will describe anything peculiar that is noted along the way. For instance, sometimes authors may name an attack that is sufficiently similar to an attack that the author has already proposed (i.e. same vulnerability, attack class, CIA failure).

Note that the surveyed papers come after the presentation of the taxonomy itself, which is presented in section 3.4. The creation of the taxonomy was an iterative process. Papers were quickly assessed, a rough taxonomy established, papers read again, the taxonomy reevaluated, and so on. Eventually we decided on a final taxonomy, and afterwards read the papers one final time to fully justify how those papers fit into the taxonomy. This is why this section appears after the taxonomy description itself.

3.6.1. Microsoft

We are considering the papers that are concerned with the numerous attempts made by Microsoft to implement the concept of FIM (Passport .NET, Microsoft Accounts and Cardspace). We found four papers in this category Alrodhan and Mitchell (2009); Gajek et al. (2009); Kormann and Rubin (2000); Oppliger (2004).

Kormann and Rubin (2000) is the first attempt to analyse the security of FIM to the best of our knowledge. Specifically, they investigate Microsoft Passport. The first attack that is proposed is a denial of service attack. Interestingly, this attack is pointed out as a risk rather than an actual attack. However, the author qualifies the risk with a vulnerability, how the attacker would exploit

that vulnerability through an attack, and how the attack impacts the user in terms of CIA which qualifies the risk to be included as an attack. The vulnerability considered is a centralised infrastructure vulnerability. Solutions stated to the DoS attack are said to be complicated because replicating the service's key distribution and databases might be hard and therefore no concrete solutions are offered.

The second attack proposed is a bogus merchant attack. This idea behind this attack is that the user gets accustomed to using passport and performs the passport authentication on a service that is not actually the legitimate passport service. For example, the attacker establishes a server with the domain name 'pasport.com' and makes the server look identical to the real passport IdP. Next, the attacker establishes an attractive SP that redirects the user to 'pasport.com' for authentication. The user does not check to ensure the legitimacy of the passport server. Ultimately, this attack is possible because there is insufficient supporting trust structures that guarantees the user knows who they are talking to. The third and fourth attack also build on this attack, but we define them as a separate attack class as they are sufficiently different from this attack class.

The third attack proposed is a MITM. The attacker sets up a bogus IdP. The difference in this attack is that instead of relying on a bogus SP that redirects to the IdP the attacker actively interposes between the user and a legitimate SP. This attack is again possible because of an insufficient trust structure in FIM because the user has no way of guaranteeing trust on the IdP.

The fourth attack proposed is a DNS poisoning attack. This attacker sets up a bogus IdP. The difference is that the attacker brings the user to the bogus IdP by poisoning a DNS server so that the legitimate domain name for passport redirects to the bogus IdP. This attack is possible because of a weak DNS. Unlike the previous attacks, a solution is suggested in the form of DNSSEC, which provides a signature scheme for DNS records.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	DoS	Centralised Infrastructure	Availability	Microsoft	Protocol	None
2	Bogus Merchant	Insufficient Trust Infrastructure	Compromised Account	Microsoft	Protocol	None
3	MITM	Insufficient Trust Infrastructure	Compromised Account	Microsoft	Protocol	None
4	DNS Poisoning	Weak DNS	Compromised Account	Microsoft	Protocol	Suggested

Table 3.2 Categorisation of Kormann and Rubin (2000)

Oppliger (2004) analyses the suitability of Passport as a SSO solution. The paper concludes stating that Passport is a suitable solution, but points out some security issues along the way. The first and only concrete issue pointed out is a replay attack. If the cookies involved in the Passport protocol are sent over a HTTP connection, they could be eavesdropped and then used in a

subsequent replay attack. We say that this attack is possible because of insecure communications as HTTP is used. The attack results in a compromised account because the attacker can access an account with replayed cookies. Solutions are suggested, for example, the lifetime of cookies could be limited. Of note is that a brute force attack possibility is considered by the author, but ultimately dismissed because passport uses a pin system which makes a brute force attack difficult to achieve.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	Replay Attack	Insecure Communications	Compromised Account	Microsoft	Protocol	Suggested

Table 3.3 Categorisation of Oppliger (2004)

Alrodhan and Mitchell (2009) focus on Microsoft Cardspace. They address two issues with FIM that are difficult to solve like the reliance on the DNS and that a centralised infrastructure could result in some privacy issues. The first issue is a DNS poisoning attack that is used to possibly redirect users to, in their words, "false websites". The reliance on the DNS is the cause of this which cannot be trusted so we say the vulnerability is insufficient trust structure. The solution to this problem is stated to be DNSSEC which is also proposed by Kormann and Rubin (2000)

The second issue is that there could be a malicious provider that is harvesting user information. This is due to the inherent centralised infrastructure vulnerability that exists in FIM. A bespoke solution is suggested that relies on zero knowledge proofs to never reveal the actual values of identity claims.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	DNS Poisoning	Weak DNS	Compromised Account	Microsoft	Protocol	Suggested
2	Malicious Provider	Centralised Infrastructure	Confidentiality	Microsoft	Protocol	Suggested

Table 3.4 Categorisation of Alrodhan and Mitchell (2009)

Gajek et al. (2009) seems to be inspired by the work of Kormann and Rubin (2000) with a very similar title and paper layout. The difference is that the focus is on Cardspace instead of Passport. Introduced is a single complex multi-layered attack that involves a number of different attack classes including DNS poisoning, replay, and bogus merchant. Since we have outlined that each issue should belong to at most one attack class for categorisation, we decide to name this as a DNS poisoning attack for two reasons. The first is that the DNS poisoning is the pivotal step to send the user to the bogus merchant which ultimately results in the replay attack. The second is that this attack is quite similar in spirit to an attack introduced by Kormann and Rubin (2000) (which we also categorise as a bogus merchant) in that a DNS is poisoned to direct the user to a bogus merchant and then a legitimate credential is obtained and replayed. The

vulnerability for this attack is a weak DNS since the DNS is required to be poisoned to launch this attack. Interestingly, this attack is actually tested on the cardspace implementation so this paper does differentiate itself from other papers by proving that the attack is possible. Solutions are suggested including cryptography improvements to tokens and augmenting browsers.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	DNS Poisoning	Weak DNS	Compromised Account	Microsoft	Implementation	Suggested

Table 3.5 Categorisation of Gajek et al. (2009)

3.6.2. OAuth

We are considering papers that have found attacks in the OAuth protocol. There are two versions of OAuth, 1.0 and 2.0. For the purposes of this work we make no distinction between these two versions. Any attack found in OAuth is valid. There are seven papers which we identify as having attacks in OAuth Alotaibi and Mahmmod (2015); Ferry et al. (2015); Grzonkowski et al. (2011); Li and Mitchell (2014); Shernan et al. (2015); Sun and Beznosov (2012); Yang et al. (2016)

Sun and Beznosov (2012) pointed out five attacks for OAuth. All of these attacks are leveraged against implementations and solutions are suggested.We identify the first attack as a message modification. The first attack involves an eavesdrop of unencrypted communications between the browser and the SP to obtain an access token which is used to compromise an account.

The second attack forces a client side login which sends an access token to the attacker using an XSS attack. The XSS attack requires the user to be logged in or to have logged in within the same browser session with the IdP. The IdP needs to offer the automatic authorisation feature of OAuth which is viewed as a vulnerability here. We could alternatively mark the vulnerability as a vulnerable SP because the SP needs to be susceptible to XSS. However, because the automatic authorisation feature is an interesting vector for the attack to take rather than a more generic vulnerable SP, we label the vulnerability as automatic authorisation. Since an access token is stolen, the eventual goal is to compromise a user account.

The third attack is described as an impersonation attack. This is where the attacker is able to guess user credentials in some way and impersonate the user. It seems that this attack is to be escalated to by other attacks since provided is a section for vulnerability interplays which lead to this attack. For instance, the attack could be possible through stolen or guessed credentials. We describe this attack as message modification, because the author describes a process in which HTTP requests are altered. The vulnerability that allows for this to occur is a lack of binding on the credential as stated in the text. The goal is to impersonate the user and therefore the CIA failure is a compromised account.

The fourth attack is known as a session swapping attack. This attack involves logging the user into the account of the attacker. The immediate goal of this attack seems to be confidentiality loss, as the user is in a precarious position to start entering in personal information which could later be read by the attacker when the attacker later logs back into the account. This is again due to lack of binding on the credential.

The fifth attack is stated as a force-login CSRF. A typical restriction of a CSRF attack is that the user must be logged on in order to conduct the attacks. If the attacker is somehow able to force the user to login, the attacker can carry out CSRF attacks. With the presence of the OAuth automatic authorisation feature the attacker can force the user to login through a CSRF attack. An additional vulnerability can be considered similar to the XSS attack above which is a vulnerable SP. The same reason we decide to state the vulnerability as automatic authorisation for the XSS attack applies here. Automatic authorisation is specific while a vulnerable SP is general. It is hard to say what the attacker can accomplish through further CSRF attacks since what is possible is going to be implementation specific. The confidentiality can be impacted by stealing data using legitimate user commands. Integrity can affected by performing actions on behalf of the user. Finally, some CSRF attacks might be able to disable a user account, impacting availability. Therefore, CSRF attacks can compromise user accounts even if the attacker never learns user credentials. The attacker could even change the password of the user through a CSRF attack, which allows a clear path for the attacker to compromise the user's account. We decide on the CIA failure for a CSRF attack as a compromised account.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	Packet Sniffing	Insecure Communications	Compromised Account	OAuth	Implementation	Suggested
2	XSS	Automatic Authorisation	Compromised Account	OAuth	Implementation	Suggested
3	Message Modification	Lack of Binding	Compromised Account	OAuth	Implementation	Suggested
4	Session Swapping	Lack of Binding	Confidentiality	OAuth	Implementation	Suggested
5	CSRF	Automatic Authorisation	Confidentiality	OAuth	Implementation	Suggested

Table 3.6 Categorisation of Sun and Beznosov (2012)

Alotaibi and Mahmmod (2015) introduces a biometric facial recognition approach to OAuth. The main idea behind this paper is that username/password combinations are not ideal because users tend to reuse credentials so a facial recognition authentication system should be considered. Pointed out is that a brute force attack might be possible because of a vulnerability where weak user credentials are used. This would compromise the user's account. The solution suggested is the facial recognition this paper revolves around.

Ferry et al. (2015) focuses on trying to launch CSRF attacks on real SPs. The attempts are successful on two sites so the vulnerability is vulnerable SP. The authors describe how access

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	Brute Force	Weak User Credentials	Compromised Account	OAuth	Protocol	Suggested

Table 3.7 Categorisation of Alotaibi and Mahmmod (2015)

tokens can be stolen in this way so we say that the CIA failure is a compromised account. The attack is tested on real SPs and so is possible on the implementation level. Interestingly, the authors inform the vulnerable SPs and the issue is fixed before the paper authored so we say that the solution is implemented.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	CSRF	Vulnerable SP	Compromised Account	OAuth	Implementation	Implemented

Table 3.8 Categorisation of Ferry et al. (2015)

Li and Mitchell (2014) performs an investigation on two different OAuth implementations and finds two issues. Therefore, both issues are possible at the implementation level. The first issue described is a CSRF attack. Described is how these attacks are possible because of flaws on the SP so we describe the vulnerability as a vulnerable SP. Furthermore described is how authentication tokens are sent to the attacker using the CSRF attack, so the CIA failure is a compromised account.

The second issue is an account binding attack. Described is how a user's SP account could be bound to the attacker's IdP account. The blame of this attack is placed on the SP and so we say that the vulnerability is a vulnerable SP. The CIA failure is a compromised account since the attacker could access the SP account using the IdP account. Both of these issues have solutions suggested: focusing on improving state parameters so that requests are harder to forge.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	CSRF	Vulnerable SP	Compromised Account	OAuth	Implementation	Suggested
2	Account Binding	Vulnerable SP	Compromised Account	OAuth	Implementation	Suggested

Table 3.9 Categorisation of Li and Mitchell (2014)

Shernan et al. (2015) performs an implementation investigation of OAuth in the context of CSRF attacks. Stated is how the SP must have a vulnerable implementation of the authorisation granting flow so we say that a vulnerable SP is the vulnerability. The CIA failure is suggested to be a compromised account because the SP will get access to the IdP account. It is not immediately clear how an account compromise is possible through this, but if we consider other work, (such as Li and Mitchell (2014)) we know that CSRF attacks on OAuth can lead to

account compromise. There is indication that real websites are scanned for specific technical vulnerabilities that make CSRF attacks possible but it is not clear if attacks are actually carried out in practice. However, because there is some evidence of the implementation investigation and we have seen that CSRF attacks have been shown to be possible through other works (such as Sun and Beznosov (2012)) we decide to specify that this attack is presented as being on the implementation level. A solution is suggested, which revolves around the state parameter which makes it non-guessable.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	CSRF	Vulnerable SP	Compromised Account	OAuth	Implementation	Suggested

Table 3.10 Categorisation of Shernan et al. (2015)

Yang et al. (2016) designs an automated tool called OAuthTester to assess OAuth implementations for vulnerabilities. Due to the extensive testing performed on OAuth implementations, we say that the issues presented are possible at the implementation level. The first issue is a CSRF attack. This is possible because of SPs not implementing a state parameter so we generalise this and say that this is a vulnerable SP. The author states that a number of different outcomes are possible from this attack but the most serious is account compromise so we say that the CIA failure is account compromise. Solutions are suggested for this issue which revolve around making the state parameter easier to implement for SPs.

The second issue is interesting and involves a number of different attack classes and vulnerabilities. At the core of the issue is a vulnerably the authors call a "dual-role IdP". This vulnerability revolves around an IdP in an OAuth federation also serving as a SP to other OAuth IdPs. Even though the issue also has elements of other vulnerabilities such as automatic authorisation and insecure communication as seen in other OAuth attacks (in particular Sun and Beznosov (2012)), we think that categorising this issue using that vulnerability takes away important information from this issue. In particular, stated is that the dual-role IdP can be compromised and then any other SP accounts that rely on that IdP are vulnerable. This is a clear example of how attacks can escalate in FIM. Therefore, we decide to state the vulnerability as "dual-role IdP". This vulnerability can be attacked in a few different ways. One attack the authors describe is how a redirect URL can be compromised which in our taxonomy we would refer to as a message modification attack class. The next attack the authors describe is more interesting to this thesis because it links in with work by Sun and Beznosov (2012) which is of particular interest. The author describes how a force-login CSRF attack can be combined with an account binding. Since we have already used the CSRF attack class for this paper and this attack is significantly different, we ultimately decide to attribute this issue to an account binding attack. The resulting CIA failure is account compromise. This attack is also said to occur at the implementation level since an indication is given as to how many IdPs are dual-role IdPs out of the evaluated IdPs (10 out of 13 are dual-role).

The third issue involves the automatic authorisation vulnerability. The difference in this issue is in how a malicious provider attack class could gain access again without the user's knowledge. Since access attempts are automatically approved if a user has granted access before, SPs can abuse that to stealthily gain access resulting in a compromised account. Also singled out are two IdPs (Sina and Renren) which are vulnerable to this which indicates that this attack is possible at the implementation level. The solution to this problem is suggested as being to not give special privilege to SPs which results in the automatic authorisation vulnerability.

The fourth issue is about insecure communications being used which results in a packet sniffing attack being possible. Specifically, if TLS is not used the attacker sniffs the access token which can be used to access a user account. The new spin on this attack that is introduced is the consideration of how some places (like China) have controlled central internet routes which could make the exploitation of this attack easier since internet traffic is concentrated in one place. Cited is a percentage of implementations which are vulnerable to this issue.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	CSRF	Vulnerable SP	Compromised Account	OAuth	Implementation	Suggested
2	Account Binding	Dual-Role IdP	Compromised Account	OAuth	Implementation	None
3	Malicious Provider	Automatic Authorisation	Compromised Account	OAuth	Implementation	Suggested
4	Packet Sniffing	Insecure Communications	Compromised Account	OAuth	Implementation	None

Table 3.11 Categorisation of Yang et al. (2016)

Grzonkowski et al. (2011) consider passwords as a weak point in some authentication schemes, namely OAuth and OpenID. We name the vulnerability as weak user credentials and the attack class as brute force. The resulting CIA failure is an account compromise. The solution to this problem is a protocol called SeDiCi which is a protocol based on zero knowledge proofs. We consider this two separate issues since both the possibility of the attack occurring on OAuth and OpenID is discussed. Because this issue is presented here, we will not be revisiting it in the OpenID subsection.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	Brute Force	Weak User Credentials	Compromised Account	OAuth	Protocol	Suggested
2	Brute Force	Weak User Credentials	Compromised Account	OpenID	Protocol	Suggested

Table 3.12 Categorisation of Grzonkowski et al. (2011)

3.6.3. OpenID

The surveyed OpenID papers are Abbas et al. (2016); Feld and Pohlmann (2011); Hsu et al. (2011); Krolo et al. (2009); Li and Mitchell (2016); Mainka et al. (2016); Oh and Jin (2008); Sovis et al. (2010); Sun et al. (2012) There are also some papers that cover a mix of OpenID and some other FIM system. Grzonkowski et al. (2011) is also presented in the OAuth subsection so is not reiterated on here. Similarly, Urueña et al. (2014) and Wang et al. (2012) both describe OpenID attacks but also cover Facebook Connect so they are included in the Facebook section and not here.

Oh and Jin (2008) considers a number of security limitations which are not considered security issues because it is not stated how a vulnerability is exploited by an attack to cause a CIA failure for the user. There is one vector of attack that is expanded on. An attacker could sniff an authentication assertion and manipulate a nonce so that the assertion could be used elsewhere. Therefore, this attack is classed as a message modification attack because the attacker cannot just trivially use the sniffed message but instead has to manipulate the message first. Stated is that this attack is possible because the nonce is not signed so we say that this vulnerability is a message formatting vulnerability. This attack results in a compromised account as the attacker could use the assertion to gain access to the user account. The attack is demonstrated at the implementation level. The presentation of the paper is rather poor and we were unable to clearly understand the meaning of the paper in general but specifically in consideration of any solutions.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	Message Modification	Message Formatting	Compromised Account	OpenID	Implementation	None

Table 3.13 Categorisation of Oh and Jin (2008)

Sovis et al. (2010) lists a number of security problems which ultimately only yields one security issue because a security issue must have an associated user CIA failure. Namely, stated is how HTTP is used over HTTPS in some cases and parameters can be injected in certain parts of a message in OpenID. These things alone do not yield in a security issue because there is no apparent CIA impact on the user. However, stated is that parameters can be forged in such a way that the attacker can change almost any user data. Interestingly, a common next step for other papers in this survey would be to describe how this exploit could be used to compromise a user account entirely. If a complete account compromise is possible from this attack, it is not made clear so the CIA failure is an integrity failure.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	Message Modificiation	Message Formatting	Integrity	OpenID	Protocol	None

Table 3.14 Categorisation of Sovis et al. (2010)

Feld and Pohlmann (2011) points out two security issues the first being a bogus merchant type attack class. We note that there are many risks discussed, such as DoS, but it is not always clear how these risks would be realised so they are not included as a security issue. Solutions suggested seems to involve nPA which is an identity card system in Germany. Specified is that a malicious SP could set up a phishing SP that imitates a real SP. We decide to categorise this as a bogus merchant attack. Note that in a similar attack described by Kormann and Rubin (2000) we state that the vulnerability as insufficient trust infrastructure because the browser does not protect the user from navigating to malicious sites. Using the same logic, this vulnerability is insufficient trust infrastructure because the user places trust in the browser and the system at large and is easy to abuse for an attacker that can imitate the system.

The second issue specified is that a due to OpenID being a centralised authentication system a malicious provider could profile users. The vulnerability is a centralised infrastructure and the attack class is a malicious provider. The overall CIA failure is just in terms of confidentiality because the profiling attack is limited to just observing the user without interacting with them.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	Bogus Merchant	Insufficient Trust Infrastructure	Compromised Account	OpenID	Protocol	Suggested
2	Malicious Provider	Centralised Infrastructure	Confidentiality	OpenID	Protocol	Suggested

Table 3.15 Categorisation of Feld and Pohlmann (2011)

Sun et al. (2012) states seven security issues but we reduce the number of issues specified to three as a number of these issues are similar to each other with only small technical details differentiating the attacks. All of the issues are possible at the implementation level as the author performs a detailed empirical investigation where real world OpenID implementations are investigated to see if the issues are actually possible.

The first issue described is a CSRF attack. The paper specifies A1-A3 as CSRF attacks but we find they are quite similar in nature with the main differences being in the HTTP methods used and whether a login or authorisation request is forged. The vulnerability here is a vulnerable SP as this attack is possible because the SP does not sign the request. The CIA failure is a compromised account as it is stated that the attacker can login through this attack. Solutions are discussed as defence mechanisms, specifically stated is that adding a cryptographic token to the login form would fix the issue.

The second issue is A4. Since the idea of the message being modified by the attacker is clear here, we specify the attack class to be message modification. The CIA failure is in terms of integrity since it is not specified how an attacker would modify the message to cause an account compromise. A solution is suggested, which is claimed to be the same as the previous solution.

The third issue the authors specify is a replay attack. This third issue is attributed to A5-A7 as all of these attacks have an element of intercepting and then sending on a message from a legitimate party in the protocol. This attack is possible because messages sent are not bound to their sender so we say that the vulnerability is a lack of binding. The CIA failure is account compromise because the attacker is able to impersonate the user. A solution is suggested which is the same as the previous solutions.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	CSRF	Vulnerable SP	Compromised Account	OpenID	Implementation	Suggested
2	Message Modification	Message Formatting	Integrity	OpenID	Implementation	Suggested
3	Replay Attack	Lack of Binding	Compromised Account	OpenID	Implementation	Suggested

Table 3.16 Categorisation of Sun et al. (2012)

Abbas et al. (2016) specifies three phishing scenarios in OpenID. We categorise this as just one issue because in essence these attacks are quite similar with the difference being how well the illegitimate site can impersonate the legitimate site. We categorise phishing attacks as bogus merchant attacks so the attack class is a bogus merchant. The vulnerability is insufficient trust infrastructure since the trust the user puts on the internet is leveraged by the attacker. The CIA failure is a compromised account because with the credentials obtained through the bogus merchant the attacker can access the user account. A solution is suggested, which is a cryptographic scheme that eliminates the dependency on a shared secret.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	Bogus Merchant	Insufficient Trust Infrastructure	Compromised Account	OpenID	Protocol	Suggested

Table 3.17 Categorisation of Abbas et al. (2016)

Hsu et al. (2011) discussed two security issues. These security issues prompt the design of WebCallerID which is an authentication scheme based on mobile phones which is claimed to be the solution to both issues. The first issue is described as a phishing which we categorise as a bogus merchant. The vulnerability is insufficient trust infrastructure because this attack leverages the user's trust in the website. The CIA failure is a compromised account because the attacker will get access to the user's account.

The second issue revolves around concerns about the user's privacy. Malicious providers could collaborate to piece together the identity of a user. Therefore, we describe the attack class as a malicious provider. This is possible because of the centralised nature of OpenID so we specify the vulnerability as a centralised infrastructure. The CIA failure is in terms of confidentiality because the data attached to the user's account is compromised.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	Bogus Merchant	Insufficient Trust Infrastructure	Compromised Account	OpenID	Protocol	Suggested
2	Malicious Provider	Centralised Infrastructure	Confidentiality	OpenID	Protocol	Suggested

Table 3.18 Categorisation of Hsu et al. (2011)

Krolo et al. (2009) discusses five separate issues for OpenID. There is an additional sixth issue proposed relating to privacy concerns but we found that the issue does not adequately describe a vulnerability or CIA impact on the user to be considered. The first issue is categorised as a DoS attack class. The idea with this attack is that a SP or IdP are tricked into downloading large files from an attacker. The vulnerability could be either a vulnerable SP or a vulnerable IdP because either can be tricked into downloading this file. We decide that the vulnerability here is a vulnerable IdP because it is rare that specific vulnerabilities on the IdP are identified. The CIA failure is in terms of availability and a solution is suggested which is to limit the amount that servers download.

The second issue is a replay attack class. A nonce used in the OpenID authentication is not correctly signed which means that the message could simply be reused by the attacker. The CIA failure from this is a compromised account because the attacker can replay the message to authenticate as the user. A solution is suggested which is to include a signature in the OpenID signature field.

The third issue is a MITM attack class. The DNS could be used to stage this attack so we say that the vulnerability is a weak DNS. It is not entirely clear what the CIA failure is which meant that this issue was almost not included but we presume that the attacker can impersonate the user to OpenID providers which is an account compromise. The solution to the issue is claimed to be certificates that are signed by a trusted authority.

The fourth issue is phishing which we categorise as a bogus merchant attack class. Stated is that an attacker can setup fake websites that look similar to real websites. We therefore attribute the vulnerability to insufficient trust infrastructure because the user places trust in the system which is leveraged by the attacker. The CIA failure is a compromised account because the attacker can use the captured credentials to login as the user. Solutions to the issue are mentioned, such as identity cards and multi-factor authentication.

The fifth issue is a CSRF attack class. The interesting spin on this attack is that the possibility of a malicious SP exploiting this attack on an IdP is considered. The vulnerability is therefore a vulnerable IdP because the target is the IdP. The CIA failure is not clearly stated, but if we consider CSRF attacks in general any kind of form could be susceptible to CSRF which has a variety of implications in terms of CIA because the attacker could forge different types of forms. We therefore say that the CIA failure is account compromise because of the wide array of

possibilities open to the attacker depending on implementation. The solution to the issue is stated which is appending secret tokens to forms so that the request has to come from the legitimate browser session.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	DoS	Vulnerable IdP	Availability	OpenID	Protocol	Suggested
2	Replay Attack	Lack of Binding	Compromised Account	OpenID	Protocol	Suggested
3	MITM	Weak DNS	Compromised Account	OpenID	Protocol	Suggested
4	Bogus Merchant	Insufficient trust infrastructure	Compromised Account	OpenID	Protocol	Suggested
5	CSRF	Vulnerable IdP	Compromised Account	OpenID	Protocol	Suggested

Table 3.19 Categorisation of Krolo et al. (2009)

Li and Mitchell (2016) performs an investigation on attacks in OpenID in particular on the hybrid server-side flow and authorisation code flow. A total of ten issues are put forward. However, there is some cross-over of the issues over these two modes. In particular, the authors state access token interception, privacy issues, and session swapping across both modes without any major differences between the two modes. We therefore decide to reduce the amount of issues we include to seven accounting for this cross-over. For all of these issues, solutions are offered for both SPs and IdPs. These solutions include measures against CSRF, changes to the state values of authorisation requests, and more restrictions on how access tokens are used. All issues also have associated evidence that indicates the issues were investigated on implementations.

The first issue is when SPs use public google ID identifiers to authenticate. The authors report that surprisingly a small amount of SPs authenticate with public google identifiers that can be found by an attacker by searching Google Plus. We say that the attack class here is message modification because the public identifier is added to a message the attacker constructs. The vulnerability is a vulnerable SP because it is the SP's choice to adopt such an insecure practice. The CIA failure is a compromised account because the attacker can access the user's account as a result of this attack. Interestingly, the authors notify the impacted SPs of the issue but not for any of the other issues they describe.

The second issue is that access tokens are not correctly verified by SPs. This means that malicious providers with a valid access token can impersonate the user to other SPs. Therefore, the attack class is a malicious provider with the vulnerability being a vulnerable SP. Because the attacker can impersonate the user to the SP the CIA failure is a compromised account.

The third issue revolves around a small number of SPs not using SSL to transport access tokens making packet sniffing possible. The vulnerability is insecure communications. With the access token, the attacker can compromise the account.

The fourth issue is to do with the privacy of users. The previous issue is about access tokens being unencrypted but for this issue it is the actual identifying information associated to users. The attack class is still packet sniffing because the information is extracted directly from the message exchanges. The vulnerability is insecure communications. However, the CIA failure is now in terms of confidentiality because only user identifying information is affected.

The fifth issue is categorised as a session swapping attack class because the attacker logs in the user to the attacker's account. This is possible because a state value is not attached to the browsing session. We therefore categorise the vulnerability here as a lack of binding.

The sixth issue revolves around the automatic authorisation feature used in particular by Google. A specific instance of an XSS attack is given on Android devices. Using the XSS attack, the attacker can send a forged authorisation request, which because of the automatic authorisation feature automatically returns an access token to the browser which is sent to the attacker. We decide to categorise the vulnerability as automatic authorisation as this seems to be the defining feature that makes this attack possible and is more relevant to FIM systems than specific Android vulnerabilities. The CIA failure is a compromised account since the attacker can use the access token on the user's account.

The seventh issue is described as a force-login CSRF attack. The attacker somehow causes the user's browser to initiate an authorisation request. We understand that this could be done through a malicious/compromised website or an email. Because of the automatic authorisation feature, the login is successful. Of note here, the author does not provide an explanation of how this attack can compromise a user in terms of CIA. They describe it as just an annoying thing the user has to deal with. As a result of this, the CIA failure is in terms of integrity.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	Message Modification	Vulnerable SP	Compromised Account	OpenID	Implementation	Notified
2	Malicious Provider	Vulnerable SP	Compromised Account	OpenID	Implementation	Suggested
3	Packet Sniffing	Insecure Communications	Compromised Account	OpenID	Implementation	Suggested
4	Packet Sniffing	Insecure Communications	Confidentiality	OpenID	Implementation	Suggested
5	Session Swapping	Insecure Communications	Confidentiality	OpenID	Implementation	Suggested
6	XSS	Automatic Authorisation	Compromised Account	OpenID	Implementation	Suggested
7	CSRF	Automatic Authorisation	Integrity	OpenID	Implementation	Suggested

Table 3.20 Categorisation of Li and Mitchell (2016)

Mainka et al. (2016) describes a model where malicious IdPs are considered. A number of security issues are proposed, which we reduce to two because the issues described in 5.2-5.4 are quite similar in nature. All of these issues occur at the implementation level as there is evidence provided that the issues have been explored on implementations. The author does claim that some of the uncovered attacks are protocol independent so are applicable to other protocols but since the testing was only carried out on OpenID we include these issues as just OpenID issues. In addition, the affected parties are notified.

The first issue is described as token recipient confusion. It seems that legitimate tokens are generated and a malicious IdP is able to obtain the token through a particular run of the protocol. This token can access user resources on a SP that trusts the malicious providers but the user is not actually registered to that IdP. We therefore say that the attack class is a malicious provider. The vulnerability is insufficient trust infrastructure because the user places trust in this malicious provider and it is possible for the malicious provider to access user information in places the user would not expect or necessarily consent to. The CIA failure is a compromised account because the attacker can login as the user upon completion of the attack.

The second issue is the aggregation of 5.2 to 5.4 that the author specifies. However, it would not be fair to say that these attacks are all the same as they are technically quite complex. Although the technical details of the attacks vary, the goal seems to be the same in that a valid access token is generated due to the token not being bound for use for the legitimate purpose that is expected of it. This is possible because of the malicious provider attack class. The vulnerability here is a lack of binding because tokens seem to not have been bound to the use expected of them. The CIA failure in all cases is a compromised account as the information on the target SP can be accessed.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	Malicious Provider	Insufficient Trust Infrastructure	Compromised Account	OpenID	Implementation	Notified
2	Malicious Provider	Lack of Binding	Compromised Account	OpenID	Implementation	Notified

Table 3.21 Categorisation of Mainka et al. (2016)

3.6.4. SAML

Armando et al. (2008) performs a formal analysis of the SAML protocol using SATMC. As a result of the formal analysis, a single security issue is uncovered. The attacker acts as a legitimate service provider and then in parallel starts a new SAML initiated session with Google when the user asks for a resource provided by the Google API (like calendar). The attacker is then able to reuse the authentication assertion provided to login as the user. Since the assertion is not correctly bound to just the service it is intended for, it can be reused, meaning that the vulnerability is a lack of binding. The attack class is MITM because the attacker relies on

starting a session to Google while posing as the user. The CIA failure is a compromised account because the attacker can login as the user. The attack seems to have been tested on an actual SAML implementation on Google so this issue is possible at the implementation level. Finally, the authors provide evidence that they have notified the vulnerable parties so we describe the issue type as notified.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	MITM	Lack of Binding	Compromised Account	SAML	Implementation	Notified

Table 3.22 Categorisation of Armando et al. (2008)

Groß (2003) describes three security issues. The first issue is a clear instance of a replay attack class as the attacker forwards an encrypted redirect to a SP. The vulnerability here is a lack of binding because the SP cannot distinguish between the user and the attacker. Because the attacker can impersonate the user to the SP, the CIA failure is a compromised account. A solution is suggested, which is for the SP to check if the sender of the particular message which is replayed has the same IP address as the user who initiated the protocol.

The second issue describes a situation where a server can be impersonated when the DNS is broken in a MITM attack class. The attacker impersonates a server between the browser and the IdP. The attacker plays the role forwarding any messages it receives until they find any unused SAML artifacts which can be used to access the user's account. We say that the vulnerability here is a weak DNS as that seems to be the defining vulnerability that makes this attack possible. The CIA failure is a compromised account because the attacker can use the artifact to impersonate the user. A solution is suggested, which is that authentication needs to occur in some way for every step in the protocol.

The third issue is described as a HTTP referrer attack. The idea is that valid SAML artifacts are leaked by interrupting the protocol at a certain point. When the protocol is interrupted, the resulting error message could be sent in HTTP and therefore is able to be read. We therefore say that the attack class is packet sniffing with the vulnerability being insecure communications. The HTTP message could contain SAML artifacts which can be used to access the user's account; therefore the CIA failure is a compromised account.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	Replay Attack	Lack of Binding	Compromised Account	SAML	Protocol	Suggested
2	MITM	Weak DNS	Compromised Account	SAML	Protocol	Suggested
3	Packet Sniffing	Insecure Communications	Compromised Account	SAML	Protocol	Suggested

Table 3.23 Categorisation of Groß (2003)

Kumar (2014) provides an analysis of SAML using the alloy framework. The attacker tricks the user into submitting a form which binds the account of the user at the SP to the account of the attacker at the IdP. The notable thing that happens here is the user account binding to the attacker account indicating account binding. We say that the vulnerability is a vulnerable SP because the SP allows for the parameters generated from one browsing session (the attacker) to be used in another (the user) which binds the accounts. The CIA failure is a compromised account because the attacker can access the SP account through the IdP account.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	Account Binding	Vulnerable SP	Compromised Account	SAML	Protocol	None

Table 3.24 Categorisation of Kumar (2014)

Mayer et al. (2014) describe two security issues. Both have been tested on implementations so we describe both of these issues as occurring at the implementation level. The first issue is describes as an ACS spoofing attack which is a novel attack proposed by the authors. We find this attack sufficiently different from already established attack classes to warrant its own attack class. It is similar to a bogus merchant, but instead of requiring user interaction on the bogus merchant a SAML logic flaw is exploited. The vulnerability here seems to lie with the IdP as the authors evaluate the IdP security and base the possibility of the attack on problems with the IdP. This logic flaw allows for valid SAML assertions to be collected by the attacker, meaning the CIA failure is a compromised account. Solutions are suggested, such as whitelisting certain SPs.

The second issue uses XSS to steal an access token stored as a cookie for the IdP. There are also elements of a UI redressing (or clickjacking) attack which is the only occurrence of this on FIM that we are aware of. We could categorise this attack class as an XSS attack. However, since this is the only occurrence of UI redressing, we decide to categorise the attack class as UI redressing so that we expand the list of known attack classes to have been studied on FIM. The attack is possible because of weaknesses within the IdP; therefore we say the vulnerability is a vulnerable IdP. The CIA failure is a compromised account because with the stolen token the user account can be accessed. Solutions are proposed, including options on web frames to mitigate the UI redressing aspects of the attack.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	ACS Spoofing	Vulnerable IdP	Compromised Account	SAML	Protocol	Suggested
2	UI Redressing	Vulnerable IdP	Compromised Account	SAML	Protocol	Suggested

Table 3.25 Categorisation of Mayer et al. (2014)

Mainka et al. (2014) outlines three attacker models with a number of technical exploits for each model. Since each model focuses on one fundamental concept (message generation for the

first one, access to a valid token for the second, and a web attacker for the third) we categorise each model as one security issue. All of these issues have also been tested on real implementations; therefore we say that all issues occur at the implementation level. It also appears that all issues had implemented solutions at the time of publication.

The first issue relies on the fact that valid messages can be generated with the exception of any secret information. One way that this can be exploited is through the generation of tokens of which the signature is not actually checked due to a logic flaw. The authors refer to this as signature exclusion. We say that the vulnerability is a lack of binding because these tokens should be bound to the users that they are intended for. The attack class is message modification because a valid message is changed to perform some malicious task. The CIA failure of this issue is a compromised account because the attacker can use arbitrarily generated tokens to associate user identity information.

The second issue assumes that the attacker already has access to a token. This token might have been valid at some point but has since expired. The idea is to simply replay this token and see if it is successful. The attack class is therefore a replay attack. Because it is the SP that fails to check the validity of the token, the vulnerability is a vulnerable SP. The CIA failure is a compromised account because the attacker can access the user account with the token.

The third issue uses a web attacker model where the attacker can influence the user to click on certain links. This immediately lends itself to the CSRF attack class as is described. The authors show a use of the CSRF attack, where the configuration of what IdPs are considered trusted can be altered. If the trusted IdPs can be altered, then the attacker can set the IdP to be one that the attacker controls and generate tokens used to access the user account. The vulnerability is a vulnerable SP because the SP does not correctly protect themselves from the well known CSRF exploit. The CIA failure is a compromised account because eventually the attacker will be able to login to the user's account.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	Message Modification	Lack of Binding	Compromised Account	SAML	Implementation	Implemented
2	Replay Attack	Vulnerable SP	Compromised Account	SAML	Implementation	Implemented
3	CSRF	Vulnerable SP	Compromised Account	SAML	Implementation	Implemented

Table 3.26 Categorisation of Mainka et al. (2014)

3.6.5. Liberty Alliance

Pfitzmann and Waidner (2003a) demonstrate one security issue in Liberty Alliance. The attack class described is a MITM attack which is possible through a dishonest SP impersonating the user to a legitimate SP. The IdP processes certain exchanges as coming directly from the

legitimate SP even though they originate from a dishonest SP. The IdP still sends the response of the request granting access but to the dishonest SP. We state that the vulnerability here is a lack of binding because the authorisation to access is not bound to the legitimate SP. The CIA failure is a compromised account because the dishonest SP can access the user account. The authors state that Liberty acknowledged and repaired the issue so the solution is implemented.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	MITM	Lack of Binding	Compromised Account	Liberty Alliance	Protocol	Implemented

Table 3.27 Categorisation of Pfitzmann and Waidner (2003a)

Ahmad et al. (2010) mentions a number of security concerns with not only Liberty but also OpenID and Windows based FIM systems. The problem with this paper is that these concerns are often not original, the author acknowledges that these are sourced from other locations. Furthermore, many of the issues are not fully explained with a clear vulnerability, attack class, and CIA failure so cannot be categorised. However, one issue that is explained is that because of the centralised infrastructure of liberty (this is the vulnerability), malicious providers can collude to piece together the identity of a user (the attack class), resulting in the confidentiality of the user being compromised (CIA failure).

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	Malicious Provider	Centralised Infrastructure	Confidentiality	Liberty Alliance	Protocol	None

Table 3.28 Categorisation of Ahmad et al. (2010)

3.6.6. Facebook Connect

Miculan and Urban (2011) describe how a replay attack can be leveraged to impersonate a user to a SP. Interestingly, this is stated to be two separate attacks which interlink. However, the taxonomy we employ requires that for each attack there is a clear CIA failure for the user. The replay attack by itself does not constitute a CIA failure however we still categorise the overall attack class as a replay attack. The authors state that this attack is possible because of unencrypted channels being used so we attribute the vulnerability to insecure communications. The CIA failure is a compromised account because the user is impersonated to the SP through the replay attack. The authors suggest a solution by employing the use of the Diffie-Hellman key exchange protocol.

Urueña et al. (2014) discusses two security issues. It should be noted that the first of these issues occurs on OpenID but since there is only one issue pertaining to OpenID and the OpenID analysis is already quite populated we include it here. Both of these issues seem to be tested on implementations so we state that these issues are implementation issues. The first issue revolves

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	Replay Attack	Insecure Communications	Compromised Account	Facebook Connect	Protocol	Suggested

Table 3.29 Categorisation of Miculan and Urban (2011)

around the fundamental design of OpenID where certain privacy leaking parameters are included in messages exchanged as part of the protocol. The vulnerability here could be in the message formatting used for OpenID, but the indicated vulnerability is insecure communications because HTTP is used. The attack class is packet sniffing as the message is simply sniffed directly for its contents. The CIA failure is in terms of confidentiality because the user's information is at risk in the message. Solutions are suggested, which include disabling the HTTP referer field in browsers.

The second issue highlights the issue of Facebook being a massive IdP and as a result there are serious privacy concerns. We state that the vulnerability here is a centralised infrastructure. How this can be exploited through an attack class is through a malicious provider. If Facebook decides to, they can keep track of all visited websites by the user in which they use Facebook. We say that the CIA failure is in terms of confidentiality as a result of this. A solution is suggested, which is to decentralise the process, like OpenID does.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	Packet Sniffing	Insecure Communications	Confidentiality	OpenID	Implementation	Suggested
2	Malicious Provider	Centralised Infrastructure	Confidentiality	Facebook Connect	Implementation	Suggested

Table 3.30 Categorisation of Urueña et al. (2014)

Wang et al. (2012) puts forward two security issues. The first of these issues actually occurs on OpenID which we cover here instead of the OpenID section. This attack seems to be possible because certain parameters in the message are not signed correctly, which means they are subject to forgery. We therefore state that the vulnerability is in the message formatting. The attack class is message modification because the attacker modifies some of the parameters in the message in order to impersonate a user. Because the attacker can impersonate the user, we state that the CIA failure is an account compromise. This attack is specifically possible on a Google implementation of OpenID so we classify this issue as an OpenID issue. It seems that this issue is tested on the implementation so we state this this is an implementation issue. The authors state that the issue has been fixed, so we say that the solution has been implemented.

The second issue is similar in nature the first issue but on Facebook Connect. The same vulnerability and attack class is applied because there are certain aspects of the message which are writable. However, when this attack is attempted in practice, the attack failed. To get the

attack to succeed, vulnerabilities in Adobe's Flash player had to be leveraged. This is interesting because in this case we see a vulnerability which should be easy to exploit but in practice is not so simple and actually more needs to be done. The CIA failure is a compromised account because the attacker can impersonate the user to relevant SPs. This issue occurs at the implementation level as the authors demonstrate their attack on an implementation. The authors state that they communicated with Facebook and fixed the issue so we mark the solution to this issue as implemented.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	Message Modification	Message Formatting	Compromised Account	OpenID	Implementation	Implemented
2	Message Modification	Message Formatting	Compromised Account	Facebook Connect	Implementation	Implemented

Table 3.31 Categorisation of Wang et al. (2012)

3.6.7. Google Accounts

There were two papers Armando et al. (2008); Wang et al. (2012) with security issues in Google Accounts but we discuss these in the SAML and Facebook section respectively. Note that for both of these issues, they are not explicitly categorised as issues for google but as the protocol suite that google is using. The rationale for using google in our search is because it is a big IdP like Facebook so there was likely to be significant research into Google. However, the difference between Google and Facebook in this categorisation approach is that Facebook used a proprietary Facebook Connect protocol while Google is implementing existing Protocol suites. Hence, Facebook is often referenced as a target protocol in this work while google is not, instead defaulting to the protocol being used.

3.6.8. Shibboleth

Chadwick (2009) briefly mentions a security issue for Shibboleth. Although the issue is described briefly, it still fulfills the requirements to be in this survey as Shibboleth is one of the FIM systems we are considering and we can gather a vulnerability, attack class, and CIA failure from the description. The issue is described as a phishing attack which we attribute to a bogus merchant attack class. The attacker forwards the user from an attacker controlled SP to an attacker controlled IdP masquerading as a legitimate IdP. We state that the vulnerability is insufficient trust infrastructure because the attacker leverages the user's trust in the system by impersonating the system. The CIA failure is a compromised account because the attacker can use the credentials stolen from the fake IdP to login to the real IdP. No direct solution is applied to this problem within Shibboleth. There is discussion on how using identity cards could prevent the issue, but this is more of a comparison to Microsoft Cardspace.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	Bogus Merchant	Insufficient Trust Infrastructure	Compromised Account	Shibboleth	Protocol	None

Table 3.32 Categorisation of Chadwick (2009)

(a) Coded Vulnerabilities

Vulnerabilities
Centralised Infrastructure
Insufficient Trust Infrastructure
Weak DNS
Insecure Communications
Automatic Authorisation
Lack of Binding
Weak User Credentials
Vulnerable SP
Message Formatting
Vulnerable IdP
Dual-Role IdP

(b) Coded Attacks

Attacks
DoS
Bogus Merchant
MITM
DNS Poisoning
Replay Attack
Malicious Provider
Packet Sniffing
XSS
Message Modification
Session Swapping
CSRF
Brute Force
Account Binding
ACS Spoofing
UI Redressing

Table 3.33 Coded Vulnerabilities and Attack Classes in FIM

3.7. Presentation of Results

In this section we present different ways of viewing the data that we have collected. We leave the interpretation of that data to a discussion section.

3.7.1. Security Issues

As we specified in 3.4.3, we analysed a total of 31 papers. From these papers, we extracted a total of 66 security issues that were found from the papers. On average each paper produced just over 2 issues. The protocol suite with the most issues is OpenID with 24 issues. The protocol suite with the least issues is Shibboleth with 1 issue.

3.7.2. Vulnerabilities and Attack Classes in FIM

We found 11 unique vulnerabilities and 15 unique attack classes from our survey which can be seen in tab. 3.33. These vulnerabilities and attack classes are introduced in our enumeration section (3.5). We show how these vulnerabilities and attack classes are categorised from the literature in section 3.6. In fig. 3.4 we can get an idea of the different vulnerabilities and attack classes that occur on each FIM protocol suite we consider.

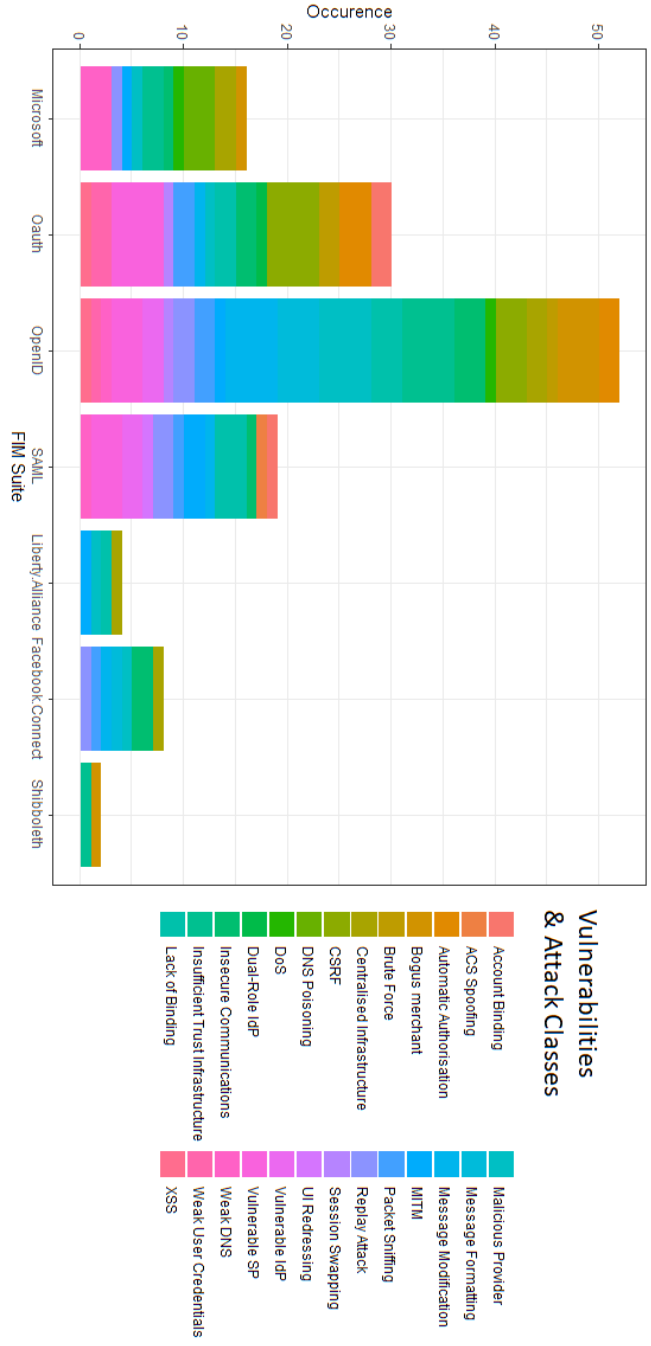

Figure 3.4 Occurence of Vulnerabilities and Attack Classes on FIM Protocol Suites

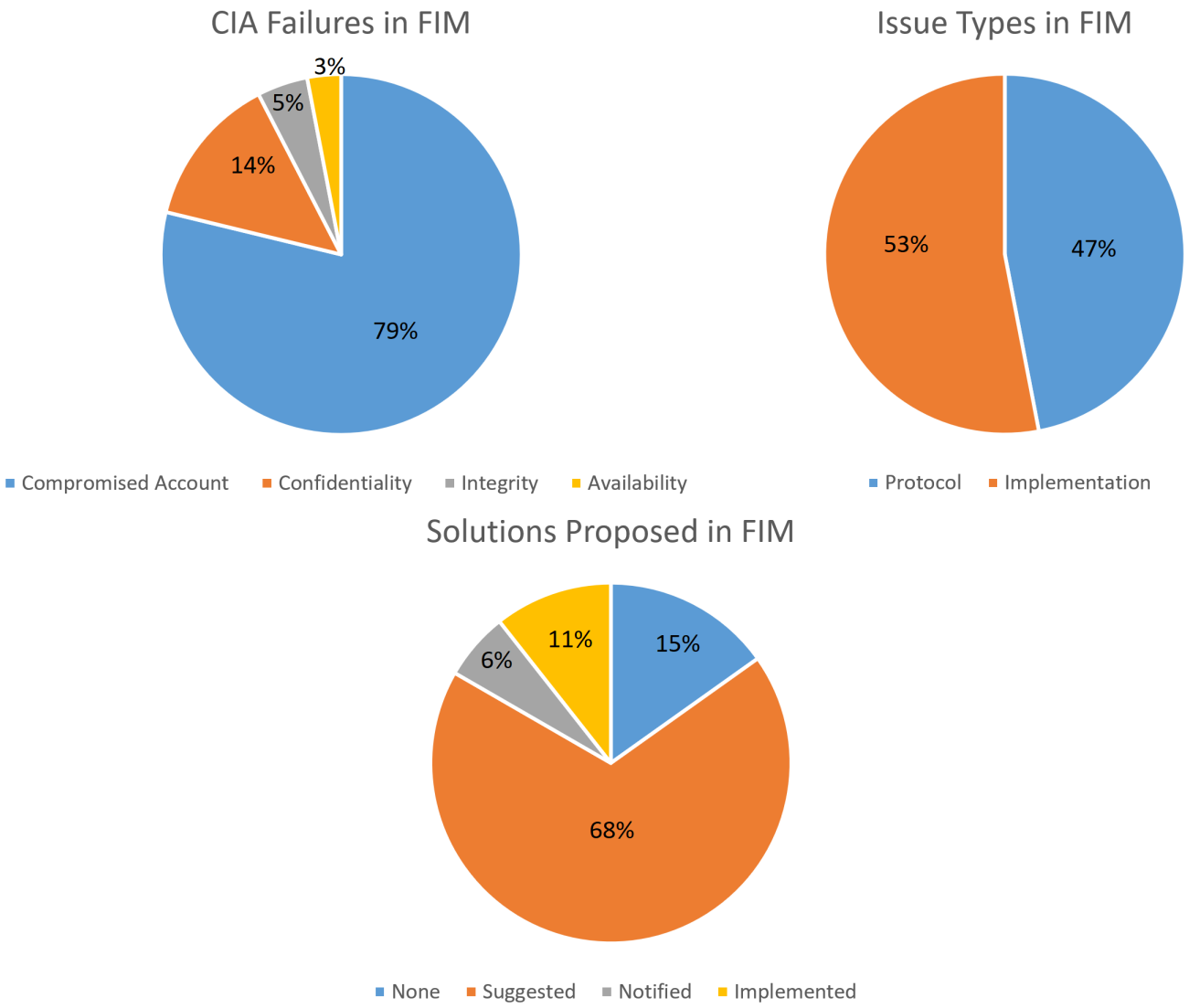

Figure 3.5 Proportion of CIA Failures, Issue Types, and Solutions Proposed in FIM

3.7.3. CIA Failures, Issue Type, and Solution Proposed

Of the 66 security issues presented, the majority of issues resulted in a compromised account at 52 (∼79%). The next most common failure is confidentiality with 9 (∼14%) security issues. Integrity and Availability occupied the rest of the security issues at 3 (∼5%) and 2 (∼3%) respectively. There was a fairly even split between the two possible categories of issue type. The implementation issue type has 35 (∼53%). The protocol issue type has 31 (∼47%). Most issues had solutions that are suggested in 45 (∼68%) instances. The next highest category for solutions proposed is none with 10 (∼15%). Issues that are implemented or relevant institutions notified are 7 (∼11%) and 4 (∼6%). All of this information can be seen in fig. 3.5.

3.7.4. Cross-Protocol Issues

We identify 15 different cross protocol issues (see tab. 3.35). We identify a cross-protocol issue when the same vulnerability, is exploited by the same attack class across more than one protocol suite. Whether or not the same CIA failure occurs is not considered, just the fact that a failure can occur.

The samples for this survey cover a long period of time and many of the specific issues that have blighted a protocol would have been fixed or be obsolete. For instance, Pfitzmann and Waidner (2003a) states that fixes were implemented directly in response to the research. Any issue that affected Microsoft Passport is not directly relevant as Microsoft Passport is now obsolete having been replaced by newer Microsoft technologies; for instance Cardspace, as reported by Gajek et al. (2009). However, while the specifics change, the general security paradigms FIM has to contend with can still be drawn from these cross-protocol issues. For instance, in 3.8.4 we note some more recent work in FIM security analysis and find that OAuth is still vulnerable to CSRF attacks as reported by Argyriou et al. (2017). To fully understand the current state of FIM in regards to susceptibility to these general paradigms, more research is needed.

For a visual representation of the relationship between vulnerabilities and attack classes consider fig. 3.6. In subsection 3.8.2 we cover some application for this graph, but here we summarise the basics of what this graph means. The graph is a way of visualising these cross-protocol issues where the emphasis is on attack classes that can be reached from the most vulnerabilities. The size of the nodes in the graph indicate how many ways that attacks can be reached from a vulnerability. The weight in the edges indicate how many different protocol suites see the same issue.

3.7.5. Solutions/Mitigations

We consider some of the solutions for dealing with the vulnerabilities and attack classes shown to occur as a result of this survey. This is in not a survey of how these attack classes can be mitigated, we simply use a solution provided in the already surveyed literature so that some reference exists for mitigating these problematic cross-protocol issues. This is in answer to RQ1.2.

Cross-Protocol issue 1: The malicious provider profiling a user is difficult to stop so most of the papers do not present a solution for it. There is one exception, Feld and Pohlmann (2011) reference a German identity card called nPA where a person attests to another person who they are, using biometrics.

Cross-Protocol issue 2: According to Feld and Pohlmann (2011), the German based identity card nPA can address bogus merchants by introducing a higher level of authentication.

Cross-Protocol issue 3: Urueña et al. (2014) suggest disabling the HTTP referrer tag which is known to leak information.

Figure 3.6 Network Graph of Vulnerability Attack Class Combinations. The size of the nodes indicate the number of ways that node is reachable. The weight of the edges indicate how many cross-protocol suites see the same issue.

ID	Vulnerability	Attack Class	Affected Protocols	# Affected Protocols
1	Centralised Infrastructure	Malicious Provider	Microsoft, Facebook, Liberty, OpenID	4
2	Insufficient Trust Infrastructure	Bogus Merchant	Microsoft, Shibboleth, OpenID	3
3	Insecure Communications	Packet Sniffing	Microsoft, Facebook, OAuth, OpenID	3
4	Vulnerable SP	CSRF	SAML, OAuth, OpenID	3
5	Message Formatting	Message Modification	Facebook, OpenID	2
6	Lack of Binding	MITM	Liberty, SAML	2
7	Lack of Binding	Replay Attack	SAML, OpenID	2
8	Lack of Binding	Message Modification	SAML, OAuth	2
9	Lack of Binding	Session Swapping	OAuth, OpenID	2
10	Vulnerable SP	Account Binding	SAML, OAuth	2
11	Automatic Authorisation	XSS	OAuth, OpenID	2
12	Automatic Authorisation	CSRF	OAuth, OpenID	2
13	Weak User Credentials	Brute Force	OAuth, OpenID	2
14	Weak DNS	MITM	SAML, OpenID	2
15	Insecure Communications	Replay Attack	Microsoft, Facebook	2

Table 3.35 Cross-protocol issues in FIM history.

Cross-Protocol issue 4: Sun et al. (2012) suggest binding requests to the session taking place by hashing a secret together with a session id and appending that token into a hidden field in the login form as a solution to CSRF.

Cross-Protocol issue 5: The basis of this attack class lies in the fact that the OpenID message (which is used in Facebook and Google implementations) can be modified by an attacker. Sun et al. (2012) present a Diffie-Hellman key exchange to mitigate the attack class. The IdP also has to sign an assertion for the Diffie-Hellman key exchange to be secure.

Cross-Protocol issue 6: When access tokens are not explicit to single SPs a MITM attack class can be launched. Pfitzmann and Waidner (2003b) propose (amongst other methods) a way of binding the token to the SP it is intended for.

Cross-Protocol issue 7: Groß (2003) suggests binding the IP address to a request to stop Replay attacks.

Cross-Protocol issue 8: Sun and Beznosov (2012) note that SPs do not actually check some credentials which allow an attacker to engineer fake credentials, so the SP checking those credentials is a mitigation to the problem.

Cross-Protocol issue 9: Li and Mitchell (2016) suggest adding a state value to bind a message in order to mitigate Session Swapping.

Cross-Protocol issue 10: Li and Mitchell (2014) remark that the simplest way of addressing the account binding is requesting the user to input the account name and password before completing the binding. This functionality should exist as a standard in the federation.

Cross-Protocol issue 11: Sun and Beznosov (2012) observes that the XSS attack is caused by the automatic authorisation functionality provided by OAuth. This functionality could be removed to stop this attack.

Cross-Protocol issue 12: Similar to 11, Sun and Beznosov (2012) observes the problematic automatic authorisation feature. This functionality could be removed to stop this attack.

Cross-Protocol issue 13: Alotaibi and Mahmmod (2015) state that biometrics can be used as a solution to weak user credentials.

Cross-Protocol issue 14: Alrodhan and Mitchell (2009) point out that the widespread use of DNSSEC could mitigate the difficult to address DNS vulnerabilities.

Cross-Protocol issue 15: The solution to the replay attack as put by Miculan and Urban (2011) is to ensure SSL/TLS is used.

3.7.6. Threat Models

Throughout the survey we observed several different threat models. For some papers, it is not always clear which threat model is used so we cannot be exhaustive. We could perhaps deduce a threat model by considering each attack in detail and interpreting the precise capabilities needed and then aggregating these capabilities together to form a complete threat model. However, this could introduce mistakes so we avoid that in favour of threat models that are made clear.

Some papers can also be specific in the threats considered in their analysis, but do not formally attribute those threats to other threat models which can make the threat model difficult to generalise. This could be because authors have introduced a bespoke threat model that is suited to the attacks they have found. For instance, Van Delft and Oostdijk (2010) is specific in the threats being considered, but does not clearly attribute those threats to other works which makes the threat model being used difficult to generalise.

Dolev-Yao: A typical threat model used to analyse the security of FIM protocol runs was Dolev-Yao. In Dolev and Yao (1983), the model is presented formally. In the survey we note that Armando et al. (2008) specifically state that the Dolev-Yao model is used. Other works that seem to be using the Dolev-Yao model in that messages are carried by the attacker and synthesised in such a way that is only restricted by the cryptography of the message are Groß (2003), Pfitzmann and Waidner (2003a), Yang et al. (2016).

OAuth Threat Model: A number of papers analysing OAuth considered the OAuth threat model presented by Lodderstedt et al. (2013). For instance, Li and Mitchell (2014), Ferry et al.

(2015) both consider the threat model as a basis. Other works in OAuth use this threat model as a basis, but build on the threat model to include some vectors that the original threat model misses. Sun and Beznosov (2012) and Shernan et al. (2015) both consider how OAuth is used in practice and consider additional web attacker scenarios which are overlooked by the OAuth threat model.

Bogus Merchant/Phishing Based Attacker: Some work focuses on an attackers ability to pose as legitimate parties which makes users susceptible to bogus merchants and phishing. Kormann and Rubin (2000) focuses on this aspect originally. Others include Feld and Pohlmann (2011), Abbas et al. (2016).

High Value Targets Compromised: Mainka et al. (2014), Mayer et al. (2014) both consider threat models in which IdPs are compromised. This vector refers back to the risks in FIM identified by Kormann and Rubin (2000) who introduces the centralised infrastructure risk. We remark that making FIM systems secure in the presence of IdPs being malicious seems like a difficult challenge due to the inherent centralising of user data on IdPs. There are concerns with large identity providers like Facebook as raised by Dwyer et al. (2007). The security of FIM with respect to this threat model remains an open question.

Coffee Break Access: One threat model we believe is largely missing from the literature in FIM security analysis is the idea of temporary access. This kind of threat model is considered by Maxion and Townsend (2002), who considers attackers masquerading as users and detecting those attackers. The scenario of temporary access is likened to a coffee break that a user could take by Maxion and Townsend, hence we refer to this threat model as coffee break access. In other security literature, cold boot attacks are considered which is where data can be extracted from RAM after a device is powered off. However, the attacker only has a limited time frame to do this in as explained by Gruhn and Müller (2013). The kind of threat model required being physical access but in a short time window which is similar to coffee break access. In FIM security research, Kormann and Rubin (2000) considers the risk of persistent cookies being left on public machines and an attacker being able to break in to that computer physically so there is some consideration for this in FIM.

However, we believe that coffee break access is a threat model that is largely unexplored in the literature of FIM analysis. Consider Facebook which stores a wealth of information on the user as identified by Krasnova et al. (2014). If a user leaves themselves logged on while on a coffee break, then this information can quickly be gathered by an attacker or authorisations to certain other apps could be granted without the user's consent. The full extent of the security of FIM given a coffee break access type threat model is not clear, and is a worthy direction for future research.

3.8. Discussion

3.8.1. The landscape of security analysis of FIM

A large part of addressing RQ1 is to get an overall picture of the issues that face FIM. In this subsection we make general observations from the results we present in section 3.7. These general observations include information on the kinds of CIA failures observed, the issue type, the protocol involved, and solutions proposed.

In the majority of security issues we survey, the Compromised Account was the most common CIA failure. This is also the CIA failure with the most impact, seeing as an attacker can potentially compromise any CIA property from a compromised user account. In addition, we do not observe many security issues that exploit solely integrity or availability. We assume that researchers know that a compromised account is the most fruitful type of compromise possible because of the range of CIA properties that can be undermined. Thus, researchers will try to develop an issue to the point in which a complete account compromise is possible. It is easy to convince audiences that compromised accounts are a problem, so it is possible that other issues are under-reported, as the impact of these other issues is harder to sell.

We have seen a fair balance between security issues at the protocol and implementation level. It is important to continue to evaluate both protocols and implementations in FIM because one can not succeed without the other. Although, we do note that some of the issues did evaluate a protocol and then test on an implementation e.g. Sun et al. (2012) finds several issues with OpenID at the protocol level and then turns attention to actually exploiting those issues at the implementation level. What this means is: we can still evaluate the security of FIM through considering issues at the implementation level. It is important to continue the research involved with the formal evaluation of protocols so that general issues with FIM can be caught before a disastrous multi-implementation level problem occurs that is the result of several implementations all following the same flawed protocol idea.

Some protocols have received more attention from analysts than others (such as OAuth) and we can therefore produce a clear history of the security issues for those protocols. Other protocols have not received such wide spread attention, especially WS-Federation which our survey did not turn up a security analysis that presented a security issue. There is work in the area, such as work done by Groß and Pfitzmann (2004) that demonstrates the protocol is secure under certain assumptions, but work investigating the flaws in WS-Federation seems to be missing. Other low attendance protocols are Shibboleth which also has not received that much attention. Are these low-attention protocols more secure or are researchers turning a blind eye to them for one reason or another i.e., perception of less people using these protocols?

A positive story is that the majority of security analysis are not only pointing out vulnerabilities and attack classes which can be used to exploit those vulnerabilities, solutions are attached also with only 15% of issues not claiming a solution. This of course does not mean that a solution is

not possible, just that the author has not attributed a solution. For example, Yang et al. (2016) considers an issue involving packet sniffing attack class on an insecure communications vulnerability on OAuth and suggests no solution. It is clear though that proper end-to-end encryption can mitigate this issue, as is discussed by Sun and Beznosov (2012) on the same kind of issue with OAuth.

The majority of FIM solutions are suggested but it is often not clear if relevant organisations are informed on issues. We consider the impracticality of ensuring the solution is implemented before the paper is published. For some papers, the solution is implemented (11%) but it is comparatively rare. There are many things that are outside the control of the researcher, such as the transparency of an organisation in the fixing of issues, or the amount of organisations that must implement change for the issue to be entirely fixed. For these reasons, we consider FIM security issues to be fixed at the time of publishing generally impractical. However, one aspect on FIM security research we think could be improved is that the relevant organisations are notified of the issue. We do not find much evidence of organisations being notified at just 6%. Those issues which were fixed the relevant organisations must have also been notified, which brings the total to 17% but we consider this to be a low rate of organisations being made aware of issues with FIM. One thing we have to consider is that for some of these issues, the relevant organisations might have been informed. However, we believe that it should be made clear in a publication if the relevant organisations have been informed. Such a change would encourage the practice to become the norm, and this would be beneficial because organisations which are informed of issues can surely better protect the CIA requirements on information, which is a larger goal of information security on the whole. Furthermore, the transparency of communication of issues with organisations in a publication can ensure non-repudiation for organisations and can help protect the integrity of individual researchers reputation in the event an issue detailed in an academic publication assists in cyber-crime.

One aspect of FIM solutions that became clear from our survey is that some suggested solutions are also outside the scope of organisations that have direct control over FIM systems and are larger problems that must be addressed with the internet in general. For example, Groß (2003) considers several issues in SAML. When considering solutions for the issues, it is mentioned that there are some aspects of the issue that are outside the scope of organisations that control FIM systems. If we consider a weak DNS as Groß does, it is clear that this problem is a problem with the internet at large, and can not just be fixed under the influence of FIM organisations alone.

3.8.2. Cross-Protocol Issues

We found 15 security issues which are cross-protocol as can be seen in tab. 3.35. The graph 3.6 (initially covered in 3.7.4) shows a clear relationship between vulnerabilities and attacks. For instance, we can ascertain that the Malicious Provider attack class is one of the most prominently discussed attack classes in FIM security analysis literature. The vulnerable SP vulnerability is a prominent vulnerability, leading to many different attack opportunities in the

literature. In this subsection, we discuss these different cross-protocol issues. We believe this to be a particularly important outcome of the research because we can see what issues are transcending individual FIM protocol suites and are issues with FIM information security in general. This is in answer to RQ1.1.

Some issues were not surprising as they capitalise on well known FIM weaknesses. Cross-protocol issue *1*, happens across FIM protocols because if an IdP is malicious, they can easily track a user. This issue was also the most prolific across protocols being reported across four different protocol suites. Cross-protocol issue *13* is also not surprising in that if a user does not use strong credentials then they are at risk of a brute force attack. However, we do note that this attack class could be more impactful on a FIM system due to the fact that access to a FIM account could be used to access several SPs in the name of the user. Similarly, cross-protocol issue *2* (bogus merchant) is also an issue not exclusive to FIM but could be more impactful on FIM systems.

We see attacks which involve common web weaknesses, like DNS flaws, being used in creative ways in FIM. For example, we see intricate MITM attacks with likeness to what is described by Groß (2003). This issue has been shown to be cross-protocol and and in the table is number *14*. Also note that liberty is based on SAML, so it makes sense that we see cross-protocol issue *6* across these protocols.

What is quite relevant to our research about attack escalation is that common web vulnerabilities can be exploited in ways that undermines the security properties a protocol is supposed to enjoy. Surprisingly, we find vulnerabilities such as insecure communications leading to packet sniffing in cross-protocol issue *3*. It is surprising that these issues, which should easily be fixed through end-to-end encryption, exist. Issues like XSS and CSRF are also prevalent in FIM, through cross-protocol issue *11* and *12*. All of these issues lead to protocols which are otherwise thought to be secure being rendered insecure due to some oversight with an implementation.

We observe cross-protocol issue *10*, which involves an account being bound to an attacker account. This particular attack is interesting because it is unique to the nature of FIM, but also is happening across protocols. Some FIM systems allow for the binding of SP accounts of a user to the IdP account of an attacker. This creates an additional attack vector exclusively for FIM.

3.8.3. Survey Limitations

We performed a systematic survey on Scopus and Google Scholar. However, there is still the possibility that papers could be missed. We are aware of one notable paper which did not appear from our search in Microsoft by Kormann and Rubin (2000) which has been included as an other source.

We do not attempt to discern the quality of the reviewed papers. This introduces the fact that security issues, which some may deem to be trivial, are represented in the survey. For example, a point of view that could be taken is that brute force attacks are always going to be a risk in

authentication. However, how we can defend against brute force attacks becomes the focus, rather than the issue itself. On the topic of the judgement of security issues, we also cannot guarantee that every security issue is correct because if we begin to use our judgement to discriminate what we perceive as incorrect results we risk compromising the survey.

The searches were performed in 2016. Any research into attacks in FIM since then have not been included in this survey. However, we are not aware of any research that has demonstrated the effectiveness of any completely novel attack classes, or leveraged some new vulnerability. In the next subsection (3.8.4), we will briefly cover some work in the security analysis of FIM that has been published since our survey was conducted.

In our effort to make our taxonomy conform to the rules of a good taxonomy outlined in 3.2.2, some issues may have been simplified. Particularly we consider the property of a specimen fitting at most one category. Some issues are influenced by multiple different attack classes. For example, Mayer et al. (2014) introduces a UI redressing attack class. The details of that paper make it clear that other attack classes have influence, we note that at the very least XSS assists in the overall attack. We have a few choices before us in handling this situation. The first is to make a new category of attack which is a super category of both of these attack classes. We chose not to do this because then we end up with attack classes which are too specific, and cannot be compared across protocol suites. The second is to list the issue as two attack classes, but that conflicts with the good taxonomy (outlined in 3.2.2 property outlined by Lindqvist and Jonsson (1997) where specimens fit into at most one category. The third choice is to pick the most influential, or the most unique property of the issue and use that to characterise the whole issue. For example, for Mayer et al. (2014), the UI redressing is significant because we already have substantial evidence XSS is an issue. The result of this decision making is that we concede some details are omitted in the taxonomy, but this was necessary to maintain a good taxonomy.

3.8.4. Recent Work

The headline for the recent work we consider is that there are some vulnerabilities and attack classes that break new ground. However, we find that our taxonomy is general enough that these attacks, of which some could be interpreted as a specific new attack, can be encapsulated in our general taxonomy already. If we performed the survey again, the thing that would be different is the instances of observed vulnerabilities and attack classes would increase. Of note, it seems that in modern times the FIM technologies receiving the most attention are OAuth, OpenID, and SAML. This could be an indication of the increasing prevalence of these technologies.

We summarise a non-complete list of work published since our survey in table 3.36. We mostly consider which FIM technology the work occurs in, with multiple representing a paper that covers a number of FIM technologies. We also include a brief summary of the types of security issues observed and why they already fit into our taxonomy.

Citation	FIM Technology	Description
Agrawall et al. (2017)	Multiple	Considers CSRF attacks on FIM, but this would not add any new vulnerabilities or attack classes to the taxonomy as CSRF attacks are already considered. The vulnerability here seems to be a vulnerable IdP. For example, one mitigation of the attack is that POST can be used for a more secure logout feature.
Chauhan and Tiwari (2018)	Multiple	Discusses several vectors for attacks and introduces a Trusted Computing framework to mitigate these attacks. All attack vectors discussed are already attack classes in this thesis, albeit the labelling might be different. For instance, escalation of privileges as described is referred to automatic authorisation in this thesis. Phishing is a bogus merchant attack.
Naik and Jenkins (2017)	Multiple	States the use of several attacks for SAML, OAuth, and OpenID connect as part of a larger assessment of these FIM technologies. The attacks specifically mentioned are DoS, MITM, and XSS. All of these attacks are covered directly in our taxonomy.
Engelbertz et al. (2018)	SAML	Provides a method for performing Document Type Definition (DTD) attacks on an implementation of SAML (eIDAS). Although these DTD attacks could make a new attack class, they can also fall under our more general message modification attack class. We try to be general in order to be able to classify cross-protocol attacks, and DTD attacks are exclusive to XML based technologies like SAML and so are not going to be possible in FIM technologies that do not utilise XML.
Fett et al. (2019)	OpenID	Provides a formal analysis of OpenID. As part of the analysis four attacks are discovered. At least some of these attacks seem to break ground for OpenID frameworks. However, we find that these attacks can be generalised into our taxonomy. For example, The first attack relies on the access token not being bound to the protocol instance and can then be used for impersonation. We therefore describe a lack of binding and a message modification attack as the token is bound to a message created by the attacker. The other attacks include CSRF and replay attack elements.
Argyriou et al. (2017)	OAuth	Discusses the benefits and flaws of OAuth. Includes several attacks that when translated to our taxonomy include MITM, CSRF, and DoS attacks.

Table 3.36 Incomplete Summary of Security Analysis in FIM Since Publication of Survey

3.8.5. Conformity to Good Taxonomy

We compare our survey to the notions introduced in 3.2.2, which considers good taxonomies. Does our taxonomy conform to these notions?

- Categories are mutually exclusive. Specimens fit into at most one category and we have no instances in the taxonomy of ambiguity in a classification.

- Every category has a description of what belongs in the category. In section 3.5 we outline what it means for specimens to fit the values in the dimensions of the taxonomy.
- It is the hope that the taxonomy is useful not only to experts but users and administrators. We have explained each category and what belongs in each category. If users cannot understand the technical details of the various attacks, they should at least be able to gain use out of the protocol that is affected by the attack. This could guide a user in what attacks to look out for while using the protocol suite that the issue affects and they could understand how the issue affects them in terms of CIA.
- The taxonomy mostly conforms to existing security terminology. Attack classes mostly use existing terminology, such as session swapping, CSRF, XSS, MITM, etc. There are some attack classes which this survey uncovers which are less well known, such as account binding. However, it was necessary to attribute these novel attack classes correctly. Failures are defined in terms of CIA which is well understood. The target of the attack specifies well known security protocols. Specifying an issue at the protocol level or the implementation level is nothing new, and is even distinguished in some papers in the survey such as Sun and Beznosov (2012). The solution of the issue is specified where it should be easily understood if a solution was published in the paper. Perhaps the area most lacking is vulnerabilities. Attempting to generalise vulnerabilities so that they are meaningful across protocol suites has been a challenge. For instance, the lack of binding vulnerability. This vulnerability considers the situation where a message has not been cryptographically bound to the context it is used in but this might not be immediately obvious security terminology. This is a concession that is made to be able to generalise security vulnerabilities and compare them across protocols.

3.9. Application and Validation of Taxonomy

We have noted one study that describes to a high standard how attacks can escalate in a FIM system. Sun and Beznosov (2012) identifies clearly how attacks can escalate in FIM. For written reference on how attacks can escalate in OAuth, consider section 5.5 in the study provided by Sun and Beznosov. For a visual, we have included fig. 3.7. We clarify that this figure is a figure copied from Sun and Beznosov.

The figure depicts a high-level overview of causality of the attacks described by Sun and Beznosov. We simplify this figure to avoid unnecessary details that are not relevant to our work in order to provide a clear view of the causality in the attacks. We state these changes and the rationale behind these changes.

Sun and Beznosov include different relationships between the attacks labelled "enables", "leads to", and "helps". The work we do here does not differentiate between these qualitative statements about the degree of causality in attacks. The important thing is that there is causality, therefore we discard the degree of causality established.

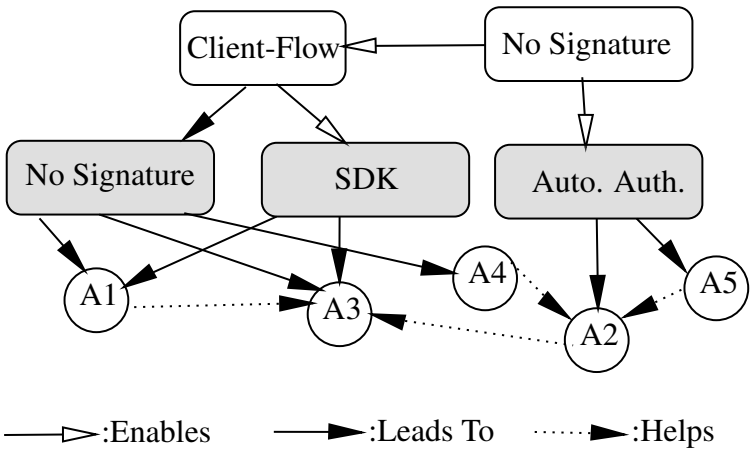

Figure 3.7 Attacks on OAuth as a Dependency Graph created by Sun and Beznosov (2012)

Sun and Beznosov provides underlying reasons as to why the vulnerabilities exist in OAuth. OAuth 2.0 has a number of different modes of operation including the mode of operation which is the main focus of analysis: client-flow. Another underlying reason is that the digital signature aspect of the protocol specification has been removed. The reasons why the specification changed and the different modes of operation for OAuth 2.0 is outside the scope of this and are removed.

Sun and Beznosov provides two different ways that access tokens are exposed to the attacker on a network. This is not to be confused by the ways an attacker can retrieve an access token in general. For example, an access token can be stolen through A2 as well. The first is through what is described as an "authentication-state gap". We find that Sun and Beznosov are referring to two different vulnerabilities when considering the "authentication state gap". Firstly, they consider when an access token is sent to a client-side script erroneously. Secondly, they refer to a "lack of contextual binding". The case for the identified "lack of contextual binding" vulnerability being its own independent vulnerability is clear when examining the textual descriptions of A3 and A4 where these attacks are specifically referred to being possible because of this "lack of contextual binding".

The second way access tokens can be exposed on the network is through what is described as "cross-domain communication SDK". This is when an IdP makes another web page usable through an iframe without having to leave the IdP (also known as cross-domain communication).

If an attacker is able to get an access token through either the "authentication state gap" or through the "cross-domain communication SDK", the attacker is immediately able to perform A3. This does not allow for much room in escalation so we do not consider these vulnerabilities in identifying causality for our model of escalation in the work provided by Sun and Beznosov.

However, the "lack of contextual binding" is still important to consider. This vulnerability allows for A3 to be possible because the token is not bound to the browser it originates on. Similarly, for A4, the "lack of contextual binding" vulnerability allows the attacker to bind a token to the session belonging to the user.

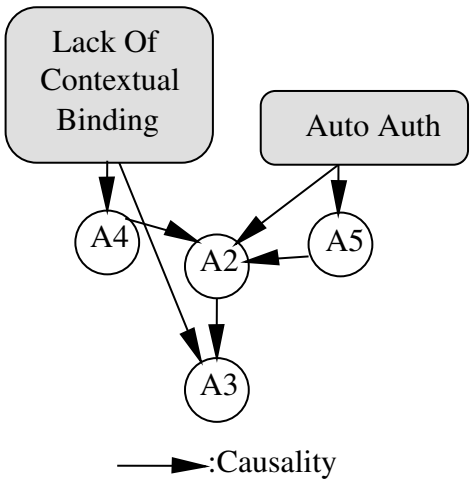

Figure 3.8 Attack Causality on OAuth Interpreted for our Work

Instead of an attacker getting an access token over a network and immediately being able to perform A3 we model getting to A3 through A2 which is possible through A4 and A5. If an access token is obtained by means other than over the network, the important part of the "authentication state gap" vulnerability is the lack of contextual binding. When considering A4, we simplify how the attacker obtains their own access token by just saying that if the attacker has an account they are able to get the access token.

We present our view of of causality of attacks in OAuth which is based on the work by Sun and Beznosov in fig. 3.8.

There are issues of attempting to model escalation directly using our attack categorisation. In this chapter we have categorised attacks in FIM. We explain in subsection 3.4.3 the different problems in using MAFTIA directly for categorisation. In essence, the level of detail provided by authors is either too vague or too detailed and would result in a categorisation that is not general enough to capture these different levels of detail. Hansman and Hunt (2005) also considers a similar problem where an attack can be hard to interpret from the literature. If we tried to use our taxonomy directly for modelling escalation there would be two main challenges.

1. **Attack Classes Are Too General.** Attack classes can be applied to different situations. Consider an XSS attack class. In Sun and Beznosov (2012) we can see how an XSS attack could be used to stage a CSRF attack by luring a user to the page in which the CSRF attack could be launched. Alternatively, the XSS attack could be used to steal an access token if the user is logged in. This is clearly two different attacks that are within the same attack class. If we use attack classes, it is difficult to reason about what attacks are possible in a given system model. If we state that the presence of an XSS attack leads to set of effects that are always possible, we risk an incorrect model which does not represent the literature.

2. **User Orientated Failure.** The primary focus of the literature is how the user's information security is affected. There is little consideration for other systems within FIM

that can fail. In order for an XSS attack to impact a FIM system as can be seen in Sun and Beznosov (2012), the Browser first has to interpret the malicious XSS code as legitimate code. Moreover, the SP has to somehow have this malicious code be injected into it in the first place. If we are trying to consider how failures on one system in FIM lead to other failures, we need to consider failures on those systems separately from the user failures.

We use MAFTIA as a framework for considering attacks in a causality chain. We apply a further level of interpretation to the attack classes we have interpreted through the study of the work provided by Sun and Beznosov (2012). The aim is to have an encoding of knowledge to be used for modelling escalation. Since Sun and Beznosov provides a clear blueprint for how attacks escalate, we choose those attacks for a case study. In particular we consider the vulnerabilities, attacks, intrusions, errors, and failures. We also consider how each attack can also be enabled by an intrusion as a result of another attack succeeding.

In addition, we also need to consider the systems targeted by the attack. For example if we consider an XSS attack a particular SP would be targeted. We also want to consider any other involved systems. For example, if the code inserted as the result of a successful XSS attack is to be executed, there must exist some kind of connection between the user and the SP.

3.9.1. Escalation of Attacks in OAuth

We consider the work by Sun and Beznosov (2012). We will focus on attacks A2, A3, A4, and A5 which have the capacity to interact with each other. We refer to A2 as access token theft via XSS, A4 as session swapping, and A5 as force-login CSRF. We refer to this entire attack suite in this thesis generally as the *OAuth Case Study*. A1 considers an access token theft via unencrypted communications. The reason we do not consider this attack is because it has less capacity for escalation than the other attacks. After an attacker is able to perform A1 the the attacker is immediately able to perform A3. The alternative path to A3 requires A2 and either A4, or A5 which represents more steps that the attacker needs to take than to just perform A1 and then A3.

Synopsis of Our Interpretation of the Overall Attack: We outline specific paths the attacker can take in the overall attack suite. The attacker wants to steal an access token via an XSS attack on a SP within the FIM federation. This can be accomplished by an XSS injected script on any page of the SP. However, we consider that the only parameters on the SP that are susceptible to XSS are behind a login as described by Sun and Beznosov in subsection 5.5 of their paper. An example of the specific parameter is the username itself. This would mean that a login is required to the SP but it does not matter how that occurs. The attacker can login the user into their user account through a force-login through CSRF. Alternatively, the attacker could login the user into the attacker's account via Session Swapping. No matter if the attacker chooses the force-login CSRF or the Session Swapping attack, the attacker first stages the attack through another XSS injection already staged on a website external to the FIM federation that the user is

browsing. This XSS injection launches the force-login CSRF or the session swapping as well as redirecting the user to the SP so that the XSS injection on the SP internal to the FIM system can be executed.

Malicious Code on the Browser Through XSS: Both the Session Swapping and the Forced-Login CSRF require that the user first renders some malicious code on the Browser. Since we are considering the Session Swapping and Forced-Login CSRF as causation for the XSS token theft the question is, how could the user render malicious code on the Browser? One way considered is through XSS attacks on a benign website outside the OAuth federation. This is not to be confused with the XSS token theft described by Sun and Beznosov as A2 because for that attack to be possible there first needs to be malicious code inserted onto the server in order to execute the XSS token theft. XSS attacks are constrained to the domain in which they operate on. An attacker can steal a cookie containing an access token through XSS, but only if the XSS code is coming from the same domain the cookie belongs to. This constraint is recognised by OWASP (2019b). This is why we cannot necessarily generalise the effects of an XSS attack, it depends on the context in which the XSS attack is applied. We consider the Browser the target of the attack XSS attack. In addition, the SP in which the XSS injection is staged and the connection from the Browser to the SP needs to be considered.

1. *Failure.* Ultimately the attacker is trying to get the browser to act in response to the XSS attack being launched by the external SP. The failure from the browsers point of view is in terms of an integrity goal. However, malicious code execution by itself does not necessarily constitute a failure. The attacker has to actually do something with the code that has been entered. This issue can be combined at the failure level with the next issue which executes a malicious redirect with the help of the malicious code.

2. *Error.* We consider a security rule where the Browser does not execute malicious code. Such malicious code could be in the form of an attacker sending cookies to an attacker controlled location, for example.

3. *Intrusion.* The attacker is attempting to render malicious code on the browser. The existence of that malicious code on the browser corresponds to an intrusion.

4. *Attack.* The attacker is carrying out the XSS attack to render the malicious code on the browser.

5. *Vulnerability and Enabling Intrusion.* The XSS injection has already been established on the SP outside of the OAuth federation. Realistically, this XSS injection could be part of another attack, but since our research is orientated around attacks on FIM systems we simplify this to a vulnerability that assumes that the XSS injection has already been staged. We consider another vulnerability on the browser, which is that the browser is unable to distinguish between malicious code inserted by an attacker on a SP and benign code that was actually placed on the SP as intended.

Malicious Redirect: The attacker is also trying to cause a redirect from the Browser to a SP within the OAuth federation in order to carry out subsequent attacks. We consider the Browser a target for this attack, but in doing so the attacker creates a system in the form of a connection between the Browser and the SP in the form of an intrusion.

1. *Failure.* The failure is in terms of integrity because the user does not initiate the redirect. We say that this failure also requires the malicious code on the browser so this issue combines with the malicious code on the browser issue above at the failure level.

2. *Error.* We consider a rule on the Browser where the endpoints are both trusted and initiated by the user.

3. *Intrusion.* The attacker is trying to create a connection between the browser and the SP within the OAuth federation to carry out subsequent attacks. This attacker created connection also corresponds to an intrusion.

4. *Attack.* The attacker is also trying to force a redirect from the Browser to the SP in order to establish an intrusion where a connection exists between the browser and the SP within the OAuth federation.

5. *Vulnerability and Enabling Intrusion.* We consider the malicious code on the browser intrusion as an enabling intrusion. This is because this intrusion is actually required to force the redirect described above. Therefore, within the Browser we do have an intrusion that causes another attack to be possible.

Force-Login CSRF: CSRF attacks can be used to perform unwanted actions by a user on a SP according to OWASP (2019a). This is done through forging requests on one website, and sending those requests through the user's browser to the target website if the links are unvalidated. According to Sun and Beznosov (2012), a typical restriction of the CSRF attack is that the user must be logged into the SP in order to carry out CSRF attacks. However, the force-login CSRF attack circumvents this restriction if the user is logged into the IdP and the IdP has automatic authorisation enabled. This attack targets a SP within the FIM federation because the CSRF requests are directed at the SP. The involved systems are the login at the account on the IdP that has a federation contract with the target SP and the account that the attacker wants to force the login on which is associated with the target SP. Also of note is that the attacker wants to cause an intrusion which is a login to the target SP as the user.

1. *Failure.* The failure is in terms of integrity because the user does not initiate the login. Note that in general the possible failures that can occur as a result of CSRF attacks is extensive. For instance, if there is an action to delete a user account that is vulnerable to CSRF, then availability can be denied. If there is an action to share information to some other entity that is vulnerable to CSRF, then confidentiality can be compromised. However, we say for this case study that the only thing the attacker is limited to is integrity related failures such as changing usernames as is described by Sun and Beznosov.

2. *Error.* We consider a rule where no unauthorised action should take place on a SP.

3. *Intrusion.* The attacker is attempting to create a login for the account associated with the target SP.

4. *Attack.* This attack interaction is described by Sun and Beznosov as a forced login CSRF. We also say that as part of this attack a XSS script is appended to the user's account name as described by Sun and Beznosov in subsection 5.5 of their paper. The XSS injection is required to do the XSS token theft. As a result of this attack we say that the XSS injection has been applied as an intrusion without explicitly stating it to simplify the case study.

5. *Vulnerability and Enabling Intrusion.* We consider the vulnerabilities to be the automatic authorisation feature described by Sun and Beznosov on the IdP associated to the target SP. The target SP also needs to be vulnerable to CSRF attacks. We describe this as an unvalidated link vulnerability since at the heart of CSRF attacks is this aspect of invalid actions not being protected against. The CSRF also needs to be staged by the malicious code that has already been inserted into the browser.

Session Swapping: Session Swapping is an alternative to force-login CSRF in exploiting the malicious code on the Browser. Instead of logging on the user into the user's account, the attacker logs the user in as the attacker. Due to the automatic authorisation vulnerability, all the attacker needs to do is get the user to view an XSS exploit on the page of the SP. Since the XSS vulnerable parameter requires the user to be logged in, the attacker must have the user log in but it does not matter which account the user is logged in on. This attack is possible because of what is described by Sun and Beznosov as a lack of contextual binding. That is, the access token that is associated with a login is not bound to the session on the browser that the access token was generated for. The attacker is then able to embed the login request into HTML (in this case injected through XSS) to log the user into an account the attacker has an access token for (the attacker's account). The target of this attack is a SP within the FIM federation. Other involved systems is the attacker's account which the attacker is trying the session swap on. Also of note is that the attacker wants to cause an intrusion which is a login to the target SP from the user as the attacker.

1. *Failure.* The failure is in terms of integrity because the user does not initiate the login into the attacker's account. In general a confidentiality failure should be possible here. It is possible that the user does not realise that they are not on their account and accidentally divulges confidential information to the attacker. However, to simplify the case study we say that the attacker uses this session swapping attack as a stepping stone for the XSS token theft.

2. *Error.* We consider a rule where no unauthorised action should take place on a SP.

3. *Intrusion.* The attacker is attempting to create a login as the attacker for the account associated with the target SP.

4. *Attack.* This attack interaction is described by Sun and Beznosov as session swapping. As part of this attack a XSS script is appended to the user's account name as described by Sun and Beznosov in subsection 5.5. The XSS injection is required to do the XSS token theft. As a result of this attack, we state that the XSS injection has been applied as an intrusion without explicitly stating it to simplify the case study.

5. *Vulnerability and Enabling Intrusion.* We consider the vulnerabilities to be the lack of contextual binding as described by Sun and Beznosov on the SP. The session swapping needs to be staged by the malicious code that has already been inserted into the browser.

XSS Token Theft: If there is a parameter which does not have sufficient validation on a website, then an XSS attack may be possible. In the case of XSS token theft the username is not validated and is vulnerable to XSS. For XSS token theft to be possible the attacker has to force the user to login to their own account through CSRF or to the attacker's account with session swapping. The actual XSS script contains a client-flow authorisation request which is made possible through the automatic authorisation vulnerability. This authorisation request returns an access token which can be sent to the attacker. The target system for this attack is the SP internal to the FIM system as that is where the XSS script is placed. There must also exist a federation between the SP and the IdP.

1. *Failure.* The failure is terms of confidentiality. Although the access token does not carry within it any inherent personal information belonging to a user, it can be used to disclose personal information about a user and therefore should be protected.

2. *Error.* We consider a rule where no resource should be disclosed without the user's permission.

3. *Intrusion.* The attacker is attempting to obtain an access token with a script which can be used to gain access to a user account.

4. *Attack.* This attack interaction is described by Sun and Beznosov as XSS token theft. The attacker initiates a client-flow authorisation request through some malicious script somehow implanted into the server that sends the access token to a remote location under the control of the attacker.

5. *Vulnerability and Enabling Intrusion.* A vulnerability that allows for this attack to occur is the automatic authorisation granting feature. There are also a number of enabling intrusions that are required. Firstly, the attacker requires that the user is logged in. This can be done through the forced login or swapped session intrusion. Secondly, the attacker requires that the user has a connection to the SP. This can be achieved through the malicious redirect intrusion.

Impersonation: The attacker impersonates the user to the SP using an access token that the attacker has obtained. In our survey we would categorise this attack generally as message modification because the attacker crafts a message which contains the access token. Sun and Beznosov prefers the use of impersonation so to avoid confusion we use the same terminology. The target system of this attack is the account of the user itself which is an abstraction of the relationship between the credentials known to a user, the resources located on an SP, and the authenticating IdP. However, once an account is compromised the overall FIM system is said to have failed. We state this because one purpose of the FIM system is to safeguard user accounts and if this attack succeeds then the overall FIM system has failed. One involved system is the federation contract that exists between the SP and the IdP. An attacker should only be able to access an account for which there actually exists a federation between the SP and the IdP. Furthermore, a user account is required for a user on the SP that the script is placed on.

1. *Failure.* When an account is compromised the potential impact the attacker can have in terms of CIA is significant and the failure is in terms confidentiality, integrity, and availability.

2. *Error.* We consider a rule where an account should only be used by the legitimate owner of that account. We refer to this rule as account compromise.

3. *Intrusion.* The attacker is attempting to breach a user account.

4. *Attack.* The attacker uses the access token associated with a certain user account to impersonate the user that account belongs to.

5. *Vulnerability and Enabling Intrusion.* A vulnerability that allows for this attack to occur is the lack of contextual binding which allows an access token to be used from any browser session. The attack is also enabled by the stealing of an access token, in this case through the access token XSS.

Knowledge Encoding: We use the description of the attack suite provided above to synthesise a knowledge encoding that can be used to establish steps in an attack in accordance with the MAFTIA model. This encoding can be seen in tab. 3.37. For the encoding itself, there are a few details to be explained.

Firstly, commas indicate conjunction of two or more dependencies. This can be seen in the vulnerability/enabling intrusions column and the systems involved column. In these cases, conjunction is straight forward as there are no further dependencies in that row. Representing conjunction for other columns is more complex as entirely different causality chains could be required. Consider a failure that requires two errors E_1 and E_2 to be possible. E_1 requires the intrusion I_1 to be possible and E_2 requires I_2. If we encode this issue on the same row, we cannot clearly distinguish between what are the precise intrusions that make each error to be possible and the causality becomes convoluted. E_1 and E_2 would be caused by both I_1 and I_2 but it is not clear which error each intrusion causes. To circumvent this problem, we use a different row to

represent this causality chain and use the keyword *with* to illustrate that there is an additional dependency in the encoding. In practice, the only time this occurs on the table is when there is conjunction in the two errors for the XSS Execution attack and Forced Redirect on row 1 and 2. Both of these errors are required to cause the eventual failure.

Some attacks can be made possible by a system existing instead of an intrusion. In this OAuth example, a login can be forced, or the login could exist in the system organically. To illustrate this, in the systems involved column we use the key word *alternative* to indicate that a different option than an intrusion could realise the attack.

Most of the information in the encoding represent variable terms that can fulfil a dependency. There could be exceptions though that a specific type of information is needed to express some dependency through a constant. For instance, consider a session swapping attack. This attack is possible if there is an attacker account present on the system. To express this, instead of stating that any account is required for this, we mean specifically that an attacker account is required which is expressed as a constant. On the table, we use the Sans-Serif font to express a constant. In practice, this is the only case we use a constant for this encoding.

Attack	Vulnerabilities and Enabling Intrusions	Resulting Intrusions	Errors	Failures	Target System	Systems Involved
XSS Execution	Undistinguished Code, XSS staging on an external SP	Malicious Code on Browser	Malicious Code Execution	Integrity *with* browser endpoint trust	Browser	Connection Between Browser and external SP
Forced Redirect	Malicious Code	Malicious Redirect connecting browser and a SP	Browser Endpoint Trust	Integrity *with* Malicious Code Execution	Browser	The SP that the redirect is aimed at
Forced Login CSRF	Automatic Authorisation, Unvalidated Link, Malicious Code on Browser	Login on the user account of the target SP	Unauthorised Action	Integrity	SP inside the FIM system	User login on the IdP account, user SP account
Session Swapping	Lack of Contextual Binding, Malicious Code on Browser	Login on the attacker account of the target SP	Unauthorised Action	Integrity	SP inside the FIM system	attacker SP account `attackerAcc`
XSS Token Theft	Automatic Authorisation, a forced login intrusion onm the browser, a malicious redirect intrusion from browser to SP	A script on the SP that will steal an access token	Resource Disclosure	Confidentiality	SP inside the FIM system	*alternative* Login on the user browser, *alternative* connection from browser to SP, Federation Contract between SP and IdP
Impersonation	Lack of Contextual Binding, access token stealing script on SP	Impersonate a User Account	Account Compromise on overall FIM system	Complete CIA of overall FIM system	User account	Federation Contract between SP and IdP. User account on SP.

Table 3.37 Knowledge Encoding of our interpretation of OAuth as according to Sun and Beznosov (2012). *with* indicates conjunction of a row to another row. *alternative* indicates that the system specified is an *alternative* to an intrusion. Sans-Serif font indicates a constant.

3.10. Summary

We had one main research question RQ1 which is to understand the security landscape of FIM. Part of answering this general question is to understand what kind of vulnerabilities exist, how can they be attacked, how do these attacks affect the user, what protocol suite is affected, is the issue with the protocol or the implementation, and was a solution proposed. We created a taxonomy to further understand this and apply that taxonomy in section 3.6. The properties of a good taxonomy are considered in section 3.2.2 and validated in 3.8.5 and the taxonomy is found to generally be valid.

Another part of the question of understanding the security landscape of FIM is to consider what issues are occurring across protocol suites which is more specifically RQ1.1. It proved challenging to try to compare security issues across works provided by different authors and over different protocol suites and we made concessions to do so (for example generalising vulnerabilities) but we have provided an overview of what we are seeing as problems across FIM in general. This can be visualised by a network graph in 3.6. We also considered solutions to these cross-protocol issues in answer to RQ1.2, which is seen in 3.7.5.

RQ1 is the main contribution of this chapter. However, there is also the question of the golden thread that runs through this thesis. In our survey, we found a particular attack suite that demonstrates how attacks can escalate in FIM from Sun and Beznosov (2012) and we show how these attack can be encoded and they link together in 3.9. Since this attack suite has been shown to be possible and an exact escalating trace has been described we can begin to model escalating attacks in FIM.

Chapter 4. Application of Security Modelling in FIM to Detect Escalating Attacks

Abstract

The aim is to perform a logical analysis on FIM systems to consider how attacks escalate in FIM. Attack escalation is useful because the reachability of some attack state can be determined by considering other attacks rather than just how the system operates. The problem with logical analysis and FIM is that first we need to consider attacks on FIM in a general way. This can be difficult when in practice there are many different researchers using different techniques to examine fundamentally different attacks on FIM. In this chapter, we describe an approach for security modelling for FIM in general. This approach should be able to capture, with enough detail, the intricacies of FIM, but still stay abstract enough that we can use it for many different kinds of FIM systems. This approach can be evaluated against the survey already performed in chapter 3 as this offers a comprehensive overview of attacks in FIM. Specifically, the approach is based on a case study introduced in subsection 3.9. We show how this approach can be extended to consider additional paradigms in FIM security modelling. We also show how original knowledge on attacks in FIM can be encoded by introducing novel attacks on the Facebook FIM system.

4.1. Introduction

The aim is to perform a logical analysis for FIM systems to model attack escalation on FIM. The idea is that the causality of attacks can be modelled by considering what other attacks might be possible. In other words, attacks can be detected by modelling the possibility of other attacks. A framework called Malicious and Accidental Fault Tolerance (MAFTIA, Powell and Stroud (2003)) offers a robust conceptualisation to breakdown issues with computer security in general. Broadly, we refer to this framework as the MAFTIA model. Of particular interest is the concept of an intrusion, which is the effect an attack has on a system. The fundamental adaption we make on the model is the idea that an intrusion could become the cause of another attack, which can be used to model escalating attacks in FIM. Intrusion states could set the stage for further attacks on a FIM system which could be difficult for a human interpreter to keep track of. This is why we use a logical analysis approach as a precise set of conditions can be modelled for an attack to become possible, and these conditions could be produced by the effect of another attack. This chapter proposes a security modelling approach that is suitable for logical analysis, which to the best of our knowledge is the first security application of logical analysis to FIM.

Taking inspiration from the MAFTIA model, the first step into considering a security model compatible with the MAFTIA framework is considering a FIM system as a system of systems. The different kind of concepts outlined in section 2.3 are applied in a systems of systems model, where failures on one system can appear as intrusions on the containing system. To do this, we have to consider how systems in FIM can be a component of FIM. In addition, how do systems interact, as system interaction is a key concept in MAFTIA as the definition of a system in MAFTIA takes inspiration from Laprie (1992) and is: a system is an entity having interacted with, interacting, or able to interact with another system. A key part of the challenge in this work is considering how FIM can be viewed as a systems of systems model by considering FIM systems as components and the interactions that take place between those systems.

MAFTIA is not the only influence of this work, as we considered more general security modelling techniques. Bau and Mitchell (2011) provides a general framework for security modelling. The three core properties of security modelling specified are the system model, the threat model, and security properties. We discuss how our MAFTIA inspired approach conforms to this view as a means of validation.

One problem of security modelling in FIM is determining how the approach can be applicable across many different attack analysis. Some FIM attack analysis might focus on how hosts within a FIM system can be compromised, and as a result of this user accounts stolen, as can be seen in Sun and Beznosov (2012). Other attack analysis might observe oversights in a FIM protocol's design that allows the protocol to be subverted under certain conditions, as can be seen in Pfitzmann and Waidner (2003b). These are clearly two different problems, and we do not want to have a different way of describing systems to suit all of the different problems we face in FIM. The goal is to have an approach which can model different problem domains in a systems of systems approach suited for modelling escalation of attacks.

There are specific technical challenges in modelling FIM systems such as how to best model user accounts. The modelling of a user account for traditional web authentication can be done by modelling the relationship between the user and the set of resources owned by the user on a SP. For a FIM system, we also have to consider that the account can be accessed via authenticating with an IdP connected to those resources, or sometimes not if the SP account is not linked to an IdP yet. We focus on the case study introduced in section 3.9 to demonstrate how some of the core concepts of a FIM system can be modelled. Then, the wider picture is considered with reference to the survey that is the focus of chapter 3. Where the survey indicates that an expansion of the approach is required to consider additional paradigms, we do so in order to have an approach which is in general applicable to security modelling in FIM. We also show how new security knowledge can be modelled, with consideration of account binding similar to what is introduced by Yang et al. (2016) but with reference to implementation issues rather than protocol logic flaws. Firstly we introduce a rich example adapted from Sun and Beznosov (2012) to cover a technical basis for security modelling and then that basis is extrapolated to cover other paradigms in FIM.

In addition, modelling which systems are at fault when the overall FIM system fails can be a challenge. We see in Kormann and Rubin (2000) that a bogus merchant is navigated to by a user. The bogus merchant redirects the user to authenticate with an IdP that is the same in appearance as the IdP the user is accustomed to interacting with. There is clear user error here, as the browser will show that the URL of the IdP is different to the URL the user expects to be on, but in this circumstance, the browser which the user trusts does nothing to correct the problem. However, the problem of a website copying another websites source code and registering a domain name similar to another well know website goes far beyond just the browser as the browser also trusts several other systems. The DNS could be said to be responsible for allowing a URL which is so similar to another well known service with clear intention for fraud to be registered. Alternatively, the certificate authority responsible for issuing a valid certificate to the website could also be judged to be at fault. Considering which systems are at fault through their security requirements is how to solve this problem.

In short, this chapter is about providing a security modelling framework that is applicable to FIM in general. To do this we take inspiration from MAFTIA and the systems of systems approach provided with additional guidance from Bau and Mitchell (2011). An approach is applied to an extensive example on an escalating attack that already exists in the literature. We extrapolate on the approach by considering the wider context of security issues in FIM and an original security problem in FIM. Finally, we consider how novel security knowledge can be modelled to show that the model is adaptable enough to be able to model attacks that have not been discovered yet. The aim is to provide a general approach to security modelling in FIM that is compatible with logical analysis and can be used in modelling escalating attacks.

4.1.1. Contribution to the field

We outline RQ2 as the primary research question in this chapter.

RQ2: An approach for Security Modelling in FIM. How can we apply security modelling techniques based on logical analysis to detect the possibility of attack escalation in FIM?

4.2. Fundamentals

4.2.1. Security Modelling

The survey presented in chapter 3 presents security issues in terms of MAFTIA concepts which are outlined by Powell and Stroud (2003). The idea is to base the security modelling on knowledge that has already been established through the survey. The approach should therefore align with MAFTIA concepts. MAFTIA uses a definition of a system by Laprie (1992). This definition outlines a system as an entity having interacted, interacting, or be able to interact with other entities. It is clear from this definition that how a system interacts with the environment is important. How we model system interactions is key to making a security model that is compatible with MAFTIA.

But why is MAFTIA a good basis for modelling security? To answer this question the key components of what makes a good security model should be explored. Bau and Mitchell (2011) describes three properties in security modelling. The System Model, Threat Model, and Security Properties. MAFTIA helps us with some of these properties.

A threat model as described by Bau and Mitchell (2011) is a clear indication of the attacker's computational resources and system access. Furthermore, threat modelling is the operations the attacker can perform on a system. MAFTIA does not seem to offer much consideration for formalising attacker capabilities in terms of resources. However, MAFTIA does offer a way of breaking down an attack into different parts. This concept introduced by MAFTIA of breaking down attacks is why MAFTIA is of particular interest; to model escalation of attacks, we aspire to break down attacks into smaller parts and establish causality from one part to another part. In a way, this can be viewed as a kind of threat model. The model is less explicit with the attacker's exact system specifications but can still describe what attacks are possible in the presence of certain vulnerabilities. Let us consider some similar approaches that employ a more abstract version of a threat model.

Looking further into the existing literature concerning how attacks are modelled, there is some disparity in how these models are presented. For example, Sheyner et al. (2002) generates attack graphs automatically. This approach does not model the capabilities of the attacker directly, but instead just models what attacks are possible irrespective of an attacker's actual capabilities. Fraile et al. (2016) demonstrates the application of attack trees to an ATM by considering what attacks are possible irrespective of the attacker's explicit capabilities. Sridhar and Govindarasu (2014) considers an attacker model which involves a template of what attacks are possible irrespective of an adversaries explicit capabilities. The point is, threat models are described throughout the literature without explicit description of the attacker's computational resources and system access, as is described by Bau and Mitchell (2011). Even though MAFTIA does not offer a formal way of viewing attacker capabilities, it does not mean it cannot be considered as a way of formalising a threat model since we see valid examples in the literature that also do not conform to explicit attacker system descriptions.

Although MAFTIA does not offer support for explicit attacker capabilities, explicit attacker capabilities are not necessarily incompatible with MAFTIA. MAFTIA considers how vulnerabilities can cause certain attacks to be possible. An explicit attacker capability can be factored in just like a vulnerability can to determine the possibility of attacks. For example, consider a SP where a XSS attack is possible because of a certain parameter being injectable. In addition to the specified vulnerability needing to be present for the attack to be possible, we can also consider if the attacker is likely to have the capability of being able to exploit that vulnerability. In this case, the attacker would need to be able to write computer code, since XSS is about running malicious code generated by the attacker.

Bau and Mitchell (2011) state that a system model must be present. This system model should be a clear representation of the system of interest and the behaviour of that system. MAFTIA is a general approach and not specific to FIM. We are adapting MAFTIA concepts for the modelling of a FIM system. Clearly, this work also needs to specify the system that is being considered. This system model, is one of the main outputs of this chapter. Another key concept that is used by Bau and Mitchell (2011) is behaviour of the system. The behaviour of the system is something we want to avoid with this work. The reason being is that this work is concerned with establishing what attacks are possible using existing knowledge. That existing knowledge has been established by considering the system behaviour already. In describing the system behaviour again in order to come to the same conclusion seems like a cumbersome way of presenting the system model. The idea of this model is to leverage existing knowledge and we can do that without explicitly providing system behaviour.

Bau and Mitchell (2011) also describe security properties being a component for security modelling. This property MAFTIA addresses directly through security rules and goals. MAFTIA uses security goals that correspond to Confidentiality, Integrity, and Availability requirements on a system. Security rules imply those goals, and any number of rules can be violated to cause a certain security goal to be violated. The violation of a security rule relates to an error in the general dependability taxonomy. The violation of a security goal relates to a failure in the general dependability taxonomy. However, since MAFTIA offers a general approach not specific to FIM, we have to consider what the security rules and goals are for the FIM system being considered. The rules/goals for the case study involving OAuth have already been considered in 3.9. Any other errors and rules needed for specific examples will be covered in this chapter.

Bau and Mitchell (2011) offers a good framework for considering what is required for a security model in security. The MAFTIA model does not fit into the the framework exactly, but the key components (system model, threat model, security properties) are there with some deviation of the specifics covered above. MAFTIA is a general framework, which is adapted in this chapter for a logical analysis. In particular, how can a FIM system be represented in a way that is compatible with MAFTIA?

Validity of Security Model Validation: We use the existing work described by Bau and Mitchell (2011) as a basis of validation for this work. The security model we introduce should therefore consist of a system model, a threat model, and security properties.

4.2.2. Logical Analysis

In 2.4 we cover the basics of logical programming. This subsection is more about the motivation and application of logical programming to FIM and what we hope to achieve through logical analysis. We consider what is possible given a logical representation of a FIM system in terms of a logical conclusion. Logical programming offers an approach for discovering what is

possible, given the condition that we can actually represent FIM in a language which can be processed by a logical programming language. This chapter is about representing FIM in a way that can be logically analysed and then applying logical analysis to the FIM system using existing knowledge of attacks.

Logical Analysis Completeness Validation: In chapter 3 an extensive survey is performed on attacks in FIM. Although we do not aim to concretely logically analyse each item in the survey, we aim to have a logical analysis framework in which different security issues in FIM can be described. To that end, we should consider the survey in an attempt to find counterexamples that require further adaption of the logical approach so that different issues can be considered.

Logical Analysis Correctness Validation: We deem the logical analysis as correct if the output of the logical analysis reflects real world observations. For instance, we refer to the work into attacks on OAuth by Sun and Beznosov (2012). The attacks that the logical analysis produce should be reflective of the source material in the literature.

4.3. Related Work

Jøsang et al. (2005) discuss several trust issues in Identity Management. A trust model for FIM is proposed. The FIM trust model outlines several trust requirements of FIM subsystems as viewed from other subsystems. For example, the trust requirements of the SP from the perspective of the user are stated. This approach is useful because it makes clear what each individual entity in a FIM system expects of every other entity. This work is similar to our own because of the view taken on FIM being a number of interacting systems and what each of these systems need to do in order to fulfill specified trust requirements.

The approach taken by Jøsang et al. (2005) would make it difficult to tell when a FIM system has failed overall since the trust requirements are from the perspective of different sub-systems. Since our general view of when a FIM system fails is based on the CIA of the user's data we could base a general FIM failure on the union of the trust requirements of the user for every other sub-system. The problem with this approach is that the trust requirements from a user's perspective are not extensive enough to cover the attacks we have observed in section 3.6. For example, there is no concept of a FIM system's availability, so any attack that leverages a FIM system's centralised infrastructure, such as an attack proposed by Kormann and Rubin (2000), cannot be fully realised using these requirements. Despite the explicit requirements listed by Jøsang et al. (2005) being not sufficient to cover all the vectors that we have observed through our survey, the concept of having requirements for individual systems in FIM is especially relevant to us as we take a systems of systems approach in which we want to describe individual requirements for systems which correspond to failures at those systems.

But the main difference is that Jøsang et al. (2005) details a number of trust requirements that can help inform on what each system within FIM is supposed to be doing, while we are performing an analysis to detect attacks on a model of the FIM system. This raises the question, what is this work doing differently to the other analysis covered at length in section 3.6? We cover a total of 31 papers all performing security analysis on FIM systems. Clearly this work is about security analysis on FIM systems, but what is different between FIM security analysis done by others and our own in FIM security analysis?

The security analysis covered in section 3.6 have an approach that involves investigating systems to establish some possible attack state either from a system model or an implementation of the system itself. What this work presents is a way of modelling the reachability of an attack state from another attack state. Why this approach is powerful is that existing knowledge can be built upon to see what attacks are possible. To the best of our knowledge this is the first logical analysis performed on FIM systems. However, there are strong examples from the literature that adopts a similar approach. Sheyner et al. (2002) produces attack graphs from a model of the network which is model checked with NuSMV. Ou et al. (2006) builds on this idea and generates attack graphs using a logical approach with emphasis on scalability. Inspiration can be taken from this general logical approach. Specifically, the trace of a Prolog execution is provided as input to an algorithm to produce an attack graph.

Let us consider the differences between this work and the work done by Ou et al. (2006):-

- The first main difference is that the work in this thesis focuses on correctness of generating attack causality in a specific problem domain: FIM. In addition, we try to consider how a particular failure state is reachable from other attacks. The work by Ou et al. (2006) focuses on scalability of attack graph generation to a more general problem domain: computer networks in general.
- The second main difference we consider is that Ou et al. (2006) focuses on generating a tree for the attacks using a logical trace and then considering how this tree can be visualised as an attack graph with possible links from the nodes in the tree to other nodes. For now, we just consider the tree produced by the Prolog trace as this is a simpler representation that suits the purpose of presenting causality of attacks in FIM without needing to go into more complex semantics of attack graphs.

4.4. Overall Approach

4.4.1. Inputs

A database containing some kind of relationships between attacks. We have performed a survey which is presented in chapter 3 where attacks are presented in terms of MAFTIA. It therefore makes sense to express the input for a logical analysis in terms of MAFTIA. To follow the MAFTIA model, we need a relationship between: vulnerabilities, attacks, intrusions, errors, and

failures. Of note here, is that an intrusion can also serve as a vulnerability for an attack. This does go outside of the definition provided by Powell and Stroud (2003). However, we realise that this is a necessary step in modelling escalation. Attacks cause intrusions, and so for an attack to be possible because of another attack (showing escalation of attacks), attacks need to sometimes be enabled by other intrusions.

MAFTIA covers the knowledge of what we need in terms of attacker manipulations, but to make the attack applicable to certain system configurations, we also need to express what the conditions are for the attack to be possible. Specifically, what system is the target of the attack and are there any additional system configurations which are required? For example, if we are expressing malicious code being rendered when a user is visiting a malicious site, the target system could be the browser and the vulnerability/enabling intrusion could be the malicious site. What is also required is a connection from the user to the malicious site which models user interaction on the malicious site. These system conditions are distinct from vulnerabilities because a connection should not be viewed as a vulnerability. Thus, each entry in the database corresponds to how the attack can be described in terms of MAFTIA, and the system configurations necessary to facilitate the attack.

The second input is a specific system configuration representing FIM. Some attacks are only effectual given some kind of system configuration. For example, if for a specific attack we are relying on a connection to some external website. A FIM system can be seen as stakeholders (user's browser, SP, IdP) and connections between these stakeholders (connection between the user and SP, connection between SP and IdP, etc.). There are also other aspects of a FIM configuration to take into account, such as user accounts and whether or not those accounts are logged in. How can this system configuration represent whether or not the system is vulnerable to a certain attack? It is within this second input that we also specify primitive vulnerabilities that exist within the system. These vulnerabilities combined with the system configuration model the FIM system at its base state before any attacker manipulations are performed on it.

An auxiliary input to this work is a syntax that represents the above two inputs that is compatible for logical analysis. Attack knowledge needs to be represented in a way that is syntactically correct. The FIM system configuration and associated vulnerabilities needs to be represented in a way that is syntactically correct. Part of this work is therefore transforming attack knowledge and FIM system configurations/vulnerabilities into Prolog syntax, since that is the logical programming language we will be using. To be clear, syntactically correct in this context means that it conforms to some language specification that is created for the purpose of logical analysis of FIM. Just because something is valid Prolog code, does not necessarily mean that it is regarded as syntactically correct for the logical analysis of FIM systems. Referring to section 2.4, we would represent the system configurations as terms and attack knowledge expressed in MAFTIA (vulnerability, attack, etc) as rules operating over those terms. Then for a logical conclusion A (the next step; a more advanced stage in an attack) is met with a formula (a term or number of terms connected via logical operands) B, which is written:-

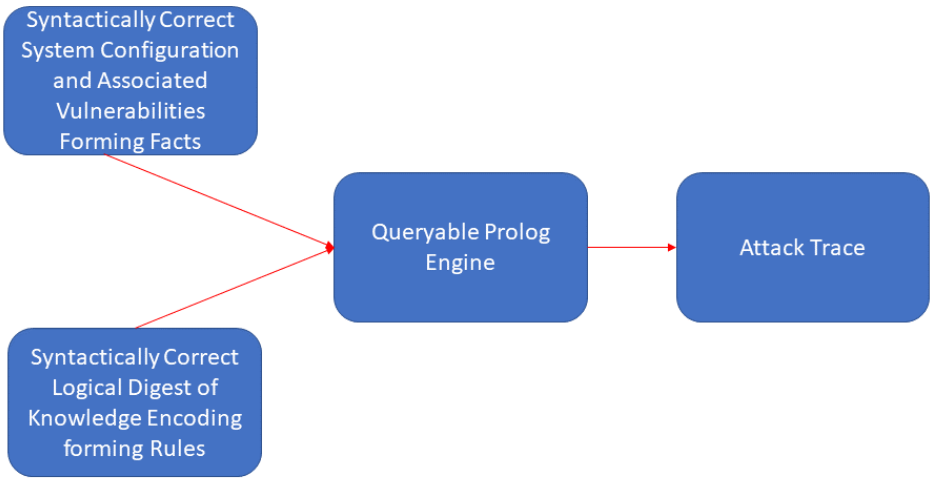

Figure 4.1 Workflow for Establishing An Attack Trace on a FIM system

$$A \leftarrow B$$

4.4.2. Outputs

The above inputs represent: a syntactically correct expression of attack knowledge inline with MAFTIA concepts; a syntactically correct representation of a FIM system with associated vulnerabilities. With that, we can use Prolog to query what attacks are possible. Thus, the first output is an engine which can be used to see which attacks are possible on a given FIM system structure. The next output requires interaction with that engine in order to produce an attack trace (specified in terms of MAFTIA) of the entire attack. In fig. 4.1 this workflow is specified.

The idea is that given these attack traces, attack escalation can be modelled as the trace breaks down the attack into stages. These stages can be represented with existing concepts provided by MAFTIA. The difference is that in this work, an intrusion can be used to cause another attack. When an attack is possible because of some intrusion, we say that attack escalation has occurred. Attack traces are a major output in modelling attack escalation in FIM.

4.5. Security Modelling Approach Applied to OAuth Case Study

In this section we describe the signature of a formal language used to describe FIM systems. That is, we consider a finite possible set of non-logical symbols that represent relations on a FIM system in terms of the FIM structure and attacker manipulations that can occur on the FIM structure. We call the aspect of the FIM structure the *System Language* and the aspect of the

attacker manipulations the *Attack Language* which overall make up the System Modelling Language. Individual terms on the system language are expressed as *System Terms* and individual terms on the attack language are specified as *Attack Terms*.

4.5.1. Systems of Systems Approach

We model the FIM system as a system of systems. The main benefit of this approach is that we can model FIM security failures in a chain rather than some byzantine monolithic failure. Using a systems of systems approach allows us to pinpoint which systems are the cause of further faults, and model that in terms of MAFTIA. To model FIM in this way, we must consider how systems can fit into a FIM system as a component or another system entirely that interacts with the FIM system. To be consistent with the MAFTIA model outlined in Powell and Stroud (2003), we also need to consider how these systems fail in terms of security rules (corresponding to errors) and security goals (corresponding to failures).

Holistic System: To account for the various kind of things that can go wrong with the FIM system, we adopt a holistic approach. This allows for the inclusion of expert knowledge in the modelling of certain attacks. The attack suite specified by Sun and Beznosov (2012) is based on attacks like XSS and CSRF which require the browser to fail in not recognising malicious code. Sun and Beznosov do not focus on the browser failing, and instead focus on how the SP and IdP fail. Thereby, focusing possible fixes to the SP and IdP. In section 3.9 we introduce the idea of also considering how the browser can fail. The benefit of this holistic approach is being able to consider additional information in an overall system failure.

There can be multiple instances of the same type of systems but not multiple instances of the overall modelling environment. The instances of systems are distinguished by an identifier which is unique to every system. Systems can also be different by the vulnerabilities which exist on that system. At the top-level of this system of systems is the overall FIM system which can contain multiple federations existing between SPs and IdPs. In general, we will be considering just one FIM system throughout this thesis.

Example. *Consider a particular SP called Service which is vulnerable to XSS attacks. Another SP, can be called Different and is vulnerable to CSRF attacks. Both of these SPs can be modelled in the same environment with the associated XSS and CSRF related vulnerabilities.*

We also consider which systems a system might interact with in some way. Since the definition for a *system* according to Laprie (1992) is just an entity having interacted, interacting or being able to interact with other entities, we aim to see what systems are associated (or interact) to other systems. This interaction is key to understanding causality of attacks in FIM systems because we want to associate systems which fail to other systems that the failure could affect. For example, a failure in an SP through an XSS attack could lead to problems in a Browser from the interpretation of malicious code but only those Browsers which are actually associated to

that SP in some way. We can associate causality of attacks through understanding what systems interact with other systems.

Example. *Consider a vulnerability at a certain SP, called Service, that somehow allows for malicious code to be injected and user accounts to be stolen, only accounts interacting with the SP Service should be impacted by an attack on Service. Consider other user accounts on a SP called Different, then these user accounts should be unaffected by the vulnerability on Service because these user accounts are not interacting with Different. Interactions are particularly useful when we consider that there can be multiple instances of the same type of system because a system can be linked to the particular system it interacts with.*

A system which is a component of another system always interacts with that system by default, and is considered within the first system's *system boundary* (2.3). The key thing which makes an interaction different from a component is that a system can have many interacting systems but at most one system it is a component of. If we consider failures on one system, one thing we want to model is that systems components can also fail which leads to the overall failure in a systems of systems approach. Although some systems are specified to interact with other systems, the system may not always have such a system to interact with yet. For example, an account interacts with a SP and an IdP. However, it could be the case that the account has not been bound to an IdP yet.

Example. *Consider an instance where an account is unbound in a FIM federation. One attack on this account could actually be to link that an account to an attacker's account, so it is important to be able to model interactions that do not always need to be filled.*

Finally, on the subject of system interaction, it can be the case that a system in FIM interacts with another system but it is not always the case that the interaction has to be modelled in the system itself. Consider a SP called service. This SP might interact with many different users (e.g. Alice, Bob) through a connection in a one-to-many relationship. If we wanted to model this from the SPs point of view, we would need to model a list of all the connections attached to the SP. We could do this, but this unnecessarily complicates the model. We can simply model this interaction from the point of view of the connection. Since the relationship between a connection and a specific SP is one-to-one, this captures the information we need in a simpler way.

System Interaction: A key aspect of the language is that the interactions between systems must be modelled. These interactions are what cause escalation of attacks to be possible.

In a systems of systems approach, one can drill down a particular system down into many components and the components of those components almost indefinitely. We introduce a set level of abstraction for FIM's interactions and components. This is necessary to allow for a generic framework that can be applicable to many different attacks on FIM.

Example. *Consider a possible malicious IdP as is introduced by Urueña et al. (2014). Such a malicious IdP could keep track of what websites are visited leading to a potential compromise of confidentiality. We could think about what components within the IdP might be responsible for compromise. Perhaps there is a file somewhere with all the websites visited by users. We could model the file that exists in this IdP as a component of the IdP. However, doing so would add unnecessary detail to the overall model. The important thing to keep in mind is that we are modelling causality and what we want to establish is that the IdP can cause the overall failure, not what the specific inner-workings are of that IdP. In addition, Urueña et al. does not go into detail about how the IdP monitors users, just that it could. Where the literature generally avoids unnecessary detail, we do the same.*

Minimum Systems Necessary: We try to describe systems with the highest level of abstraction as possible. This reduces the complexity of the language we are trying to describe because there are less terms. In addition, the details that do not matter in an attack can be avoided. For example, we could model an account as resources at a SP, some identity linkage at an IdP, and some credentials for the user. Considering an attack described by Sun and Beznosov (2012), such as the XSS token theft, all of these aspects of an account are important. We need to know the IdP responsible for the account because of the automatic authorisation vulnerability. We need to know which SP is involved because that is where the XSS attack is staged from. We need to know which user the account belongs to because ultimately that is the user the attack will affect in terms of CIA. Therefore, one possible way we could model the account as a system located at each of these three systems. The alternative to this is to create an account system as a component of the FIM system, and have have some kind of link from the account system to each of these three systems. We prefer the latter option because then we can consider less systems.

One thing that we avoid in this holistic approach is the evaluation of secure behaviour by the user. We are investigating the security of FIM systems, and not necessarily the usage of those systems. This kind of view is consistent with at least some computer security literature. Notably, Adams and Sasse (1999) raises the argument that users are not the enemy. We should not necessarily blame users for the problems that exist in computer security. Why this is an important point to note is because if we are to remain consistent with this argument, we should seek to find fault with the FIM system and not the user of that system. Ultimately, this is to consider faults at the technical level rather than the user level.

The User is Not the Enemy: We try not to blame the user when examining failures of systems, and adopt a similar philosophy to Adams and Sasse (1999). We believe it is important that systems are as secure as possible despite how they are used. For example, Kormann and Rubin (2000), describes a bogus merchant attack. The user is at error here for not properly checking certificates/URLs. However, the origin of the attack could be attributed to the Browser for allowing the user to browse to a website that is obviously impersonating another site without warning.

Security Properties: Bau and Mitchell (2011) puts forward the notion that security properties are important to security modelling. MAFTIA has the notions of security rules and security goals which correspond to errors and failures. In order for the approach we adopt to be compatible with these two views, the security properties need to be considered. These properties have already been briefly considered in section 3.9 to encode the errors and failures but in this chapter the properties are referred to more formally in the context of the system the properties belong to in order for a logical programming approach to be adopted. The violation of security properties on systems leads to errors and failures. In the presence of certain intrusions, a specific security rule can be violated corresponding to a certain error. A single error or a number of errors can correspond to a failure.

4.5.2. High-Level Overview of Systems

We offer a high-level overview of considered systems for the model with reference to the systems needed to describe the attacks introduced by Sun and Beznosov (2012) and encoded in section 3.9. Let us consider the systems involved in the case study introduced in section 3.9 as a starting point. What is important to represent these systems logically? For further background on systems of systems approaches we highlight background section 2.3.

- **Component:** For a systems of systems approach, systems can be *component* of another system according to Laprie (1992). A component is a system within the system boundary of another system. When a system is not a component of another system, we refer to that system as a *top-level* system. A top-level system is the highest level of abstraction being considered. In reality, this top-level system could be viewed as a component of a larger system. However, this larger system would be considered outside the scope of analysis.

- **One-to-One Interactions:** The other systems that a particular system interacts with. Specifically we consider one-to-one interactions. These one-to-one interactions are *non-symmetric*. This is because otherwise some systems would need to contain information in a list which would complicate the modelling process. For instance, a SP could have many connections connecting to the SP. The logical representation of the SP would then need to keep a list of all of these different connections. A more simple approach is to just specify the SP connection for the connection itself and omit this information for the SP. Storing the information in a symmetric way would complicate the model unnecessarily. This is also not a limitation, because the Prolog logic engine can always find every system that interacts with another system even if the relationship is non-symmetric. For example, we can find every connection belonging to a particular SP by finding each connection term and checking to see if the connection term has a reference to that SP. The nature of Prolog allows us to trivially find all solutions exhaustively, which is in this case every connection which belongs to that particular SP. The only thing that we have to assume is that each non-symmetric interaction is properly initialised and maintained, because if that is incorrect, there is no redundancy to recover

the situation. Lastly, some systems do not have one-to-one interactions because for those systems, every system that does interact with them is not on one-to-one basis but is actually one-to-many. For instance, a SP has many connections, which are stored as terms for that particular connection. As per the previous point about storing these interactions as a list being unnecessary, systems with only one-to-many interactions can have no one-to-one interactions.

- **Security Properties:** Security properties in MAFTIA are considered in terms of security goals and security rules. Security goals are in terms of CIA. These goals link to security rules in that the security rules imply the security goals. For some high-level system being considered we consider the rules and goals with respect to the attack suite specified by Sun and Beznosov (2012). We infer the relevant security properties for these high-level systems from the attack suite. The security properties are not representative of every relevant security rule and goal that could be considered for that system. Furthermore, not every high-level system has security properties because no failure for a particular system was interpreted from Sun and Beznosov. Sun and Beznosov mostly focused on failures from the browser and SP perspective, and our model reflects that.

Federated Identity Management: One of the top-level systems that we consider is the FIM system itself. There are systems which can exist outside the FIM system, for example an external SP that may be considered outside of the FIM system. This is a possibility in the attack suite demonstrated by Sun and Beznosov (2012) and encoded in this thesis in section 3.9. Specifically, an XSS attack can be staged from an SP external to the FIM system. It is important to model this because we can deduce what aspects of an attack occur internally and what aspects of an attack occur externally.

The literature regarding attacks on FIM focus on how accounts can be compromised by attackers which undermines the CIA of the user. In chapter 3 the details of how an account compromise can result in CIA failures are described. From this information, we consider a security rule stating that accounts should not fall into the hands of the attacker, and if this happens then the CIA security goals can be undermined and result in a failure of the overall FIM system.

- *Component Of:* Top-Level.
- *One-to-One Interactions:* No one-to-one interactions.
- *Components:* SP, IdP, Active Connection, Federation Contract, Authentication Connection.
- *Security Properties:* Account Compromise $\rightarrow$ CIA.

Browser: The Browser system captures the nature of the agent the user wields to browse the World Wide Web. The browser interacts with the user. To be consistent with our philosophy to try and find fault in the design of a FIM system rather than the user as is introduced by Adams

and Sasse (1999), we attempt to shift the blame of attacks from the user to the browser wherever possible.

We consider the browser as part of the FIM system. This would mean that faults occurring on the browser are internal to the FIM system and not some external threat. This is perhaps an unfair responsibility to place on FIM system designers, but faults in the browser can be dealt with by FIM stakeholders. For example, Kormann and Rubin (2000) point out that user's browser can be tricked into a bogus merchant attack. This particular attack could be avoided by FIM stakeholders if they considered who owns similar domain names, or perhaps implement a more involved login procedure so that users are less likely to auto-pilot login to an IdP. Overall the approach of considering the browser as part of the FIM system is a more holistic approach.

There are two situations in which the browser can fail which we consider. Malicious code must not be executed on the browser, as this is the cause for many XSS type attacks. Such a rule being broken could lead to an integrity failure of the browser, where certain actions occur that are outside of the user's interests. The second rule is that the browser should not erroneously navigate to a website not specified by the user. This can lead to another integrity failure.

- *Component Of:* FIM.
- *One-to-One Interactions:* User.
- *Components:* Login.
- *Security Properties:* Malicious Code Execution → Integrity, Endpoint Trust → Integrity.

User: We consider the user a top-level system. We adopt a similar philosophy to Adams and Sasse (1999), as explained in sub-section 4.5.1, where the user is not the enemy.

- *Component Of:* Top-Level.
- *One-to-One Interactions:* No one-to-one interactions.
- *Components:* No components.
- *Security Properties:* None.

Service Provider: The SP in a FIM system. Alternatively, a SP could just be a regular website. There are cases in which a regular website can facilitate an attack in some way. For example, Sun and Beznosov (2012) state how a SP not involved in a federation can serve as an intrusion for malicious scripts inserted via XSS vulnerabilities. For this reason, an SP can be considered outside of a FIM system.

A rule for the SP is that no action should take place on the SP that is not prompted by the legitimate user. Such an action is said to be an unauthorised action and can lead to an integrity failure. A second rule is that no resource belonging to the user should be disclosed, when this occurs resource disclosure is said to have occurred and can cause a confidentiality failure.

- *Component Of:* FIM or Top-Level.
- *One-to-One Interactions:* No one-to-one interactions.
- *Components:* No components.
- *Security Properties:* Unauthorised Action → Integrity, Resource Disclosure → Confidentiality.

Identity Provider: The IdP in a FIM system. IdPs vary in their function. Some IdPs are entirely devoted to authentication on behalf of others, such a shibboleth. Other IdPs can also act as SPs, with some performing authentication on behalf of others whilst also offering a service of their own, such as Facebook.

For the example involving Sun and Beznosov (2012), no failures are said to occur on the IdP so no security properties are specified. In section 4.7 we discuss adding security properties to the IdP.

- *Component Of:* FIM.
- *One-to-One Interactions:* No one-to-one interactions.
- *Components:* No components.
- *Security Properties:* None.

Account: The account resides in the FIM system and represents a relationship between the user, the SP in which accounts resources are located on, and the IdP the user authenticates to. An account in FIM is not as simple to model as accounts in more traditional authentication approaches. A user knows some credentials/key to access the account, a SP holds resources, and an IdP, who the SP trusts, vouches on behalf of the user that the user is who they say they are.

We could model an account as a number of different subsystems. For example, a resource at an SP could be a component, and the identity linker at the IdP could be another component. Perhaps the credentials for the user could even be a component under the user. Rather than splitting an account into many different subsystems, we decided to make an account to be a component contained in the main FIM system that represents a relationship between these different entities and opted for the simpler approach as is consistent with our general philosophy outlined in subsection 4.5.1.

- *Component Of:* FIM.
- *One-to-One Interactions:* User, SP, IdP.
- *Components:* No components.
- *Security Properties:* None.

Login: The login system represents a relationship between the user, and the account. If the login system exists, then that means the user is logged into that account. If there is no associated login system, then the account is not logged on. We say that the login system is a component of the browser. This is because the necessary login token is stored on the browser and this is susceptible to attacks. In particular, XSS attacks target login tokens at the browser. SPs, are involved with the login system also, and we could model this by having another login at the SP that links to the browser login. We chose the simpler option, as according to our philosophy outlined in subsection 4.5.1.

- *Component Of:* Browser.
- *One-to-One Interactions:* Account, SP.
- *Components:* No contained systems.
- *Security Properties:* None.

Active Connection: Connections capture the situation in which two hosts are connected in some way. The nature of these connections can be different. A user interacting with a website is different to the type of connection that might occur between an SP and an IdP in federation. Here we describe the type of connection between a user and a SP, but we will go on to describe the remaining types of connections.

An active connection captures a broad interaction between a browser and a SP. This is equivalent to having a tab open in a browser. The point of this is to state that the user is currently browsing a SP and so is susceptible to possible vulnerabilities existing on that SP. A SP could have some kind of vulnerability, like susceptibility to XSS, but that does not matter unless this connection is in place.

Connections in general are a component of the FIM system itself. The overall FIM system is the orchestrator of most data that is sent in FIM. Browser's, SPs, and IdPs could also have a claim for aspects of connections being a component. We simplify this by just saying that connections are components of FIM.

- *Component Of:* FIM.
- *One-to-One Interactions:* Browser, SP.
- *Components:* No contained systems.
- *Security Properties:* None.

Authentication Connection: An Authentication Connection captures the messages exchanged by a user and an IdP when the user authenticates themselves to the IdP.

- *Component Of:* FIM.

- *One-to-One Interactions:* Browser, IdP.
- *Components:* No contained systems.
- *Security Properties:* None.

Federation Contract: This connection exists between a SP and an IdP and models that an IdP and a SP have agreed to operate a FIM protocol to authenticate a user. If there are any exchanges of messages as part of a FIM protocol, the messages are sent over this connection.

- *Component Of:* FIM.
- *One-to-One Interactions:* SP, IdP.
- *Components:* No contained systems.
- *Security Properties:* None.

System Name	Component Of	One-to-One Interactions	Components	Security Properties
FIM	None	None	SP, IdP, Active/Federation/Auth Connection	CIA
User	None	None	None	None
Browser	FIM	User	None	Integrity
SP	FIM or None	None	None	Confidentiality, Integrity
IdP	FIM	None	None	None
Account	FIM	User,SP,IdP	None	None
Login	Browser	Account, SP	None	None
Active Connection	FIM	Browser, SP	None	None
Authentication Connection	FIM	Browser, IdP	None	None
Federation Contract	FIM	SP, IdP	None	None

Table 4.1 High-Level Overview of Systems

4.5.3. Process of System Language Creation

In subsection 4.5.2, we introduce a high-level overview of the systems we consider including the important aspects of modelling that particular system. This can be summarised in tab. 4.1. In this section we show how we can represent those systems as a term. We represent systems in such a way that the systems can be interpreted by Prolog. The necessary background including valid grammar for Prolog tools is provided in section 2.4. Ultimately, we aim for systems to be interpreted by Prolog so that the extent of which escalating attacks can happen is evaluated. We attach certain non-logical meaning to these terms. We generally refer to individual terms within the systems language as *System Terms*. There are several things that are important to include directly in the logical term representing a system:-

- **Component:** If the system is a component of another system. For instance, a login is a component of a browser. There is no one way in which this modelling process has to be

done, but we outline our approach in 4.5.2, which acts as a reference point for logically representing a system. For instance, an account could be a component of a SP or an IdP. Ultimately, for simplicity we model an account as a component of the FIM system which represents an abstraction of the account interacting with other systems in the FIM system such as the SP and IdP. Some systems could even only sometimes be a component of the FIM system, such as an SP which can be external or internal to the FIM system. In this case the empty keyword is used to designate a SP as an external SP not being a component of the FIM system. Each particular instance of a system can only be a component of at most one system.

- **Interactions:** What other systems are interacted with. In particular, if the system could interact with another system in a one-to-one relationship. If the system interacts with another system on a one-to-many basis then leave it to that other system to include the interaction. This approach means we can avoid using lists in descriptions of systems and also eliminate redundancy in describing information. For example, a SP could have many connections interacting with the SP which could be stored in a list. But a more simple approach is to simply have the connection contain the SP that is being interacted with.

The first thing to consider when designing this system is the type of system that we are dealing with. This is the key component in each of the systems we describe. To represent this, we say that the type of system structure we are representing forms a *functor* of everything to do with the system structure. So if we have a system x, and a tuple of system properties $(y_1, ..., y_n)$ we can represent the system:-

$$x(y_1, ..., y_n)$$

The next question is, how do we consider the system properties of a system in a consistent way? Since we can have multiple instances of the same type of system, the first essential thing to consider is an identifier. We always position the identifier first in a tuple of system properties associated with a system. The identifier must always be unique, there can be no duplicate identifiers for system terms.

If the system is a component of another system, we need to consider that. We need to clearly link a system to the other system that the system is a component of. Although a system can be a component of systems of different types, a particular instance of a system can only be a component of one other instantiated system.

Finally, we consider the one-to-one interactions for a system. Although a type of system can have more than one interaction, the number of interactions for a particular system is always the same. This keeps the definitions of a type of system the same and allows for those systems to be in rules without ambiguity to the arity of the system. There are cases where an interaction is not

filled, such as when an account does not have an IdP linkage yet. We reserve the keyword *empty* for these cases.

The top-level FIM system is modelled because we need to distinguish between systems which are considered inside and outside the FIM system. In practice, we only ever consider one FIM system. We do also have examples of systems that can exist either inside or outside the FIM system. For instance, an SP can either be internal or external to a FIM system.

Consider a set of high-level systems described in 4.1 X. For a system $x \in X$ the properties Y can be expressed with a tuple addition of identifier (i), the system x is a component of (c), and interactions $(a_1, ..., a_n)$. The properties for a system are then expressed:-

$$Y = \begin{cases} (i), & \text{if the system is a top-level system} \\ (i,c), & \text{if the system could be a component of another system} \\ & \text{and has no possible interactions} \\ (i,c,a_1,...,a_n), & \text{otherwise} \end{cases} \tag{4.1}$$

The system x is then ultimately expressed: $x(Y)$ and we apply this process to each system in X.

There is room for various semantic errors that are syntactically correct. In particular usage of the keywords, such as *empty*. The keywords *empty* could be used anywhere. We do not deal with the possibility of using the keywords incorrectly, and stick to what we consider as correct use of the keywords. Specifically, using *empty* to indicate that a system that could exist in a FIM system but does not (a possible external SP), and using *empty* to specify that sometimes the specified system interacts with another systems, but in this case does not (such as an account which is not linked to an IdP). Applying the general methodology for creating a language to model systems to the Sun and Beznosov (2012) case study yields a number of systems associated with FIM described in syntax compatible with logical analysis. We apply that methodology to the list of systems presented in 4.1 to generate the language in tab. 4.2. The system terms represent what is necessary to represent the attack suite introduced by Sun and Beznosov and encoded in this thesis in section 3.9

4.5.4. Process of Attack Language Creation

We move onto how we model faults, erroneous states, and failures in FIM. The background for this section is described in 2.3. We generally refer to the individual terms describing faults, erroneous states, and failures as *attack terms*. In general, the faults we consider (vulnerabilities, attacks, intrusions) link to the literature as shown in section 3.9. The erroneous states link to security properties which are mainly outlined in subsection 4.5.2, but also briefly outlined in 3.9 to encode the knowledge of the attack as to encode the attacks security rules/goals are a dependency. Attack terms are concerned with three things:-

Functor	Arity	Example	Semantic Summary
fim	1	fim(fbConnect)	The overall FIM system.
user	1	user(alice)	A user.
browser	3	browser(aliceFirefox, fbConnect, alice)	A browser, linking to the FIM system the browser is contained in and the user that the browser is linked to.
sp	2	sp(instagram, empty)	A Service Provider. We want to say if a SP is internal or external to the FIM system we consider, in order to establish if the FIM system is at least partially responsible for the SP. For this circumstance the *empty* keyword is used.
idp	2	idp(facebook, fbConnect)	An Identity Provider contained by the FIM system.
account	5	account(aliceAccount, fbConnect, alice, instagram, empty)	An account. The containing FIM system is specified. The account links to the user who owns it, the SP the account acts on, and the IdP which authenticates the account. The IdP is *empty* here, which represents an account which is not currently linked to an IdP.
login	3	login(instagramLogin, aliceFirefox, aliceAccount,)	A login, linking to the browser of which this login is a component, and the account the login is acting on.
actCon	4	actCon(active1, fbConnect, aliceFirefox, instagram)	An active connection contained in the FIM system. Links to the browser and the SP.
autCon	4	autCon(auth1, fbConnect, aliceFirefox, facebook)	An authentication connection contained in the FIM system. Links to the browser and the IdP.
fedCon	4	fedCon(fed1, fbConnect, instagram, facebook)	A federation contract contained in the FIM system. Links to the SP and the IdP.

Table 4.2 List of System Language Terms

- **Attack Term Type:** The type of the attack term, such as a vulnerability or an intrusion. The type of the attack term is represented in logic by the functor. This allows for the different types of an attack term to be easily distinguished for analysis.
- **Attack Value:** What the attack term actually represents, such as XSS vulnerability or a CSRF vulnerability.
- **Location:** We consider the location of which the attack term occurs. This location is expressed as a system. As part of our systems of systems approach, attack terms should link to locations in a system. For a location to be valid, it also must be able to be expressed using the system language in 4.2. Typically, attack terms link to systems existing in the system model but sometimes systems can actually be created through an intrusion. A vulnerability which causes an XSS attack might exist on a SP, but the code injected through the XSS might initiate a malicious redirect, which could be represented as a connection. The extent of which attacks impact individual systems is observed through this approach as we can refer to the location the attack occurs on.

For an overview of attack terms see tab. 4.3. An attack term type is represented as x where $x \in \{vulnerability, attack, intrusion, error, failure\}$. The attack value a is a member of the universe of all attack values $a \in A$. Finally, all attacks occur at a location represented as a term l

where l refers to systems considered (e.g. 4.5.3) and belongs to the universe of all possible system terms L. Attack terms are then ultimately expressed:-

$$x(a,l)$$

Functor	Arity	Example	Semantic Summary
vulnerability	2	vulnerability(CSRFVulnerability, sp(instagram))	Links a vulnerability to a system.
attack	2	attack(CSRFAttack, login(instagramLogin, aliceFirefox, instagram))	Links an attack to a system.
intrusion	2	intrusion(maliciousLogin, login(instagramLogin, aliceFirefox, instagram))	Links an intrusion to a system.
error	2	error(outsideInfluence, sp(instagram))	Links an error to a system.
failure	2	failure(integrity, sp(instagram))	Links a failure to a system.

Table 4.3 List of Attack Language Terms

4.5.5. Simplified Syntax

For illustration purposes, we often use a simplified syntax for terms. Typically, we express system terms $x(Y)$ where x is the system and Y is the set of properties of that system described in tab. 4.2. For the simplified syntax, Y consists of only the unique identifier of that system. For example, the system term $idp(identity, oauth)$ becomes $idp(identity)$ for the simplified syntax. This syntax is not compatible with the approach in general, and is simply used for conciseness in figures. Every figure that utilises this simplified syntax will reference this subsection for clarity.

4.5.6. Rules

Note that the terminology used in MAFTIA describing the violation of a rule causing an error and the use of a logical rule conflicts. Where there might be confusion, we use the terminology *security rule* to describe a rule in a MAFTIA context and a *logical rule* to describe a rule that is used for logical analysis.

A rule advances one stage of a security issue to the next. Rules enable modelling of attacks by coming to a logical conclusion on the possibility of an overall security issue. MAFTIA concepts and terminology are used and allow for the breaking down of an attack into logical stages that progresses through the use of multiple rules. These rules can also take information from the system model in order to form a conclusion. In section 3.9 a case study is introduced that produces a knowledge encoding which can be seen on tab. 4.4 (tab. repeated from chapter 2.3). This subsection describes the process of going from a knowledge encoding to rules which are syntactically correct from the point of view of the language we describe and also can be interpreted by Prolog to realise logical conclusions of possible attack states.

Attack	Vulnerabilities and Enabling Intrusions	Resulting Intrusions	Errors	Failures	Target System	Systems Involved
XSS Execution	Undistinguished Code, XSS staging on an external SP	Malicious Code on Browser	Malicious Code Execution	Integrity *with* browser endpoint trust	Browser	Connection Between Browser and external SP
Forced Redirect	Malicious Code	Malicious Redirect connecting browser and a SP	Browser Endpoint Trust	Integrity *with* Malicious Code Execution	Browser	The SP that the redirect is aimed at
Forced Login CSRF	Automatic Authorisation, Unvalidated Link, Malicious Code on Browser	Login on the user account of the target SP	Unauthorised Action	Integrity	SP inside the FIM system	User login on the IdP account, user SP account
Session Swapping	Lack of Contextual Binding, Malicious Code on Browser	Login on the attacker account of the target SP	Unauthorised Action	Integrity	SP inside the FIM system	attacker SP account `attackerAcc`
XSS Token Theft	Automatic Authorisation, a forced login intrusion onm the browser, a malicious redirect intrusion from browser to SP	A script on the SP that will steal an access token	Resource Disclosure	Confidentiality	SP inside the FIM system	*alternative* Login on the user browser, *alternative* connection from browser to SP, Federation Contract between SP and IdP
Impersonation	Lack of Contextual Binding, access token stealing script on SP	Impersonate a User Account	Account Compromise on overall FIM system	Complete CIA of overall FIM system	User account	Federation Contract between SP and IdP. User account on SP.

Table 4.4 Knowledge Encoding of our interpretation of OAuth as according to Sun and Beznosov (2012). *with* indicates conjunction of a row to another row. *alternative* indicates that the system specified is an *alternative* to an intrusion. Sans-Serif font indicates a constant.

An entire row in the tab. 4.4 corresponds to a chain of states and events that relates to the MAFTIA fault model conceptualised by Powell and Stroud (2003). Columns in the table mostly refer to different stages of those attacks in terms of MAFTIA (e.g. vulnerability, attack). Attack terms must also have a location in which to act on, which is given by the target system in the table. In some cases, the location of an attack term might not be the target of the attack. For example, if the attack creates a system through an intrusion, say an attacker caused login, then this location must be referred to in the entry directly. In these cases, the encoding should make it clear that the location of the attack is different to the target of the attack.

Not every chain necessarily needs to have all values for each part of the chain. For instance, vulnerability, attacks, and intrusions could occur without a corresponding error and failure. The intrusion could still be part of a condition for an attack for a different chain. Although, the chain cannot have any breaks. A vulnerability cannot result in an intrusion without a corresponding attack. Conceptually, the start of a chain could also not include values to encode how a certain intrusion caused a particular error and failure without the need to encode the specific vulnerabilities that cause the intrusion. However, for this case study all chains are complete with all stages of the MAFTIA fault model containing values as this represents the case study that we attempt to capture. As a side effect, having each value filled in does make for a simpler case study.

Any other systems which are required for the attack to be possible are listed in the *systems involved* column. The systems involved could be used in any step in the process, but in practice we use the systems involved to establish the possibility of an attack in combination with a single, or combination of vulnerabilities/intrusions.

Security issues can be influenced by other security issues by encoding an intrusion as the cause for an attack. Furthermore, security issues can cause other security issues through the intrusion that can be produced. Therefore, we can show escalating attacks in FIM by modelling an attack being logically possible from some intrusion caused by some other security issue. The entire process of modelling a security issue can be seen in fig. 4.2.

4.5.7. Genesis of Rules from Knowledge Encoding

How do we go from a knowledge encoding like is presented in tab. 3.37 to logical rules? For each chain, four rules are considered. There needs to be a rule that goes from a vulnerability to an attack, attack to intrusion, intrusion to error, and error to failure. This subsection is about how exactly information from an encoding is interpreted in the formation of logical rules. The background about logical rules is covered in section 2.4.

The encoding itself is provided in a natural language format. The advantage of this approach is that the encoding can be more easily understood and more easy to encode. But one problem is interpreting this encoding is a way to interpret the encoding in a logical format systematically. Part of this subsection is therefore about interpreting the encoding systematically. When creating

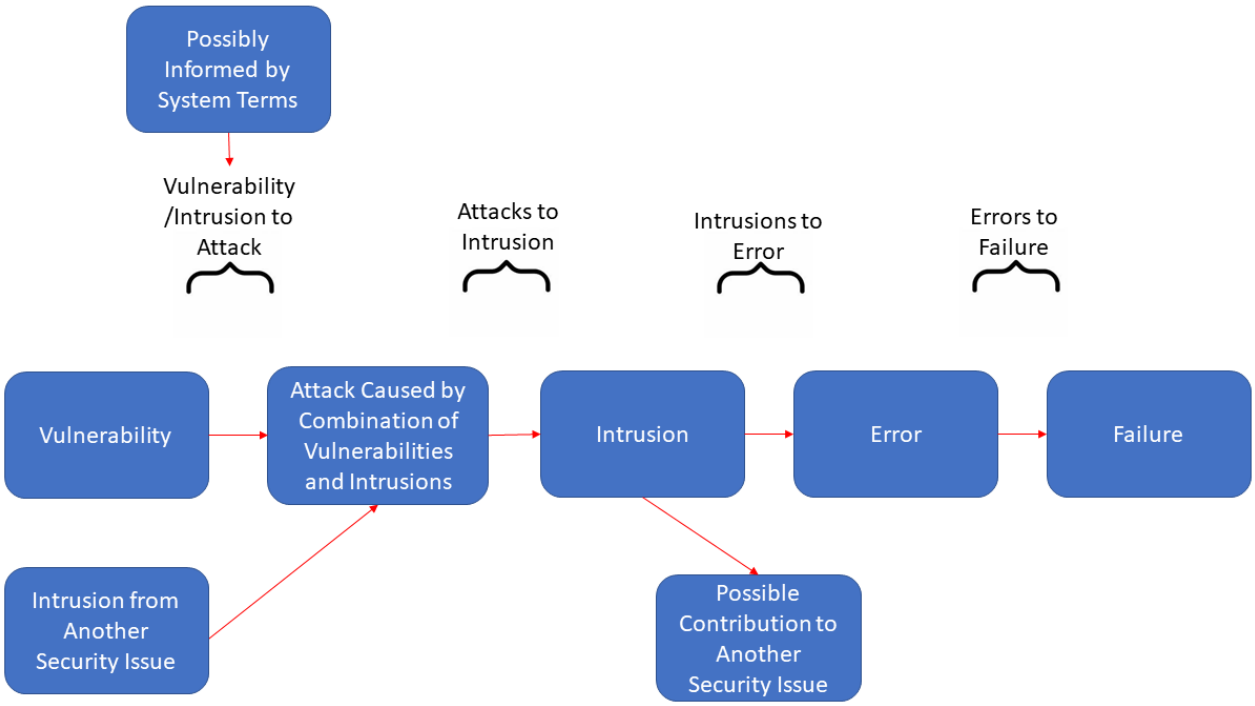

Figure 4.2 Security Issue that can contribute to and be made possible by other Security Issues

the rules from the encoding the following principles need to be followed and applied to arrive at logical rules:-

- **Understand the Goal and Condition:** Every rule has a condition and a goal. We represent the goal as the LHS of the rule and the condition as the RHS. The first step is to understand generally what is forming the condition and goal for each rule. For example, a rule that dictates what intrusion is possible consists of the intrusion as the goal and the attack as the condition.
- **List of Conditions:** Vulnerabilities and systems involved are typically given as a list of conditions which are all required and so must be logically conjoined. Note that it is more difficult to represent other values (attacks, intrusions, errors, failures) as a list in the encoding because logical conclusions in Prolog cannot be a list of terms and only one term. We specify a method for conjoining a chain to another chain further on in this list.
- **Enabling Intrusion:** An attack can have a list of vulnerabilities and intrusions and the vulnerabilities need to be distinguished from the intrusions. To do that, the encoding can be searched to see if the term occurs as an intrusion for another chain. Assuming no duplicates, if this term does occur elsewhere then it is an intrusion and should be attributed to an intrusion instead of a vulnerability.
- **Map the Encoded Description to the Language Term:** The encoding is written using natural language but the logical rules need to follow a format which is viable in Prolog (for instance excluding spaces). The format we use is described in section 4.5.3.

- **Attack Term Logical Representation:** Attack terms have both an identifier describing that attack term and a location. The knowledge encoding must provide both of these. If no location is provided explicitly, then the target system is assumed.
- **Variable Representation:** When encoding the natural language description to a variable, the name of the variable is arbitrary, but variables used on systems in the LHS must be the same as the variables on systems on the RHS. Furthermore, any other systems that contain interactions or is a component of other systems within that same rule must have the same variable designation. Therefore, any system that interacts with another system or is a component of another system are all to be bound to the same variable.
- **Anonymous Variables:** Variables used only once in a rule should be made anonymous as these are singleton variables.
- **Constants:** Certain terms can be identified by a constant in the encoding. In this case, the term cannot just be any term and must be of the specific value specified by the constant. These are denoted by the font `Sans-Serif`.
- **Systems Involved Inclusion:** Systems involved are included in the vulnerability/enabling intrusion to attack part of the chain. These systems ensure that an attack is only possible given a certain set of circumstances beyond just vulnerability designations. For instance, some attacks rely on logins existing, or connections existing.
- **Intrusion-System Permutations:** Some chains require that certain systems be involved in order for an attack to be possible. However, these systems could be introduced via an intrusion instead. In the encoding, the *alternative* keyword is used. A complication introduced by this way of thinking is that for every system in a rule that could be introduced via an intrusion instead, there must be 2^n rules where n is the number of systems that can be introduced via an intrusion.
- **No Duplicate Rules:** If a circumstance would arise to cause a duplicate rule then do not duplicate rule as this could introduce confusion over two different paths to a failure where actually, the application of a duplicate rule has occurred. Note also that a condition requiring two terms to be true, say term A and term B then $A \wedge B$ is equivalent to $B \wedge A$ and this would constitute as a duplicate rule.
- **Rule Chain Conjoining:** Some values in the table indicate a conjoining rule from some other chain. Specifically, for the browser, the integrity failure can only occur with an additional error from another chain. This is specified using the *with* keyword. That other error is specified in the value containing the goal itself. If this is specified in the condition of a rule, then ignore it as it does not matter how that condition is met just that the condition is met. Note, that in the provided encoding it is not possible for a condition to have a conjunction because the only occurrence of conjunction in the chains is for a failure, which cannot form the condition of any other rule. Also note that duplicate rules

cannot occur, the rule will not be made twice when looking at the other chain which corresponds to the conjoining rule.

What is important is that an encoding can be followed to systematically reproduce a ruleset. Let us apply these principles to some extracts from the encoding to show that the process is systematic and reproducible.

Simple Case: In a simple case the goal of a rule is achieved through a condition with one term. For instance, for the XSS Execution attack chain, malicious code on the browser results in malicious code execution. From this we use the *Understand the Goal and Condition* principle to establish that the goal (LHS) is malicious code execution and the condition is malicious code on the browser. Next, we *Map the Encoded Description to the Language Terms* and find that these two terms are attack terms because they appear in the intrusion and error column. The *Attack Term Logical Representation* principle requires an identifier and a location. The identifier is provided by a Prolog compatible string describing the attack. For example, malicious code on the browser simply becomes "maliciousCode". Next we consider the location for these attack terms which is specified as a system. For the intrusion, the location of the system is specified as the browser. For the error, the system is not specified and since no system is explicitly stated the target system is assumed which is the browser. For each of these systems, the variables need to be specified. The *Variable Representation* principle is considered and since the browser occurs on both the LHS and the RHS the same variables are used resulting in the rule:-

$$err(maliciousCodeExecution, browser(A,B,C)) \leftarrow int(maliciousCode, browser(A,B,C))$$

Vulnerabilities and Enabling Intrusions to Attack: Attacks being made possible by vulnerabilities and enabling intrusions typically include lists of vulnerabilities but also could include a list of systems involved. Since this part of the chain is the vulnerability and enabling intrusions to attack, the *List of Conditions* principle is used to separate the list into terms. The *Systems Involved Inclusion* principle is also used to obtain a list of systems that make the attack possible. Consider the forced login CSRF attack. This attack includes a list of vulnerabilities including "automatic authorisation" and "unvalidated link". Also, the malicious code intrusion is in the list. But how do we distinguish that this is an intrusion rather than a vulnerability? Applying the *Enabling Intrusion* principle, this malicious code entry is interpreted correctly as an intrusion instead of a vulnerability. Putting this together the resulting rule is:-

$$att(forceLogCsrf, login(forceLog, A, B)) \leftarrow$$
$$vuln(autoAuth, idp(\_, C)), vuln(unvalidLink, sp(B, C)), int(maliciousCode, browser(A, C, \_))$$

Intrusion-System Permutations: Some rules can have an intrusion take the place of some involved system. These intrusions need to be clearly marked as we see in the XSS Token Theft chain. In this case, a login can be created through an intrusion by a forced login, or a connection can be created through a malicious redirect. There must exist a rule for every permutation of this chain (2^n) as per the *Intrusion-System Permutations* principle which for this example would be four different rules. Below is the rule which corresponds to both intrusions forming the condition:-

$$att(xssTheft, sp(A, B)) \leftarrow$$
$$vuln(autoAuth, idp(C, B)), fedCon(\_, B, A, C),$$
$$int(forcedLog, login(\_, D, A)), int(malRed, actCon(\_, B, A, D))$$

Rule Chain Conjoining: Having lists describing the logical conjunction of vulnerabilities/intrusions and systems is easy for vulnerability to attack rules as the conditions are not used in the creation of more rules. However, for any other stage in the rule genesis process this could become confusing as only one goal is allowed in Prolog which would make lists of attacks, intrusions, errors, and failures problematic. To address this, we say that some goals can only be realised with another chain in the encoding. For instance, consider the XSS Execution rule. Specified is a failure of integrity *with* browser endpoint trust. Referring to the other chains in the encoding as per the *Rule Chain Conjoining* principle, we find that there is another error called browser endpoint trust produced in the Forced Redirect chain so we add this as a condition of the integrity failure. Note that when the error to failure part of the Forced Redirect chain is processed no rule is produced because of the *No Duplicate Rules* principle.

This process is applied to every chain until we get a full rule-set which describes the attack suite initially described by Sun and Beznosov (2012) and encoded as part of this thesis in section 3.9. The full list of rules used can be found in appendix B.

4.5.8. System Model for Case Study

The aim is to provide a system model suitable for describing the vulnerability interplay introduced by Sun and Beznosov (2012) (specifically in the paper presented in section 5.5). A

broad description is given, such as the kinds of attacks that are needed and some of the systems that are required to be in place such as logins. Other parts of what is required in the system can be inferred by considering other parts of the paper, like what specific vulnerabilities cause forced login CSRF to be possible. In subsection 4.5.2, a general idea of what systems are required to represent what is happening in this attack suite is presented. The aim for this subsection is to collate all of the information provided by Sun and Beznosov to form a precise system model of what causes the specific vulnerability interplay to be possible.

User: To describe this issue one user is considered. Adopting standard Alice-Bob naming conventions, we will call this user Alice.

FIM System and Primary Actors: The overall FIM system is described as an OAuth system. The primary actors within this FIM system are a SP which we will call Service, an IdP we will call Identity, and a browser that is supposed to interact with Alice, which we will call Alice Browser.

External SP: This attack is possible because of malicious code that is initially injected onto the browser. We model this process as occurring from some SP external to the FIM system we will call Untrusted.

Connections: A connection must be made between Alice's browser and the untrusted service provider. This connection makes the malicious code execution attack possible. In addition, there must exist a federation contract between service and identity. These are not just two arbitrary SPs and IdPs in the FIM system, they must be connected, otherwise attacks like the force login would not be possible.

Accounts: Accounts are represented as interactions between the SP and IdP within a FIM system. There is an account on the SP called "aliceSpAcc" that the attacker is trying to compromise. There also exists an IdP account called "aliceIdPAcc" which belongs solely to the IdP. The rationale for this decision becomes clear when we consider any logins that exist in the system. Another account that is considered as part of the system model is an attacker account as an attacker account is required to perform session swapping.

Login: As part of this attack the attacker has to force a login for the SP account. However, the force login attack for the SP is only possible if the user is logged into the IdP account. This is why two accounts are modelled one for the SP and one for the IdP. Because a login can be specified as pre-existing for the IdP, but the other SP login the attacker has to somehow force.

Vulnerabilities: There are a number of vulnerabilities that are considered by Sun and Beznosov (2012), which need to be attached to this model. The first is that an XSS attack is staged on the untrusted SP. This vulnerability models that this untrusted SP has some malicious code on it. How exactly that code can end up on the untrusted SP is a matter that is considered out of scope since the SP is outside of the FIM system and not relevant to this work.

Specifically mentioned by Sun and Beznosov (2012) as a vulnerability is how IdPs have an automatic authorisation feature which grants access to certain resources, based on the user having previously given permission to access this resource. We could consider this vulnerability more deeply and realise that this could actually be broken down into many vulnerabilities occurring at different entities. For instance, we could consider how this problem can be attributed to the protocol specification and how the IdP also failed to correct this problem in an implementation; meaning that the overall cause can be attributed to a conjunction of these two things. However, this goes deeper than is needed to demonstrate the causality for this attack, so we just consider the vulnerability on the IdP.

For a CSRF attack to be possible, there has to exist some unvalidated request that causes some state change in the server (OWASP (2019a)). Possibly the most straight-forward way for a CSRF to be possible is through an unvalidated link for the server, which is the approach we take. The unvalidated link occurs on Service.

The last vulnerability considered is a lack of contextual binding. This vulnerability is stated to occur on the SP so we model this vulnerability as occurring on Service.

There is one other vulnerability that we a consider which Sun and Beznosov (2012) does not explicitly state. That vulnerability is the inability of the browser to distinguish the legitimate website code from the illegitimate website code. This is because we aim to consider a more holistic view of causality beyond just the SP and IdP.

For a full overview of the system model including the attached vulnerabilities see appendix tab. B.1 which is how the system model is formally specified in Prolog using the language introduced in 4.5.3. Also, consider fig. 4.3 for a visualisation of this system model.

4.6. Attack Traces and Escalation in OAuth

4.6.1. Establishing Attack Traces

Attack Trace: We can consider the causality of attacks in FIM using an attack trace. An attack trace is represented as a tree where each node on the tree can have any number of conjunctive conditions which cause that goal to be true unless that node is a leaf node. Each node must have exactly one parent unless the node is the root of the tree. An attack trace can then be interpreted as a sequence of events from bottom to top. Consider subsection 2.5.3 for a formal definition of attack trees and attack traces.

The system model and accompanying rules are not sufficient for generating an attack trace. For a concrete attack trace to be established, the system model needs to be queried against the knowledge base systematically. If causality for a security failure on the overall FIM system is to be established, the possibility of that security failure needs to be queried and any subsequent dependencies that security failure requires needs to be established.

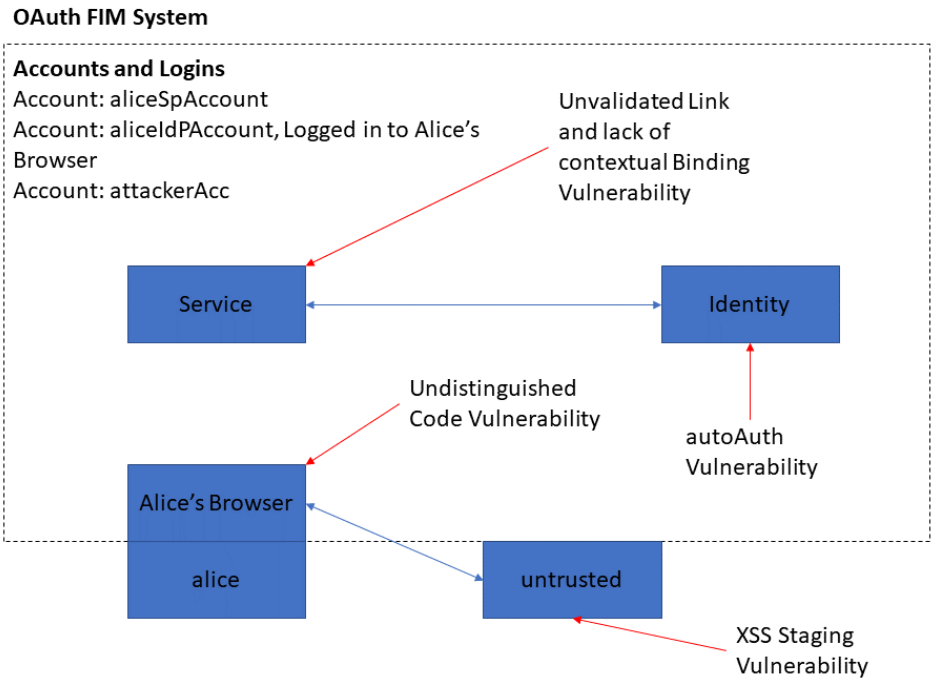

Figure 4.3 Visualisation of OAuth System Model

As described in section 2.4, Prolog only instantiates the arguments for a particular term which is presented. For example, if we were looking to see if a failure of confidentiality is possible in FIM we could query "fail(fimSystem, confidentiality)". In this case Prolog would return true if the query is satisfiable (i.e. the failure is possible), and false if the query is non-satisfiable (i.e. the failure is not possible. We could also query "fail(fimSystem, X)". Prolog would return all the values of the variable X that would make the query satisfiable, and false if the query is non-satisfiable. This behaviour allows us to see what is possible, but does not allow us to see how it is possible through an attack trace. Modelling escalation of attacks requires showing how an attack can be caused by the effect of another attack.

Trace Building: Prolog does offer extension functionality that allows for investigation into the chain of logical conclusions. If trace mode in Prolog is enabled, the calls that Prolog makes to come to a logical conclusion can be considered. Using this functionality as a basis, an attack trace can be built. An official overview of the Prolog trace tool is available at SWI-Prolog (n.d.). Let us consider the Prolog trace mode and how it can be used to generate attack traces.

This kind of approach is not unprecedented, Ou et al. (2006) does something similar and provides an algorithm for generating an attack graph from a trace. Ou et al. takes the entirety of a trace and forms an attack graph complete with any disjunctive nodes being linked. In this chapter, we focus on an algorithm that treats each individual Prolog solution as its own tree. This approach allows us to break down the problem to each specific component of the FIM system.

The difference in method of the algorithm we use is that we focus on failures for each system individually in a conjunctive chain by considering the security properties for each system.

When a goal is input into Prolog with trace enabled the steps Prolog takes to prove that the goal is true are shown unlike in normal Prolog operation. Prolog will exhaustively test a a goal for truth. When no more solutions are available, the trace outputs false. Prolog uses four standard ports in a trace: *Call*, *Exit*, *Redo*, and *Fail*:-

- **Call:** attempts to prove a goal as true, and in the case that a rule is applied could Call another goal. When Call is first used for a particular goal, the initial variables can be unknown in the case that Prolog has not yet unified them.
- **Exit:** is used when a goal is proved to be true by calling a base-level fact or when one goal has been satisfied by proving another goal. When Exit is called, any unknown variables will be unified with the term.
- **Fail:** is similar to Exit but is used when a particular goal is not satisfiable.
- **Redo:** is used to satisfy a goal that has been attempted before but using different sub-goals. Once a goal has been established to be true, more solutions can be considered under different sub-goals using Redo.

Using this information a general algorithm can be considered for generating attack traces from a Prolog trace. The algorithm is called TraceBuilder. The input to this algorithm are the lines produced by the Prolog trace. These lines can be thought of as a list where each line corresponds to a single item on the list. Each line consists of two important parts (there are other details but we ignore them for this thesis), the first is a descriptor (call, exit, etc.) and the second is the body of the line which represents the goal of a particular line. The exception is where a line just has the result of a particular trace being stated to be simply true or false. Truth can also be presented as variables satisfying the goal. A data structure is required to represent the trace. Since goals can have multiple conditions a tree data-structure is a reasonable approach to representing the trace. The output of the algorithm is a list of all the traces which are possible.

Summary of Algorithm: We use an object orientated approach to describe the algorithm. Specifically, this algorithm was written in Java and involves several user defined classes that have dependencies on the standard Java library. We provide a UML diagram in fig. 4.4 and omit the full class descriptions for conciseness. To summarise, the main class reads a Prolog trace from a file using the CustomIO class which converts a Prolog trace debug as detailed by SWI-Prolog (n.d.) and converts each debug statement into an object in the Line class that contains the relevant aspects of each debug statement. These lines are input into the TraceBuilder algorithm class to produce a trace in the form of the Trace class. The core algorithm, TraceBuilder, is provided in Algorithm 1. In addition, we explain several intricacies of the algorithm:-

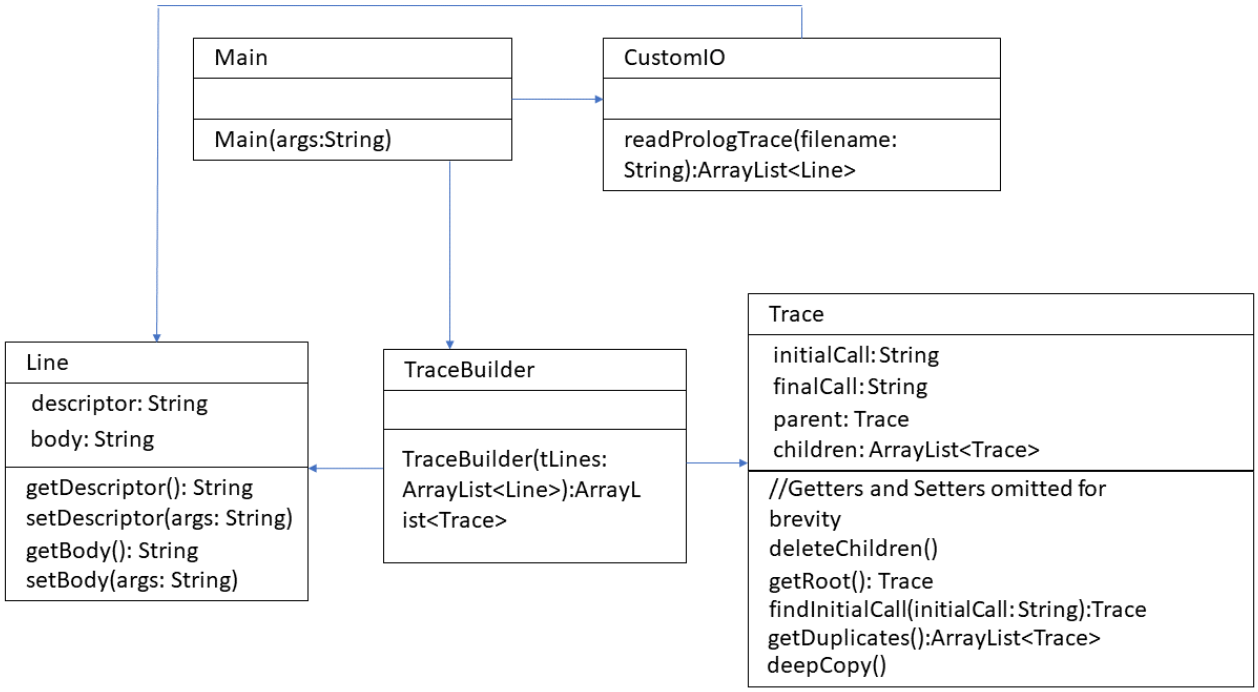

Figure 4.4 UML Diagram of Java Implementation of Trace Builder

- Each trace object refers to a term in Prolog. When each term is attempted for the first time, the call descriptor is used. It is possible that Prolog does not yet know the final values of this term as the term is yet to be bound. When the exit descriptor is used, these values are bound. This is why a trace object has both *initialCall* and *finalCall* variables.
- We maintain the *initialCall* variable because the Redo descriptor retries the *initialCall* of the term, which means we need to find this term in the tree and perform the Redo.
- The *finalCall* variable represents the term with all properties in the term bound. When a Redo descriptor is encountered, Prolog can check the truth of terms above the redone trace object which would have never been deleted because only trace objects beneath the Redo are deleted. This leaves the possibility of duplicates, which we delete by seeing if there is more than one node in a list of children with the same *finalCall* variable.
- We do not consider traces where the initial query has variables in it. The variables can still be used in the Prolog program itself to understand different ways in which goals can be realised. However, when variables are involved in the initial query, the trace output becomes more complex as instead of simply stating the truth of a query Prolog also considers what values the variables take. It would not be difficult to modify the algorithm to take account for this, but in the interest of simplicity we do not consider this detail.

```
algorithm : TraceBuilder
input     : A list of trace statements tLines
output    : A list of traces traces

Trace trace = new Trace("New Trace");
for each tl in tLines do /* (1)                                              */
    if tl.getDescriptor() == 'Call' then /* (2)                              */
        if trace.getInitialCall() != "New Trace" then
            Trace child := new Trace(tl.getBody(),trace);
            trace.addChild(child);
            trace := child;
        else
            trace = new Trace(tl.getBody());
        end
    else if tl.descriptor == 'Redo' then/* (3)                               */
        trace = trace.getRoot().findInitialCall(tl.getBody());
        trace.deleteAllChildren();
    else if tl.descriptor == 'Exit' then/* (4)                               */
        trace.setFinalCall(tl.getBody());
        Duplicates can occur after a Redo retries a goal, and then retries another
        goal further up the tree without ever calling Redo again.
        List duplicates = trace.getDuplicates();
        if !duplicates.isEmpty() then
            for each traceDuplicate in duplicates do
                trace.getChildren().remove(traceDuplicate);
            end
        if trace.getParent() != null then
            trace = trace.getParent();
    else if tl.descriptor == 'Fail' then/* (5)                                */
        trace = trace.getParent();
    else if tl.descriptor == 'false' then/* (6)                              */
        Do nothing, as the trace is not added unless a true statement is
        encountered.
    else /* (7)                                                              */
        The trace is true
        traces.add(trace);
        trace = trace.deepCopy();
end

return    : traces
```

Algorithm 1: List of trace statements to trace data structure. Comments in blue and numbers in parenthesis are annotations to be referred to for consideration of time computational complexity.

4.6.2. Traces

Applying the rule base describing the attacks described by Sun and Beznosov (2012) to a system model which represents the applicable system in its beginning state outputs several possibilities in Prolog. Using an algorithm for building traces by considering the possibilities provided by Prolog yields several attack traces. These traces are made possible when failures are queried for

Figure 4.5 Attack trace on the Browser using simplified syntax described in subsection 4.5.5.

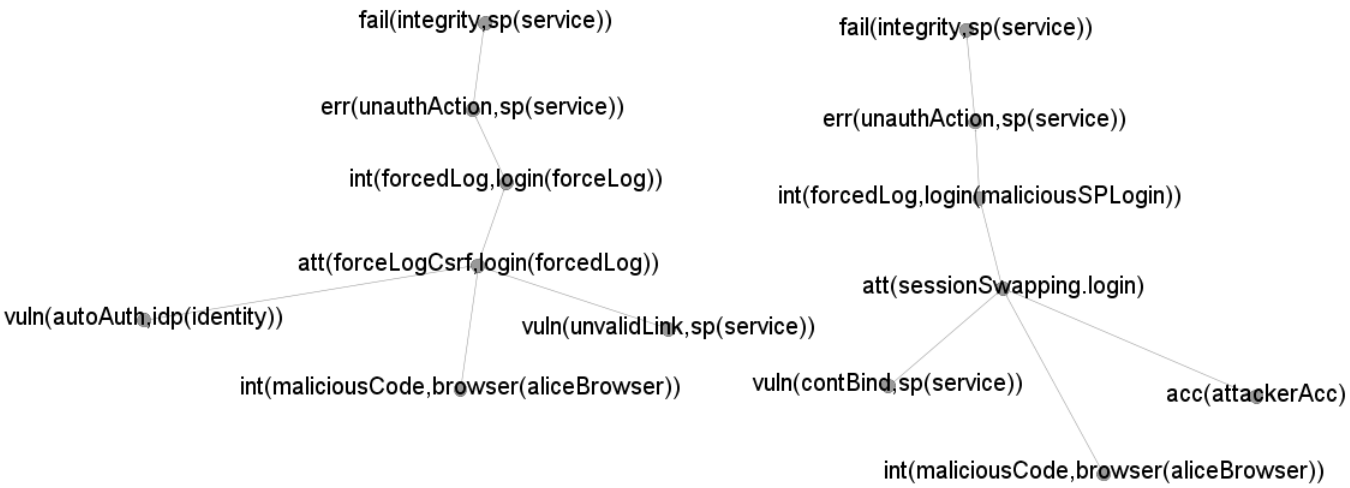

Figure 4.6 Attack traces on the SP's Integrity using simplified syntax described in subsection 4.5.5. Note that the malicious code intrusion escalates from the browser attack trace (fig. 4.5).

each specific system. Failures are possible within the browser, SP, and overall FIM system. These failures are illustrated respectively in fig. 4.5, 4.6, 4.7, and 4.8.

Note that these traces are an original contribution from our interpretation of the attack suite described by Sun and Beznosov (2012). Sun and Beznosov do not focus on algorithmic computation of attack traces. We use their work as a basis for evaluating correctness of the traces we generate and our attack traces should conform.

The traces link to nodes in other traces, for example, the FIM failure requires a forced login that is only possible through the the attacks realised on the SP. The idea is that this attack trace demonstrates attack escalation.

Each trace corresponds to a single security failure. Some traces require multiple security issues, for example, the browser requires two errors for an overall failure to be possible. Other traces

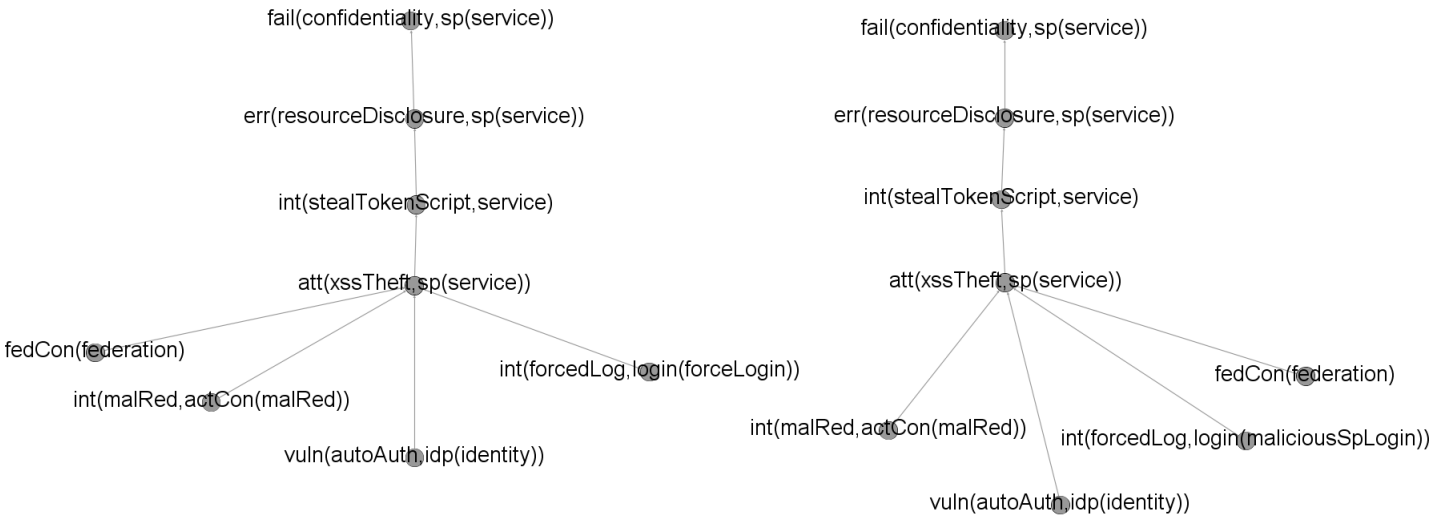

Figure 4.7 Attack traces on the SP's Confidentiality using simplified syntax described in subsection 4.5.5. Note that the forced logins for each trace escalate from the Integrity failures on the SP (fig. 4.6) and the malicious redirects escalate from the browser attack trace (fig. 4.5).

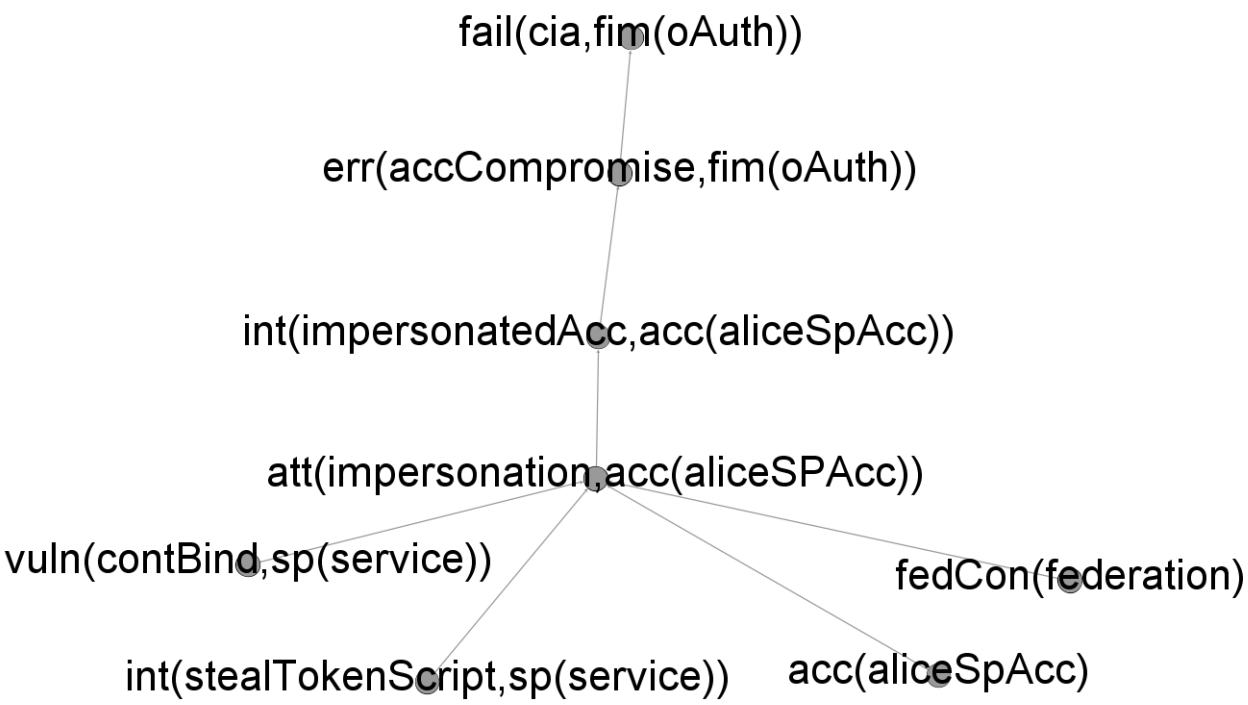

Figure 4.8 Attack Trace FIM Overall using simplified syntax described in subsection 4.5.5. Note that the steal token script intrusion links to the SP confidentiality attack trace (fig. 4.7).

have a common cause, but that cause can be realised in different ways. For example, the FIM trace shows that a failure is possible through a forced login. But that forced login could be realised in two different ways: forced login CSRF, or session swapping. Therefore, there are two different ways that the overall attack on FIM can be realised, which is not reflected in the FIM trace itself.

These traces show the effect of the attack through an intrusion becoming the cause of another attack. The traces present a model of attack escalation to address RQ2. These traces can be realised automatically through a logical analysis using terms to represent the system model and rules to represent a threat model and what is possible for an attacker to do.

Computational Complexity of Trace Builder algorithm (alg. 1): Analysing algorithms often depends on the input according to Cormen et al. (2009). The input to this algorithm is the number of lines in a trace statement, let us call this n. Something which complicates the analysis is the algorithm is also dependent on the trace structure. Let us consider m as the worst case size of any trace in terms of number of entries. The algorithm has numbers annotated to refer to when considering the time complexity. We will break the algorithm down into discussion points with reference to these annotations. Once the time complexity for each of these annotations is understood, we will show a bigger picture of the overall time complexity of the algorithm.

Annotation *(1)* refers to a for loop which is dependent on *O(n)* amount of trace lines that are input into the algorithm. Consider that annotation *(5)* and *(6)* clearly occur in constant time as only comparisons and variable setting occurs. Note that also, before the for loop denoted by *(1)* occurs, a new trace object is made, which is also a constant time operation.

Annotation *(2)* contains an if statement. If the condition is met, a new trace object is made using data from the trace line and another trace. These are both constant time operations. Else, the trace is initialised for the first time using data from the trace lines, which also occurs in constant time. Therefore, the overall time complexity for *(2)* is *O(1)*.

Annotation *(3)* navigates to the root of the trace and then finds the point in the trace which is being redone. This part of the algorithm is therefore dependent on the size of the trace. Note that for simplicity the root of the trace is not stored in any of the nodes, meaning we instead find the root by recursively calling get root on the parent of each node until no parent is found. In the worst case this could be the entire size of the trace if we assume that each node in the trace has at most one child node, making the trace into what is effectively a linked list. Subsequently, we then find a node in the trace with the initial call specified by the redo, which could be near the end of the trace, making the while operation of setting the trace position. In the worst case this operation is *2m* simplifying to *O(m)*.

The next line for *(3)* deletes all nodes which are children for a particular trace. If we assume that all nodes in the trace have one parent, then this is in the worst case *O(m)*. However it is worth noting that for the previous line of code we assume that the worst case happens when the trace is

effectively a linked list. These worst case assumptions can not be true at the same time. Regardless, deleting all nodes for a particular trace could still grow linearly in relation to the size of the trace, meaning that in the worst case this is *O(m)*. The overall complexity for *(3)* is therefore *O(m)*.

Annotation *(4)* refers mainly to getting duplicates for a particular set of children to a node. A trace is considered to be a duplicate if the *finalCall* variable is the same as another trace. We assume a trace where each node has no parent or the same parent as every other node. To do this in the simple case, since this list is not sorted in any meaningful way, the time complexity is $O(m^2)$. Then, if the list of duplicates is not empty, we iterate through every one of these duplicates. In the worst case, this list of duplicates grows linearly with the size of the trace, meaning this iteration is *O(m)*. For each of these iterators, a remove operation is done which could be done in constant time. The duplicate removal process is therefore $O(m)$. Since the removal of duplicates process follows on directly from the finding of duplicates, the overall time complexity for *(4)* is $O(m^2)$.

Lastly, annotation *(7)*. The first line of code simply adds the trace to the list of traces which is the output of the algorithm and this occurs in constant time. However, Prolog can find other solutions. We handle this by making a copy of the trace as it is currently, which represents a solution, and then performing a deep copy to be further worked on, potentially finding other solutions. To do this, we only need to visit each node in the trace once and initialise each trace with a new reference. The overall time complexity of *(7)* is *O(m)*.

Let us now consider the big picture of how this algorithm performs overall. This algorithm linearly iterates through the trace lines in annotation *(1)* and this is expressed as *O(n)*. Within this iteration in some cases, the algorithm performs in *O(1)* as per annotation *(2)*, *(5)*, and *(6)*. In other cases, the algorithm performs in *O(m)*, where m is the size of the trace as per annotation *(3)* and *(7)*. In the worst case, the algorithm performs in $O(m^2)$ as per annotation *(4)*. Assuming the algorithm executes in the worst case, the overall time complexity is stated to be $O(n+m^2)$

4.7. Applying the Language to the Wider Context of FIM

The language presented so far in this chapter is specific to the case study initially provided by Sun and Beznosov (2012) and then encoded in section 3.9. But security issues in FIM are wider in nature than the language provided thus far and to better satisfy RQ2 a wider scope should be considered. In chapter 3 a comprehensive survey of security issues in FIM is described. Let us consider the language and see how it is applicable to the wider context of FIM. In particular, focus is placed on how we expand on the concepts already introduced to ensure that the language can represent the breadth of security issues uncovered throughout the survey. We do this as a means of validating our work by focusing on completeness as defined in 4.2.2.

There are several key dimensions of security issues that are not covered by the Sun and Beznosov (2012) example alone. The idea is to show how the approach we introduce can be

adapted to cover these different security issues. Overall, we intend to validate our logical analysis approach by showing how it can be used in a variety of different circumstances. What are some additional vulnerabilities and attacks to consider? What are additional locations for these attacks to consider in order for the attack to be properly represented? Are there any other security rules and goals to be considered? What are some of the key logical rules that are needed to consider the issue? The goal of this section is not to show how every identified issue in the survey would be encoded as many are similar conceptually. The goal is to show the flexibility and power of the approach applied to different paradigms in FIM.

The argument being made in this section is that any issue in the survey can be logically analysed using the language already introduced with some additional considerations. The paradigms being introduced in this section are realised from considering the security issues in the survey systematically and asking the following question: can this issue be described using the existing language; if not, what needs to be added? Let us introduce overlooked paradigms discovered from the survey and show how these issues can be represented in the language.

4.7.1. Attacker Capability

Bau and Mitchell (2011) introduces the idea that security modelling should include the capabilities of an attacker. One problem with using existing knowledge for security modelling is that complete information required to make a precise model is not always presented. This is especially true for attacker capabilities, which vary across different work. The case study we have used so far from Sun and Beznosov (2012) outlines a suite of attacks. This suite of attacks outlines a kind of threat model for OAuth but does not give indication of the exact attacker capabilities that are needed in order for the attack suite to be realised. In subsection 4.2.1 the idea of not using explicit attacker capabilities is discussed. There are many examples in the literature which are valid and do not fit the exact definition provided by Bau and Mitchell (2011). But these capabilities can be included easily as part of an approach leveraging logical analysis.

Even though Sun and Beznosov (2012) do not explicitly mention the attacker needing to know how to script, this kind of capability would be required by an attacker in order to successfully exploit a system using XSS. This is made clear by referring to OWASP (2019b), where specifically the requirement of injecting a script is considered. The attack is not possible if the attacker cannot write this script. Why this piece of information is useful, is because when considering an overall model of the risk for a particular system there might be cause for an accepted risk. If an attack is possible, but the capability required for an attack is advanced to the point at which the risk is considered to be unlikely to be exploited then the risk could be allowed to exist. This kind of approach seems unwise with the attack suite outlined by Sun and Beznosov because the capabilities required to carry out the attacks is not extensive.

More interesting attacker capabilities exist. Consider a particular formal analysis on OAuth by Chari et al. (2011) which serves as a basis for the analysis performed in Sun and Beznosov (2012). This formal analysis is based on the assumption that an attacker is bound by polynomial

time. The protocol is considered to be secure under this assumption. This attacker model is clearly a sensible attacker model, but for the sake of fully assessing the avenues of attack, a capability of somehow circumventing the polynomial time restriction could be considered.

For the sake of modelling different kinds of attacker, capabilities can be introduced to logical analysis. To do this is a straight forward addition in our approach. Included in the system model, a number of capabilities can be introduced that the model should be evaluated against. These capabilities could represent, for example, that an attacker is capable of writing a script. To use proper Prolog compatible syntax, $capability(canScript)$.

The next step is to incorporate capabilities as part of a logical rule. Take any given rule that has a goal A and a set of conditions B. We can then apply a union operation of a set of capabilities C to the set of conditions B. This approach can be performed systematically for a given ruleset R by searching for the goal that needs to have a capability appended for every rule r in R.

4.7.2. Availability Issues

We have seen several issues that pertain to the availability of a FIM system. For example, the issue proposed by Kormann and Rubin (2000) which relates to a centralised infrastructure vulnerability. To logically capture this vulnerability, the term $vuln(centralised, idp(identity))$ can be used to describe the vulnerability on an IdP called identity. An attack can be caused by this vulnerability using a rule such as $att(dos, idp(X)) \leftarrow vuln(centralised, idp(X))$ which would identify that an attack is possible on any IdP where the centralised vulnerability is present. This attack could then be used by other rules until a failure of availability is possible.

Another thing to consider with these kind of attacks is that often an attacker with significant capabilities is required. For instance, an attacker that has access to a large number of hosts that can be used to deliver the DoS attack. In subsection 4.7.1, the idea of adding attacker capabilities to the language is considered. In the situation of describing DoS attacks, this capability is particularly relevant. For instance, an attacker could be said to have a capability like $capability(hostNetwork)$ to describe that the attacker has access to a large network that could be used to facilitate a DoS attack. This allows for consideration of the risk of the system against different attacker models.

4.7.3. Connection Issues

Issues exist because of an implementation error occurring across two stakeholders in a FIM system. For instance, consider an access token being exposed over the internet without end to end encryption as is discovered by Sun and Beznosov (2012). Such an issue issue is perhaps best suited to be described as a vulnerability with the connection itself. For instance, consider an active connection called "con". A vulnerability describing inadequate end to end encryption on con between a browser called "alice" and a SP called "service" could be described

$vuln(unencrypted, actCon(con, alice, service))$. Any rule can then leverage this vulnerability, for instance, an attack such as packet sniffing.

4.7.4. Protocol Issues

The approach taken by Sun and Beznosov (2012) mainly focuses on implementation shortcomings which can easily be attributed to different stakeholders such as the SP and IdP. However, a significant number of issues can also be attributed to the FIM protocol in general beyond any implementation shortcomings as can be seen in chapter 3. The problem is that the FIM protocol itself is not an entity in the system model. Similar to the connection issues discussed in subsection 4.7.3, protocol issues can also be represented in the connections.

A key goal in the system model is to avoid protocol complexities. But we can still describe protocol issues using existing knowledge without having to describe the operational semantics of the protocol itself. The vulnerability in a protocol can be represented in the connection between the two hosts and this vulnerability can be attacked. Using this approach, we can model escalating attacks as we model what is possible once that initial exploitation has been done.

Pfitzmann and Waidner (2003a) introduces a MITM type of attack to Liberty Alliance. A dishonest SP can impersonate a user to an honest SP because there is a lack of binding in the protocol which binds the access token to the context of the intended token. This lack of binding can be represented as a vulnerability in the authentication connection between a browser and a IdP. The example given by Pfitzmann and Waidner is that a single SP is compromised. This attack could be applied to every SP in the federation in which this vulnerability is present. If a more extensive system model is considered with more SPs, using our approach the potential scale of the attack can be more fully realised. A more extensive system model is presented in fig. 4.9.

How can a logical analysis approach capture the extent of the issue across the entire federation? The entire process that Pfitzmann and Waidner (2003a) describe can be broken down into two parts. The first is the construction of an unbound access token using the lack of contextual binding vulnerability. This can be represented as a vulnerability, attack, intrusion chain. The second is how that unbound access token can be used to access user accounts across the FIM federation. This can also be represented as a chain with the difference being that the intrusion from the first part causes the second part. An example of a ruleset that could capture this can be seen in tab. 4.5. Note that for simplicity the exact security properties that intrusions could violate are omitted but a security goal on the overall system to maintain the CIA of a user account could be specified, which is implied through a security rule that states no unauthorised access should happen on an account. Also note that on row 3 of the table the rule condition includes an account system. This is to ensure that an account actually exists for this rule to apply to.

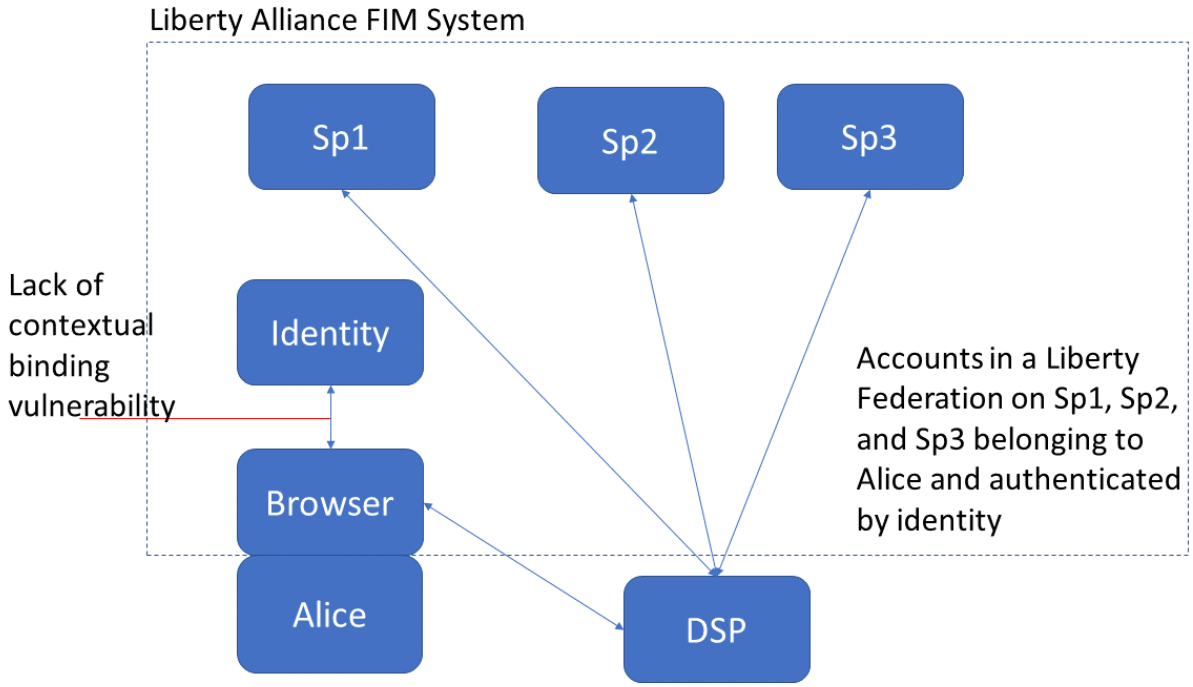

Figure 4.9 Liberty Alliace Federation with Multiple SPs

Rule Goal	Rule Condition
att(msgCreation,authCon(X,Y,Z))	vulnerability(contBind,authCon(X,Y,Z))
int(unboundMsg,authCon(X,Y,Z))	att(msgCreation,authCon(X,Y,Z))
att(compromiseAcc,acc(_,_,_,X,_))	int(unboundMsg,authCon(_,_,X),acc(_,_- ,_,X,_)
int(compromisedAcc,acc(V,W,X,Y,Z))	att(compromiseAcc,acc(V,W,X,Y,Z))

Table 4.5 Limited ruleset for Pfitzmann and Waidner (2003a)

4.7.5. New Systems - DNS

The systems already described are not adequate to holistically model all of the issues observed in the survey. There are times when entirely new systems need to be added to the overall model. When adding a system, a number of things need to be considered which are outlined in subsection 4.5.2. In brief, we have to consider:-

- **Components:** What is this system a component of? What components does this system have?
- **Interactions:** How does this system interact with the environment?
- **Securtiy Properties:** What are this system's security goals if any? How do security rules imply those goals?

Complications occur when adding a system and having existing systems interact with that system. This is because the existing systems would need to be updated to include the additional system. Any rule bases using these system descriptions would also need to be updated. The

DNS, and connections to that DNS, can be represented without needing to update the logical representation of a system and any existing rules.

The case of a DNS is interesting in that sometimes it could be included as part of the FIM system for holistic analysis, and sometimes it could not. Groß (2003) investigates how a DNS vulnerability can be used in compromising FIM. In the possible solutions for this attack, improvements to the protocol specification are suggested rather than how the DNS system could be fixed. We interpret this to mean that the DNS system is viewed as some external entity that cannot be controlled while the protocol specification can be controlled. But for some FIM systems, the DNS could be controlled. For instance, Google operates a FIM system but also offers one of the largest public DNS systems in the world (Google (n.d.c)). In this situation, the DNS can be considered internal to the whole FIM system. A high-level representation of the DNS can be seen below where we also consider a new security rule that implies integrity. The rule specifically refers to the DNS cache which should only be edited through an authorised party.

- *Component Of:* FIM or External.
- *One-to-One Interactions:* None.
- *Components:* No components.
- *Security Properties:* Authorised Changes to Cache $\rightarrow$ Integrity

How does a DNS interact with other systems in FIM? The DNS resolution process can be abstracted to a connection that occurs between a SP or IdP and the DNS. This is not the only way the process could be represented, as it could also be attached to existing connections between, for instance, browsers and SPs. But to avoid having connections that are dependent on other connections, for this example a system is considered which connects the DNS and the SP/IdP. The connections could also occur within the FIM system or externally depending on if the DNS is seen as some external or internal system. The DNS interacts with the FIM system through these connections. A high-level overview of the DNS connection can be seen below.

- *Component Of:* FIM or External.
- *One-to-One Interactions:* SP/IdP and DNS.
- *Components:* No components.
- *Security Properties:* None

If a DNS is susceptible to tampering, this can be modelled as a vulnerability on the DNS. The attacker model could leverage this from a rule where if a vulnerability exists on a DNS, and a connection between the DNS and the SP exists, then an intrusion could be realised where the DNS fails for a particular SP/IdP for which the DNS resolution has been compromised. This would immediately constitute a failure of the DNS system, but could also lead to other failures

in the FIM system. In this case, the logical definition of the FIM system changes so that the FIM system can be considered a component of another system.

4.7.6. Changing Existing Systems - FIM

The way systems are logically represented in this work is not necessarily the only way that those systems can be logically represented. For instance, more interactions could be interpreted. One idea asserted in the OAuth case study presented in 4.5, is that FIM is represented as a top-level system. But this does not necessarily have to be the case. For instance, FIM could be considered a component of a larger access control system.

4.8. Applying the Approach to Novel Security Issues

This section shows how new knowledge of attacks can be encoded for attacks in FIM. The issue presented is takes inspiration from Li and Mitchell (2014) which outlines how a SP account of a user can be linked to an attacker IdP account by exploiting a flaw with the protocol. We consider a similar issue, where a user SP account is linked to an attacker IdP account, but the way this issue is exploited is through the attacker manually exploiting the user interface on the SP to link the SP account to the IdP account. How this attack would be exploited in practice is by the user leaving themselves logged in on a device and the attacker stealthily accessing the device and maliciously linking the user SP account with the attacker IdP account. This allows for an attacker to access the user account in a time and place more appropriate for deeper exploitation of the SP account where the attacker does not have temporary access to the user's device. The attacks involved are said to be escalating because the effect of the account linkage can cause another attack to be possible, which is the attacker using the account linkage as a backdoor for more exploitation. This attack modelling is done as a completeness validation of the logical approach used in this work. Our approach should be used to model escalation of attacks in FIM beyond the existing literature to better satisfy RQ2.

4.8.1. Evidence for the Account Linkage and Threat Model

The concept for this issue was tested on the Facebook FIM system with Facebook serving as an IdP and the Spotify Desktop app serving as the SP. Spotify is a platform for hosting and sharing music online. Most existing literature on FIM attacks surveyed in chapter 3 provided evidence that the described attack is possible. To meet the standards of the existing literature, evidence should be provided for the issue proposed in this work. This evidence exists at either the protocol or implementation level.

One aspect of an issue that the taxonomy does not consider is the threat model of the attacker. Threat models in FIM are discussed in subsection 3.7.6. One threat model that is considered to some extent by Kormann and Rubin (2000) and features in other research such as Maxion and Townsend (2002); Gruhn and Müller (2013), and is discussed in section 3.7.6 is temporary

access. We also refer to temporary access as coffee break access. The issues we put forward require the attacker to have at some point coffee break access to the user's device.

Note that coffee break access is a weaker threat model than physical access. An attacker gaining physical access to a computer system is the subject of much research. Depoy et al. (2005) refers to physical attacks as an attacker gaining physical access to an asset in order to damage or disable it. Specific application areas include automotives according to Checkoway et al. (2011), who considers a threat model where an attacker has indirect access (through an intermediary specialist interface for automotives). Side-channel attacks are another kind of attack that often include physical access and Joy Persial et al. (2011) conducts a survey on side-channel attacks where as part of the threat model the attacker has physical access. Physical access receives much attention from the literature, and is a stronger threat model than coffee break access. Coffee break access should be a cause for concern in computer security as it is a weaker threat model than physical access which receives much attention.

We evidence the attack on the implementation level by simulating the attack from the point of view of the attacker. Since this is an implementation level simulation, we try to consider possible mitigations that the implementation could employ and show that at least for this simulation those mitigations were not present. First an attacker IdP account on Facebook is needed. We created a new Facebook account to represent the attacker under the name Bob Mallory with the email address "s.simpson@ncl.ac.uk". The attacker then attempts to link the IdP account to the Spotify SP account. Note that the Spotify app does not request that the user re-authenticate at this stage, which would mitigate the issue as we are presuming the attacker does not know the user's credentials. On the success of the operation, the Spotify account is linked to the attacker's IdP account. This process can be summarised in fig. 4.10.

Once the account has been linked, a number of things happen. Spotify does state in the settings page that a Facebook account is connected and it is a straightforward process of disconnecting the account. Spotify does usually send emails to the registered email address of new logins so the user may be able to ascertain a problem from that. What is perhaps the most worrying about this attack is that to the ordinary user, it might be difficult to know that the attacker now has a backdoor to the Spotify account even with the information provided by Spotify. From the attacker's point of view, the success of the account linkage can easily be seen from the settings page of the IdP Facebook account. This part of the issue does constitute an integrity failure for the user account because this is an action that the user did not intend. Evidence for what happens on completion of the attack can be seen in fig. 4.11.

The next step for the attacker to take is to login to the account in a setting which is more appropriate for deeper exploitation than the user's device. There is no apparent mechanism Spotify employs to block what seems to be a suspicious login attempt from happening. Login attempts were attempted via VPN changing the geolocation of the attempt to Japan, United States, United Kingdom, and Spain. For these attempts, a different device was used so there is

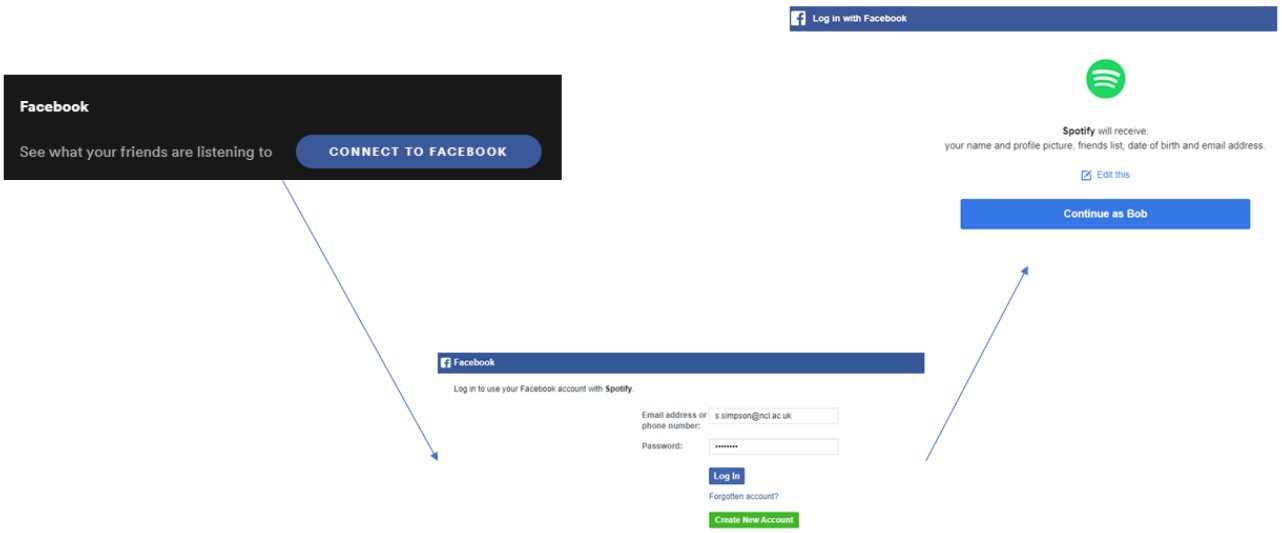

Figure 4.10 Account Linkage Process A.

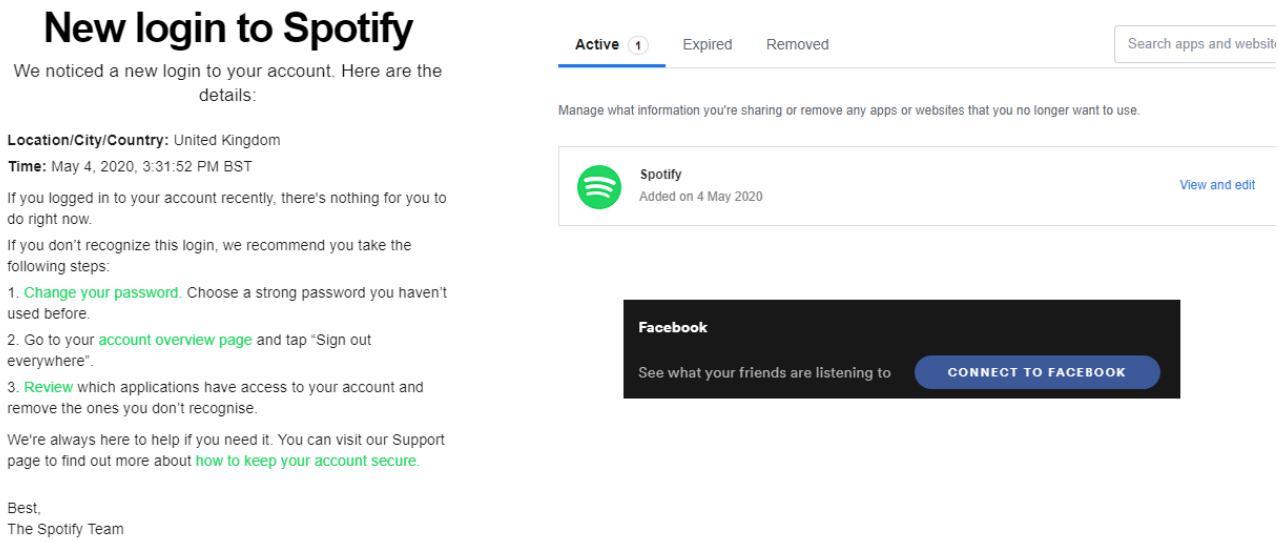

Figure 4.11 Account Linkage Process B.

no apparent blockage based on device fingerprint. According to Spotify (2021) the only circumstances a user account would be blocked is for fraudulent activity or payment issues with the account. Upon investigation of the terms and conditions of Spotify (2019), fraudulent activity pertains mostly to activities such as pretending to be an official from a certain organisation. However, it does seem somewhat ambiguous what is meant by fraudulent activity here. What we know is that from the tests we conducted there was no apparent intrusion detection system that automatically blocked the account from being accessed.

Once the attacker has logged onto the account a number of CIA failures are possible:-

- *Confidentiality:* Basic information about the user can be seen in the account settings page. Such information include the username, the email address, the date of birth, and the payment plan/payment method of the Spotify account. Note that the attacker did not necessarily need to know this information about the user before-hand to link the accounts so any of this information could be used maliciously. For example, attempting to brute force the user's email address. Since the attacker can also learn about the user's music tastes, there is also the possibility of making an informed choice in guessing the user's password. The payment plan is obscured to the last four digits on the expiry date so there is limited options on that vector for the attacker.

- *Integrity:* The attacker can perform most actions the legitimate user could do with access to the Spotify Account. Such actions could include annoyances like subscribing to music the user would not typically enjoy which can influence further music suggestions in the event the user fully regains control of the account. In some cases though, if this account represented a popular artist or brand any unauthorised actions could be more serious and misrepresent this account to a large group of other users in the Spotify service. One critical action that could not be used without knowing the original user password, and cannot be overridden by an IdP login, is a change of user password. It seems likely that the user could gain control of the account back because of this.

- *Availability:* Since the attacker cannot change the user password without knowing the original password, it seems unlikely that the attacker can permanently deny availability to the user. However, Spotify accounts can only stream music to one device at once. If the user is somehow unable to figure out and deal with that has happened to their account, the attacker could deny the user availability of the streaming service.

4.8.2. Taxonomy Categorisation

In chapter 3 we propose a framework for the categorisation of security issues in FIM. We do this as a means of further qualifying the attack we have discovered. The attack should be able to be categorised alongside existing literature and a taxonomy has already been proposed to do that.

This issue can be separated into two parts. The first is the process of linking the account. The second is for the attacker to login to the account in a more appropriate location. In the taxonomy,

this would be categorised as a single issue. This is because the remote login step is rather trivial and does not constitute its own attack class. If this step were given its own attack class, then most attacks in the survey would also involve this attack class as at some stage in the process remote exploitation is expected. The overall issue can be summarised with one attack class already introduced in the survey: Account Binding. The vulnerability could be attributed to a vulnerable SP because The SP does not request the user re-authenticate to bind the account. Alternatively, the vulnerability could be attributed to the Facebook IdP because re-authentication is not part of the specification. We choose to use the vulnerable SP as this is consistent with other literature descriptions of account binding such as Kumar (2014). Since the end result is account compromise, the overall failure is in terms of CIA. This is an implementation level issue. We suggest a solution that forcing the user to authenticate again should mitigate the issue.

Attack Number	Attack Class	Vulnerability	CIA Failure	Target Protocol	Issue Type	Solution Proposed
1	Account Linkage	Vulnerable SP	CIA Failure	Facebook Connect	Implementation	Suggested

Table 4.6 Categorisation of Facebook Account Linkage

4.8.3. Encoding and Rules

Similar to how a knowledge encoding is provided of Sun and Beznosov (2012) in section 3.9, the same can be done for this issue. This encoding is eventually transformed to actual terms and rules that can be processed by Prolog for logical analysis. The synopsis for this attack is that an attacker attacks a user by maliciously linking the attacker IdP account with the user SP account. This linkage is possible because of a vulnerability with the the user interface in how accounts can be linked for users that are logged in without any additional verification. The attacker also needs to somehow have access to the browser. Then, the attacker logs in with the established backdoor. We encode this as two separate chains where the first involves the initial account linkage and the second is the remote login. Overall, this is intended to simulate the account binding attack class.

Attack on the SP: The SP is attacked such that a user account becomes erroneously bound with an account on the IdP. The target of this attack is the SP, as that is the user interface which is interacted with during the attack. However, the intrusion is said to occur with the user account as the problem is that the attacker can now access that account through a backdoor using the attacker IdP account in the SP settings. The user must be logged on.

1. *Failure.* The failure is in terms of integrity because an action is provoked on the user account that is not from the user themselves.
2. *Error.* The error is in the rule being broken about an unauthorised action taking place.
3. *Intrusion.* The intrusion is that the attacker can login to the user account.

4. *Attack.* The attack is the process of the attacker linking the IdP account to the SP account.

5. *Vulnerability and Enabling Intrusion.* Vulnerabilities occur on the SP and IdP for this attack to be possible. The SP has to have the linking of the accounts option available in the user interface. The IdP does not request that the attacker has to re-authenticate themselves, which would stop the attack because the attacker does not necessarily know user credentials. In addition, the browser also has to be accessible to the attacker somehow by the user leaving their computer unattended with the SP account logged on. We model this vulnerability broadly by simply stating that the app is insecure (because this is a Spotify application not a traditional browser). In reality, the situation might be more complex and involve office security policies. To eliminate the complexity of this, we broadly model the issue with the browser being accessible to the attacker as an insecure app.

Overall FIM System Failure: The attacker can use a backdoor on the user account to login remotely. This is an attack on the overall FIM system because the goal is to compromise a user account and the protection of the information on the user account is a security goal of the FIM system.

1. *Failure.* The failure is in terms of CIA because the attacker is able to login remotely to a user account. Logging in to a user account has far reaching impact on CIA because the attacker can carry out varied operations such as viewing user account information or attempting to disable the account.

2. *Error.* The error is that an account has been compromised.

3. *Intrusion.* The intrusion is that there exists a login of the legitimate user account by the attacker.

4. *Attack.* The attack is the process of the attacker logging in with the user account.

5. *Vulnerability and Enabling Intrusion.* The intrusion that enables this attack to be possible is that the attacker has setup a backdoor into the SP account using the attacker IdP account. An additional vulnerability that can be considered at this point is how the SP does not have any systems in place to detect erroneous logins. For instance, the geographic location of the login attempt.

In a similar way in which rules are derived from a knowledge encoding in section 4.5.7, rules are derived from the encoding for the account binding. These rules can be seen in appendix tab. B.6 and tab. B.7.

4.8.4. System Model

The system model represents the scenario in which a user has left an account logged in but is not physically at the computer. This is broadly modelled as an insecure browser vulnerability. The Spotify SP has two vulnerabilities: one that enables the account linkage option and another that

Attack	Vulnerabilities and Enabling Intrusions	Resulting Intrusions	Errors	Failures	Target System	Systems Involved
Account Linkage on user account	linkage option on SP, no forced re-authentication from IdP, insecure app	account backdoor on SP account	Unauthorised Action	Integrity	SP	Attacker IdP account, User Login to SP
Login backdoor for the user account	User account linked to attacker account, SP does not check for erroneous logins	Attacker login to User Account	Account Compromise	CIA	FIM System	None

Table 4.7 Knowledge Encoding of Account Linkage

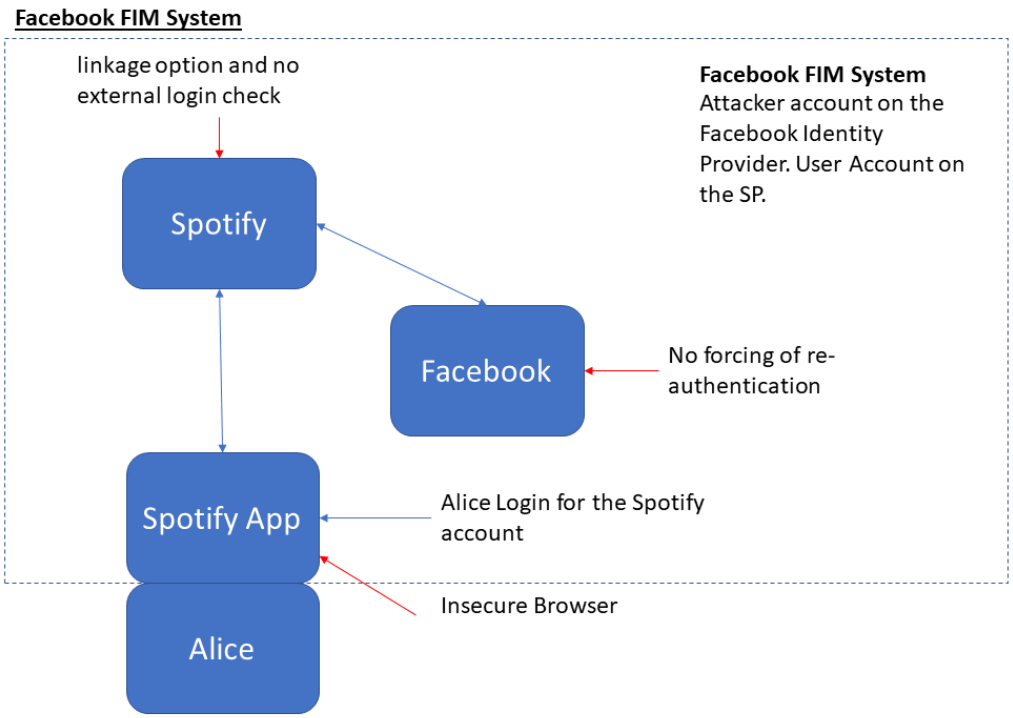

Figure 4.12 System Model Overview with Appended Vulnerabilities

does not check for suspicious login activity. The IdP has a vulnerability in which it does not force the SP to authenticate again when attempting the account linkage. Note that we could also use a capability term to describe the attacker as having physical access to the user device as we describe in 4.7.1. To be consistent with our previous example concerning OAuth and to not conflate the model, we do not include capabilities for this model. The system model is depicted in fig. 4.12. The precise system model specified in logical terms can be seen in appendix tab. B.2.

4.8.5. Traces

We employ the set of rules on the system model. All possible failures can be queried and two distinct failures are found. The first failure occurs on the SP and involves the SP when the account is linked. This can be seen in fig. 4.13. The second failure escalates from 4.13 and can be seen in 4.14.

4.9. Discussion

4.9.1. Attack Detection in FIM via Modelling Escalation

For this work we aim to model attack escalation in FIM as per RQ2. The idea is that by modelling attack escalation we can detect a primary attack that is possible through considering how a secondary attack could be the cause for the primary attack. This work introduces a general approach for how attacks can be detected on a system model given the presence of some other attack. Using this approach, the possibility of escalating attacks can be evaluated on a

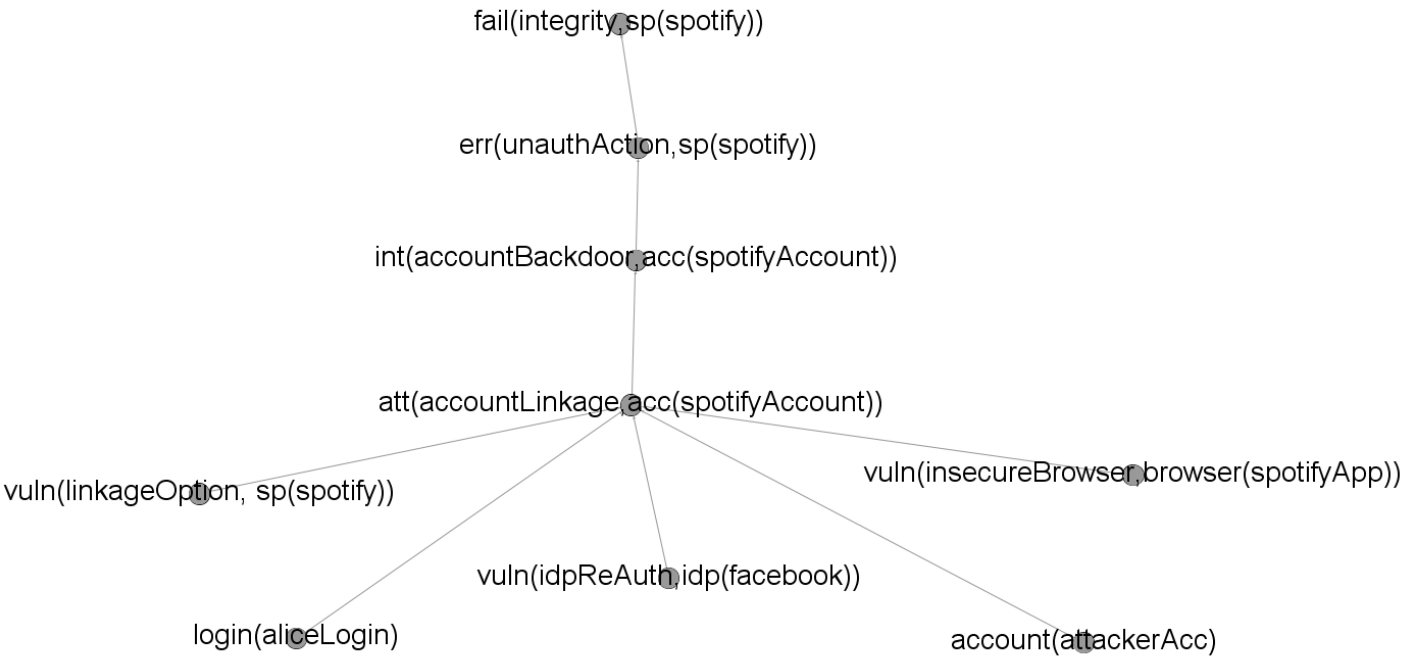

Figure 4.13 Attack Trace on SP with Account Linkage using a simplified syntax described in subsection 4.5.5.

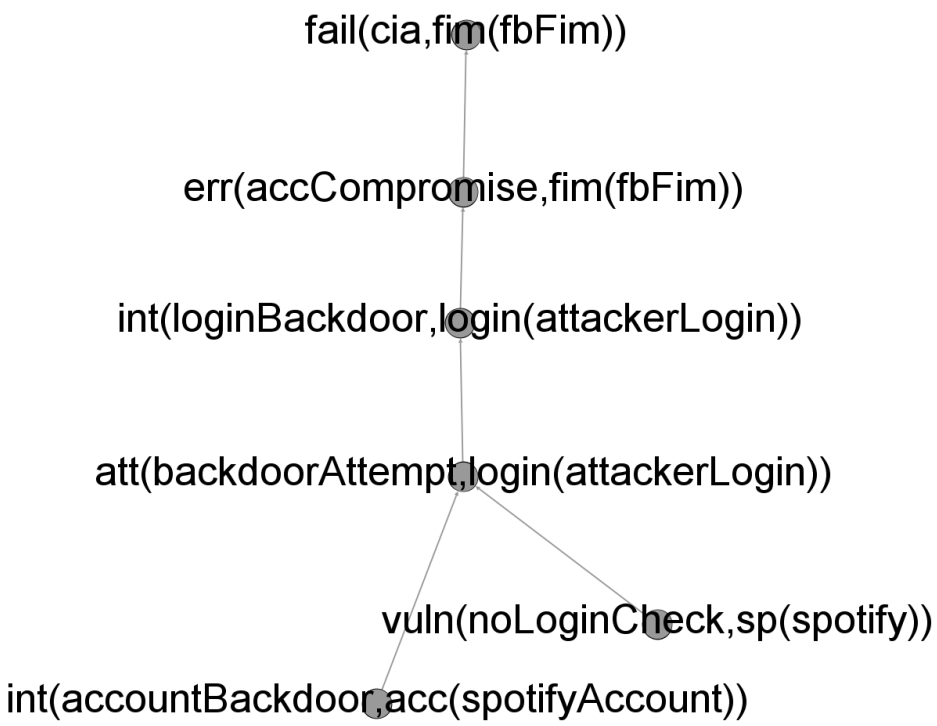

Figure 4.14 Attack Trace on FIM Overall with Account Linkage using a simplified syntax described in subsection 4.5.5. This attack trace escalates from 4.13.

system model and the causality of the attack considered through attack traces. To demonstrate this, we consider an example attack suite from the literature as specified by Sun and Beznosov (2012). In addition, we consider an example of a novel attack introduced as part of this work. This shows that modelling escalating attacks is a suitable method for evaluating the security of a system beyond just modelling the functionality.

This approach could possibly be applied in a way to help detect entirely new methods of compromising a system. The approach relies on encoded knowledge, so it is unlikely that entirely new attack classes can be found, but ways in which the attack classes can be applied could be discovered. For example, consider the account linkage attack introduced as part of this work. It could occur to an individual researcher that a user SP account bound to an attacker IdP account could be compromised. The exact way in which this account can be linked, in this case physical access to the user's device, could be discovered by another researcher and then the rulesets interact to discover the possibility of escalation for this attack.

The problem with this is that labels are used to represent conditions, and without the same label, the connection between the two issues would not be made. These labels are not arbitrary as they represent the semantics of an attack. The label connecting these two issues needs to be realised. If one of the researchers is exploring the ruleset and realises the possible connection between these two issues and changes the ruleset so the labels correlate, then the link between these two attacks can be established. In this hypothetical case, the overall approach has assisted in discovering the new attack. Tools could be used to help assist this process, like a searchable ruleset to try and find correlations, but this is beyond the scope of this work.

Alternatively, more general conditions for escalation could be considered. Instead of the specific intrusion being used as a condition for another attack, the more general failure states could be used. The limitations of this approach is that false positives could occur. For instance, either a malicious insider or a malicious script on a SP could cause the same kind of failure but in practice what is possible for the attacker to do using each of these attacks might be different. Another alternative might be to categorise intrusions into more abstract states. This is something that would help attack detection as rulesets get more complex, but would complicate our case studies, so in favour of avoiding convolution of the case studies further, this idea is left to those who would consider adapting this work for larger scale modelling.

4.9.2. Escalation Via An Attack Orientated Approach Over an Operational Approach

In this work an approach is taken in which the primary consideration in the security modelling is attack escalation: how attacks lead to other attacks. The actual operation of a system is largely ignored with only reference to some core concepts like accounts and logins. This approach to some extent goes against the general outline of what a model in security should consist of according to Bau and Mitchell (2011), who puts forward the idea that a system model should consist of operational semantics.

The benefit of this approach is that what we already know to be possible can be encoded without having to define the precise operational semantics of what caused the issue in the first place. We avoid specifying the same operational semantics just to come to the same conclusion as proposed by others. Using this approach, we reduce the complexity of introducing attacks as proposed by others into this security model. This way, we are able to reuse large amounts of information produced by others from existing literature without having to model the specifics of how that information was produced.

The trade-off made in this approach is that new conclusions about attacks on FIM that could, from the collation of all this knowledge, be missed. Universal operational semantics could be expressed in FIM using approaches like Universally Composable Security as introduced by Canetti (2001). In this work we focus on how knowledge can be encoded based on what we know about attacks. For example, we extend the attack suite described by Sun and Beznosov (2012) with deeper consideration of how the browser can be involved. A model checking approach did not come to the conclusion that the browser is involved. Conversely, encoding operational semantics more formally would probably leave more room for a model checker to figure out new knowledge. This is evidenced by the new attacks that are found on FIM through model checking solutions such as work done by Armando et al. (2008) which finds a new attack on the Google implementation of SAML.

However, this approach has been shown to be incomplete in the past. Formal descriptions, such as the work from Chari et al. (2011), state that OAuth is insecure while Sun and Beznosov (2012) proves that in practice OAuth is not secure. This was in large part a motivation for this thesis as it was considered that only modelling the functionality of FIM protocols has failed in the past. A universal composition of FIM operational semantics would be a worthy goal which could detect new attacks as well as prove the existence of other attacks known to be possible. However, it remains an open question. How exactly the various topologies, protocols, and vulnerabilities occurring both at the implementation and protocol level can be considered as part of one operational model is not known. Future work into model checking solutions leveraging the operational semantics of FIM could pave the way for finding new escalating attacks on FIM.

4.9.3. Scalability

This thesis overall does not focus on the scalability of attack detection in FIM. However, it is useful to consider this mindset to show how our approach can be adapted to solve different kinds of challenges in attack detection. We do this to consider some problems with our approach that are not obvious until scalability is considered. Scalability is an issue with attack detection approaches as is pointed out by Ou et al. (2006). We do not wish to convolute the original approach with scalability strongly in mind, as scalability is not strictly necessary to achieve our primary research questions. Scalability still remains an interesting problem, so let us consider some problems with our approach in terms of scalability.

False Positives in Logical Approach: To some extent, we do consider false positives in the logical rules we introduce for the OAuth example detailed in section 4.5. For instance, an XSS attack is only possible if the browser is connected to the website hosting the malicious script. In practice, there is only one browser in the system model so this connection is not strictly necessary. However, if the system model were to consider two browsers (perhaps in an effort to demonstrate how attacks for one user might influence another user), then it would be necessary to state that this connection is required. If the connection was not required, then any browser in the system model would be attacked by XSS which is not necessarily the case as XSS requires the connection. As a system model increases in complexity, more care would have to be paid to what it means for an attack to be possible, and if any unintentional false positives exist. What Prolog does well is find any solution for a given problem. Sometimes though, more solutions are found than are intended and this could be the case with our approach unless careful consideration is given to rules constraining their applicability.

For a given system model, the testing for false positives could occur. Does the detection of a certain attack reflect the real world possibility of that attack existing? In this case, the real world possibility of attacks existing is provided by empirical investigation from Sun and Beznosov (2012). If the model is not consistent with the real world, then what part in the system model or the rules oriented around that system model causes this inconsistency to exist. In our experience of using this modelling technique, it is an iterative process. We observed situations where an attack that should not be possible, did occur, usually because the system model is not adequately described or there is some semantic mistake with a rule. For instance, an attack happening for a certain user even if that user does not have an account. In this case, we can consider how the rule should be changed so that an account is required for an attack.

State Space Explosion: It is not clear how well this approach will handle very large systems which can be attacked in many ways requiring many rules. In addition to Ou et al. (2006); Vigo et al. (2014) considers how automated approaches to threat detection (specifically attack trees), often suffers from state space explosion. Specifically, our approach uses Prolog. Leuschel and Butler (2003) use Prolog for the model checking of the B programming language. Scaling was not a primary focus for Leuschel and Butler, but they claim that testing was done for reasonably large systems and conjecture that their approach could handle even larger systems. Blanchet (2001) use Prolog for the verification of cryptographic protocols and show how Prolog can avoid state explosion through unification. However, the speed of the approach considered also relies on having just a few rules which might not be realistic for a large complex model of a FIM system.

For clarity, we did not explore the extent of the issue with state space explosion in this work. No model checker was applied to our model to evaluate the extent of this problem. The effectiveness of this model in relation to state space explosion remains an open question.

Generality of Rules: In the case studies given we explicitly state intrusion states which could limit the logical programming approach in organically realising new possibilities for attacks. This issue is already discussed in subsection 4.9.1 but this also relates to scalability. One solution is to provide intermediary rules for how failures of systems are perceived by another system. For instance if a SP is compromised in some way, the IdP might not be interested in exactly how that SP is compromised, just that it has been compromised; then have some intermediary rules that go from a specific problem with the SP to something more general.

4.9.4. Validation

In this work a logical analysis approach is introduced based on the MAFTIA framework. In this subsection, several arguments are made as to why this is a valid approach. We base these arguments on validation considerations outlined in 4.2.

Validity of our General Security Model: Bau and Mitchell (2011) puts forward several properties of a security model. These properties include: system model, threat model, and security properties. MAFTIA allows for the consideration of a system model in terms of a systems of systems approach. In subsection 4.2.1 the MAFTIA model in general is discussed in terms of how it fits into these three security properties. Here we discuss how our adaption of the MAFTIA model fits into these security properties.

As already outlined in subsection 4.9.2, the system model we use is not a model based on the functional operation of the system but is specified using a systems of systems model based on interactions which conforms to the MAFTIA model. The benefits and drawbacks of this approach are mainly discussed in that subsection. Reiterating a key point here, this approach allows us to model attacks that are known to be possible without necessarily specifying the exact operating conditions just to come to the same conclusions. This allows us to easily encode attacks which in turn could escalate to other attacks.

The threat model is described as the rules that model the necessary conditions for attacks to be possible. MAFTIA can consider a threat model by breaking down security issues into attacks, vulnerabilities, and intrusions. However, Bau and Mitchell (2011) does consider that the exact attacker capabilities should be stated specifically in terms of the attacker's available computer resources. MAFTIA does not consider specific attacker resources. In the extension to our approach, specified in subsection 4.7.1, we consider how attacker capabilities can be easily added to a model.

Security properties in MAFTIA are expressed via security goals which are typically stated as CIA requirements. In our approach, some systems do not have security rules/goals as the systems are more used to describe some elementary condition in the system. For example, accounts do not have security rules/goals. The reason being for this is that failures with an account can be captured at the general FIM level. Having security rules/goals for every system

would not necessarily be a wrong approach. However, we chose to use security rules/goals for the systems that are the key participants like the Browser, SP, IdP, and FIM system as a whole. The benefit to this approach is that we reduce the complexity of the overall model and can focus on these key participants.

In general we find that our model mostly conforms to the properties of a security model as proposed by Bau and Mitchell (2011) with some reasoned deviation.

Correctness: We consider the correctness of the case studies introduced as part of this thesis. The first is a model of an attack suite described initially by Sun and Beznosov (2012). We know this attack to be possible as it has been proven to be possible under certain conditions by an empirical study. We present a model of the conditions necessary for the attack and the different stages of the attack expressed via an attack trace. In section 3.9 we introduce our full interpretation and adaptation of the attack. This chapter, namely section 4.6, reproduces the process as it is originally described. We make the addition of how the browser can also contribute to the failure, but we know from sources that investigate the specific techniques used in attacks like CSRF (OWASP (2019a)) that the browser is involved in the overall attack as the browser ultimately causes the execution of the malicious code. Other than including the browser, we believe that we have faithfully reproduced the issue of accounts being compromised as proposed by Sun and Beznosov.

We also consider the correctness of the model by modelling an attack which is introduced as part of this work. This attack is broadly described as an account linkage attack on Facebook and the details of this are in section 4.8. The qualifiers of this new attack are discussed in subsection 4.8.1, but in short we seek to qualify the attack by showing how the attack is possible by providing evidence of the implementation exploit, showing how we can describe the attack in our taxonomy for describing attacks in FIM, and showing how the threat model is a valid threat model in security literature. We represent traces that describe the attack in 4.8.5. The idea is that these traces represent the sequential steps that the attacker takes as described in sub-section 4.8.1.

Completeness: We provide a comprehensive survey of attacks in FIM in chapter 3. When considering the attacks in this survey, we found no issue that could not be represented using our approach initially detailed in 4.5 and expanded on as described in section 4.7. In general, we find that when the initial approach is not enough to fully describe an issue, we can consider the extensions. Sometimes, more than one of the extensions need to be applied. For example, when describing an issue described by Kormann and Rubin (2000) which involves a DNS issue, we should use a new system extension and it could also be argued that a capability to undermine the DNS should be included.

We also test the completeness of the model by introducing a new attack to Facebook. The intention is that the approach can be adapted for new knowledge. It is not clear if the overall approach can capture every new attack that could be discovered. However, we try to build a case that the approach is extendable and adaptable to different scenarios and if it is the case that something new is discovered that cannot be properly represented using this approach the approach could be extended.

4.10. Summary

In this chapter a security modelling technique is considered for FIM. This modelling technique broadly follows the three properties of a security model introduced by Bau and Mitchell (2011) which are: the system model, the security properties, and the threat model. The overall approach is inspired by MAFTIA (Powell and Stroud (2003)), as MAFTIA is a general conceptualisation for viewing a system as a system of systems and considering how security failures can escalate in a system. The system model is described using logical terms which represent different aspects of a FIM system. Security properties are described as security rules and goals as proposed by MAFTIA on the different systems in FIM. The threat model uses a logical programming approach to consider how security properties can be violated on a given FIM system model.

Through this security model, we seek to show how attacks can escalate in FIM. Escalation is useful for showing how the security properties of a system can be violated through other attacks rather than just considering the functionality of the system. We model this by considering how intrusions for one security issue can be the condition for another attack on a FIM system. The causality of an attack can be viewed as an attack trace which models the different stages in a security issue.

We show how the modelling technique can be used to model an escalating attack in the existing literature. Specifically, an attack suite described by Sun and Beznosov (2012) is modelled. The basis for the overall approach is described in section 4.5, including how systems in FIM can be modelled logically in a systems of systems approach that focuses on interactions between those systems. How logical rules are used to represent a threat model based on existing knowledge is presented. Traces are then built by querying the models security properties to observe the causality of a failure and this is shown in section 4.6.

However, there are many aspects to security modelling in FIM that cannot be covered using just the example introduced by Sun and Beznosov (2012). For example, protocol level issues, issues that involve entirely different systems like the DNS, and issues that need to be modelled with respect to some attacker capability. We use the survey in chapter 3 as a basis for considering other problems in FIM. We find that there are specific problems which could not be modelled using only our existing approach so we expand on the approach to consider other problems in section 4.7. The aim is to validate that the approach is applicable to more than one specific example.

Finally, we consider how an original security issue that we discover on Facebook and Spotify can be modelled using our approach in section 4.8. This is intended to validate our approach by demonstrating how new knowledge can be used to model escalation of attacks in FIM. Also, we show how escalation can demonstrate how protocols which might be secure under normal means can be undermined. It could be the case that Facebook is secure from an attacker logging in remotely, but this model shows that under certain conditions this remote login is possible through first performing a different attack.

Chapter 5. Attack Tree Assisted Security Analysis of FIM

Abstract

We present attack trees as a useful abstraction for considering different security threats in a FIM system. The problem is that while attack trees are easy to read and understand, that is not necessarily the case for the creation of attack trees by a stakeholder who wants to analyse the risk of a particular system model. We underline the basis of a technique for automatic attack tree generation in FIM. We consider the ways an attack tree can assist analysis of FIM systems.

5.1. Introduction

In chapter 4 an approach for considering security issues on a FIM system in terms of the Malicious and Accidental Fault Tolerance (MAFTIA) is introduced. The causality of issues is expressed as an attack trace which is a sequence of steps the attacker must go through to reach some final goal in the system exploitation. We realise that this way of considering security issues on FIM can be likened to research into attack trees. Attack trees are a useful analysis tool, which provides motivation for being able to consider FIM systems with attack tree analysis.

Attack trees were first introduced by Schneier (1999). Considerable research effort has gone into attack tree research since then. Formal models for attack trees have been introduced by Mauw and Oostdijk (2005), and these models have been further refined including a model for attack-defence trees as considered by Kordy et al. (2010). The applicability of attack trees has been examined in case studies, such as a case study on bank ATM machines as proposed by Fraile et al. (2016).

However, the synthesis of attack trees is considered to be error-prone and often impractical as systems get larger according to both Vigo et al. (2014) and Gadyatskaya et al. (2017), who focus on automatic generation of attack trees. In this work, we already have a basis for the automatic generation of attack trees with attack traces introduced in chapter 4. The first step to considering attack trees for FIM systems is to ensure attack traces can be leveraged for automatic attack tree generation.

Considerable work has gone into analysing attack trees. In this chapter we focus on new analysis techniques for attack trees rather than considering how existing analysis techniques can be applied to FIM systems. However, we note that existing analysis techniques should be compatible with our approach. There are a number of problems with FIM systems that can be considered by using attack trees. For instance, the issue of presenting problems with a FIM

system to a group of stakeholders. The stakeholders are the main actors in a FIM system like the SP and the IdP. However, some of the issues with a FIM system might occur externally and outside of the stakeholders control. For example, a XSS attack staged on a SP not within the FIM system could contribute towards an issue. How can we use attack trees to present the issues to the FIM system in such a way that only the issues fixable by the stakeholders are considered? Alternatively, there could be issues that affect multiple stakeholders. For instance, a prevalent risk of using FIM systems is that the IdP account is compromised, potentially leading to every SP stakeholder being impacted in addition to the IdP stakeholder. Can we design an approach to consider how attack trees are used to model the impact to multiple stakeholders?

We emphasise that the novelty of this approach and the contribution to the wider attack tree field is a model of attributing individual elements of attack trees to the actual stakeholders who are invested in the system. The idea of attributing attack trees to invested stakeholders is at least informally discussed; for example by Fraile et al. (2016), but to the best of our knowledge this is the first model for that idea. The contribution to FIM is that this is the first attempt on generating attack trees for FIM systems.

5.1.1. Contribution to the field

We outline RQ3 as the primary research question in this chapter.

RQ3: Automatic Attack Tree Generation and Manual Analysis Techniques. In what ways can attack trees automatically generated by our attack tree synthesis method for FIM be analysed?

5.2. Fundamentals

The main background for attack trees is considered in section 2.5, including a mathematical formal definition. In this section we deal more with the motivation of applying attack trees to FIM. In chapter 4, a method for finding the reachability of certain attack states from other attack states is considered. In particular, we consider attack traces, which are a tree-like data structure for which every node corresponds to some attacker manipulation on the system model, or an additional system that is required for the attack to be possible. An attack trace could be analysed directly. However, there are existing analysis structures and techniques available in the literature. One such structure is attack trees which were originally proposed by Schneier (1999).

Attack traces align well with attack tree research. The major difference is that certain states are possible through alternative paths or disjunction while attack traces only allow for a conjunctive approach. To deploy attack traces into the widely researched field of attack trees, we only have to consider where traces could be combined together to form an alternative path to some ultimate failure state.

What is attractive about attack trees is how they are easily analysable as Schneier (1999) demonstrates several intuitive ways in which attack trees can be analysed including cost analysis. The drawback of attack trees according to Vigo et al. (2014) is that the construction of attack trees can be error-prone. However, since the logical approach in chapter 4 has already done most of the work of the attack tree construction, a good option for this work is to consider the analysis of FIM systems using attack trees.

One technical detail that is not made clear by Schneier (1999) is how nodes with multiple kinds of connection are considered. The two different kinds of connection being logical and/or. In the examples given by Schneier, there is only ever one kind of connection belonging to a single node. How do we consider situations in which multiple kinds of connection are considered? Specifically, two alternatives (a logical or kind of connection) where each alternative has multiple requirements (a logical and kind of connection). Mauw and Oostdijk (2005) answers this question in work which focuses on the formalisation of attack trees. The idea of a *bundle* is conceived. We cover the idea of a bundle mainly in 2.5, but to recap, a bundle is an entire set of nodes which are all required as part of a logical and type connection. Nodes can have as children multiple bundles where each bundle corresponds to a logical or type connection. Thus, we can represent situations where multiple complex alternatives exist for a particular node in an attack tree. In addition, implementing trees in this way should make our work compatible with existing attack tree research.

Attack Tree Compatibility Validation: The attack trees produced should be compatible with the formal attack tree approach provided by Mauw and Oostdijk (2005).

5.3. Related Work

Automatic attack tree synthesis is a topic of much research. Attack trees are easily understood, but not so easy to create. Fraile et al. (2016) demonstrates the difficulty of manually creating attack trees even with a team of experts and an established knowledge base on ATMs. Others report on the process of attack tree creation being error-prone so they implement automatic attack tree synthesis such as Vigo et al. (2014); Gadyatskaya et al. (2017); Wideł et al. (2019).

Attack trees are not the only graphical threat analysis solution. Attack graphs are a slightly different method of modelling security threats and are proposed by Phillips and Swiler (1998). Automated generation of attack graphs is something that has been studied and first introduced by Sheyner et al. (2002). Both attack graphs and attack trees are valid ways of determining threats to a system and are similar in function. Sheyner et al. provides seminal work on the generation of attack graphs. An attack graph is generated from a system model of a network. Of particular interest is that the attack graph can be parsed to produce a minimal cut set, which is the set of sets of vulnerabilities that if present, will always cause a failure. In addition, support for probabilistic modelling is provided through the use of Markov Decision Processes. However, there is no logical separation between knowledge of attacks and the system model which might

cause problems if we wanted to apply the same knowledge base of attacks to different system models or vice versa. Ingols et al. (2006) also practice automated attack graph generation but also do not consider a logically separate system model and knowledge base.

Ou et al. (2006) does seem to have a separation between the system being considered and the knowledge base for deciding possible attacks. The approach taken is largely similar to our own and Prolog is used as the logic engine. The difference is that attack graphs are used over attack trees. In addition, Ou et al. considers the wider area of computer networks in general while we focus on the specific problem domain FIM. In contrast to Ou et al. we decide to use attack trees because Prolog generates proofs in a tree like structure as is shown in chapter 4. Also, attack trees have been the subject of much research in recent years so we decide to move in that direction.

Automated generation of attack trees has been studied. Vigo et al. (2014) generate attack trees automatically given a labelled process in the π-calculus family. This approach relies on encoding both the system model and the knowledge of attacks in the structure, which could be problematic when generating attack trees based on the same knowledge base for different systems. One key benefit our approach has is the separation of the system model and the knowledge base. Gadyatskaya et al. (2017) generate attack trees with respect to a specified refinement structure. Attack trees are generated where intermediary nodes actually have semantic meaning, rather than being sequential steps in a system model. The two inputs to the technique, being the intended semantics and the refinement specification are derived from the system model. This again raises the problem of wanting to apply the same knowledge base to a different system model and vice versa. It is also worth mentioning Jhawar et al. (2018), who does not generate attack trees automatically, but does use Prolog to augment already generated attack trees with annotations indicating that the use of attack trees and Prolog is an active area of research.

Birkholz et al. (2010) outline an approach for generation of attack trees from a common vulnerabilities database. The idea is that a network is modelled with a number of components that have installed vulnerable software with a score rating. An attack graph is then generated by looking for the shortest route to a goal based on the score rating. The method proposed is a weighted physical path through system components to reach an end goal component. This approach does enjoy a logical separation between system model and knowledge base, and is useful for determining a weighted path through a network based on some score. However, the knowledge base does not allow for the encoding of dependencies, which can be used to break down an attack into steps, which is suited for modelling escalating attacks. The input is dependent on the pre-specified CVE database, which is certainly useful. However, the CVE database does not clearly define how vulnerabilities can interact with one another as for example Sun and Beznosov (2012) demonstrates with OAuth.

The analysis of attack trees has also received attention. Schneier (1999) originally outlines several methods for analysing attack trees. Since then, Kumar et al. (2015) shows how attack

analysis can be expanded automatically taking account for different attack tree constructions. Dewri et al. (2012) considers several different cost operations for attack trees but does not consider how those cost operations can be applied to different stakeholders in a system. Specifically in this thesis we focus on how attack trees can be analysed in a way that is useful to FIM focusing on how stakeholders in the FIM system can be attributed to the attack tree.

The related work for this chapter is summarised with the following. Attack trees are regarded as useful by the academic community in general which is why attack trees are the subject of much research. We have briefly outlined attack tree research as much as is relevant to this thesis, for instance focusing on automatic synthesis. For more, Kordy et al. (2014) have performed a survey on attack trees (formally, directed acyclic graphs). However, the actual creation of attack trees is still an open area of research with Gadyatskaya and Trujillo-Rasua (2017) stating several new directions and modern challenges regarding attack trees.

5.4. Attack Tree Synthesis

5.4.1. Synthesising Attack Trees from Existing Traces

In this work, specifically the focus of chapter 4, we already have a method for generating attack traces from a Prolog trace statement. These traces can be consolidated to create attack trees. Specifically, for a trace we find the *point of divergence* from the tree. This is the location on an attack tree and a trace, in which an alternative for an attack tree is discovered from a trace and added to the tree. A graphic overview of an example is shown in fig. 5.1. There are a number of preconditions that are necessary for this approach:-

- *Consistency of ruleset and system model:* Assume that every trace is generated from the same ruleset and system model.
- *All traces synthesised from the same Prolog query:* Attack traces need to be synthesised from the same instance of a Prolog query. In other words, we are dealing with generating a single attack tree from one query rather than an attack forest from different queries.
- *No variables in the Prolog query:* No variables are used for these queries to simplify the overall approach. Not using variables is not a severe limitation as Prolog can first be used to identify the possibilities and then specific queries not using variables can be used.
- *The list of traces in non-empty:* We create an attack tree from at least one trace.

Summary of Algorithm for Building An Attack Tree: We have a list of traces that conform to the preconditions specified above. The idea is to consider these traces in one attack tree structure. For the first trace in a list, this is simple as this can be represented as an attack tree without any disjunctions. For subsequent traces, the *point of divergence* needs to be found which is where an alternative path through the tree is discovered. An outline of the algorithm written

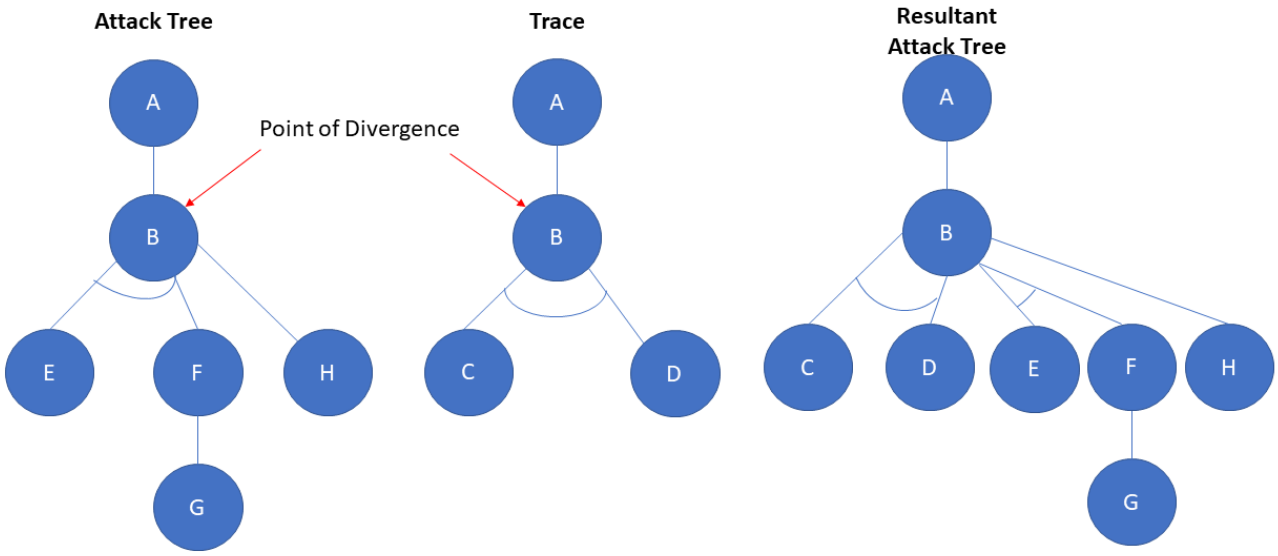

Figure 5.1 Example Merge of an attack tree with a trace to make a new attack tree.

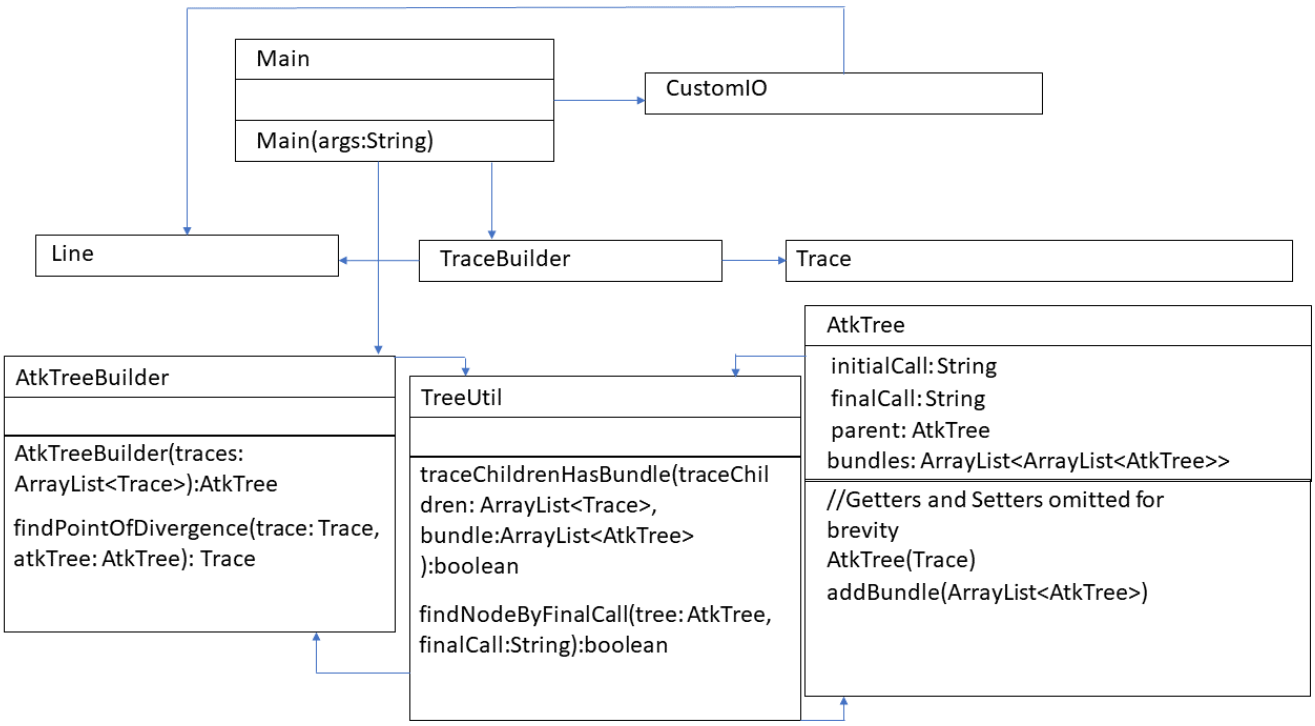

Figure 5.2 UML Diagram of Java Implementation of AtkTreeBuilder. Previously defined classes omitted for conciseness and can be found in 4.4.

from an object orientated programming point of view can be seen in algorithm 2 with reference to the UML diagram in fig. 5.2.

algorithm : AtkTreeBuilder
input : A list consisting of trace objects *traces*
output : An attack tree object atkTree

```
Trace nextTrace := traces.next();
atkTree := new AtkTree(nextTrace);
Trace pointOfDivergence;
for each trace in traces do /* (1)                                          */
    pointOfDivergence := findPointOfDivergence(trace,atkTree);
    List<Trace> pChildren := pointOfDivergence.getChildren();
    List<AtkTree> newAtkTreeBundle := new List();
    AtkTree pointofDivergenceInTree :=
     TreeUtil.findNodeByFinalCall(atkTree,pointOfDivergence.getFinalCall());
    for each pChild in pChildren do /* (2)                                  */
        AtkTree nextAtkTree = new AtkTree(pChild);
        nextAtkTree.setParent(pointofDivergenceInTree);
        newAtkTreeBundle.add(nextAttackTree);
    end
    pointOfDivergenceInTree.addBundle(newAtkTreeBundle);
end
return : atkTree
```

Algorithm 2: Attack Traces to Attack Trees . Comments in blue and numbers in parenthesis are annotations to be referred to in the discussion of time complexity.

Summary of Algorithm for Finding The Point of Divergence: We need to find the point in a trace where the trace diverges from what is already on the attack tree. One way of doing this is to take the path down an attack tree dictated by the attack trace, and then as soon as we can no longer take that path because the attack trace differs, then the divergence must be found. An outline of the algorithm written from an object orientated programming point of view can be seen in algorithm 3 with reference to the UML diagram in fig. 5.2.

Computational Complexity of Finding the Point of Divergence (alg. 3): The attack tree building algorithm relies on finding the point of divergence. Thus, let us consider the complexity of finding the point of divergence first. We consider Cormen et al. (2009) as a guide for this. An input trace is considered to have a worst case size of m. The algorithm also involves the creation of attack trees, which we assume as the worst size of l.

Annotation *(1)* refers to a constant time operation where if a trace does not have any children, a return operation is performed. Outside of the if statement, there are a number of get operations which return a reference to a field within the class, which are also constant time operations. Therefore, the time complexity for *(1)* is $O(1)$.

```
algorithm : findPointOfDivergence
input     : A Trace object, An AtkTree object
output    : A Trace object

//This path is exhausted so return null
if !trace.hasChildren() then /* (1)                                    */
  | return     : null
//Is there a bundle in the attack tree that matches the trace children?
List<Trace> traceChildren := trace.getChildren();
List<List<AtkTree» bundles := atkTree.getBundles();
List<AtkTree> tracePath := null;
for each bundle in bundles do /* (2)                                   */
  | if TreeUtil.traceChildrenHasBundle(traceChildren,bundle) then
  | | tracePath := bundle;
end
//Recursive call of each atkTree in the bundle
if tracePath != null then /* (3)                                       */
  | for each nextAtkTree in tracePath do
  | | for each nextTrace in traceChildren do
  | | | if nextAtkTree.getFinalCall() == nextTrace.getFinalCall() then
  | | | | Trace returnDiff := findPointOfDivergence(nextTrace,nextAtkTree);
  | | | | if returnDiff != null then
  | | | | | return     : returnDiff
  | | end
  | end
//If the trace has no children and there is no path through the attack tree
  identical to the trace children we have found the point of divergence
return     : trace
```

Algorithm 3: Finding the Point of Divergence in An Attack Trace Compared to An Attack Tree

Annotation *(2)* refers to a for loop which iterates over each bundle in a tree. In the worst case, every child node for an attack tree can be its own bundle, and thus this is $O(l)$. The TreeUtil function then takes each of these bundles and sees if they exist within one level of the trace. The worst case trace size would be $O(m)$, assuming each node in the trace is on that level, and the algorithm must check to see if each of these trace nodes in a particular bundle. The complexity for *(2)* is $O(m+l)$.

Annotation *(3)* most notably refers to several for loops which iterate through the bundle and trace path. In essence, for each attack tree in the bundle, we recursively call the divergence algorithm again. In the worst case, the size of the attack tree bundle is $O(m)$ and the size of the trace children is $O(l)$ which means that the algorithm is recursively called $O(l)$ times. The worst case for *(3)* is therefore $O(m+l)$.

Let us now consider the big picture for this algorithm. Annotation *(1)* is the smallest term, and the overall complexity for this algorithm is clearly going to be dependent on the other two annotations. Both Annotation *(2)* and *(3)* have a complexity of $O(m+l)$. Of note, Annotation *(3)* makes a recursive call. However, the worst case for each individual call is only ever going to

be $O(m+l)$. Also of note, when we assume a tree data structure with one very big level that would lead us to our worst case, that tree cannot also be deep, meaning that each recursive call would not lead to another recursive call. The time complexity for this algorithm is $O(m+l)$.

Computational Complexity of Attack Tree Builder (alg. 2): Note that this algorithm is dependent on 3. Input into the algorithm is a list of traces n. Also input is a trace, which we assume as the worst case size of m. Another important dependent is an attack tree, of which the worst case size is l.

Annotation *(1)* mostly refers to $O(1)$ operations. For example, getting certain references and instantiating new objects. However, there is also a loop on the list of traces which is a $O(n)$ operation. The algorithm for finding the point of divergence is also called here. For this algorithm let us assume the worst case trace object and worst case tree object (which is initialised from an existing trace). We could consider a trace where every node is a child of the root node, with the exception of the root node. This will result in the execution of finding the point of divergence being $O(m+l)$. We also must consider that the TreeUtil function to find a node by the final call in the worst case will iterate through the entire attack tree initialised by the trace trying to find the node identified as the point of divergence resulting in a $O(l)$ operation. However, since there is already a $O(l)$ operation this is insignificant result in *(1)* being $O(n+m+l)$.

Annotation *(2)* is embedded within *(1)* and starts with iteration over the children in the trace, which assuming the worst case where each node in the trace is being iterated over, is $O(m)$. The other operations for *(2)* are constant time operations where variables are set. Adding a bundle to an attack tree is also a constant time operation as the bundles are not sorted. The time complexity for *(2)* is therefore $O(m)$.

The big picture for this algorithm is that although *(1)* and *(2)* do not by themselves have any quadratic elements, since *(2)* is embedded within *(1)*, the time complexity for processing the worst case trace statement becomes quadratic. The overall time complexity for the algorithm is $O(n+m^2+l)$. The quadratic element has to be considered for the scalability of these algorithms, as clearly this will be a limiting factor. In addition, we must also consider that we also need the attack trace list, which is generated from alg. 1 and is $O(n+m^2)$.

Compared to other literature in the area, notably Ou et al. (2006), who generates attack graphs in the with complexity $O(n^2 log(n))$, our method seems generally inline. We also require attack traces for our algorithm to function, which is a quadratic operation presented in alg. 1. Also, we do not take into account the complexity of Prolog solving the logical problem. Ou et al. do strongly consider the performance and scalability of the algorithm, and have also provided rigorous proofs on the matter. Since this thesis does not focus strongly on performance and scalability, it is difficult to say for certain how our approach compares without taking a strong focus on this and adopting the same level of rigor.

5.4.2. *Considering Strict Attack Trees By Stripping Systems*

Formal approaches to attack tree synthesis often consider the types of nodes that can appear on an attack tree. For instance, Mauw and Oostdijk (2005) considers attack nodes that are explicitly concerned with attacker actions. Kordy et al. (2010) considers defence nodes in addition to attacker nodes. The attack trees that we introduce can include nodes which represent the presence of certain systems in attacks. For instance, a browser might need to be connected to some SP for an attack to be possible. However, these systems can be stripped from the tree to make an attack tree consisting of only attack nodes. The problem is: is information ever lost from the tree if system nodes are stripped? Consider for instance if other attacks caused that system to exist.

Fig. 4.2 presents an overview for how rules interact with each other to form an attack tree. Systems are only ever used as facts in conjunction with some vulnerability. Systems that are created as part of the course of an attack are represented as an intrusion which would not be removed in the stripping of system nodes. Since system nodes are only ever facts in Prolog, and facts are leaves in an attack tree, no additional information is lost from the tree in the stripping of system nodes. This is a useful property of the attack trees we specify. If there is a need to have these attack trees conform to existing literature specifications, that only consider attack nodes (such as Mauw and Oostdijk (2005)), this can be done without losing information except for the information contained in the system nodes themselves.

5.4.3. *Other Attack Tree Synthesis Methods*

The method outlined is not the only method that can be used to generate attack trees using a logical approach:-

- *From Prolog trace:* Consider how attack traces are generated in chapter 4. The Prolog trace output is used to generate the attack traces. This Prolog trace could also be used to generate an attack tree rather than using the attack trace as an intermediary structure.
- *Directly from Prolog rules:* Rules can be expanded so that a goal also has a variable which consists of the path taken so far. Briefly, a goal x could be represented $t(x, path)$ and path consists of all of the conditions that allowed for that goal to be true.

Ultimately, we chose to use the outlined method, because we already have an intermediary trace structure (see 4.6), and an attack tree can be synthesised by utilising this structure. In the literature, there is also (as already stated in section 5.3) Ou et al. (2006) who, to the best of our knowledge, is the closest comparison to our Prolog approach of generating attack trees. The difference is, Ou et al. generate attack graphs while we focus on attack trees. Ou et al. seems to use the Prolog trace to generate the attack graph, while we use an intermediary data structure called an attack trace. The output of our approach is the attack tree from the Prolog rules and FIM system model, while Ou et al. outputs an attack graph.

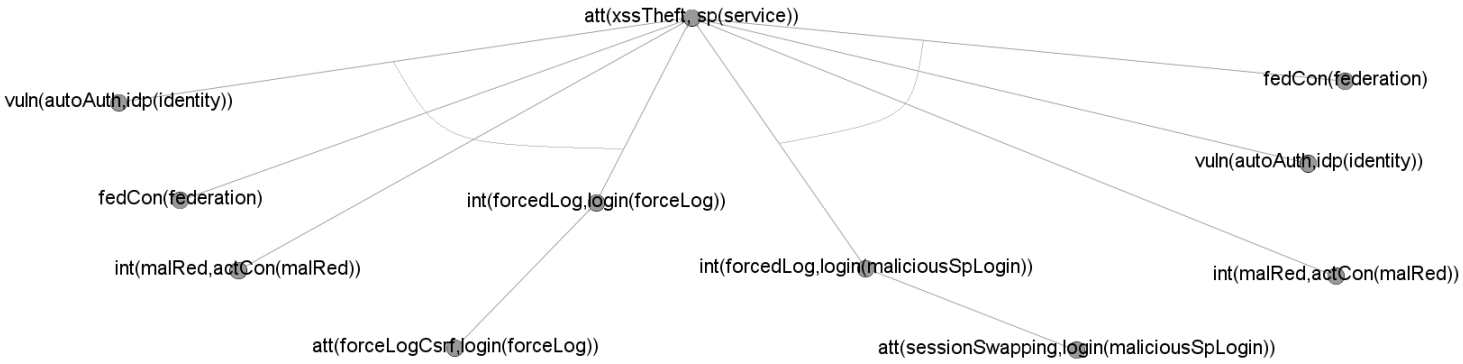

Figure 5.3 Subtree of Overall attack in OAuth adapted from Sun and Beznosov (2012) showing the different paths to a forced login via session swapping and CSRF. Curved lines denote conjunction of nodes into bundles.

5.4.4. Application of Synthesis Methods to Case Studies

In chapter 4, two different examples are considered for security modelling in FIM. The first is as adaption of the attack suite proposed by Sun and Beznosov (2012). The second revolves around a security issue proposed as part of this thesis involving an attack on accounts on Facebook through an account linkage backdoor. The attack tree specification for both of these case studies is not semantically much different to the traces. Especially the Facebook Account Backdoor attack which is a single path.

The OAuth case study based on Sun and Beznosov (2012) does have two different paths to ultimate failure of the FIM system. The attacker can force a login by performing a force-login CSRF attack or a session swapping attack. The way in which the login is forced for these two attacks is different so the point of divergence is the XSS theft with the two bundles containing the CSRF and session swapping. Both force login methods also need a number of other conditions, which are duplicated in each bundle. A sub-tree of the overall attack tree showing this point of interest on the attack tree can be seen in fig. 5.3.

Full Attack Trees: For clarity, we also include the full attack trees. The attack tree for the OAuth case study can be seen in fig. 5.4. The attack tree for the Facebook account linkage case study can be seen in fig. 5.5.

5.5. Stakeholder Attribution

5.5.1. System to Stakeholder Attribution

The attack trees we use have the systems that the attacks occur on directly in the attack tree. This gives our approach a benefit over more abstract approaches that map attacks to a general attack

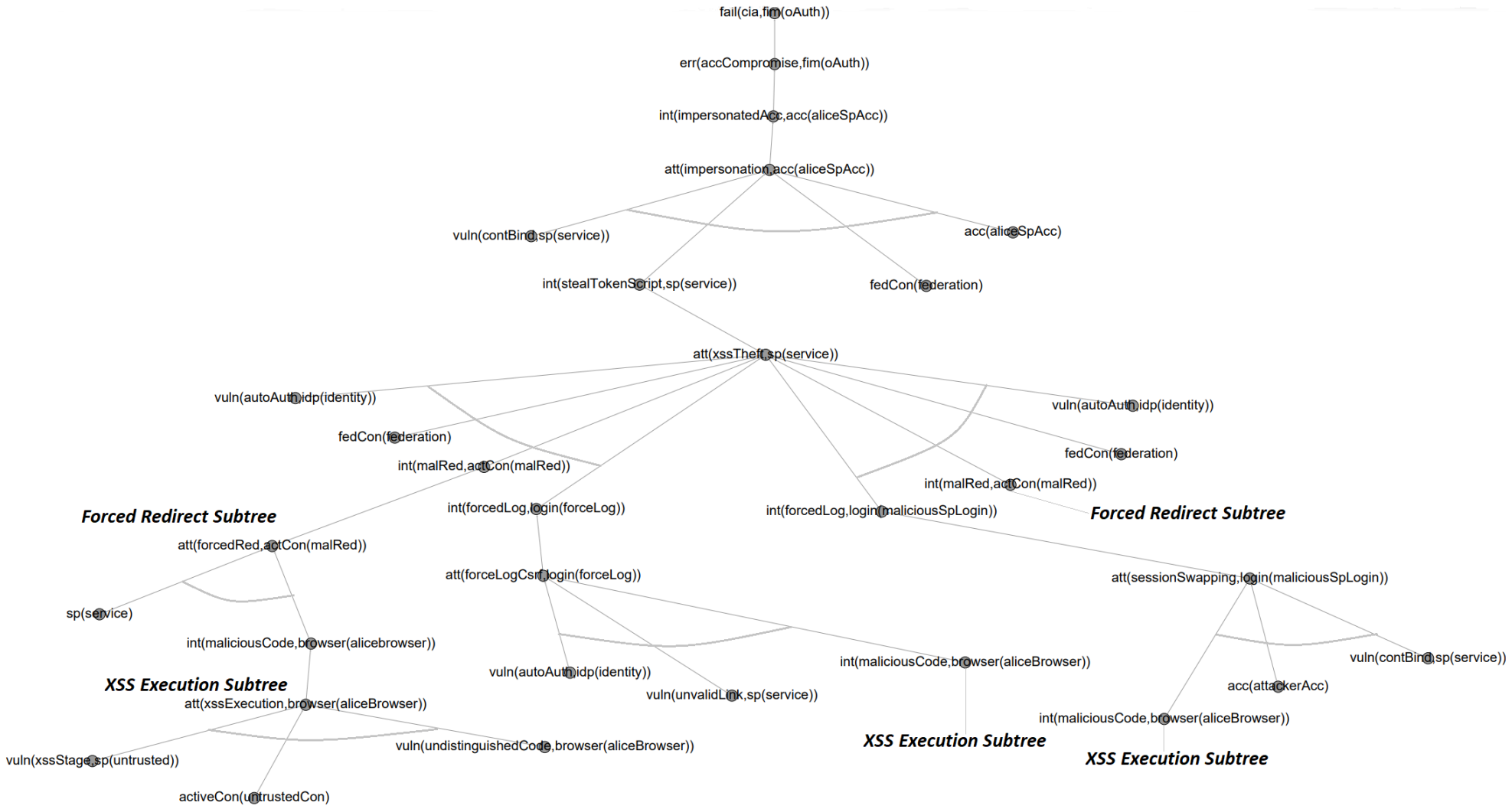

Figure 5.4 Overall Attack Tree for OAuth case study adapted from Sun and Beznosov (2012). Exact query used is 'fail(cia,fim(oAuth))'. Duplicate Subtrees highlighted and omitted for conciseness

Figure 5.5 Overall Attack Tree for Facebook case study adapted from introduced in section 4.8. Exact query used is 'fail(cia,fim(fbFim))'.

tree similar to that used by Fraile et al. (2016). Having this information in the tree allows us to consider attacks with a concrete system model in mind. There could exist a large number of attacks for the general system we are considering, but what attacks are actually applicable? Furthermore, of those attacks that are applicable, how are those attacks causing issues at each individual component? The systems of systems approach we take allows for analysis of attack trees with respect to some system model.

One concept Fraile et al. (2016) discusses is how attack trees can help diverse stakeholders consider a security problem. But one thing Fraile et al. do not show is how an attack tree can be systematically mapped to the stakeholders. Up until this point we have been considering the FIM systems in our system model in isolation, but how do these components relate to the actual stakeholders for the FIM system? This is important, because ultimately the goal is to present analysis to stakeholders and how components relate to stakeholders is not always clear from the system model. Stakeholder attribution on attack trees forms the basis for answering RQ3.

Sometimes, the attribution of a component to a stakeholder is simple, for instance, a SP that relates to a SP owner. But consider that a stakeholder actually owns both the SP and the IdP. In this case, any concerns with the SP or IdP system can be attributed to the same stakeholder. We can also have systems that might be of concern to more than one stakeholder. For instance, if there is a vulnerability on a protocol as we explore in subsection 4.7.4, that vulnerability could be attributed to multiple stakeholders like the SP, IdP, and perhaps even the protocol

specification. With these more complex examples, we can see that how components are attributed to stakeholders is not as simple as it first seems.

Abstraction Level: How stakeholders are chosen depends on the level of abstraction taken by the analyser. For instance, certain teams or groups for one system could be responsible for certain vulnerabilities. Perhaps there is one team that is responsible for database vulnerabilities and another is responsible for front-end website vulnerabilities. It could be the case that vulnerabilities for a certain system are to be divided between these teams. The level of abstraction we take for these examples will be higher level than this where we consider the owners for each system rather than specific entities within an organisation.

There is No One Way: Which systems are attributed to certain stakeholders is ultimately the decision of the analyser. For example, Sun and Beznosov (2012) considers SPs and IdPs to be the main stakeholders for the analysis of OAuth. However, authors like Jøsang et al. (2005) consider a more holistic approach where also the browser is considered a stakeholder in the overall FIM system. There is a choice that the analyser makes on which system is to be attributed to which stakeholder. This decision might be made because of the perceived control the stakeholder has over a system, or the impact a security issue with a system would have on the stakeholder, the ownership of digital assets on a system (such as a user account on an SP), or the target audience for the analyser (some analysts might not be concerned with the browser).

How Stakeholder Attribution is Done: We then consider exactly how systems can be linked to stakeholders. For each system that is attached to a MAFTIA attack term (vulnerability, attack, intrusion, etc. more in tab. 4.3) in an attack tree, there must exist some attribution to one or more stakeholders. A stakeholder can have multiple systems attributed to that stakeholder. Systems can be attributed to more than one stakeholder in the event that it is perceived that more than one stakeholder could be responsible for that system. Sometimes this can result in having to attribute a system that we do not seek to analyse any further to a stakeholder. For instance, a user, external SP, or DNS. In these situations, we can assign these systems to a stakeholder called *external*, to signal in analysis that this is not a system of concern.

More formally, the set of systems S derived from all systems contained in attack terms within a tree maps to a set of stakeholders σ. This is described as the mapping function:-

$$shMap : S \rightarrow \sigma$$

.

5.5.2. *Stakeholder Attribution Facebook Example*

In section 4.8, an attack trace for the Facebook Account Linkage case study is presented. Semantically, the overall attack trees for this system are not different to the attack trace as there is only one path to failure. We start with this case study as it is simpler than the OAuth case study. However, there are still some things to consider with this case study. In particular, what is interesting about this case is that the SP owner, Spotify, is also the stakeholder for the Spotify app. This means that we have a single stakeholder that is responsible for two different systems that are in the attack tree structure.

Let us consider how the systems referenced by attack terms can be attributed to Stakeholders. The set of all systems that require attribution are all the systems that are represented by an attack term in the tree. Note that we use a simplified syntax (see subsection 4.5.5) for the tree referring to just the unique identifier of the system. For the Facebook Account Linkage example, these systems are referred to as S_f:-

$$S_f = \{fim(fbFim), idp(facebook), sp(spotify), browser(spotifyApp), login(attackerLogin), acc(spotifyAccount)\}$$

With the set of affected systems specified, we aim to attribute these systems to a set of stakeholders. As stated in subsection 5.5.1, there is no one way in which stakeholders are chosen so this can change depending on the analyser. For instance, one aspect we do not focus on is presenting issues to the user so the user does not appear as a stakeholder. However, others might aim to present issues with the user in mind as a stakeholder in a system so the exact set of stakeholders can be changed depending on different viewpoints and different aims of the analysis of attack trees. The set of stakeholders considered for this example are σ_f:-

$$\sigma_f = \{FacebookCorporation, SpotifyCorporation\}$$

We consider a mapping $shMap_f : S_f \rightarrow \sigma_f$ that maps systems to stakeholders for the Facebook case study specified in subsection 4.8. A full overview of the mappings between the systems and the stakeholders and the reason for those mappings can be seen in tab. 5.1.

This mapping is to link the issues that occur on the attack tree to some analysis. In essence, the analysis we introduce on attack trees in FIM revolves around this idea of stakeholder attribution. The idea is to be able to present the issues to stakeholders with some idea of how these issues can be dealt with. Of particular interest with this example is that one stakeholder is responsible for multiple systems. This means that the stakeholder has to account for issues occurring on distinct systems.

System	Stakeholder	Reason
fim(fbFim)	Facebook Corporation, Spotify Corporation	The problems with the overall FIM system is attributed to the Facebook Corporation since Facebook is the corporation which provide this method of authentication to SPs globally. We also consider the Spotify Corporation for attribution as these attacks are possible due to some implementation oversights by Spotify.
idp(facebook)	Facebook Corporation	The IdP in the system is attributed to the Facebook Corporation.
sp(spotify)	Spotify Corporation	The SP in the system is attributed to the Spotify Corporation.
browser(spotifyApp)	Spotify Corporation	Spotify is responsible for the app that this attack is staged on.
login(attackerLogin)	Spotify Corporation	Since Spotify is responsible for the app, we attribute the login to the Spotify Corporation.
acc(spotifyAccount)	Facebook Corporation, Spotify Corporation	The account of a user is an abstraction of the information stored on a user between the SP and IdP and as a result could be attributed to all of these stakeholders including the user, but since we are not considering the user as a stakeholder for this example the account is attributed to only the SP and IdP stakeholders which are the Spotify Corporation and the Facebook Corporation.

Table 5.1 Mapping of Systems to Stakeholders in the Facebook Account Linkage Case Study using a simplified syntax described in subsection 4.5.5.

5.5.3. Stakeholder Attribution OAuth Example

In section 4.6 we introduce attack traces for OAuth based on the attack suite presented by Sun and Beznosov (2012). This is further refined to an attack tree, where the alternative between the CSRF and session swapping attack is shown in subsection 5.4.4. We consider the attack tree from the perspective of the overall FIM system. This means that we take account for all of the failures of components that led to the overall failure. This overall attack tree can be seen in fig. 5.4. For this tree we consider the set of all systems which are the location in an attack term. The set of systems being considered, using simplified syntax as per section 4.5.5 is S_o:-

$$S_o = \{fim(oAuth), idp(identity), sp(service), browser(aliceBrowser), acc(aliceSPAcc), \\ actCon(malRed), sp(untrusted), login(maliciousSpLogin), login(forceLog)\}$$

Each of these systems has to be mapped to a set of stakeholders and how these stakeholders are chosen depends on the level of abstraction taken by the analyser. The level of abstraction we take for these examples will be high, in that the owners for each system will be considered. Consider below a description of each stakeholder and a more formal specification σ_o:-

- *Service Owner:* Represents the stakeholder who is ultimately responsible for the SP. In reality, this could be a group of individuals or even the same stakeholder as the IdP. For the simplicity of this example, we assume there to be one central entity responsible for the SP and acts as the stakeholder.
- *Identity Owner:* Represents the stakeholder who is ultimately responsible for the IdP. Similar to the Service Owner stakeholder in framing, the stakeholder could in reality be a group but we assume a central entity which is responsible for the IdP and acts as the stakeholder.
- *Browser Community:* Represents the stakeholder who is ultimately responsible for the Browser. In reality, it is difficult to reason a central authority for the browser. To the best of our knowledge, the closest representation to a central authority is W3C, which is a community concerned with the growth of the World Wide Web in general. Other candidates for stakeholders are Microsoft, Google, and Mozilla, as these organisations deploy some of the most used web browsers. However, this thesis is not concerned with these details, so for the simplicity of this example, one abstract stakeholder for the browser community is named, even if in reality it is far more likely to be several separate and potentially competing entities.
- *Concerned Users:* Represents the users of the system. For example, could be a group of employees for a company. Again for simplicity, represented as a single stakeholder.
- *External:* Represents systems which are considered outside of the control of the model. For example, this could be a website which is not in any way affiliated with the FIM system, but is still has potential to introduce security issues for the users of the FIM system. These issues could still have a cost, but this cost is not applicable to the FIM system.

$$\sigma_o = \{ServiceOwner, IdentityOwner, BrowserCommunity, ConcernedUsers, External\}$$

Now let us consider a concrete mapping between the systems $shMap_o : S_o \rightarrow \sigma_o$ and the stakeholders which can be seen in tab. 5.2. This mapping allows for a basis of the analysis because we can link the components into the FIM system to actual stakeholders.

5.6. Cost Analysis

Cost analysis using attack trees is nothing new and is a problem which has been approached before. The idea of a cost analysis is first introduced by Schneier (1999) and notably refined by Kumar et al. (2015); Dewri et al. (2012). We aim to show how our approach can offer some

System	Stakeholder	Reason
fim(oAuth)	Service Owner, Identity Owner, Browser Community	Sun and Beznosov (2012) presents overall solutions to the SPs and IdPs involved. We extend this viewpoint to also include the browser.
idp(identity)	Identity Owner	The IdP is the sole stakeholder concerned with issues on the IdP.
sp(service)	Service Owner	The SP is the sole stakeholder concerned with issues on the SP.
browser(aliceBrowser)	Browser Community	The browser is the sole stakeholder concerned with issues on the browser.
acc(aliceSpAcc)	Service Owner, Concerned Users	It could be useful to demonstrate to users how their accounts can be compromised. The Service Owner might also be interested in knowing how the accounts they are trying to protect can be compromised.
actCon(malRed)	Browser Community	We consider this mainly a browser issue as the browser does not detect that a redirect has not been initiated by a user.
sp(untrusted)	External	This untrusted sp is outside of the control of the FIM system but still contributes to the overall attack. For this situation, we can refer this system to some external stakeholder which signals that this system is beyond the control of the overall FIM system.
login(maliciousSpLogin)	Browser Community, Service Owner, Concerned Users	The browser could assist in detecting fraudulent logins not initiated by a user. The service owner could also use additional authentication controls to stop fraudulent logins. Users could be made more aware that this is a possibility, and if possible, act accordingly.
login(forceLog)	Browser Community, Service Owner, Concerned Users	This is similar to the login(maliciousSpLogin), the difference being the type of attack realised to reach each login state so the reasoning is the same.

Table 5.2 OAuth System to Stakeholder Mapping using a simplified syntax described in subsection 4.5.5.

additional considerations to the existing cost methods by directly applying cost to stakeholders in an attack tree.

5.6.1. Traditional Attack Tree Cost Analysis as Introduced by Schneier

Schneier (1999) (see above for reference on more recent work) demonstrates how cost can be applied to attack trees in general and considers how much an attack would cost from the attacker perspective. Therefore, attack trees can be used as a risk analysis tool to consider how much an attacker would have to invest into compromising a system. This could trivially be done with the attack trees presented as part of this thesis, by applying arbitrary cost values to vulnerabilities that indicate the cost of a vulnerability being exploited, and then propagating this cost through the attack tree. Note that for the attack trees used in this work, some leaf nodes are systems which might not be appropriate for cost evaluation. Schneier only has cost annotations for leaf

nodes, but all of those leaf nodes correspond to vulnerabilities rather than just systems being present. One way of handling this is to remove the systems from our attack trees which we describe in 5.4.2.

Traditional Cost Analysis of OAuth Tree: The cost of exploitation to the attacker should be low. The only thing the attacker might need is a server in which to send stolen XSS tokens. This information could possibly be annotated to the attack tree seen in fig. 5.4. Specifically to the intrusion regarding the stolen access token. Every other attack would only require the knowledge itself of how to exploit the system and no need to solve hard cryptography problems. This means that the attack could easily be scaled up to many users with little cost to the attacker.

Traditional Cost Analysis of Facebook Account Linkage Tree: The coffee break access required can be attributed to the insecure browser vulnerability. This is the only exploitation that could have considerable associated cost as the other vulnerabilities, like the OAuth tree, only require knowledge of the attack to execute. The cost of getting this temporary access could be non-trivial for an attacker and include bribing individuals to get access or using special equipment to break in. In fact, this vulnerability could even be treated as an intrusion for other physical systems such as a company or a residential home. How exactly this vulnerability would be exploited is out of scope, but we will state that this attack has less scalability than the OAuth attack as the cost of exploiting a required vulnerability could be considerable and would also require the physical presence of the attacker at a location, rather than the attack being possible to launch remotely.

5.6.2. Additional Preliminaries

In fig. 5.2 an object orientated approach is outlined for creating attack trees. We further refine this design for cost analysis and this is presented in UML form in fig. 5.6. The first step is to actually set the costs for an attack tree. For these examples, we are only concerned with vulnerability removal which closely aligns with how Schneier (1999) analyses attack trees. The difference being, Schneier focuses on the attacker point of view while we focus on the defender. To do this we create a mapping between the set of all possible identifiers in attack terms I and a cost which is the set of all natural numbers $\mathbb{N}$. We do not take the entire attack term because we could have the same kind of vulnerability in different places. A vulnerability mapping is then expressed:-

$$vulnMap : I \rightarrow \mathbb{N}$$

We then run the *setCosts* method for the attack tree with the specified vulnerability map. For these examples, we choose arbitrary cost values that could denote some monetary or

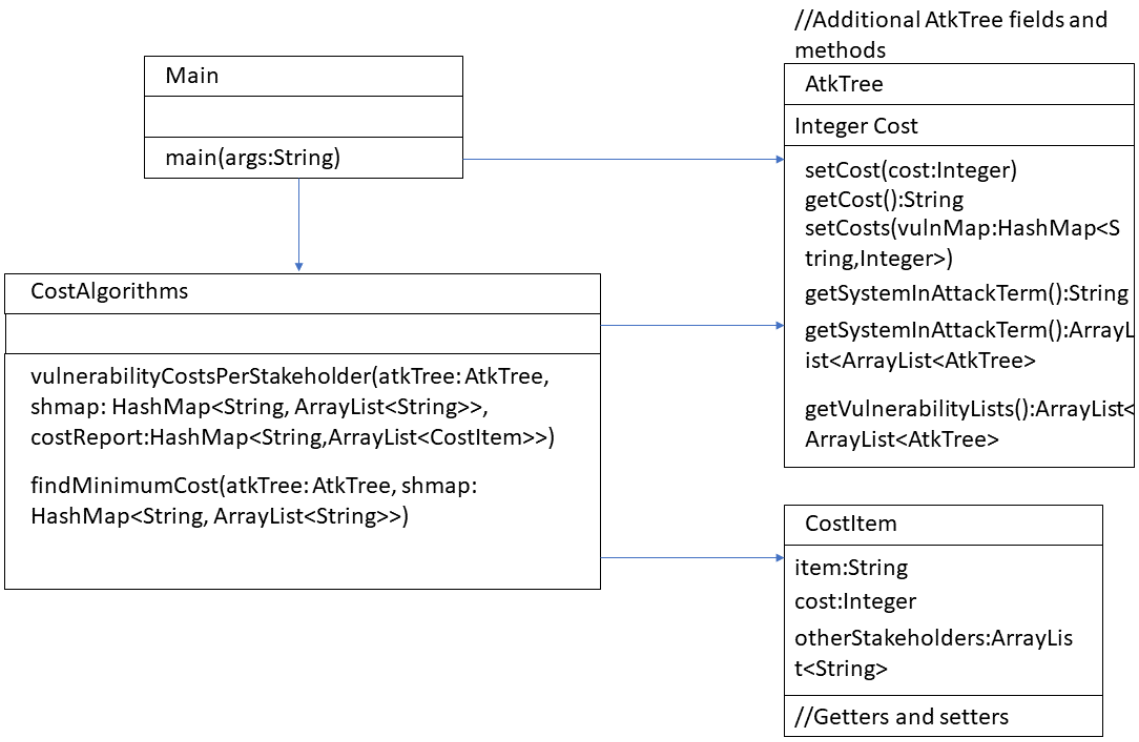

Figure 5.6 UML Diagram of Java Implementation of Cost Algorithms. Omitted previously presented classes, focusing on the classes needed for cost analysis. See 5.2 for more.

development time cost. For the Facebook Account Linkage case study, we refer to $vulnMap_f$. For the OAuth case study, we refer to $vulnMap_o$. Both of these mappings can be seen in tab. 5.3.

Mapping	Vulnerability	Cost
$vulnMap_f$	linkageOption	100
	idpReAuth	200
	insecureBrowser	300
	noLoginCheck	400
$vulnMap_o$	undistinguishedCode	100
	xssStage	200
	autoAuth	300
	unvalidLink	100
	contBind	500

Table 5.3 The vulnerability mappings for the Facebook and OAuth case study.

For each stakeholder, the aim is to generate cost reports consisting of cost items. Cost items are triples (as seen in fig. 5.6) which describe a relation between a node in the attack tree (or item), the cost of that item expressed as an Integer, and other stakeholders (any stakeholder which is also attributed to this problem which is not the stakeholder this cost item is attributed to). A cost report is then a mapping between the set of all stakeholders σ and the set of all cost items C:-

$$costReport : \sigma \rightarrow C$$

Goal of Cost Analysis: The goal is to present cost analysis to a group of stakeholders through cost reports. The central idea being that the systems on an attack tree can be attributed to stakeholders and those stakeholders can consider approaching a security problem in different ways. The remaining part of this section demonstrates different ways in which cost reports could be rendered and how they can be analysed.

5.6.3. Calculating Vulnerability Fixing Cost for Each Stakeholder

The Schneier (1999) approach calculates the cost from the perspective of the attacker. We instead consider cost from a defender point of view by removing vulnerabilities. The Facebook Account Linkage case study is a simpler attack tree so let this serve as a basic approach to generating cost reports where we simply find all vulnerabilities in an attack tree and consider these vulnerabilities as a cost item attributed to each stakeholder for the system the vulnerability exists on. Vulnerabilities with the same identifier are not recorded twice, as the vulnerability would only need to be fixed once. The idea for this approach is that each stakeholder can consider all of the vulnerabilities affecting their systems and potentially fix all of them knowing the full cost.

Summary of Algorithm: An attack tree, a stakeholder mapping, and an empty cost report global variable which is to be populated by mappings between stakeholders and vulnerabilities is input. We first check if the node has a cost, and if it does we create a new cost item for every stakeholder involved. If this cost item is new to the mapping (we do not map vulnerabilities more than once), then add it to the cost report. If the attack tree has children, call this algorithm recursively for all children. This algorithm is detailed in alg. 4.

Applying the algorithm using $shMap_f$ and $vulnMap_f$ as inputs, the following cost report is produced as per tab. 5.4.

5.6.4. External Cost

One other application for algorithm 4 is to calculate the total external cost for an attack tree. In other words, we can separate the cost of issues for relevant stakeholders and irrelevant stakeholders. Let us consider the application of the stakeholder map $shMap_o$ and the vulnerability map $vulnMap_o$ to algorithm 4. This application yields the cost report shown in tab. 4.5.5. This information can be presented to stakeholders to show what is controllable and what is not controllable.

5.6.5. Minimum Cost of Fixing The Overall Issue

Schneier (1999) approach calculates the cost of attacking a system by taking the cheapest cost of each leaf node (or, in the case of conjunction adding those together) and attributing that cost to the parent node. Repeat this process for every leaf node, until the root node is reached with the cheapest possible attack for the attacker to execute. We shift the perspective from the cost of

```
algorithm : VulnerabilityCostPerStakeHolder
input     : AtkTree atkTree, Map<String, List<String» shMap, Global Map<String>,
            List<CostItem> costReport
if atkTree.getCost() != null then
    String atkTreeSystem = atkTree.getSystemInAttackTerm;
    List<String> stakeholders = shMap.get(atkTreeSystem);
    for each sh in shers do
        List<String> otherStakeholders = new List<>();
        for each innerSh in shers do
            if innerSh != sh then
                otherStakeholders.add(innerSh);
        end
        CostItem nodeAsCostItem = new CostItem(atTree.getFinalCall(), atkTree.getCost(),
          otherStakeholders);
        List<CostItem> currentCostReport = costReport.get(nodeAsCostItem);
        if currentCostReport == null then
            List<CostItem> newReport = new List<>();
            newReport.add(nodeAsCostItem);
        else if !currentCostReport.contains(nodeAsCostItem) then
            currentCostReport.add(nodeAsCostItem);
    end
if atkTree.getBundles() != null then
    for each bundle in bundles do
        for each atkTree in bundle do
            vulnerabilityCostPerStakeholder(atkTree,shMap,costReport);
        end
    end
```

Algorithm 4: Vulnerability Cost Per Stakeholder

executing the attack for the attacker to the cost of defending against attacks from the defender perspective by removing vulnerabilities. Sheyner et al. (2002) inspired this direction for cost analysis. Although Sheyner et al. focuses on attack graphs, the idea of minimum sets required for an attacker is considered which is similar to the approach we take here.

When taking the defender perspective we have to account for each of the different paths that the attacker can take in stopping an attack. These different paths are determined by disjunctions in the attack tree. Each disjunction in the tree will allow for a different set of vulnerabilities being exploited to realise the attack. In the OAuth case study, we have an alternative between the session swapping and CSRF attack which results in two different vulnerability sets that the attacker must exploit to succeed in the root level goal on the attack tree. Therefore, if the defenders can remove one vulnerability from each of these sets, the root level goal will not be reachable. We consider generating cost reports for each possible vulnerability set on an attack tree. The algorithm used to achieve this can be seen in alg. 5.

Stakeholder	Cost Item (item on tree, cost, other stakeholders)
Facebook Corporation	vuln(idpReAuth,idp(facebook)) 200 empty
Spotify Corporation	vuln(linkageOption,sp(spotify)) 100 empty _____ vuln(insecureBrowser,browser(spotifyApp)) 200 empty _____ vuln(noLoginCheck,sp(spotify)) 400 empty

Table 5.4 The cost report for the Facebook Account Linkage case study produced by algorithm 4. Note that we use a simplified syntax for systems as described in 4.5.5.

In essence, we consider each possibility for an attack and then consider the cheapest option for each of these possibilities. Then a report is offered for each attack option, indicating the cheapest specific part of the attack tree to fix in order to stop that attack option. Therefore, we enumerate over each possibility for a specific instance consisting of vulnerabilities mapped to cost and a relationship between those vulnerabilities and stakeholders. There is no optimisation performed here. One thing to consider would be to use sensitivity analysis to determine the propensity of change in our results in response to changes in the cost. For example, we could investigate the propensity for our results to change based on changing select values. There could be uncertainty with the cost of fixing a specific vulnerability, and sensitivity analysis could be used to account for that. We note this as a future avenue for research, but the cost analysis we describe in this thesis does not consider this avenue any further.

Applying the algorithm to the OAuth case study, we get a list of cost reports where each item in the list corresponds to a possible option in the attack tree. One cost report corresponds to the session swapping attack, and the other corresponds to the CSRF attack. This can be seen in tab. 5.6. The cost for CSRF attack features two separate vulnerabilities because there is a tie between the cheapest items. The Session Swapping attack consists of an item attributed to the Browser Community, indicating that perhaps the best option for the FIM system overall is to petition the browser for changes.

5.6.6. Disputing Cost

The case studies we describe do not have other stakeholders in any of the cost items. However, it is not unrealistic to imagine a situation where several stakeholders are attributed to a single vulnerability. For example, if for an OAuth based FIM system the IdP has some claim to SPs in addition to the SP owner. In these cases, the stakeholders can see what other stakeholders are attributed and an internal discussion can be had regarding how to split costs.

Stakeholder	Cost Item (item on tree, cost, other stakeholders)
Service Owner	vuln(contBind,sp(service)) 500 empty _____ vuln(unvalidLink,sp(service)) 100 empty
Identity Owner	vuln(autoAuth,idp(identity)) 300 empty
Browser Community	vuln(undistinguishedCode,browser(aliceBrowser)) 100 empty
External	vuln(xssStage,sp(untrusted)) Not Applicable as External empty

Table 5.5 The cost report for the OAuth Case Study produced by algorithm 4 with the cost of the external labelled as non-applicable. Note that we use a simplified syntax for systems as described in 4.5.5.

5.7. Discussion

5.7.1. Further Cost Analysis

This chapter demonstrates some cost analysis techniques for showing how stakeholder attribution in an attack tree can be useful. This is in fulfilment of RQ3, but is not an exhaustive approach. There are many other ways cost analysis could be applied:-

- Minimum cost reports could be further analysed and consolidated into a single cheapest vulnerability to fix for an attack tree in general. The most cost-efficient vulnerabilities could be considered across different attack options so that the overall minimal cost vulnerabilities can be chosen in solving a security problem.
- More complex cost models could be considered rather than just a single unit of measurement. For instance, consider a vulnerability in a browser that also needs to be adopted by all browsers in order for the fix to be effective. This cost goes beyond just a simple numerical expression. At the same time, costs need to be in some way comparable so a more complex cost model with different heuristic parameters as input could be considered. The way that costs are compared could influence the overall decisions being made and this is a possible direction of future research.
- Empirically sourced cost values could be considered to see how effective these ideas are in real life security scenarios.
- The impact of a successful attack on a stakeholder could be studied by considering the cost of failures or intrusions. We can then consider the extent of the damage in terms of

```
algorithm : findMinimumCost
input     : AtkTree atkTree, Map<String, List<String» shMap
List<List<AtkTree» vulnerabilityLists = atkTree.getVulnerabilityLists();
List<List<AtkTree» cheapestOptions = new List<>();
for each nextList in vulnerabilityLists do
    List<AtkTree> cheapList = new List<>();
    Integer currentCheapest = null;
    for each nextTree in nextList do
        if currentCheapest == null || nextTree.getCost() < currentCheapest then
            cheapList = new List<>();
            currentCheapest = nextTree.getCost();
            cheapList.add(nextTree);
        else if nextTree.getCost() == currentCheapest) then
            cheapList.add(nextTree);
    end
    cheapestOptions.add(cheapList);
end
List<Map<String,List<CostItem»> costReports = new List<>();
for each nextOption in cheapestOptions do
    Map<String,List<CostItem» costReport = new Map<>(); for each nextTree in
     nextOption do
        String systemForTree = nextTree.getSystemInAttackTerm();
        List<String> stakeholders = shMap.get(systemForTree);
        List<CostItem> costItems = new List<>();
        for each nextStakeholder in stakeholders do
            List<String> otherStakeholders = new List<>();
            for each nextInnerStakeholder in stakeholders do
                if nextInnerStakeholder != nextStakeholder then
                    otherStakeholders.add(nextInnerStakeholder)
                CostItem nextCostItem = new
                  CostItem(nextTree.getFinalCall(),nextTree.getCost(),otherStakeholders);
                costItems.add(nextCostItem);
            end
            costReport.put(nextStakeholder,costItems);
        end
    end
    costReports.add(costReport);
end
```

Algorithm 5: Minimum cost per attack tree option

cost for one stakeholder, perhaps by another stakeholder neglecting to handle an issue that could be taken care of for comparatively much cheaper. High threat vulnerabilities could be highlighted by evaluating the extent of the damage that could be caused from their exploitation.

Attack Option	Stakeholder	Cost Item (item on tree, cost, other stakeholders)
Session Swapping	Browser Community	vuln(undistinguishedCode,browser(aliceBrowser)) 100 empty
CSRF	Browser Community	vuln(undistinguishedCode,browser(aliceBrowser)) 100 empty
	Service Owner	vuln(unvalidLink,sp(service)) 100 empty

Table 5.6 Minimum Cost Reports for the Session Swapping and CSRF case study produced by algorithm 5. Note that we use a simplified syntax for systems as described in 4.5.5.

5.7.2. Wider System Security Considerations

The idea introduced in this chapter that system components can be mapped to stakeholders could be useful for attack tree research in general. Fraile et al. (2016) specifically considers how attack trees can be used to relate security problems to stakeholders, but does not provide a clear way on how to do this. There could be many system components for an overall system but how do we relate all of these components to the stakeholders responsible for those components? This question has been considered in this chapter, and the information gleaned could be useful in designing security strategies where certain responsibilities or tasks have to be delegated across different organisations or groups. We have used some simple cost analysis techniques, and have discussed further ways that this can be built upon in subsection 5.7.1. However, this is not the only way stakeholder attribution could be used. Another application could be to consider overall cyber-defence strategies for stakeholders based on the systems under their control. We believe stakeholder attribution to be a good foundation for considering further attack tree research.

Importantly, attack trees would not need to be synthesised in the exact way specified in this thesis. The only quality that the attack tree needs to incorporate: for each attack node there must be information that allows us to identify the attacked system component. The way the attack trees are synthesised (in this case using a logical programming approach to create traces and then trees) does not need to be followed. Also, the problem domain being considered in this thesis (FIM), is not the only applicable area for this to be used as there are many other systems that can be separated into components such as networks in general or even physical systems.

5.7.3. Validation

Attack Tree Compatibility: The attack trees synthesised mostly conform to attack tree literature in that there are abstract attacker goals that can be reached via the condition goals. These condition goals are bundles, as is presented initially by Mauw and Oostdijk (2005). The attack trees presented conform to this idea of nodes being made possible by bundles representing logical and conditions, and the different bundles an attack tree has representing logical or

conditions. Traditional cost analysis (specified by Schneier (1999)), should be possible on these attack trees.

Attack Tree Correctness: In subsection 4.2.2 the idea of traces produced being correct with respect to the source material that they are produced from is introduced. The main distinction when considering the validation from an attack tree perspective is that the alternatives are also present on the tree and we can see the full picture. In section 3.9, the case study for OAuth is introduced adapted from Sun and Beznosov (2012). We adapt this case study to narrow the scope as per fig. 3.8. The causality of vulnerabilities is followed in this work where the two main vulnerabilities considered by Sun and Beznosov, lack of contextual binding and automatic authorisation granting, are followed correctly. We also extend the case study to show how the browser is involved and how an outside influence is also present through XSS staging. While this is not made entirely explicit by Sun and Beznosov, the chain of events is highlighted in their work specifically in section 5.5 titled "Vulnerability Interplays". We believe our adaption to be a faithful interpretation of the vulnerability interplays described by Sun and Beznosov.

5.8. Summary

In this chapter, we introduce the synthesis of attack trees. We base the attack tree synthesis on the consolidation of attack traces (which are introduced in chapter 4 through a logical programming approach). Attack trees are a good security analysis tool, which offered sufficient motivation for the synthesis of attack trees.

One benefit to the way in which we have built attack trees is that security issues are linked directly to the system that the issue is present on. This allows for what we describe as stakeholder attribution which is a mapping between systems that exist on the attack tree and the actual entities that are in control of that system. Stakeholder attribution allows for different kinds of analysis than we have seen before regarding attack trees. For instance, we demonstrate some simple cost based examples on the case studies we have considered as part of this thesis.

Attack trees are a good analysis tool that interfaces well between security experts and those with little security knowledge. However, we believe that there is more potential there, and stakeholder attribution is one possible direction for this. As part of the discussion subsection 5.7.1, we have also suggested further directions for cost analysis based on stakeholder attribution. FIM systems are based on collaboration between different stakeholders, which makes these ideas applicable to FIM, but not specific to only FIM. Further research is needed to further develop these ideas and assess their quality for computer security research in general.

Chapter 6. Conclusion

Overall, we show how attacks can be modelled in FIM by using a logical programming approach supplemented by attacks existing in the literature and new attacks that are discovered as part of this thesis. The hope is that this research can provide researchers with a method of considering how some undesirable security failure is reachable through escalating attacks for a particular system. Beyond the reachability of an undesirable failure state, the causality which allowed for that failure state to occur can be presented and analysed through attack trees.

This thesis performs a systematic survey in chapter 3 on security analysis papers in Federated Identity Management (FIM). We collate this knowledge to establish a taxonomy for considering attacks in FIM. The work in this chapter was performed in support of RQ1. The main contribution of this chapter is an overview of the landscape of security issues in FIM expressed in a taxonomy based on the Malicious and Accidental Fault Tolerance Framework (MAFTIA). This is the most extensive collation of specific security issues in FIM to date, to the best of our knowledge. This landscape of attacks provides insight into security issues in FIM occurring across protocols and potential solutions for those issues. The collation of this information also led to the idea of escalating attacks on FIM.

We introduce the idea of escalating attacks in FIM. Escalating attacks are a phenomena where the effect of one attack causes another attack. From the survey of security analysis in FIM, we observed an interesting situation where OAuth 2.0 is claimed to be secure under certain assumptions as is considered by Chari et al. (2011). However, when OAuth is exposed to other attacks, the assumption that OAuth 2.0 is secure no longer holds, and Sun and Beznosov (2012) proves this. We devise an approach based on logical programming which is able to model this situation and this is the main contribution of chapter 4 and in answer of RQ2. The approach should be adaptable to different scenarios in FIM, therefore we also consider how different problems can be considered using the survey of security analysis in FIM as a basis. In addition, we show how novel security knowledge of attacks can be considered using our approach.

Using attack traces as a basis, we automatically synthesise attack trees for FIM. A specific quality of the attack trees we use is that the system being attacked is interwoven into the tree. This allows for stakeholder attribution, which is the linking of analyst specified stakeholders to the FIM system. Stakeholder attribution is a novel idea for attack trees which allows for some interesting analysis approaches. We consider some simple cost analysis applications where we attribute the total cost of fixing vulnerabilities for each stakeholder, and the minimum cost of

fixing each possible route in an attack tree for each stakeholder. We also discuss several routes for deeper analysis based on stakeholder attribution, but these would require further research.

6.1. Contributions

The overall aim of this thesis is to model escalating attacks in FIM. Towards this main aim, we offer three contributions which we believe to answer RQ1, RQ2, and RQ3 respectively:-

- A taxonomy for security issues on FIM systems.
- A logical programming approach to detect escalating attacks on a FIM system model.
- Automated synthesis of attack trees, stakeholder attribution, and cost analysis for the stakeholders of a FIM system.

6.2. Limitations and Future Work

6.2.1. Performance Analysis and Alternative Attack Tree Synthesis Methods

Analysis of performance was done and some consideration for alternative methods in the generation of attack trees was considered (see chapter 5). However, the analysis is rather limited, and not the same standard as similar work in the area like Ou et al. (2006). What would be interesting is to compare different methods against each other to find what is the best way of synthesising an attack tree using a model of a FIM system operated on by logical rules in Prolog.

6.2.2. Vulnerability Analysis and Exploitation Tools

We have introduced a logical approach that is informed by a declaration of a vulnerability existing. However, there could be disparity of vulnerabilities that actually exist for a system and vulnerabilities that are thought to exist. An avenue for supplementing this approach would be to include automated vulnerability analysis tools. For instance, a specific FIM system can be analysed with reference to some automated analysis tools. In particular, there are many tools that are used as part of an ethical hacking approach. Kali (2021) is an operating system which comes prepackaged with many different tools for finding specific vulnerabilities such as sqlmap for finding sql vulnerabilities or XSSer for finding XSS vulnerabilities. Alternatively, Kali has complete vulnerability assessment tools to find a number of different vulnerabilities like Nikto. With automated vulnertability assessment, a model of a FIM system can be constructed based on vulnerabilities that actually exist for a particular FIM system.

We have developed a rule base and system model for a limited number of case studies. These case studies suffice in demonstrating escalating attack modelling in FIM but do not paint a complete picture of all the ways FIM can be logically analysed. The work by Sun and Beznosov (2012) was especially useful because there was considerable empirical testing of issues, detail about these issues, and a clear description of how these issues interlink. Therefore, it was easy to

logically analyse compared to the wider FIM security analysis literature. What would be interesting is to empirically verify the work on security analysis in FIM by others to see exact methods of exploitation and the system model required in order to devise a logical model. Kali (2021) could come in use here as the various analysis and exploitation tools could be used to simulate the attacks described in FIM security analysis literature and recreate the conditions necessary for exploitation which could then form the basis of a logical analysis model.

6.2.3. Intrusion Tolerance

One of the main goals of MAFTIA as described by Powell and Stroud (2003) is intrusion tolerance which is concerned with providing a service capable of implementing the system function, despite intrusions. For FIM systems, this would mean that instead of trying to remove, prevent, or forecast errors we try to tolerate the errors once they do happen. One avenue of interest is access tokens once they get into the hands of an attacker and this is discussed by Sun and Beznosov (2012). Specifically, Sun and Beznosov describe how stolen access tokens could be used by an attacker to act as the user in a social graph and compromise user data. The work in this thesis sets the basis for considering FIM using an intrusion tolerance approach by conforming to the MAFTIA model. We consider stakeholder attribution which could be used to attribute intrusion tolerance techniques to stakeholders. The next step would be to apply the intrusion tolerance techniques. This would allow for a more formal consideration of tolerating intrusions in FIM.

Appendix A. Final Survey Sample

Paper Number	Title	Protocols Involved	Reference
1	Risks of the passport single signon protocol	Microsoft	Kormann and Rubin (2000)
2	Microsoft .Net Passport: a security analysis	Microsoft	Oppliger (2004)
3	Improving the security of cardspace	Microsoft	Alrodhan and Mitchell (2009)
4	Risks of the cardspace protocol	Microsoft	Gajek et al. (2009)
5	The devil is in the (implementation) details: an empirical analysis of OAuth SSO systems	OAuth	Sun and Beznosov (2012)
6	Enhancing OAuth services security by an authentication service with face recognition	OAuth	Alotaibi and Mahmmod (2015)
7	Security evaluation of the oauth 2.0 framework	OAuth	Ferry et al. (2015)
8	Security issues in oauth 2.0 sso implementations	OAuth	Li and Mitchell (2014)
9	More guidelines than rules: Csrf vulnerabilities from non compliant oauth 2.0 implementations	OAuth	Shernan et al. (2015)
10	A security analysis of the oauth protocol	OAuth	Yang and Manoharan (2013)
11	Security analysis of authentication protocols for next-generation mobile and CE cloud services	OAuth, OpenID	Grzonkowski et al. (2011)
12	The security limitations of sso in openid	OpenID	Oh and Jin (2008)
13	Security analysis of openid	OpenID	Sovis et al. (2010)
14	Security analysis of openid, followed by a reference implementation of an npa-based openid provider	OpenID	Feld and Pohlmann (2011)
15	Systematically breaking and fixing openid security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures	OpenID	Sun et al. (2012)
16	Identifying an openid anti-phishing scheme for cyberspace	OpenID	Abbas et al. (2016)
17	Webcallerid: Leveraging cellular networks for web authentication	OpenID	Hsu et al. (2011)
18	Security of web level user identity management	OpenID	Krolo et al. (2009)
19	Analysing the security of google's implementation of openid connect	OpenID	Li and Mitchell (2016)
20	Do not trust me: Using malicious idps for analyzing and attacking single sign-on	OpenID	Mainka et al. (2016)
21	Formal analysis of saml 2.0 web browser single sign-on: breaking the saml-based single sign-on for google apps	SAML	Armando et al. (2008)
22	Security analysis of the saml single sign-on browser/artifact profile	SAML	Groß (2003)
23	A lightweight formal approach for analyzing security of web protocols	SAML	Kumar (2014)
24	Guardians of the clouds: when identity providers fail	SAML	Mayer et al. (2014)
25	Your software at my service: Security analysis of saas single sign-on solutions in the cloud	SAML	Mainka et al. (2014)
26	Analysis of liberty single-sign-on with enabled clients	Liberty	Pfitzmann and Waidner (2003a)
27	Trusted computing based open environment user authentication mode	Liberty	Ahmad et al. (2010)
28	Formal analysis of facebook connect single sign-on authentication protocol	Facebook	Miculan and Urban (2011)
29	Analysis of privacy vulnerabilities in single sign-on mechanisms for multimedia websites	Facebook, OpenID	Urueña et al. (2014)
30	Signing me onto your accounts through facebook and google: A traffic-guided security study of commercially deployed single-sign-on web services	Facebook, OpenID	Wang et al. (2012)
31	Federated identity management	Shibboleth	Chadwick (2009)

Table A.1 Final Sample of Papers in Survey

Appendix B. Prolog System Models, Rule Sets, and Trace Outputs

Line Number	Term
1	fim(oAuth).
2	sp(service,oAuth).
3	sp(untrusted,empty).
4	idp(identity,oAuth).
5	user(alice).
6	browser(aliceBrowser,oAuth,alice).
7	activeCon(untrustedCon,oAuth,untrusted,aliceBrowser).
8	acc(aliceSpAcc,oAuth,alice,service,identity).
9	acc(aliceIdPAcc,oAuth,alice,identity,empty).
10	acc(attackerAcc,oAuth,empty,service,empty).
11	login(identityLog,aliceBrowser,aliceIdPAcc).
12	fedCon(federation,oAuth,service,identity).
13	vuln(undistinguishedCode,browser(aliceBrowser,oAuth,alice)).
14	vuln(xssStage,sp(untrusted,empty)).
15	vuln(autoAuth,idp(identity,oAuth)).
16	vuln(unvalidLink,sp(service,oAuth)).
17	vuln(contBind,sp(service,oAuth)).

Table B.1 This model is adapted from Sun and Beznosov (2012). System Model of specified in terms compatible with the system language specification described in tab. 4.2. Additional vulnerabilities specified as per the attack language specification described in tab. 4.3.

Line Number	Term
1	fim(fbFim).
2	sp(spotify,fbFim).
3	idp(facebook,fbFim).
4	user(alice).
5	browser(spotifyApp,fbFim,alice).
6	activeCon(appConnection,fbFim,spotify,spotifyApp).
7	fedCon(federation,fbFim,spotify,facebook).
8	acc(spotifyAcc,fbFim,alice,spotify,facebook).
9	acc(attackerAcc,fbFim,empty,empty,facebook).
10	login(aliceLogin,spotifyApp,spotifyAcc).
11	vuln(linkageOption,sp(spotify,fbFim)).
12	vuln(idpReAuth,idp(facebook,fbFim)).
13	vuln(insecureBrowser,browser(spotifyApp,fbFim,alice)).
14	vuln(noLoginCheck,sp(spotify,fbFim)).

Table B.2 This is the system model for the Facebook Account Linkage Introduced in 4.8. System Model specified in terms compatible with the system language specification described in tab. 4.2. Additional vulnerabilities specified as per the attack language specification described in tab. 4.3.

```
   Call: (8) fail(cia, fim(oAuth)) ? creep
   Call: (9) err(accCompromise, fim(oAuth)) ? creep
   Call: (10) int(impersonatedAcc, acc(_6258, oAuth, _6262, _6264, _6266)) ? creep
   Call: (11) att(impersonation, acc(_6258, oAuth, _6262, _6264, _6266)) ? creep
   Call: (12) vuln(contBind, sp(_6264, oAuth)) ? creep
   Exit: (12) vuln(contBind, sp(service, oAuth)) ? creep
   Call: (12) int(stealTokenScript, sp(service, oAuth)) ? creep
   Call: (13) att(xssTheft, sp(service, oAuth)) ? creep
   Call: (14) vuln(autoAuth, idp(_6300, oAuth)) ? creep
   Exit: (14) vuln(autoAuth, idp(identity, oAuth)) ? creep
   Call: (14) login(_6320, _6322, _6324) ? creep
   Exit: (14) login(identityLog, aliceBrowser, aliceIdPAcc) ? creep
   Call: (14) acc(aliceIdPAcc, oAuth, _6324, service, identity) ? creep
   Fail: (14) acc(aliceIdPAcc, oAuth, _6324, service, identity) ? creep
   Redo: (13) att(xssTheft, sp(service, oAuth)) ? creep
   Call: (14) vuln(autoAuth, idp(_6300, oAuth)) ? creep
   Exit: (14) vuln(autoAuth, idp(identity, oAuth)) ? creep
   Call: (14) int(forcedLog, login(_6306, _6308, service)) ? creep
   Call: (15) att(forceLogCsrf, login(_6306, _6308, service)) ? creep
   Call: (16) vuln(autoAuth, idp(_6322, _6324)) ? creep
   Exit: (16) vuln(autoAuth, idp(identity, oAuth)) ? creep
   Call: (16) vuln(unvalidLink, sp(service, oAuth)) ? creep
   Exit: (16) vuln(unvalidLink, sp(service, oAuth)) ? creep
   Call: (16) int(maliciousCode, browser(_6308, oAuth, _6338)) ? creep
   Call: (17) att(xssExecution, browser(_6308, oAuth, _6338)) ? creep
   Call: (18) vuln(undistinguishedCode, browser(_6308, oAuth, _6338)) ? creep
   Exit: (18) vuln(undistinguishedCode, browser(aliceBrowser, oAuth, alice)) ? creep
   Call: (18) activeCon(_6372, oAuth, _6376, aliceBrowser) ? creep
   Exit: (18) activeCon(untrustedCon, oAuth, untrusted, aliceBrowser) ? creep
   Call: (18) vuln(xssStage, sp(untrusted, _6360)) ? creep
   Exit: (18) vuln(xssStage, sp(untrusted, empty)) ? creep
   Exit: (17) att(xssExecution, browser(aliceBrowser, oAuth, alice)) ? creep
   Exit: (16) int(maliciousCode, browser(aliceBrowser, oAuth, alice)) ? creep
   Exit: (15) att(forceLogCsrf, login(forceLog, aliceBrowser, service)) ? creep
   Exit: (14) int(forcedLog, login(forceLog, aliceBrowser, service)) ? creep
   Call: (14) int(malRed, actCon(_6364, oAuth, service, aliceBrowser)) ? creep
   Call: (15) att(forcedRed, actCon(_6364, oAuth, service, aliceBrowser)) ? creep
   Call: (16) int(maliciousCode, browser(aliceBrowser, oAuth, _6388)) ? creep
   Call: (17) att(xssExecution, browser(aliceBrowser, oAuth, _6388)) ? creep
   Call: (18) vuln(undistinguishedCode, browser(aliceBrowser, oAuth, _6388)) ? creep
   Exit: (18) vuln(undistinguishedCode, browser(aliceBrowser, oAuth, alice)) ? creep
   Call: (18) activeCon(_6422, oAuth, _6426, aliceBrowser) ? creep
   Exit: (18) activeCon(untrustedCon, oAuth, untrusted, aliceBrowser) ? creep
   Call: (18) vuln(xssStage, sp(untrusted, _6410)) ? creep
   Exit: (18) vuln(xssStage, sp(untrusted, empty)) ? creep
   Exit: (17) att(xssExecution, browser(aliceBrowser, oAuth, alice)) ? creep
   Exit: (16) int(maliciousCode, browser(aliceBrowser, oAuth, alice)) ? creep
   Call: (16) sp(service, oAuth) ? creep
   Exit: (16) sp(service, oAuth) ? creep
   Exit: (15) att(forcedRed, actCon(malRed, oAuth, service, aliceBrowser)) ? creep
   Exit: (14) int(malRed, actCon(malRed, oAuth, service, aliceBrowser)) ? creep
   Call: (14) fedCon(_6428, oAuth, service, identity) ? creep
   Exit: (14) fedCon(federation, oAuth, service, identity) ? creep
   Exit: (13) att(xssTheft, sp(service, oAuth)) ? creep
   Exit: (12) int(stealTokenScript, sp(service, oAuth)) ? creep
   Call: (12) fedCon(_6428, oAuth, service, _6266) ? creep
   Exit: (12) fedCon(federation, oAuth, service, identity) ? creep
   Call: (12) acc(_6258, oAuth, _6262, service, identity) ? creep
   Exit: (12) acc(aliceSpAcc, oAuth, alice, service, identity) ? creep
   Exit: (11) att(impersonation, acc(aliceSpAcc, oAuth, alice, service, identity)) ? creep
   Exit: (10) int(impersonatedAcc, acc(aliceSpAcc, oAuth, alice, service, identity)) ? creep
   Exit: (9) err(accCompromise, fim(oAuth)) ? creep
   Exit: (8) fail(cia, fim(oAuth)) ? creep
true ;
```

Figure B.1 First part of a trace with the system model B.1, rulesets B.3; B.4; B.5, and query 'fail(cia,fim(oAuth)).'.

```
   Redo: (12) acc(_6258, oAuth, _6262, service, identity) ? creep
   Fail: (12) acc(_6258, oAuth, _6262, service, identity) ? creep
   Redo: (14) int(forcedLog, login(_6306, _6308, service)) ? creep
   Call: (15) att(sessionSwapping, login(_6306, _6308, service)) ? creep
   Call: (16) vuln(contBind, sp(service, _6324)) ? creep
   Exit: (16) vuln(contBind, sp(service, oAuth)) ? creep
   Call: (16) int(maliciousCode, browser(_6308, oAuth, _6332)) ? creep
   Call: (17) att(xssExecution, browser(_6308, oAuth, _6332)) ? creep
   Call: (18) vuln(undistinguishedCode, browser(_6308, oAuth, _6332)) ? creep
   Exit: (18) vuln(undistinguishedCode, browser(aliceBrowser, oAuth, alice)) ? creep
   Call: (18) activeCon(_6366, oAuth, _6370, aliceBrowser) ? creep
   Exit: (18) activeCon(untrustedCon, oAuth, untrusted, aliceBrowser) ? creep
   Call: (18) vuln(xssStage, sp(untrusted, _6354)) ? creep
   Exit: (18) vuln(xssStage, sp(untrusted, empty)) ? creep
   Exit: (17) att(xssExecution, browser(aliceBrowser, oAuth, alice)) ? creep
   Exit: (16) int(maliciousCode, browser(aliceBrowser, oAuth, alice)) ? creep
   Call: (16) acc(attackerAcc, oAuth, empty, service, _6380) ? creep
   Exit: (16) acc(attackerAcc, oAuth, empty, service, empty) ? creep
   Exit: (15) att(sessionSwapping, login(maliciousSpLogin, aliceBrowser, service)) ? creep
   Exit: (14) int(forcedLog, login(maliciousSpLogin, aliceBrowser, service)) ? creep
   Call: (14) int(malRed, actCon(_6358, oAuth, service, aliceBrowser)) ? creep
   Call: (15) att(forcedRed, actCon(_6358, oAuth, service, aliceBrowser)) ? creep
   Call: (16) int(maliciousCode, browser(aliceBrowser, oAuth, _6382)) ? creep
   Call: (17) att(xssExecution, browser(aliceBrowser, oAuth, _6382)) ? creep
   Call: (18) vuln(undistinguishedCode, browser(aliceBrowser, oAuth, _6382)) ? creep
   Exit: (18) vuln(undistinguishedCode, browser(aliceBrowser, oAuth, alice)) ? creep
   Call: (18) activeCon(_6416, oAuth, _6420, aliceBrowser) ? creep
   Exit: (18) activeCon(untrustedCon, oAuth, untrusted, aliceBrowser) ? creep
   Call: (18) vuln(xssStage, sp(untrusted, _6404)) ? creep
   Exit: (18) vuln(xssStage, sp(untrusted, empty)) ? creep
   Exit: (17) att(xssExecution, browser(aliceBrowser, oAuth, alice)) ? creep
   Exit: (16) int(maliciousCode, browser(aliceBrowser, oAuth, alice)) ? creep
   Call: (16) sp(service, oAuth) ? creep
   Exit: (16) sp(service, oAuth) ? creep
   Exit: (15) att(forcedRed, actCon(malRed, oAuth, service, aliceBrowser)) ? creep
   Exit: (14) int(malRed, actCon(malRed, oAuth, service, aliceBrowser)) ? creep
   Call: (14) fedCon(_6422, oAuth, service, identity) ? creep
   Exit: (14) fedCon(federation, oAuth, service, identity) ? creep
   Exit: (13) att(xssTheft, sp(service, oAuth)) ? creep
   Exit: (12) int(stealTokenScript, sp(service, oAuth)) ? creep
   Call: (12) fedCon(_6422, oAuth, service, _6266) ? creep
   Exit: (12) fedCon(federation, oAuth, service, identity) ? creep
   Call: (12) acc(_6258, oAuth, _6262, service, identity) ? creep
   Exit: (12) acc(aliceSpAcc, oAuth, alice, service, identity) ? creep
   Exit: (11) att(impersonation, acc(aliceSpAcc, oAuth, alice, service, identity)) ? creep
   Exit: (10) int(impersonatedAcc, acc(aliceSpAcc, oAuth, alice, service, identity)) ? creep
   Exit: (9) err(accCompromise, fim(oAuth)) ? creep
   Exit: (8) fail(cia, fim(oAuth)) ? creep
true ;
   Redo: (12) acc(_6258, oAuth, _6262, service, identity) ? creep
   Fail: (12) acc(_6258, oAuth, _6262, service, identity) ? creep
   Fail: (11) att(impersonation, acc(_6258, oAuth, _6262, _6264, _6266)) ? creep
   Fail: (10) int(impersonatedAcc, acc(_6258, oAuth, _6262, _6264, _6266)) ? creep
   Fail: (9) err(accCompromise, fim(oAuth)) ? creep
   Fail: (8) fail(cia, fim(oAuth)) ? creep
false.
```

Figure B.2 Remainder part of a trace with the system model B.1, rulesets B.3; B.4; B.5, and query 'fail(cia,fim(oAuth)).'.

```
   Call: (8) fail(cia, fim(fbFim)) ? creep
   Call: (9) err(accCompromise, fim(fbFim)) ? creep
   Call: (10) int(loginBackdoor, login(attackerLogin, empty, _5114)) ? creep
   Call: (11) att(backdoorAttempt, login(attackerLogin, empty, _5114)) ? creep
   Call: (12) int(accountBackdoor, acc(_5114, _5128, _5130, _5132, _5134)) ? creep
   Call: (13) att(accountLinkage, acc(_5114, _5128, _5130, _5132, _5134)) ? creep
   Call: (14) vuln(linkageOption, sp(_5132, _5128)) ? creep
   Exit: (14) vuln(linkageOption, sp(spotify, fbFim)) ? creep
   Call: (14) login(_5170, _5172, _5114) ? creep
   Exit: (14) login(aliceLogin, spotifyApp, spotifyAcc) ? creep
   Call: (14) vuln(idpReAuth, idp(_5134, fbFim)) ? creep
   Exit: (14) vuln(idpReAuth, idp(facebook, fbFim)) ? creep
   Call: (14) acc(attackerAcc, fbFim, empty, empty, facebook) ? creep
   Exit: (14) acc(attackerAcc, fbFim, empty, empty, facebook) ? creep
   Call: (14) vuln(insecureBrowser, browser(spotifyApp, fbFim, _5130)) ? creep
   Exit: (14) vuln(insecureBrowser, browser(spotifyApp, fbFim, alice)) ? creep
   Exit: (13) att(accountLinkage, acc(spotifyAcc, fbFim, alice, spotify, facebook)) ? creep
   Exit: (12) int(accountBackdoor, acc(spotifyAcc, fbFim, alice, spotify, facebook)) ? creep
   Call: (12) vuln(noLoginCheck, sp(spotify, fbFim)) ? creep
   Exit: (12) vuln(noLoginCheck, sp(spotify, fbFim)) ? creep
   Exit: (11) att(backdoorAttempt, login(attackerLogin, empty, spotifyAcc)) ? creep
   Exit: (10) int(loginBackdoor, login(attackerLogin, empty, spotifyAcc)) ? creep
   Exit: (9) err(accCompromise, fim(fbFim)) ? creep
   Exit: (8) fail(cia, fim(fbFim)) ? creep
true.
```

Figure B.3 Trace with the system model B.2, rulesets B.6; B.7, and query 'fail(cia,fim(fbFim)).'.

Line Number	Goal	Condition
1	att(xssExecution,browser(A,B,C))	vuln(undistinguishedCode,browser(A,B,C)), activeCon(_,B,D,A),vuln(xssStage,sp(D,_))
2	att(forcedRed,actCon(malRed,A,B,C))	int(maliciousCode,browser(C,A,_)),sp(B,A)
3	int(maliciousCode,browser(A,B,C))	att(xssExecution,browser(A,B,C))
4	int(malRed,actCon(A,B,C,D))	att(forcedRed,actCon(A,B,C,D))
5	err(maliciousCodeExecution,browser(A,B,C))	int(maliciousCode,browser(A,B,C))
6	err(browserEndpointTrust,browser(A,B,_))	int(maliciousCode,browser(C,A,_)),sp(B,A)
7	fail(integrity,browser(A,B,C))	err(maliciousCodeExecution,browser(A,B,C)), err(browserEndpointTrust,browser(A,B,C))

Table B.3 Prolog Ruleset for Browser attacks on OAuth adapted from Sun and Beznosov (2012)

Line Number	Goal	Condition
1	att(forceLogCsrf,login(forceLog,A,B))	vuln(autoAuth,idp(_,C)),vuln(unvalidLink,sp(B,C)),int(maliciousCode, browser(A,C,_))
2	att(sessionSwapping,login(maliciousSpLogin, A,B))	vuln(contBind,sp(B,C)),int(maliciousCode,browser(A,C,_)),acc(attackerAcc,C,empty,B,_)
3	att(xssTheft,sp(A,B))	vuln(autoAuth,idp(C,B)),login(_,_,D),acc(D,B,_,A,C)
4	att(xssTheft,sp(A,B))	vuln(autoAuth,idp(C,B)),int(forcedLog,login(_,D,A)),int(malRed,actCon(_,B,A,D)),fedCon(_,B,A,C)
5	int(forcedLog,login(A,B,C))	att(forceLogCsrf,login(A,B,C))
6	int(forcedLog,login(A,B,C))	att(sessionSwapping,login(A,B,C))
7	int(stealTokenScript,sp(A,B))	att(xssTheft,sp(A,B))
8	err(unauthAction,sp(A,B))	int(forcedLog,login(_,_,A))
9	err(resourceDisclosure,sp(A,B))	int(stealTokenScript,sp(A,B))
10	fail(integrity,sp(A,B))	err(unauthAction,sp(A,B))
11	fail(confidentiality,sp(A,B))	err(resourceDisclosure,sp(A,B))

Table B.4 Prolog Ruleset for SP attacks on OAuth adapted from Sun and Beznosov (2012)

Line Number	Goal	Condition
1	att(impersonation,acc(A,B,C,D,E))	vuln(contBind,sp(D,B)),int(stealTokenScript,sp(D,B)),fedCon(_,B,D,E),acc(A,B,C,D,E)
2	int(impersonatedAcc,acc(A,B,C,D,E))	att(impersonation,acc(A,B,C,D,E))
3	err(accCompromise,fim(A))	int(impersonatedAcc,acc(_,A,_,_,_))
4	fail(cia,fim(A))	err(accCompromise,fim(A))

Table B.5 Prolog Ruleset for Overall FIM attacks on OAuth adapted from Sun and Beznosov (2012)

Line Number	Goal	Condition
1	att(accountLinkage,acc(A,B,C,D,E))	vuln(linkageOption,sp(D,B)),login(_- ,F,A),vuln(idpReAuth,idp(E,B)),acc(attackerAcc, B,empty,empty,E),vuln(insecureBrowser,browser(F,B,C))
2	int(accountBackdoor,acc(A,B,C,D,E))	att(accountLinkage,acc(A,B,C,D,E))
3	err(unauthAction,browser(A,B))	int(accountBackdoor,acc(_,B,_,A,_))
4	fail(integrity,browser(A,B))	err(unauthAction,browser(A,B))

Table B.6 Prolog Ruleset for Spotify attacks on Facebook Account Linkage attack introduced in section 4.8.

Line Number	Goal	Condition
1	att(backdoorAttempt,login(attackerLogin, empty,A))	int(accountBackdoor,acc(A,B,_,C,_)), vuln(noLoginCheck,sp(C,B))
2	int(loginBackdoor,login(attackerLogin, empty,A))	att(backdoorAttempt,login(attackerLogin,empty,A))
3	err(accCompromise,fim(_))	int(loginBackdoor,login(attackerLogin,empty,_-))
4	fail(cia,fim(A))	err(accCompromise,fim(A))

Table B.7 Prolog Ruleset for Overall attacks on Facebook Account Linkage attack introduced in section 4.8.

References

Abbas, H., Qaemi Mahmoodzadeh, M., Aslam Khan, F., and Pasha, M. (2016). Identifying an OpenID anti-phishing scheme for cyberspace. *Security and Communication Networks*, 9(6):481–491.

Adams, A. and Sasse, M. A. (1999). Users are not the enemy. *Communications of the ACM*, 42(12):41–46.

Agrawall, A., Maheshwari, S., Bandyopadhyay, P., and Choppella, V. (2017). Modelling and Mitigation of Cross-Origin Request Attacks on Federated Identity Management Using Cross Origin Request Policy. In *International Conference on Information Systems Security*, pages 263–282. Springer.

Ahmad, Z., Ab Manan, J.-L., and Sulaiman, S. (2010). Trusted computing based open environment user authentication model. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, volume 6, pages V6–487. IEEE.

Alotaibi, A. and Mahmmod, A. (2015). Enhancing OAuth services security by an authentication service with face recognition. In *Systems, Applications and Technology Conference (LISAT), 2015 IEEE Long Island*, pages 1–6. IEEE.

Alrodhan, W. and Mitchell, C. (2009). Improving the security of cardspace. *EURASIP journal on information security*, 2009:1–8.

Amoroso, E. G. (1994). *Fundamentals of computer security technology*. Prentice-Hall, Inc.

Andress, J. (2014). *The basics of information security: understanding the fundamentals of InfoSec in theory and practice*, 2nd ed. Syngress.

Anicas, M. (2014). An Introduction to OAuth 2. Retrieved from https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2.

Argyriou, M., Dragoni, N., and Spognardi, A. (2017). Security flows in OAuth 2.0 framework: a case study. In *International Conference on Computer Safety, Reliability, and Security*, pages 396–406. Springer.

Arias-Cabarcos, P., Almenárez-Mendoza, F., Marín-López, A., Díaz-Sánchez, D., and Sánchez-Guerrero, R. (2012). A metric-based approach to assess risk for "on cloud" federated identity management. *Journal of Network and Systems Management*, 20(4):513–533.

Armando, A., Carbone, R., Compagna, L., Cuellar, J., and Tobarra, L. (2008). Formal analysis of SAML 2.0 web browser single sign-on: breaking the SAML-based single sign-on for google apps. In *Proceedings of the 6th ACM workshop on Formal methods in security engineering*, pages 1–10. ACM.

Avizienis, A., Laprie, J.-C., and Randell, B. (2001). *Fundamental concepts of dependability*. University of Newcastle upon Tyne, Computing Science.

Baldwin, A., Casassa Mont, M., Beres, Y., and Shiu, S. (2010). Assurance for federated identity management. *Journal of Computer Security*, 18(4):541–572.

Bartsch, M. and Dienlin, T. (2016). Control your Facebook: An analysis of online privacy literacy. *Computers in Human Behavior*, 56:147–154.

Bau, J. and Mitchell, J. C. (2011). Security modeling and analysis. *IEEE Security & Privacy*, 9(3):18–25.

BBC (2017). Cyber-attack: Europol says it was unprecedented in scale. Retrieved from https://www.bbc.co.uk/news/world-europe-39907965.

Bell, D. E. and La Padula, L. J. (1976). Secure computer system: Unified exposition and multics interpretation. Technical report, MITRE Corp Bedford MA.

Birkholz, H., Edelkamp, S., Junge, F., and Sohr, K. (2010). Efficient automated generation of attack trees from vulnerability databases. In *Working Notes for the 2010 AAAI Workshop on Intelligent Security (SecArt)*, pages 47–55.

Bishop, M. (1995). A taxonomy of unix system and network vulnerabilities. Technical report, Technical Report CSE-95-10, Department of Computer Science, University of California at Davis.

Blanchet, B. (2001). An efficient cryptographic protocol verifier based on prolog rules. In *CSFW*, volume 1, pages 82–96.

Bonneau, J., Herley, C., Van Oorschot, P. C., and Stajano, F. (2012). The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 553–567. IEEE.

Brewer, D. F. and Nash, M. J. (1989). The chinese wall security policy. In *Proceedings. 1989 IEEE Symposium on Security and Privacy*, pages 206–214. IEEE.

Cambiaso, E., Papaleo, G., Chiola, G., and Aiello, M. (2013). Slow DoS attacks: definition and categorisation. *International Journal of Trust Management in Computing and Communications*, 1(3-4):300–319.

Canetti, R. (2001). Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE.

Cantor, S., Hodges, J., Kemp, J., and Thompson, P. (2003). Liberty id-ff architecture overview. *Wason, Thomas (Herausgeber): Liberty Alliance Project Version*, 1.

Chadwick, D. W. (2009). Federated identity management. In *Foundations of security analysis and design V*, pages 96–120. Springer.

Chari, S., Jutla, C. S., and Roy, A. (2011). Universally Composable Security Analysis of OAuth v2. 0. *IACR Cryptology ePrint Archive*, 2011:526.

Chauhan, A. and Tiwari, V. K. (2018). A Trusted Computing Solution for Security Threats in Federated Identity Management. *American Research Journal of Computer Science and Information Technology*, 3(1):1–8.

Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czeskis, A., Roesner, F., and Kohno, T. (2011). Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*, volume 4, pages 447–462. San Francisco.

Chen, H., Dean, D., and Wagner, D. A. (2004). Model Checking One Million Lines of C Code. In *NDSS*, volume 4, pages 171–185.

Chiu, G. and Chan, B. (2006). A Client-Side Browser-Integrated Solution for Detecting and Preventing Cross Site Scripting (XSS) Attacks. *Citeseer*.

Christey, S. and Martin, R. A. (2007). Vulnerability type distributions in CVE. *Mitre report, May*.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to algorithms*, 3rd ed. MIT press.

Cremers, C. and Mauw, S. (2012). Security properties. In *Operational Semantics and Verification of Security Protocols*, pages 37–65. Springer.

Denning, D. E. (1987). An intrusion-detection model. *IEEE Transactions on software engineering*, SE-13(2):222–232.

Depoy, J., Phelan, J., Sholander, P., Smith, B., Varnado, G. B., and Wyss, G. (2005). Risk assessment for physical and cyber attacks on critical infrastructures. In *MILCOM 2005 IEEE Military Communications Conference*, pages 1961–1969. IEEE.

Dewri, R., Ray, I., Poolsappasit, N., and Whitley, D. (2012). Optimal security hardening on attack tree models of networks: a cost-benefit analysis. *International Journal of Information Security*, 11(3):167–188.

Dolev, D. and Yao, A. (1983). On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208.

Dwyer, C., Hiltz, S., and Passerini, K. (2007). Trust and privacy concern within social networking sites: A comparison of Facebook and MySpace. *AMCIS 2007 proceedings*, page 339.

Engelbertz, N., Erinola, N., Herring, D., Somorovsky, J., Mladenov, V., and Schwenk, J. (2018). Security analysis of eidas–the cross-country authentication scheme in europe. In *12th USENIX Workshop on Offensive Technologies (WOOT) 2018*.

Facebook (n.d.). Facebook Login for the Web with the JavaScript SDK. Retrieved from https://developers.facebook.com/docs/facebook-login/web/.

Feld, S. and Pohlmann, N. (2011). Security analysis of OpenID, followed by a reference implementation of an nPA-based OpenID provider. In *ISSE 2010 Securing Electronic Business Processes*, pages 13–25. Springer.

Ferry, E., O Raw, J., and Curran, K. (2015). Security evaluation of the OAuth 2.0 framework. *Information & Computer Security*, 23(1):73–101.

Fett, D., Hosseyni, P., and Küsters, R. (2019). An extensive formal security analysis of the openid financial-grade api. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 453–471. IEEE.

Fette, I., Sadeh, N., and Tomasic, A. (2007). Learning to detect phishing emails. In *Proceedings of the 16th international conference on World Wide Web*, pages 649–656.

Fraile, M., Ford, M., Gadyatskaya, O., Kumar, R., Stoelinga, M., and Trujillo-Rasua, R. (2016). Using attack-defense trees to analyze threats and countermeasures in an ATM: a case study. In *IFIP Working Conference on The Practice of Enterprise Modeling*, pages 326–334. Springer.

Gadyatskaya, O., Jhawar, R., Mauw, S., Trujillo-Rasua, R., and Willemse, T. A. (2017). Refinement-aware generation of attack trees. In *International Workshop on Security and Trust Management*, pages 164–179. Springer.

Gadyatskaya, O. and Trujillo-Rasua, R. (2017). New directions in attack tree research: Catching up with industrial needs. In *International Workshop on Graphical Models for Security*, pages 115–126. Springer.

Gajek, S., Schwenk, J., Steiner, M., and Xuan, C. (2009). Risks of the cardspace protocol. In *International Conference on Information Security*, pages 278–293. Springer.

Gallon, L. and Bascou, J. J. (2011). Using CVSS in attack graphs. In *2011 Sixth International Conference on Availability, Reliability and Security*, pages 59–66. IEEE.

Gaw, S. and Felten, E. W. (2006). Password management strategies for online accounts. In *Proceedings of the second symposium on Usable privacy and security*, pages 44–55.

Ghazizadeh, E., Zamani, M., Ab Manan, J., and Pashang, A. (2012). A survey on security issues of federated identity in the cloud computing. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, pages 532–565. IEEE.

Gomi, H. (2010). An authentication trust metric for federated identity management systems. In *International Workshop on Security and Trust Management*, pages 116–131. Springer.

Goodner, M., Hondo, M., Nadalin, A., McIntosh, M., and Schmidt, D. (2007). Understanding ws-federation. *Microsoft and IBM*.

Google (n.d.a). Integrating Google Sign-In into your web app. Retrieved from https://developers.google.com/identity/sign-in/web/sign-in.

Google (n.d.b). OpenID Connect. Retrieved from https://developers.google.com/identity/protocols/oauth2/openid-connect.

Google (n.d.c). Public DNS. Retrieved from https://developers.google.com/speed/public-dns.

Groß, T. (2003). Security analysis of the SAML single sign-on browser/artifact profile. In *19th Annual Computer Security Applications Conference, 2003. Proceedings.*, pages 298–307. IEEE.

Groß, T. and Pfitzmann, B. (2004). Proving a WS-Federation passive requestor profile. In *Proceedings of the 2004 workshop on Secure web service*, pages 77–86.

Gruhn, M. and Müller, T. (2013). On the practicability of cold boot attacks. In *2013 International Conference on Availability, Reliability and Security*, pages 390–397. IEEE.

Grzonkowski, S., Corcoran, P. M., and Coughlin, T. (2011). Security analysis of authentication protocols for next-generation mobile and CE cloud services. In *2011 IEEE International Conference on Consumer Electronics-Berlin (ICCE-Berlin)*, pages 83–87. IEEE.

Gupta, S. and Gupta, B. (2015). BDS: browser dependent XSS sanitizer. In *Handbook of Research on Securing Cloud-Based Databases with Biometric Applications*, pages 174–191. IGI Global.

Hansman, S. and Hunt, R. (2005). A taxonomy of network and computer attacks. *Computers & Security*, 24(1):31–43.

Hardt, D. (2012). The OAuth 2.0 Authorization Framework. RFC 6749, RFC Editor.

Herley, C. and Van Oorschot, P. (2012). A research agenda acknowledging the persistence of passwords. *IEEE Security & Privacy*, 10(1):28–36.

Hoellrigl, T., Dinger, J., and Hartenstein, H. (2010). A consistency model for identity information in distributed systems. In *2010 IEEE 34th Annual Computer Software and Applications Conference*, pages 252–261. IEEE.

Howard, J. D. and Longstaff, T. A. (1998). A common language for computer security incidents. Technical report, Sandia National Labs., Albuquerque, NM (US).

Hsu, F., Chen, H., and Machiraju, S. (2011). WebCallerID: Leveraging cellular networks for Web authentication. *Journal of Computer Security*, 19(5):869–893.

Ingols, K., Lippmann, R., and Piwowarski, K. (2006). Practical attack graph generation for network defense. In *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*, pages 121–130. IEEE.

Isaac, M. and Frenkel, S. (2018). Facebook Security Breach Exposes Accounts of 50 Million Users. Retrieved from https://www.nytimes.com/2018/09/28/technology/facebook-hack-data-breach.html.

ITU-T (2009a). Baseline capabilities for enhanced global identity management and interoperability. Retrieved from https://www.itu.int/rec/T-REC-X.1250-200909-I/en.

ITU-T (2009b). NGN identity management framework. Retrieved from https://www.itu.int/rec/T-REC-Y.2720-200901-I.

Ives, B., Walsh, K. R., and Schneider, H. (2004). The domino effect of password reuse. *Communications of the ACM*, 47(4):75–78.

Jenkins, J. L., Grimes, M., Proudfoot, J. G., and Lowry, P. B. (2014). Improving password cybersecurity through inexpensive and minimally invasive means: Detecting and deterring password reuse through keystroke-dynamics monitoring and just-in-time fear appeals. *Information Technology for Development*, 20(2):196–213.

Jensen, J. (2012). Federated identity management challenges. In *2012 Seventh International Conference on Availability, Reliability and Security*, pages 230–235. IEEE.

Jhawar, R., Lounis, K., Mauw, S., and Ramírez-Cruz, Y. (2018). Semi-automatically augmenting attack trees using an annotated attack tree library. In *International Workshop on Security and Trust Management*, pages 85–101. Springer.

Jøsang, A., Fabre, J., Hay, B., Dalziel, J., and Pope, S. (2005). Trust requirements in identity management. In *Proceedings of the 2005 Australasian workshop on Grid computing and e-research-Volume 44*, pages 99–108. Australian Computer Society, Inc.

Joy Persial, G., Prabhu, M., and Shanmugalakshmi, R. (2011). Side channel attack-survey. *Int. J. Adv. Sci. Res. Rev*, 1(4):54–57.

Juels, A. and Rivest, R. L. (2013). Honeywords: Making password-cracking detectable. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 145–160. ACM.

Kali (2021). Penetration Testing and Ethical Hacking Linux Distribution. Retrieved fromhttps://www.kali.org/.

Kelley, P. G., Komanduri, S., Mazurek, M. L., Shay, R., Vidas, T., Bauer, L., Christin, N., Cranor, L. F., and Lopez, J. (2012). Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In *2012 IEEE symposium on security and privacy*, pages 523–537. IEEE.

Kitchenham, B. (2004). Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33(2004):1–26.

Kordy, B., Mauw, S., Radomirović, S., and Schweitzer, P. (2010). Foundations of attack–defense trees. In *International Workshop on Formal Aspects in Security and Trust*, pages 80–95. Springer.

Kordy, B., Pièvre-Cambacédès, L., and Schweitzer, P. (2014). DAG-based attack and defense modeling: Don't miss the forest for the attack trees. *Computer Science Review*, 13:1–38.

Kormann, D. P. and Rubin, A. D. (2000). Risks of the passport single signon protocol. *Computer Networks*, 33(1-6):51–58.

Krasnova, H., Eling, N., Abramova, O., and Buxmann, P. (2014). Dangers of 'Facebook Login' for Mobile Apps: Is There a Price Tag for Social Information? In *ICIS*.

Krolo, J., Šilić, M., and Srbljić, S. (2009). Security of web level user identity management. In *Proceedings of the 32nd International Convention Proceedings: Digital Economy-6th ALADIN, Information Systems Security, Business Intelligence Systems, Local Government and Student Papers*, pages 93–98. Citeseer.

Kumar, A. (2014). A lightweight formal approach for analyzing security of web protocols. In *International Workshop on Recent Advances in Intrusion Detection*, pages 192–211. Springer.

Kumar, R., Ruijters, E., and Stoelinga, M. (2015). Quantitative attack tree analysis via priced timed automata. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 156–171. Springer.

Landwehr, C. E., Bull, A. R., McDermott, J. P., and Choi, W. S. (1994). A taxonomy of computer program security flaws. *ACM Computing Surveys (CSUR)*, 26(3):211–254.

Laprie, J.-C. (1992). Dependability: Basic concepts and terminology. In *Dependability: Basic Concepts and Terminology*, pages 3–245. Springer.

Le, T., Kim, J., and Kim, H. (2017). An effective intrusion detection classifier using long short-term memory with gradient descent optimization. In *2017 International Conference on Platform Technology and Service (PlatCon)*, pages 1–6. IEEE.

Leiba, B. (2012). Oauth web authorization protocol. *IEEE Internet Computing*, 16(1):74–77.

Leuschel, M. and Butler, M. (2003). ProB: A model checker for B. In *International Symposium of Formal Methods Europe*, pages 855–874. Springer.

Li, W. and Mitchell, C. J. (2014). Security issues in OAuth 2.0 SSO implementations. In *International Conference on Information Security*, pages 529–541. Springer.

Li, W. and Mitchell, C. J. (2016). Analysing the Security of Google's implementation of OpenID Connect. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 357–376. Springer.

Lindqvist, U. and Jonsson, E. (1997). How to systematically classify computer security intrusions. In *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No. 97CB36097)*, pages 154–163. IEEE.

Lloyd, J. W. (2012). *Foundations of logic programming*. Springer Science & Business Media.

Lodderstedt, T., McGloin, M., and Hunt, P. (2013). OAuth 2.0 threat model and security considerations. RFC 6819, RFC Editor.

Mainka, C., Mladenov, V., Feldmann, F., Krautwald, J., and Schwenk, J. (2014). Your software at my service: Security analysis of saas single sign-on solutions in the cloud. In *Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security*, pages 93–104. ACM.

Mainka, C., Mladenov, V., and Schwenk, J. (2016). Do not trust me: Using malicious IdPs for analyzing and attacking Single Sign-On. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 321–336. IEEE.

Maler, E. and Reed, D. (2008). The venn of identity: Options and issues in federated identity management. *IEEE Security & Privacy*, 6(2):16–23.

Martin, B., Brown, M., Paller, A., Kirby, D., and Christey, S. (2011). 2011 CWE/SANS top 25 most dangerous software errors. *Common Weakness Enumeration*, 7515.

Mauw, S. and Oostdijk, M. (2005). Foundations of attack trees. In *International Conference on Information Security and Cryptology*, pages 186–198. Springer.

Maxion, R. A. and Townsend, T. N. (2002). Masquerade detection using truncated command lines. In *Proceedings International Conference on Dependable Systems and Networks*, pages 219–228. IEEE.

Mayer, A., Niemietz, M., Mladenov, V., and Schwenk, J. (2014). Guardians of the clouds: when identity providers fail. In *Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security*, pages 105–116. ACM.

Mayer, R. C., Davis, J. H., and Schoorman, F. D. (1995). An integrative model of organizational trust. *Academy of Management Review*, 20(3):709–734.

Miculan, M. and Urban, C. (2011). Formal analysis of Facebook Connect single sign-on authentication protocol. In *SOFSEM*, volume 11, pages 22–28. Citeseer.

Naik, N. and Jenkins, P. (2017). Securing digital identities in the cloud by selecting an apposite Federated Identity Management from SAML, OAuth and OpenID Connect. In *2017 11th International Conference on Research Challenges in Information Science (RCIS)*, pages 163–174. IEEE.

Necula, G. C. (1997). Proof-carrying code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 106–119.

NIST (n.d.). CSRC Glossary. Retrieved from https://csrc.nist.gov/glossary.

Nordbotten, N. A. (2009). XML and web services security standards. *IEEE Communications Surveys & Tutorials*, 11(3):4–21.

Oasis (2008). Security Assertion Markup Language (SAML) V2.0 Technical Overview. Retrieved from http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html.

Oasis (2009). Web Services Federation Language (WS-Federation) Version 1.2. Retrieved from http://docs.oasis-open.org/wsfed/federation/v1.2/ws-federation.html.

OAuth (n.d.). Access Tokens. Retrieved from https://www.oauth.com/oauth2-servers/access-tokens/.

Oh, H.-K. and Jin, S.-H. (2008). The security limitations of sso in openid. In *Advanced Communication Technology, 2008. ICACT 2008. 10th International Conference on*, volume 3, pages 1608–1611. IEEE.

OpenID (2014). Welcome to OpenID Connect. Retrieved from https://openid.net/connect/.

Oppliger, R. (2004). Microsoft. net passport and identity management. *Information Security Technical Report*, 9(1):26–34.

Ou, X., Boyer, W. F., and McQueen, M. A. (2006). A scalable approach to attack graph generation. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, pages 336–345. ACM.

OWASP (2019a). Cross-Site Request Forgery (CSRF). Retrieved from https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF).

OWASP (2019b). Cross-site Scripting (XSS). Retrieved from https://www.owasp.org/index.php/Cross-site_Scripting_(XSS).

Pai, S., Sharma, Y., Kumar, S., Pai, R. M., and Singh, S. (2011). Formal verification of OAuth 2.0 using Alloy framework. In *2011 International Conference on Communication Systems and Network Technologies*, pages 655–659. IEEE.

Pfitzmann, B. and Waidner, M. (2003a). Analysis of liberty single-sign-on with enabled clients. *IEEE Internet Computing*, 7(6):38–44.

Pfitzmann, B. and Waidner, M. (2003b). Federated identity-management protocols. In *International Workshop on Security Protocols*, pages 153–174. Springer.

Phillips, C. and Swiler, L. P. (1998). A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 workshop on New security paradigms*, pages 71–79. ACM.

Portswigger (n.d.). Clickjacking (UI redressing). Retrieved from https://portswigger.net/web-security/clickjacking.

Powell, D. and Stroud, R. (2003). Conceptual model and architecture of MAFTIA. *Technical Report Series-University of Newcastle Upon Tyne Computing Science*.

Rieger, S. (2009). User-centric identity management in heterogeneous federations. In *2009 Fourth International Conference on Internet and Web Applications and Services*, pages 527–532. IEEE.

Robertson, D. S. (1998). Unification. Retrieved from http://www.dai.ed.ac.uk/groups/ssp/bookpages/quickprolog/node12.html.

Roesch, M. (1999). Snort: Lightweight intrusion detection for networks. In *Lisa*, volume 99, pages 229–238.

Sandhu, R. S. and Samarati, P. (1994). Access control: principle and practice. *IEEE communications magazine*, 32(9):40–48.

Scambray, J., McClure, S., and Kurtz, G. (2000). *Hacking exposed*. McGraw-Hill Professional.

Schneier, B. (1999). Attack Trees. Retrieved from https://www.schneier.com/academic/archives/1999/12/attack_trees.html.

Shernan, E., Carter, H., Tian, D., Traynor, P., and Butler, K. (2015). More guidelines than rules: Csrf vulnerabilities from noncompliant oauth 2.0 implementations. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 239–260. Springer.

Sheyner, O., Haines, J., Jha, S., Lippmann, R., and Wing, J. M. (2002). Automated generation and analysis of attack graphs. In *Proceedings 2002 IEEE Symposium on Security and Privacy*, pages 273–284. IEEE.

Shibboleth (n.d.). Shibboleth Wiki. Retrieved from https://wiki.shibboleth.net/confluence/#all-updates.

Simpson, S. and Groß, T. (2016). A survey of security analysis in federated identity management. In *IFIP International Summer School on Privacy and Identity Management*, pages 231–247. Springer.

Smith, D. (2008). The challenge of federated identity management. *Network Security*, 2008(4):7–9.

Sovis, P., Kohlar, F., and Schwenk, J. (2010). Security Analysis of OpenID. In *Sicherheit 2010. Sicherheit, Schutz und Zuverlässigkeit*, pages 329–340.

Speltens, M. and Patterson, P. (2007). Federated ID Management—Tackling Risk and Credentialing Users. In *ISSE/SECURE 2007 Securing Electronic Business Processes*, pages 130–135. Springer.

Spotify (2019). Spotify Terms and Conditions of Use. Retrieved from https://www.spotify.com/uk/legal/end-user-agreement/.

Spotify (2021). Disabled Accounts. Retrieved from https://support.spotify.com/uk/article/why-has-my-account-been-disabled/.

Sridhar, S. and Govindarasu, M. (2014). Model-based attack detection and mitigation for automatic generation control. *IEEE Transactions on Smart Grid*, 5(2):580–591.

Sun, S.-T. and Beznosov, K. (2012). The devil is in the (implementation) details: an empirical analysis of OAuth SSO systems. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 378–390. ACM.

Sun, S.-T., Hawkey, K., and Beznosov, K. (2012). Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures. *Computers & Security*, 31(4):465–483.

SWI-Prolog (n.d.). Overview of the Debugger. Retrieved from https://www.swi-prolog.org/pldoc/man?section=debugoverview.

UK Government (2019). Cyber Security Breaches Survey 2019. Retrieved from https://www.gov.uk/government/statistics/cyber-security-breaches-survey-2019.

UK Government (2021). Gov.uk verify. Retrieved from https://www.gov.uk/government/publications/introducing-govuk-verify/introducing-govuk-verify.

Urueña, M., Muñoz, A., and Larrabeiti, D. (2014). Analysis of privacy vulnerabilities in single sign-on mechanisms for multimedia websites. *Multimedia Tools and Applications*, 68(1):159–176.

Van Delft, B. and Oostdijk, M. (2010). A security analysis of OpenID. In *IFIP Working Conference on Policies and Research in Identity Management*, pages 73–84. Springer.

Vesely, W. E., Goldberg, F. F., Roberts, N. H., and Haasl, D. F. (1981). Fault tree handbook. Technical report, Nuclear Regulatory Commission Washington DC.

Vigo, R., Nielson, F., and Nielson, H. R. (2014). Automated generation of attack trees. In *2014 IEEE 27th Computer Security Foundations Symposium*, pages 337–350. IEEE.

Wang, R., Chen, S., and Wang, X. (2012). Signing me onto your accounts through facebook and google: A traffic-guided security study of commercially deployed single-sign-on web services. In *2012 IEEE Symposium on Security and Privacy*, pages 365–379. IEEE.

Wideł, W., Audinot, M., Fila, B., and Pinchinat, S. (2019). Beyond 2014: Formal Methods for Attack Tree–based Security Modeling. *ACM Computing Surveys (CSUR)*, 52(4):1–36.

Wolf, M., Thomas, I., Menzel, M., and Meinel, C. (2009). A message meta model for federated authentication in service-oriented architectures. In *2009 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 1–8. IEEE.

Yan, J., Blackwell, A., Anderson, R., and Grant, A. (2004). Password Memorability and Security: Empirical Results. *IEEE Security & Privacy*, 2(5):25–31.

Yang, F. and Manoharan, S. (2013). A security analysis of the OAuth protocol. In *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, pages 271–276. IEEE.

Yang, R., Li, G., Lau, W. C., Zhang, K., and Hu, P. (2016). Model-based security testing: an empirical study on OAuth 2.0 implementations. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pages 651–662. ACM.