

Quantitative Analysis of the Release Order of Defensive Mechanisms

Thesis by

Suliman Abdullah Alsuhibany

In Partial Fulfilment of the Requirements for the Degree of
Doctor of Philosophy



School of Computing Science, Newcastle University

Newcastle upon Tyne, UK

June 2014

ABSTRACT

Dependency on information technology (IT) and computer and information security (CIS) has become a critical concern for many organizations. This concern has essentially centred on protecting secrecy, confidentiality, integrity and availability of information. To overcome this concern, defensive mechanisms, which encompass a variety of services and protections, have been proposed to protect system resources from misuse. Most of these defensive mechanisms, such as CAPTCHAs and spam filters, rely in the first instance on a single algorithm as a defensive mechanism. Attackers would eventually break each mechanism. So, each algorithm would ultimately become useless and the system no longer protected. Although this broken algorithm will be replaced by a new algorithm, no one shed light on a set of algorithms as a defensive mechanism.

This thesis looks at a set of algorithms as a holistic defensive mechanism. Our hypothesis is that the order in which a set of defensive algorithms is released has a significant impact on the time taken by attackers to break the combined set of algorithms. The rationale behind this hypothesis is that attackers learn from their attempts, and that the release schedule of defensive mechanisms can be adjusted so as to impair the learning process. To demonstrate the correctness of our hypothesis, an experimental study involving forty participants was conducted to evaluate the effect of algorithms' order on the time taken to break them. In addition, this experiment explores how the learning process of attackers could be observed. The results showed that the order in which algorithms are released has a statistically significant impact on the time attackers take to break all algorithms. Based on these results, a model has been constructed using Stochastic Petri Nets, which facilitate theoretical analysis of the release order of a set of algorithms approach. Moreover, a tailored optimization algorithm is proposed using a Markov Decision Process model in order to obtain efficiently the optimal release strategy for any given model by maximizing the time taken to break a set of algorithms. As our hypothesis is based on the learning acquisition ability of attackers while interacting with the system, the Attacker Learning Curve (ALC) concept is developed. Based on empirical results of the ALC, an attack strategy detection approach is introduced and evaluated, which has achieved a detection success rate higher than 70%. The empirical findings in this detection approach provide a new understanding of not only how to detect the attack strategy used, but also how to track the attack strategy through the probabilities of classifying results that may provide an advantage for optimising the release order of defensive mechanisms.

ACKNOWLEDGEMENTS

This four year period of study has been a long journey with many trials and tribulations. There have been ups and downs, roadblocks and dead ends. However, at the same time, I also have gained various skills and knowledge during this period. All these are not possible without the help, guidance, and support of several people during my time here at Newcastle University.

Firstly and foremost, I would like to express my gratitude to my supervisor Professor Aad van Moorsel for his valuable guidance and support throughout the duration of my study here at Newcastle University. His comments and insight during our various discussion sessions help me to shape my research and solve many research problems. Special thanks for Dr. Peter Andras and Dr. Jeff Yan for being members of the supervision committee.

I would like also to extend my gratitude to these people individually, who has been involved in my research work both directly and indirectly namely; Dr. Charles Morisset, Dr. Chris Smith, Dr. Nur Haryani Zakaria, Ahmad Alonaizi and the rest of the participants that have took part in any of the experiment work that I have conducted; for which without the full commitment and co-operation from each of you, the research would not have been a success.

Last but not least, my special gratitude is due to my father Abdullah and my mother Nourah for always being there when I needed them most. They deserve far more credit than I can ever give them. I could not imagine the level that I have reached without their warm-heartedness, support and help. Also, I would like to sincerely thank from the bottom of my heart, my beloved wife Nourah who always provides support and love to me and my daughter Tamara who bring joy and smile at all times. Finally, I dedicate this research to my brother, and my sisters, who have supported and encouraged me.

Suliman Abdullah Alsuhibany

TABLE OF CONTENTS

LIST OF TABLES	VII
LIST OF FIGURES	VIII
GLOSSARY OF TERMS	X
CHAPTER 1. INTRODUCTION	1
1.1 CONTEXT & MOTIVATION.....	1
1.2 RESEARCH HYPOTHESIS & QUESTIONS	7
1.3 RESEARCH AIM & OBJECTIVES	8
1.4 RESEARCH METHODOLOGY	9
1.5 CONTRIBUTIONS OF THE THESIS	10
1.6 PUBLICATION HISTORY	12
1.7 STRUCTURE OF THE THESIS	13
CHAPTER 2. BACKGROUND AND LITERATURE REVIEW	15
2.1 ATTACKERS, DEFENDERS AND THEIR MOTIVES	16
2.1.1 Attackers	16
2.1.2 Defenders	17
2.1.3 Motivation of Attackers and Defenders.....	17
2.2 DEFENSIVE MECHANISM	18
2.2.1 What is a Defensive Mechanism?.....	18
2.2.2 Interactive Defensive Mechanism	18
2.2.3 Examples of Interactive Defensive Mechanisms.....	21
2.3 AN OVERVIEW OF THE LEARNING ACQUISITION PROCESS	27
2.3.1 Learning Curve Theory.....	27
2.3.2 Problem-Based Learning Approach.....	31
2.3.3 Effects of Information Order in Learning Process.....	33
2.4 SECURITY QUANTIFICATION AND ATTACK MODELLING	37
2.4.1 Security Quantification and Attacker Behavior.....	38
2.4.2 Time-To-Compromise System	41
2.4.3 Game Theoretic Security Approach	42
2.5 ANOMALY DETECTION TECHNIQUES.....	44
2.6 STOCHASTIC MODELLING FORMALISMS.....	45

2.6.1	Continuous-Time Markov Chain	48
2.6.2	Continuous-Time Markov Decision Process	49
2.6.3	Stochastic Petri Nets	51
2.6.4	Software Tools for Building and Solving SPN Models.....	53
2.7	SUMMARY	56
CHAPTER 3. EXPERIMENTAL STUDY		57
3.1	EXPERIMENT SCOPE	58
3.2	HYPOTHESES	58
3.3	EXPERIMENT SETUP	59
3.3.1	The Experimental Design	59
3.3.2	System.....	60
3.3.3	Attackers	61
3.3.4	Algorithms	62
3.3.5	Materials: stimulus and rationale.....	65
3.3.6	Variables	65
3.3.7	Measurement Units	65
3.3.8	Generalisation and Threats Validity	66
3.3.9	Avoiding Bias and Control Measures	67
3.4	EXPERIMENT PROCEDURE	68
3.4.1	Instructions to subjects.....	68
3.4.2	Procedures.....	68
3.4.3	Collected Data.....	69
3.5	RESULTS AND ANALYSIS	69
3.5.1	Testing Hypothesis: Does Order Matter?	69
3.5.2	The Influence of Order on Defeating Future Algorithms.....	72
3.5.3	The Influence of Order on Defeating All Algorithms	73
3.5.4	Attacking Process	73
3.6	DISCUSSION	74
3.7	SUMMARY	76
CHAPTER 4. MODELLING OF RELEASE ORDER STRATEGIES		78
4.1	STOCHASTIC MODEL	79
4.2	PROPOSED STOCHASTIC PETRI NET (SPN) MODEL	80
4.2.1	SPN: An Overview	80
4.2.2	Evaluation Tools	82
4.2.3	Model Assumptions	83
4.2.4	Model Metrics.....	84
4.3	MODEL DESIGN AND PERFORMANCE	84
4.3.1	Performance Metrics Calculation	87
4.3.2	Further Details for the Used Functions.....	88
4.4	CASE STUDY	90
4.5	RESULTS AND ANALYSIS.....	92
4.5.1	Replicating the Results of Release Order	92

4.5.2	Algorithms Order vs. MTTSF	93
4.5.3	Attacker's Knowledge Acquisition Process	94
4.6	DISCUSSION	95
4.7	SUMMARY	96
CHAPTER 5. OPTIMAL RELEASE ORDER STRATEGIES.....		98
5.1	DERIVING OPTIMAL RELEASE ORDER STRATEGY	99
5.1.1	Modelling the Release Order Strategy	100
5.2	OPTIMISATION ALGORITHM	102
5.3	APPLICATION TO THE EXAMPLE	104
5.3.1	Markov Decision Process for Example	105
5.4	DISCUSSION	106
5.5	SUMMARY	107
CHAPTER 6. ATTACKER LEARNING CURVE.....		109
6.1	AN OVERVIEW OF ATTACK SCENARIO.....	110
6.2	ATTACKER LEARNING CURVE.....	111
6.2.1	Extracting Similarity-Based Data	113
6.2.2	Accumulative Manipulation	114
6.2.3	Illustrative Example to ALC.....	115
6.3	STRATEGIES APPLIED IN ATTACK PROCESS	116
6.3.1	Observed Strategies	117
6.3.2	Impact of All Strategies in Breaking Algorithms	119
6.4	ATTACKER LEARNING CURVE MODEL (ALCM).....	120
6.4.1	Knowledge Model.....	120
6.4.2	Proposed ALCM.....	121
6.4.3	Performance of the Proposed Model	122
6.5	DISCUSSION	124
6.6	SUMMARY.....	126
CHAPTER 7. DETECTION OF ATTACK STRATEGIES.....		128
7.1	STRATEGY-BASED DETECTION APPROACH: AN OVERVIEW	129
7.1.1	Types of Attack Strategies	130
7.1.2	Detection Approach: DLDA.....	130
7.1.3	The Workflow of the Detection Approach	131
7.2	EXPERIMENTAL EVALUATION	133
7.2.1	Experiment Setup.....	133
7.2.2	Experiment Procedure.....	137
7.3	RESULTS OF THE EVALUATION.....	138
7.3.1	Overall Rates of Success Detection	138
7.3.2	Probabilities of Classification.....	139
7.4	DISCUSSION	141
7.5	SUMMARY	143
CHAPTER 8. CONCLUSION AND FUTURE WORK.....		144

8.1	SUMMARY OF CONTRIBUTIONS	145
8.2	REFLECTIONS ON RESEARCH OUTCOMES	147
8.2.1	The first research question	147
8.2.2	The second research question	148
8.2.3	The third research question	149
8.2.4	The fourth research question	150
8.2.5	The fifth research question	150
8.2.6	Overall Reflection	151
8.2.7	Applicability to Other Security Scenarios	152
8.3	FUTURE WORK	155
	APPENDIX A: EXPERIMENT MATERIALS	158
	APPENDIX B: ACCUMULATIVE MANIPULATION OF TEST SETS	169
	APPENDIX C: PROBABILITY OF CLASSIFICATION RESULTS	172
	BIBLIOGRAPHY	176

LIST OF TABLES

Table 1.1: The Confusion Matrix.....	2
Table 2.1: Attacker Types [135].	17
Table 2.2: Basic Security Tradeoffs [9].	19
Table 2.3: Guidelines for estimating the learning rates in different fields [137].	29
Table 2.4: Hybrid approaches in which an unsupervised machine learning algorithm is applied as a first layer.	44
Table 3.1: Symbols used in pseudo code and their values in the experiments.	63
Table 3.2: The Experiment Measurement Units.	65
Table 3.3: Order of two algorithms A1 and A2.	70
Table 3.4: Breaking A1 for each group.....	71
Table 3.5: Breaking A2 for each group.....	71
Table 3.6: Breaking A3 for each group.....	72
Table 3.7: Breaking all algorithms for each group.	73
Table 4.1: Identifying the Symbols.....	85
Table 4.2: Fire rate of Break_Released_Algo transition.	91
Table 6.1: Calculating the Accumulative Manipulation.	115
Table 6.2: Fitted Parameters for Group 1.	123
Table 6.3: Fitted Parameters for Group 2.	123
Table 7.1: The Number of Samples of each Strategy for Group 1.	134
Table 7.2: The Number of Samples of each Strategy for Group 2.	135
Table 7.3: The Results of Classifying each Strategy.	139
Table 7.4: The probability of classifying one of the correctly classified samples.....	140
Table 7.5: The probability of classifying one of the incorrectly classified samples....	140

LIST OF FIGURES

Figure 1.1: Abstract System Model.	5
Figure 2.1: An example Tradeoff in Intrusion Detection System [9].	20
Figure 2.2: An example of a text-based CAPTCHA (Hotmail, 2013).....	21
Figure 2.3: An example of an image-based CAPTCHA [76].	22
Figure 2.4: CAPTCHA Developing System [10].	23
Figure 2.5: Source of IP addresses of spam emails [151].	24
Figure 2.6: An example of a passive warning page vs. a suspected unsafe site [142]....	25
Figure 2.7: Learning Curve in the simulation [58].	30
Figure 2.8: Performance over time for all three presentation order types [91].	37
Figure 2.9: A typical attacking process [70].	39
Figure 2.10: The standard attack phase [70].	39
Figure 2.11: A Graphical Petri Net Example.	51
Figure 3.1: Pseudo code of A1.	63
Figure 3.2: Pseudo code of A2.	64
Figure 3.3: Pseudo code of A3.	64
Figure 3.4: The average time (in minutes) for ‘attackers’ to break the algorithms.	70
Figure 4.1: A graphical illustration of the proposed SPN model.	85
Figure 4.2: A Guard Function for Superset Algorithm.	88
Figure 4.3: A Guard Function for Subset Algorithm.	88
Figure 4.4: A Guard Function for Independent Algorithm.	88
Figure 4.5: A Distribution Function for controlling the Attack Rate λ	89
Figure 4.6: A Reward Function for Evaluating the Security of a System.	90
Figure 4.7: A Reward Function for Evaluating Knowledge Gained by the Attacker.	90
Figure 4.8: Replicating the results of release order.	92
Figure 4.9: MTTSF vs. Algorithm Orders.	93
Figure 4.10: Attacker’s Knowledge Acquisition Process.	94
Figure 5.1: The Backward Optimization Algorithm.	103
Figure 5.2: Markov Decision Process for Example.	105
Figure 6.1: Attack Scenario.	110
Figure 6.2: The structure of the ALC.	112
Figure 6.3: Example of Structured Attacker Performance.	113
Figure 6.4: Example of Unstructured Attacker Performance.	114

Figure 6.5: The Attacker Learning Curve Based on the Accumulative Manipulation.	116
Figure 6.6: Original Spam E-mail.	118
Figure 6.7: Using Random Addition.	118
Figure 6.8: Using Thesaurus Substitution.	118
Figure 6.9: Using Perceptive Substitution.	118
Figure 6.10: Using Add Spaces.	119
Figure 6.11: Using Delete Spaces.	119
Figure 6.12: The average Accumulative Manipulation vs. all strategies.	120
Figure 6.13: Result of running the proposed model on Group 1 and Group 2.	124
Figure 7.1: The workflow of the Detection Approach.	132
Figure 7.2: The average of ALC-Based accumulative manipulation for each strategy.	134
Figure 7.3: The average of ALCM-Based accumulative manipulation for each strategy.	135
Figure 7.4: The ALC-Based accumulative manipulation of attacker's attempts.	136
Figure 7.5: The ALCM-Based accumulative manipulation of attacker's attempts.	136

GLOSSARY OF TERMS

Term	Definition
Security	The sum of all measures taken to prevent loss of any kind, which can occur due to user error, hardware failure, malicious acts, acts of nature and defects in code [139].
Security Attack	Any action, such as interruption, interception and modification that compromises the security of information [26].
Successful Attack	Any attempt of attack that is executed successfully.
Attacker	An agent (human or computational) that attempts the attack [111].
Attacker's manipulation	The distance between one attempt from an attacker and his previous attempts [14].
Accumulative manipulation	Attacker's aggregated amount of knowledge, which effectively represents how close the attacker is to breaking the defensive mechanism [14].
Computer Security	The prevention of, or protection against, access to information by unauthorised users, as well as intentional but unauthorised destruction or alteration of that information [26].
Prevention	The security procedures undertaken by implementing safeguards, such as a defensive mechanism, in order to make a secure system [111].
Defensive Mechanism	As stated by Bishop in [26], is to institute controls that preserve secrecy, confidentiality, integrity and availability.
Interactive Defensive Mechanism	A defensive mechanism that operates while interacting with an attacker.
Non-Interactive Defensive Mechanism	A defensive mechanism that operates without interacting with an attacker.
Release Order of Defensive Mechanism	The arrangement of releasing a set of defensive mechanisms according to a particular sequence [11].
System resources	A finite set of resources e.g. communication or computational resources that an attacker intends to misuse [139].
System's feedback	The simple Boolean response or reasons that attackers receive from the system for the failure of their request [11].
Model	As defined by Wilson in [146] is "the explicit interpretation of a situation, or one idea about that situation. It can be expressed in mathematics, symbols or words, but is essentially a description of entities, processes or attributes and the relationships between them."

Chapter 1. INTRODUCTION

This chapter introduces the idea of the research undertaken; in particular, the background, motivation, and problems of the research are detailed. This chapter also includes the research hypothesis with several research questions set out for further investigation. Furthermore, the research aim and objectives are stated.

This chapter also explains the methodology of the research, which is empirically based in nature with a novel experimental work that is carried out to evaluate the proposed idea. Moreover, the main contributions of the research are described and the publication history is presented. Towards the end of this chapter, the structure of this thesis is detailed.

1.1 Context & Motivation

Dependency on information technology (IT) and computer and information security (CIS) has become a critical concern for many organisations. This concern has been mainly about protecting secrecy, confidentiality, integrity and availability of information when using computer systems. For this reason, much research has been conducted in this area by proposing defensive mechanisms, which are based on algorithms that aim to protect system resources from misuse. These algorithms encode a set of rules that characterize and recognize attempts at misuse, and prevent any adverse effect on system resources.

In this thesis, a defensive mechanism is used as a broad term which encompasses a variety of services and products that preserve secrecy, confidentiality, integrity and availability. In order to precisely define a defensive mechanism in terms of classification

performance, we begin with the *confusion matrix*¹ that helps to understand how a defensive mechanism performs correct/incorrect classification, as shown in Table 1.1. Terminologies presented in Table 1.1 are explained as follows. True Positive (TP) indicates that the defensive mechanism correctly classifies a malicious attempt as malicious. True Negative (TN) indicates that the defensive mechanism classifies a normal attempt as normal. These two classification results (i.e. TP and TN) are accurate. Moreover, False Negative (FN) indicates that the defensive mechanism classifies a malicious attempt as normal. Finally, False Positive (FP) indicates that the defensive mechanism classifies a normal attempt as malicious. These two classification results (i.e. FN and FP) are inaccurate. The key point to highlight is that an attacker is successful if it is classified as a FN.

Table 1.1: The Confusion Matrix

		Prediction of a defensive Mechanism	
		Attack	Not attack
Reality	Attack	True Positive (TP)	False Negative (FN)
	Not attack	False Positive (FP)	True Negative (TN)

Given this confusion matrix, we categorise defensive mechanisms, based on their predicted results, into two main categories: *Assertive*² and *Predictive*³ defensive mechanisms. The predicted results of an *Assertive* defensive mechanism are always accurate: TP or TN. For instance, an access control system [53] restricting entrance to a property, a building or a room to authorised persons under a well-defined policy (i.e., a policy either clearly denying or allowing and that is typically defined as “to set who can use what information in a computer system” [69]) is an *Assertive* defensive mechanism.

¹ The term confusion matrix is generally known as a contingency table or an error matrix that represents a specific table layout, which gives visualization of the performance of an algorithm, where each column of the matrix states the examples in a predicated class, whereas each row states the examples in an actual class [136].

² Since assertive theories throughout the history of philosophy have sprung from diverse motives and considerations, in this thesis assertiveness is the idea that everything that happens is completely determined by prior conditions.

³ A prediction refers to a quantitative description about what will happen under specific conditions [121].

On the other hand, due to uncertainty as well as several essential tradeoffs⁴ between security and other forms of important factors such as usability or accessibility in designing a defensive mechanism, the predicted results of a predictive defensive mechanism, in addition to the accurate classification results, can also include FP and FN. This category can be divided into two types: *Interactive* and *non-Interactive* defensive mechanisms. With the interactive type, the attackers can get feedback from the system with regards to their attempt, in particular after each failed attempt. This feedback may be a simple Boolean response or may include a reason for the failure. For example, in a CAPTCHA system [143], an attacker gets feedback from the system indicating whether an attempt is successfully passed or not. Similarly, with a spam-filter system (e.g., [4, 42, 116]), an attacker can obtain a response with regards to a spam email that is submitted to target users. Furthermore, in a biometrics authentication⁵ system, an attacker can know the result of cheating attempts such as a cheated face or voice. As a result of this feedback on the attacker's performance, the prediction results of the interactive defensive mechanism type can gradually identify a malicious attempt as a normal attempt (i.e. attackers can learn while a TP attempt becomes a FN), where feedback gained can be a key factor to revealing a defensive algorithm's rules. For instance, the attacker can send spam messages as normal messages. The underlying assumption is that the protection accuracy of a system against an attack begins to gradually decrease over time⁶.

In the non-Interactive type, the attacker, in contrast, cannot get feedback from the system. For example, in airport security identification devices, the prediction results are observed only by the responsible person behind these devices. Even though a FN prediction result can occur, for example when dangerous items, such as weapons, are mistaken for keys or coins, the attacker still cannot estimate the prediction results of this defensive mechanism type. Another example is in antivirus systems [114]; the prediction results of these systems can be observed by only the user. Not only can a FN prediction result occur, but also a FP prediction result, such as when a normal file is identified as a virus.

At present in the literature, interactive defensive mechanisms are qualitatively studied; that is, once a defensive mechanism is broken, the security officer must deploy another

⁴ More details regarding these tradeoffs are presented in Chapter 2 (Table 2.2).

⁵ The term biometric authentication refers to the identification of humans by their characteristics or features.

⁶ This type of protection mechanisms will be referred to as systems that are eventually breakable.

one, which will in turn eventually be broken, leading the officer to deploy a new defensive mechanism, and so on and so forth. For instance, once several CAPTCHA schemes such as Microsoft and Yahoo are released in 2007, an attack is improved and subsequently these released schemes are broken [150]. This causes to develop new schemes. In 2011, these new schemes are broken again by an attack that has been applied in [32]. Additionally, deploying a defensive mechanism has a cost; for example, Caliendo et al. calculated in [33] the cost of deploying a spam-filter within a particular organization at about fifteen thousand Euros for the first year. Furthermore, the security officer must usually work within a given budget constraint, and has therefore a limited number of defence mechanisms to deploy. For this reason, it is important to look at the problem of interactive predictive defensive mechanisms across several of attacks.

With the background issues presented above, this research intends to fill the gap by placing its focus on interactive predictive defensive mechanism. The rationale for this limited focus is explained as follows. In the practical use of interactive predictive defensive mechanism, attackers and defenders exchange ‘victories,’ each celebrating (temporary) success in breaking and defending. That is, as attackers interact with the system, they receive feedback that augments their knowledge of the rules of the algorithm which is used by the system to characterize misuse. They are then able to adapt their future interactions in accordance with this augmented knowledge, increasing their ability to break the defensive algorithms deployed, until eventually reaching the point where the defensive mechanism is broken, the spam-filter rules are overridden or CAPTCHAs are automatically deciphered.

Since most interactive predictive defensive mechanism rely on a single algorithm as a defence mechanism [11, 12] and when this algorithm is broken, it will be replaced with another, we investigate a methodology to prolong the interaction between an attacker and a security system for as long as possible. As an attacker can gradually derive the rules that are used by the defensive mechanism to classify attempts as a result of interacting with the system, the aim of this study is not to prevent breaking the defensive mechanism as much as lengthening the time necessary to break it. Therefore, the method proposed in this research is *a set of algorithms approach* as a holistic defence

mechanism. When studying sets of algorithms, various issues arise about how to construct the algorithms and in which order⁷, or in which combination to release them.

In order to precisely define the problem under consideration, we provide an abstract system model of the attack scenario, involving the attacker and the system as shown in Figure 1.1. This model describes a general class of interactive security solutions, which includes CAPTCHAs, certain spam filters, and intrusion tolerance algorithms. This class of mechanisms is characterized by an intelligent defensive algorithm being attacked and eventually broken, and then being replaced by a new intelligent defensive mechanism. Furthermore, this model provides the basis of the experimental study presented in Chapter 3, which is revised into a stochastic model in Chapter 4 in order to analyse the release order of security algorithms, and is refined into a stochastic model in Chapter 5 to derive optimal release strategies. Each component of this abstract system model is detailed as follows:

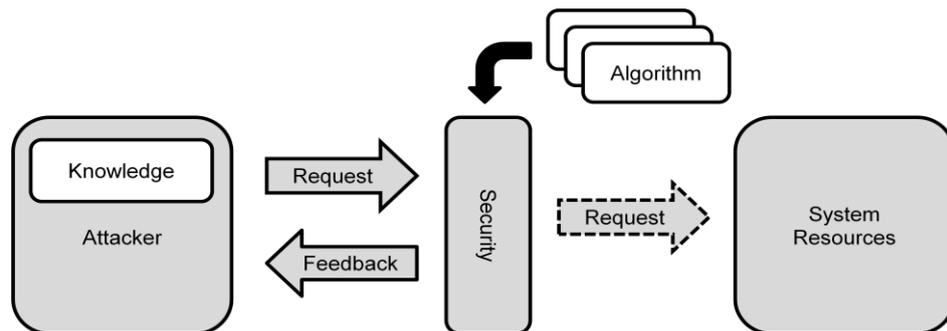


Figure 1.1: Abstract System Model.

Security Layer. To maintain the security of the system, the security layer must be updated. Within this update, the algorithm used by the security layer is replaced by another algorithm from the pool to encapsulate a different set of rules such that requests that are misusing system resources are no longer permitted to pass through the security layer. The attacker must repeat the process of knowledge acquisition in order to determine the new classification rules so that he can continue sending requests to misuse system resources. This process of learning takes time and the overall aim of the algorithms is to maximize the time until all are broken.

⁷ The order terms refers in this thesis to the arrangement or disposition of defence algorithms in relation to each other according to a particular sequence, pattern, or method.

System Resources. In the system model, it is assumed that a finite set of resources can be used, e.g. communication or computation resources. The pool of security algorithms is contained by a security layer deployed to protect the system resources from misuse, e.g. excessively high consumption or consumption for unacceptable purposes. These algorithms classify requests to the system as acceptable or unacceptable based upon a set of rules. If a request is classified as acceptable, then the request proceeds and the system resources are consumed. A request that is classified as unacceptable cannot proceed through the security layer, and feedback is provided to the user regarding the failed request.

Algorithms. The selection of algorithms when updating the security layer determines the subsequent security of the system. In particular, a set of algorithms D , $D = \{d_1, d_2, \dots, d_n\}$, $n > 1$, $d_n \in D$, represents the security layer of the system. Each of these algorithms includes a number of rules $R = \{r_1, r_2, \dots, r_n\}$, $n \geq 1$ to protect the system. Based on the rules of an algorithm, a set of algorithms is classified into three types: overlapping rules, non-overlapping rules or mixed. In the first type, some of the rules are a subset of each other $r_i \subset r_{i+1}$. The importance of this type can be in *breaking up a defense algorithm into a set of algorithms*, and more details about this will be given later in Chapter 3. In the non-overlapping rules type, all the rules are independent $r_i \not\subset r_{i+1}$. The importance of this type can be in using *variance algorithms* that might force the attacker back to the beginning of the learning phase. For example, when an algorithm is broken by an attacker, the attacker needs to discover a new approach to break the next release algorithm. The possible reason behind this can be that there were not patterns that could be exploited from the first released algorithm in order to break the next release algorithm, and more details about this will be given later in Chapter 4. The third type is using mixed overlapping rules and non-overlapping rules algorithms $r_i \subset r_{i+1}$ and $r_{i+1} \not\subset r_{i+2}$. The importance of this type can be, in addition to the importance of the first and second type, that releasing an independent algorithm which has non-overlapping rules between dependent algorithms that have overlapping rules might *impair the attacker's learning process*.

Attacker. An attacker is an agent (human or computational) that attempts to misuse system resources. The attacker makes requests for the system resources, which pass through the security layer as described above. The attacker has some prior *knowledge* about the rules used to classify requests, and attempts to design requests to be classified

as acceptable. On each failed attempt, the attacker receives some feedback from the system. This feedback may be a simple Boolean response, or may include reasons for the failure. The attacker can add this feedback to his knowledge, and use this knowledge to inform his subsequent requests. By repeatedly performing this knowledge acquisition process⁸, the attacker can derive the rules that are used by algorithms to classify requests. This includes both the parameters used and the values of these parameters. The attacker can then misuse system resources by sending requests that are structured in such a way that they fulfil the rules of the algorithm in the security layer. In this case, the algorithm is considered ‘broken’.

Intuitively, releasing a set of algorithms one by one sequentially extends the required time to break a system, rather than releasing only one algorithm. However, the order in which algorithms are released may be essential in terms of extending the time taken to break all algorithms. The question for the defender is then to find out the order in which to release these algorithms so that the time until all algorithms are broken is maximized.

1.2 Research Hypothesis & Questions

Based on the aforementioned problems discussed, the order in which algorithms are released may thus be important. As a consequence, this research postulates that the longer it takes for the attacker to acquire the necessary knowledge regarding classification, the longer the system is protected from misuse. In line with this, the research hypothesis is:

Hypothesis: *“The time taken by a series of attackers to break a set of interactive probabilistic defensive mechanism is dependent on the order in which the algorithms are released.”*

The main reasoning behind this hypothesis is that the time it takes an attacker to break a defensive algorithm may depend on what the attacker has learned from earlier successful attacks on similar algorithms. Therefore, if this hypothesis is valid, one may be able to defend better against attacks by impairing the process of learning of attackers. There may be many ways in which this can be achieved, but in this thesis the most direct im-

⁸ In the education context, for example Kahn and O’Rourke stated in [72] that learning is driven by a process of inquiry.

plication of this reasoning is considered to be that the order in which defensive algorithms are released may impact the learning process.

Bearing in mind the hypothesis mentioned above, a number of research questions have been set forward to guide the research work, as well as the construction of this thesis:

Research Question (1):

“Does the order in which different defensive mechanisms are released impact the time an attacker needs to break each one of them?”

Research Question (2):

“Could we optimize the order in which defensive mechanisms are released?”

Research Question (3):

“How does dependency between algorithms impact on ability to answer question 2?”

Research Question (4):

“Could we model the learning acquisition process of attackers?”

Research Question (5):

“Based on understanding the learning acquisition, can we devise an attacker detection approach?”

1.3 Research Aim & Objectives

Following the research problem, the hypothesis and the research questions presented in the previous two sections, the aim of this research is to investigate a set of algorithms approach for an interactive defensive mechanism which focuses on the order in which to release these algorithms, so that the time until all algorithms are broken is maximised. With several research questions to be answered as listed above, the objectives of this research to achieve its aim are:

Objective (1):

- To conduct a review of relevant existing approaches and related theories.

Objective (2):

- To identify and propose an ordered releasing strategy for a set of algorithms approach.

Objective (3):

- To develop and conduct an evaluation experiment for the proposed order.

Objective (4):

- To build a model and analyse different algorithm orders.

Objective (5):

- To derive an optimisation algorithm for releasing security algorithms.

Objective (6):

- To devise an attack detection approach based on the learning curve that can provide an advantage to optimising the release order algorithm.

The above objectives have been reached throughout the course of this research. Many interesting insights were gained and lessons learned from this research and these are discussed in Chapter 8.

1.4 Research Methodology

In order to achieve the research aim, a research methodology was prepared which covered all research aspects, from collecting the preliminary research data to the evaluation of the work. The research methodology is summarised as follows:

- I. Literature review:** This stage of the study aims to identify the weaknesses of interactive defensive mechanism types. In addition, it aims to review the existing interactive defensive mechanism approaches (methods, models, techniques, frameworks) in order to investigate their ability to accommodate to weaknesses.
- II. Proposing a new approach:** As a result of this research, an approach will be proposed which is expected to address the weaknesses of the existing approaches to interactive defensive mechanism types, and which includes various aspects that have not yet been covered.
- III. Evaluation:** In this stage, the proposed approach will be evaluated. Specifically, an empirical-based approach undertaken with a novel experimental study was set up in order to evaluate the proposed approach.

The participants involved in all of the experimental works were recruited from the student body of this university. The recruitment process was managed through emails and posters distributed among the students on the campus (Appendix A). In order to guarantee that ethical procedures were followed, the methods of recruitment and the process of informed consent (Appendix A) for all the experimental work carried out, was approved by the university ethical committee (UEC).

The experimental work focused on the ordering of algorithms, which aimed to investigate the question of whether the order in which a set of defensive algorithms is released has a significant impact on the time taken by attackers to break the combined set of algorithms. Several simplified but representative spam filter algorithms and a web-based system on which to perform the experiment were developed to support this experiment. This study of the ordering of several algorithms involved forty participants. Details of the ordering of several algorithms, including the results and analysis, will be discussed in Chapter 3.

1.5 Contributions of the Thesis

This thesis addresses the issue of the release of a set of algorithms as an interactive defensive mechanism in general and focuses on the issue of maximising the time taken by an attacker to break all algorithms. The key area of concern is, when a released algorithm is broken by an attacker, how much knowledge is gained that can provide the attacker with patterns to break the next released algorithm.

This thesis provides the following five key contributions to address the issue of the release order of security algorithms in interactive defensive mechanisms:

- An introduction to the issue of interactive defensive mechanisms in a system, which includes (i) appropriate categorisation of defensive mechanisms, and (ii) a definition of interactive defensive mechanism types (Addressed in Chapter 1). The result of the categorisation scheme of defensive mechanisms is based on the expected results of a defensive mechanism from the confusion matrix and is useful to interested parties such as researchers, defensive mechanism designers and developers as a tool to classify a defensive mechanism. The view of interactive

defensive mechanisms is based on the perspective of depending on a single algorithm as a defensive mechanism and the learning acquisition ability of attackers. This view is important and useful since it provides a consistent and clear understanding of the problem of interactive defensive mechanisms in a system. Having such a view enables various interested parties, such as researchers, defensive mechanism design and defensive mechanism developers, to work from the same reference point, which is as unambiguous as possible

- A novel controlled experimental study for evaluating whether the order of utilising a set of algorithms methodology as a defensive mechanism matters (Addressed in Chapter 3). This includes its design, in addition to numerous developed secure algorithms. The developed controlled experimental study facilitates a real-life interaction between a system that includes a set of algorithms and an attacker in terms of time taken to break the system
- A model for the release order of security algorithms using Stochastic Petri Nets (Addressed in Chapter 4). The proposed model is designed based on the underlying principles of the developed controlled experimental study, which can describe the interaction between an attacker, the set of algorithms used by a system and the knowledge gained by an attacker with each attack. This framework facilitates theoretical analysis of the releasing order of a set of algorithms methodology. Based on empirical results achieved from the controlled experimental study, the proposed model is parameterised and evaluated
- An optimisation algorithm to compute the optimal release strategy for a set of defensive algorithms (Addressed in Chapter 5). Due to the fact that the results of the experiment showed that the release order of defensive algorithms has a significant impact on the time needed to break a set of defensive algorithms, the problem of the release order strategy for a set of defensive algorithms is mathematically modeled as a Markov Decision Process and provides a tailored algorithm to efficiently solve any model within the class of models presented. The model solution should scale without problems to optimize the release order of tens of defensive algorithms

- A notion of accumulative manipulation of an attacker that is derived from the experimental study. Since breaking an interactive defensive algorithm typically requires several attempts by the attacker, the attacker’s knowledge gradually increases with each attempt. As such, the attacker performs each attempt following a structured strategy rather than randomly modifying their requests. Thus, the accumulative manipulation is defined as the attacker’s aggregated amount of knowledge that effectively represent how close the attacker is to breaking the defensive mechanism. This notion also forms the basis of the Attacker Learning Curve (ALC) approach, which reflects the influence of the strategy used by the attacker. Based on this approach, a detection of attack strategy mechanism is introduced and implemented. This detection mechanism can provide an advantage for the developed optimisation algorithm with regard to prolonging the protection of an interactive defensive mechanism as long as possible. Furthermore, the results of evaluating this detection mechanism indicate that the ALC offers a set of efficient features and heuristics for a machine learning technique in order to detect the attack strategy qualitatively. (Addressed in Chapters 6 and 7).

1.6 Publication History

This thesis includes work that has been published in peer-reviewed publications written by the author. These publications are as follows:

1. Alsuhibany, S. A., Alonaizi, A., Morisset, C., Smith, C., and van Moorsel, A. “Experimental Investigation in the Impact on Security of the Release Order of Defensive Algorithms,” *Proceeding in 3rd IFIP International Workshop on Security and Cognitive Informatics for Homeland Defence (SeCIHD’13)*, volume 8128 of Lecture Notes in Computer Science (LNCS), Springer, September 2–6, 2013, pp. 321–336.
2. Alsuhibany S. A., and van Moorsel, A. “Modelling and Analysis of Release Order of Security Algorithms Using Stochastic Petri Nets,” *Proceeding in 8th International Conference on Availability, Reliability and Security (ARES’13)*, IEEE Computer Society, September 2-6, 2013, pp. 437–445.

3. Alsuhibany, S. A., Alonaizi, A., Morisset, C., and van Moorsel, A. “Optimizing the Release Order of Defensive Mechanisms,” *Proceeding in 29th Annual UK Performance Engineering Workshop (UKPEW’13)*, 4th Jul, 2013, pp. 34–41.
4. Alsuhibany, S. A., Morisset C., and van Moorsel, A. “Detection of Attack Strategies,” *Proceeding in 8th International Conference on Risks and Security of Internet and Systems (CRiSIS’13)*, IEEE Computer Society, October 23-25, 2013, pp. 1–8.

In addition to peer-reviewed papers, a technical report has been written and published in the School of Computing Science Technical Reports Series.

1. Alsuhibany, S. A., Alonaizi, A., Smith, C., and van Moorsel, A. *Optimizing the Release Order of Defensive Mechanisms*. School of Computing Science. 2012. School of Computing Science Technical Report Series 1333, available at: <http://www.cs.ncl.ac.uk/publications/trs/papers/1333.pdf>

This was a preliminary version of the results of validating the main hypothesis and the derivation of the optimisation algorithm.

1.7 Structure of the Thesis

The remainder of the thesis is structured as follows:

Chapter 2 provides a relevant literature review and offers background information that allows the reader to understand the topics and related work in the area of defensive mechanisms. The chapter presents examples of different interactive defensive mechanisms. It focuses on the effectiveness of information order in the education and psychology fields, as well as shedding light on the learning curve theory. It also explains related works on game theory, attack modelling and attacker’s behaviour in terms of time-to-compromise a system. Moreover, it mainly highlights the anomaly detection technique work on hybrid, machine learning based classifiers in which an unsupervised machine learning algorithm is applied as a first layer for observing an attack attempt. Finally, stochastic modelling formalisms and a number of tools that are used for stochastic modelling and solving are presented.

Chapter 3 describes the controlled experimental study, which examines the hypothesis that the time spent on breaking a set of defensive algorithms depends on the order in which these algorithms are released. This chapter explains the experiment setup, including the system developed and the experiment procedures, including data collection. The findings of this evaluation study are detailed and discussed in the chapter.

Chapter 4 introduces the proposed model of the release order of security algorithms using Stochastic Petri Nets (SPN). In this chapter, the SPN model, assumptions and performance metric calculations are described. Based on the results achieved from the experimental study, the model is parameterised. Furthermore, the results obtained by evaluating the SPN model are presented.

Chapter 5 explains the approach to the derivation of optimal release strategies. Based on a continuous time Markov Decision Process, a tailored optimisation algorithm is provided. An application is presented based on the empirical results of the experimental study. This optimisation algorithm can efficiently obtain the optimal release strategies for any given model.

Chapter 6 demonstrates the Attacker Learning Curve (ALC) that is formed by accumulative manipulation notions of an attacker. This ALC is derived from the data collected in Chapter 3. This chapter also describes an Attacker Learning Curve Model (ALCM) that is inspired by a previous model used for describing developers' learning curve during software development.

Chapter 7 shows the proposed attack strategy detection approach that builds upon the ALC concept of knowledge that is presented in Chapter 6. The results of evaluating this detection approach include not only an ALC experiment-based, but also an ALCM model-based. A discussion regarding the probability results of classifying correct/incorrect classified samples is also provided.

Chapter 8 concludes the thesis by examining all research questions thoroughly and in parallel to meet the research aim and objectives set out earlier. Implications of this work and the benefits of the findings will also be discussed. Also, this final chapter includes all contributions of this thesis to this field of study followed by several possible future research avenues.

Chapter 2. BACKGROUND AND LITERATURE REVIEW

This chapter provides background information and a literature review related to the research conducted in this thesis in order to give the context for the problem of interactive defensive mechanisms. The background gives information about the theoretical and practical problems for the defender of the system, with regard to the time needed to break a set of algorithms, and for the attacker, with respect to the knowledge gained. The fundamental concepts used in this thesis are introduced, and the relationship between these concepts is investigated. The relevant literature pertaining to these concepts is then reviewed, drawing from several disciplines including: presenting different perspectives on the motivations behind investigating the learning acquisition process (such as learning curve theory); describing different quantitative security techniques used by researchers (such as Time-To-Compromise-System); and presenting approaches to detect anomaly attacks in order to improve the protection of a system. Such an approach differs from previous works, which focus on concepts and methodologies from the perspective of a single discipline. The value of this cross-disciplinary approach is to provide a theoretical and practical basis for the study of interactive defensive mechanisms in a secure system.

The remainder of this chapter is structured as follows. Section 2.1 gives an overview regarding attackers, defenders and their motives. Section 2.2 highlights the definition of a defensive mechanism and interactive defensive mechanism with several examples. Section 2.3 presents the learning acquisition process with respect to the learning curve theory, a problem-based learning approach and the effectiveness of information order in the learning process. Section 2.4 discusses the quantitative security methodology of at-

tack modelling. Section 2.5 outlines anomaly detection techniques in which an unsupervised machine learning algorithm is utilised as a first layer for observing an attack attempt. Section 2.6 explores stochastic modelling formalisms, and then describes those that are relevant to this thesis. This section also summarizes a number of tools used to build a model. Finally Section 2.7 concludes this chapter.

2.1 Attackers, Defenders and their Motives

In general, the importance of utilising the Internet as an integral way of conducting daily business has increased continuously in several areas including banks, schools and services providers. However, it is not only those with good intentions who can connect to the Internet, but also those with malicious goals. Since the Internet has become ubiquitous, computer security has become more important than ever. Security researchers have therefore long been interested in understanding what an attacker can do to the Internet and what can be done to prevent attacks through defenders. In light of this, the definition of attackers, defenders and their motives are presented in the following subsections.

2.1.1 Attackers

The term ‘attacker’ is defined as people who attempt to compromise the confidentiality, integrity, or even control of a computer network without its owner’s knowledge [9]. Although this definition reflects a very broad umbrella that covers people with very different backgrounds and motivation, a possible classification of this definition is proposed in [135] as shown in Table 2.1. The following explains briefly each attack type that is shown in Table 2.1.

In an early period of networks and the Internet, *Gray Hat* hackers dominated the attacker scene [9]. Curiosity and fame were key in motivating these attackers who were often asocial individuals. Furthermore, *Script Kiddies* mimic hackers for fame by using self-developed tools. When a widespread device, such as a game, is protected, a *Cracker* removes the protection of this device. Nowadays, this concept has been extended to include various activities such as blackmailing individuals and stealing money from bank accounts. The *Malicious Users* and system administrators are an important class of attacker. This is because attackers are inside an organisation; therefore, the po-

tential for causing damage can be high. For example, malicious users can steal corporate secrets or costumers' data [9].

Table 2.1: Attacker Types [135].

Actor	Description
Script Kiddie	Often young, no sophisticated skills, motivated by fame.
Gray hat hacker	Semi-Professional, criminal intent, sophisticated attack tools and programs.
Cracker	Modifies software to remove protection.
Malicious user	Inside organisation, criminal intent.
Malicious system administrator	Control of network, criminal intent, potentially significant damage.

2.1.2 Defenders

Ideally, defensive actions should take place against security issues that are raised by attackers, when using a networked system. Indeed, the responsibility for securing network and systems lies with the system administrators. In addition to configuring and monitoring the networks against attacks, the system administrators are responsible for enforcing formal and informal security polices and educating users on possible vulnerabilities [18].

2.1.3 Motivation of Attackers and Defenders

The motivation of attackers seems difficult to be determined. In particular, several places indicate that they are very appealing to attacker such as law enforcement and defence department. Also, as some systems are easy to be attacked, attackers use them as targets [139].

There are several challenges of network security from the defender's perspective. One of these challenges is the lack of motivation. This challenge partly stems from the difficulty in quantifying the value added by network security. Furthermore, misalignment of incentives is another challenge. That is, since several researchers exaggerate the risks for their own benefits, management usually has an incentive to cover up security breaches. There is thus a huge gap between system administrators and people suffering security breaches (e.g., customers of a bank) [9].

However, various recent optimistic developments for defensive mechanisms have taken place. Firstly, the level of awareness of network security has increased in different ar-

eas, ranging from government and business to the general public. Secondly, perceiving that security is an important feature leads to making demands for secure networks and systems. As a result of this, security services for both organisations and individuals have emerged. Finally, the nature of emerging security facilities, whether for individuals or organisations, supports dynamic prevention and improved response defence [9].

2.2 Defensive Mechanism

Given the importance of both computer security and the existing vulnerabilities, this section presents the definition of a defensive mechanism, interactive defensive mechanism and examples for interactive defensive mechanisms.

2.2.1 What is a Defensive Mechanism?

A defensive mechanism, as stated by Bishop in [26], is that which institutes controls that preserve secrecy, confidentiality, integrity and availability. The interpretations of these four aspects vary due to the contexts in which they arise. In the computer security context, these aspects are interpreted as follows [18, 26]:

- *Secrecy* refers to the effects of the defensive mechanism used to limit the number of principals which can access information, such as cryptography or computer access controls.
- *Confidentiality* indicates the concealment of information or recourses against an unauthorised person.
- *Integrity* refers to preventing unauthorised persons from modifying the information.
- *Availability* refers to the ability to use the information or resources requested by the user. The aspect that is relevant to security is preventing Denial of Service (DoS) attack in which someone might deliberately arrange to deny access to the information by making it unavailable.

2.2.2 Interactive Defensive Mechanism

Considering the definition of a defensive mechanism, a variety of defensive mechanisms have been developed. Despite the main categories encompassing widely deployed

solutions including firewalls and antivirus, this subsection essentially reviews the interactive defensive mechanism type that is pointed out in Chapter 1. In particular, based on the predicted results from the confusion matrix that is shown in Figure 1.1, defensive mechanisms are categorised into two major types: Assertive and Predictive. As mentioned previously, the results of an assertive defensive mechanism are always a TP prediction result or TN prediction result, such as an access control system restricting under a well-defined policy. On the other hand, due to several fundamental tradeoffs in designing a defensive mechanism, as summarised in Table 2.2 [9], a predictive defensive mechanism can also include FP prediction results and FN prediction results. The following sheds light on the tradeoffs indicated in Table 2.2 that cause a FP prediction result and a FN prediction result, as well as a TP prediction result and a TN prediction result.

Table 2.2: Basic Security Tradeoffs [9].

Tradeoff	Security versus
Usability	Difficulty of use and mental overhead
Accessibility	Access restrictions based on location or role
Overhead	Costs on system and network resources
Economics	Monetary and manpower costs

Typically, *usability* represents a basic tradeoff that lies between security risk and ease of use, where additional security mechanisms impair usability, therefore making the system less usable for its user. A real-life example is when the user of a computer network wants to access data easily while expecting the sensitive data to be protected from unauthorised access. Since the application of such defensive mechanisms is for achieving a satisfactory level of security between the user and sensitive data, a level of usability is sacrificed [9].

Accessibility is another factor that needs to be balanced in the network security, where a service is accessible to as many people as possible. For instance, a spam email can represent a downside to the unrestricted accessibility of a network, while a network with restricted accessibility that protects users against spam emails can have accessibility tradeoffs.

Furthermore, the *overhead* that is caused by security systems with regard to system and economic resources is also an important tradeoff. As such, the defensive mechanism used, such as antivirus or firewall, utilises both network and system resources, such as bandwidth and memory. Consequently, the system administrator needs to maintain the defensive mechanism, even though it is open source and free [9].

In addition to the aforementioned fundamental tradeoffs, a number of tradeoffs that can occur during the operation of defensive mechanisms need to be taken into account. For instance, a basic performance criterion for an intrusion detection system is the FP prediction result. The tradeoff can be a decrease in the FP prediction result, which leads to decreasing the system's sensitivity, and increasing the FN prediction result, which in turn leads to decreasing the system effectiveness, as shown in Figure 2.1. As a solution, upper bounds for the FP prediction result and lower bounds for the FN prediction result should be determined according to the specifications of the deployed network.

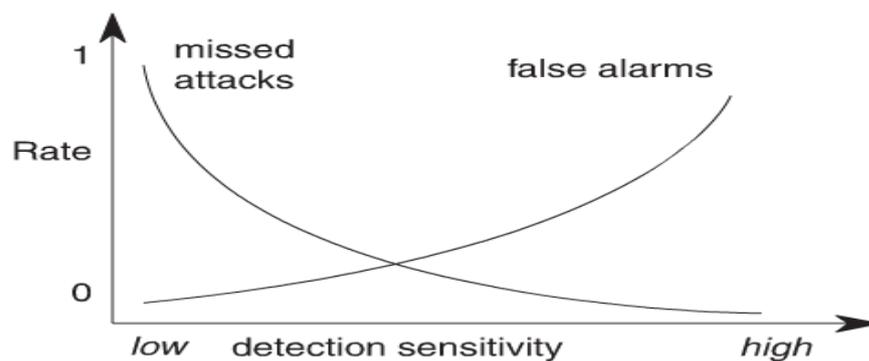


Figure 2.1: An example Tradeoff in Intrusion Detection System [9].

Hence, the predictive defensive mechanism is divided into two types: interactive defensive mechanisms and non-interactive defensive mechanisms. In the interactive defensive mechanism, the attacker can get feedback from the system after each attempt, whether the attempt passes through the defensive mechanism or not. This feedback may be a simple Boolean response or may include a reason for the failure. Accordingly, the feedback plays an important role in breaking this type of defensive mechanism gradually over time. In contrast, with a non-interactive defensive mechanism, the attacker cannot obtain feedback from the system. For example, in antivirus systems, the prediction results of these systems can be observed by only the user. Also, in airport security identification devices, the prediction results are observed only by the responsible person behind these devices. As noted previously in Chapter 1, this thesis focuses mainly on the

problem of the interactive defensive mechanism. Therefore, the following provides several examples of interactive defensive mechanisms.

2.2.3 Examples of Interactive Defensive Mechanisms

Despite the fact that there are several interactive defensive mechanisms, this section outlines those most widely deployed, including CAPTCHAs, Spam-Filters and Anti-Phishing systems. Moreover, these mechanisms provide an indication to systems that are eventually breakable over time, as mentioned in Chapter 1.

CAPTCHAs

In 1996, Moni Naor was the first person who proposed to use automated Turing tests to verify that a human is in the loop [101]. For the same reason, *the CAPTCHA* system was developed. CAPTCHA is a *Completely Automated Public Turing test to tell Computers and Humans Apart*. It is a program that generates and grades tests that humans can pass easily, whereas computers cannot [143]. CAPTCHA is defined formally in [22] by Baird and Papat. Since then, it has been established for several applications; for instance, it protects against spammers who abuse email accounts by writing programs which automatically sign up for thousands of email accounts for this purpose. As a result, CAPTCHA plays a significant role in reducing spam and has been adopted by various websites, including Microsoft, Google, and Yahoo.

A high-quality CAPTCHA must satisfy two main requirements: *robustness* and *usability*. The robustness aspect is the strength of CAPTCHAs to defend against adversarial attacks, while the usability aspect is, by definition, in the effortlessness for humans to pass its challenges.

So far, three main types of CAPTCHAs have been deployed. The first is Text-based, which consists of sophisticated distorted text images that are unrecognisable to even state of the art of pattern recognition programs, though recognisable to users' eyes. The users are typically required to perform a text recognition task to pass these tests. Figure 2.2 shows an example of a text-based CAPTCHA.



Figure 2.2: An example of a text-based CAPTCHA (Hotmail, 2013)

The second type is Image-based, which consists of images that are unrecognisable to state of the art of image recognition programs, but remain recognisable to users. The users are typically required to perform an image recognition task to pass these tests. For example, *PIX* is one of the first Image-based schemes; more details about *PIX* and *EPS-Game*, which are used for creating the required image database, can be found in [144]. Figure 2.3 shows an example of an image-based scheme.

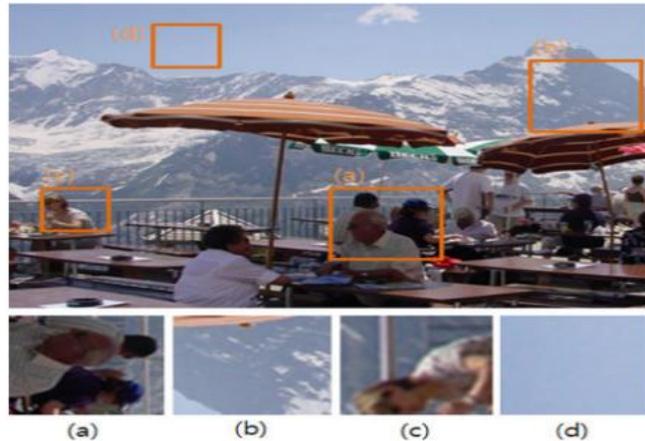


Figure 2.3: An example of an image-based CAPTCHA [76].

The third type is Sound-based and consists of sophisticated distorted speech that is again unrecognisable to state of the art of speech recognition programs, but recognisable to users. Users are typically required to perform a speech recognition task to pass these schemes. For example, a word is said and the user must type the word [35]. These systems are usually used alongside text-based schemes for disabled people.

Although developers have proposed several mechanisms to create more robustness CAPTCHAs against the attackers, a number of successful attacks have been reported. For example, Moy et al. [100] defeated EZ-Gimpy (99% success rate) and the 4-letter Gimpy-r (78% success rate). In addition, a number of CAPTCHAs have been broken [150] by counting the number of pixels of each segmented character, achieving a success rate of almost 100%. More recently, a systematic evaluation methodology has been applied in [32] to 15 current CAPTCHA schemes from popular web sites, in which the authors found that 13 of these schemes were vulnerable to automated attacks.

Therefore, once a new CAPTCHA is released, a new attack is developed. In light of this, Alsubibany proposed in [10] a CAPTCHA developing system, as shown in Figure 2.4, which shows the targeted area that satisfies security and usability aspects, although

as a result of this, once the developed scheme is released, an attack is improved and subsequently the released scheme is broken.

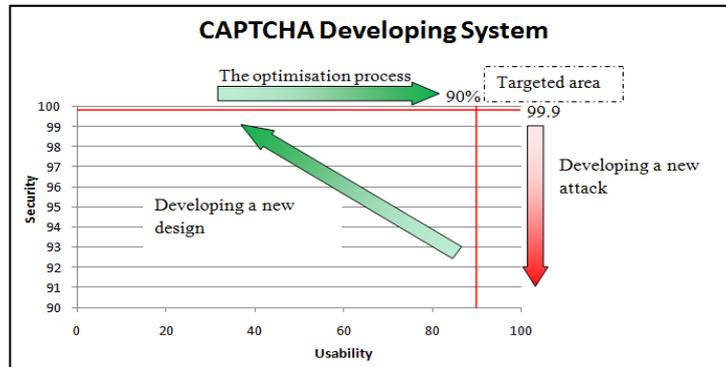


Figure 2.4: CAPTCHA Developing System [10].

Spam-Filters

Despite the fact that CAPTCHAs are applied to protect against spammers who abuse email accounts, unsolicited emails have become a serious problem with tangible costs felt by virtually every internet user. Several approaches have been proposed by researchers for filtering unsolicited emails. These approaches are classified into two major methods [4]. The first method is the reputation-based filters that rely on information outside of the content of the individual email messages. For example, origin based techniques classify spam email based on network information including black lists [42] and white lists [51, 116]. Furthermore, social networks aim to assign to each message a probability of it being spam, based on the past history of the user. Implicit techniques [82] and explicit techniques [52] are examples on the social network method. Besides this technique, traffic analysis is used in [103] to identify when a host or network issues an abnormally large number of emails. Although this method can mitigate the impact of spam email, studies in [61] and [52] stated numerous disadvantages can take place for the black list and white list.

In contrast to the reputation-based filter, content-based filters detect spam email by examining the content of the email regardless of its origin. For example, in heuristic filters, an email is classified as spam by searching for patterns that are commonly identified in spam, such as in [42]. For fingerprinting filters, spam emails are detected by computing and comparing the fingerprint of any incident email, for example, via an approximate or exact hashing algorithm or digest [44]. Moreover, machine learning filters aim to automatically derive spam and not spam classifiers, therefore avoiding the hu-

man labour required to maintain rule-based filters. There are several categories of machine learning including: statistical filters [50], genetic algorithms [103], artificial immune systems [75], and artificial neural networks [75]. Density-based clustering [152] is considered on the server side, which can process hashed versions of messages. As content-based filters can detect spam with such accuracy, spammers are increasingly devising attacks to thwart them. For example, four groups of attacks have been introduced in [147]: tokenization, obfuscation, weak statistical and strong statistical. Furthermore, an extensive survey has been conducted in [41] on content-based spam filters. This survey concludes that, as computers improve continually and processing power becomes cheaper, it may become more likely that the better developed mechanisms against spam-emails can be more widely employed. However, it seems that 100% accuracy is not expected to be achieved by any automated filter or even a combination of filters (e.g., naïve Bayes and genetic programming). The reason behind this is the changing nature of spam and improving attacks on statistical filters.

Moreover, Yeh et al. observed and analysed in [151] a large number of spam-emails. For each spam email, they collected source IPs, the URLs within the emails, and the web sites of the URL. Figure 2.5 illustrates the geographic distribution of the spam source IP addresses. They observed by tracing some spam campaigns that most are likely manipulated by various automatic programmes.

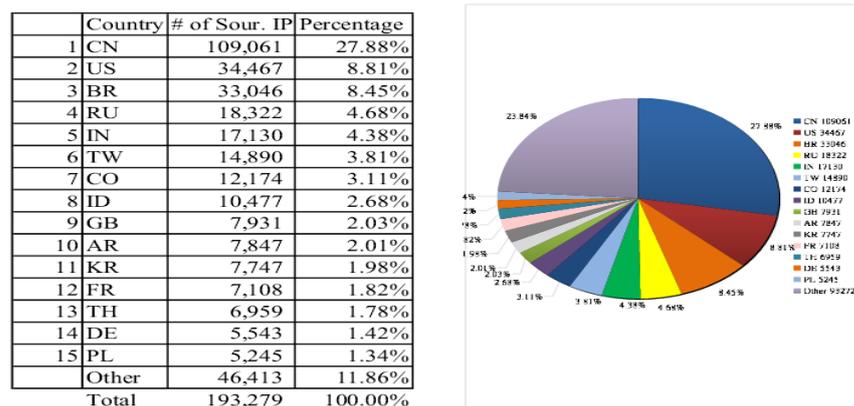


Figure 2.5: Source of IP addresses of spam emails [151].

In 2012, Caliendo et al. investigated in [33] the cost impact of spam filters by measuring the effect of information system technologies in organisations. They found the cost of deploying a SPAM filter within a particular organization to be approximately fifteen thousand Euros for the first year, which means that while the security officer must usu-

ally work within a given budget constraint, and has therefore a limited number of defence mechanisms to deploy, there is a need to investigate a method to mitigate this dilemma.

Anti-Phishing

Phishing attack is the king of the social-engineering attack in which people are tricked into sharing sensitive information or installing malware on their devices by using a spoof email message [63]. Phishing attack is increasingly pervasive and sophisticated. That is, it has spread further than email messages by means of SMS, instant messaging, social networking and massive multiplayer games [60].

To protect users against this attack, a number of Anti-Phishing algorithms have been developed. For example, Fette et al. developed in [49] the first email phishing filter. Afterwards, blocking phishing sites was proposed as a defence mechanism against phishing attack. As such, numerous commercial browsers are designed to block phishing attempts by means of a number of approaches (e.g. [1]). Figure 2.6 shows an example of a passive warning page against a suspected unsafe site. Since these approaches are installed in browsers, their efficiency can be evaluated empirically. For instance, Sheng et al. examined in [133] the most important block-lists and browser tools, showing that zero-hour protections achieved by block lists had a TP (i.e., True Positive) rate of less than 20%. They found that, despite the fact that deployed heuristics were rather effective in identifying phishing attack attempts, they were simply warning people in web browsers rather than blocking probable phishing sites.



Figure 2.6: An example of a passive warning page vs. a suspected unsafe site [142].

In light of the protection approaches against phishing attacks, taking down phishing sites has also been proposed by several organizations. Therefore, end users who click on a phishing website should be shown a message such as “The requested page is not available” [63].

Furthermore, a very recent broad survey conducted in [74] reviewed a number of anti-phishing software techniques. This survey concluded that applying *machine learning* techniques as a defence mechanism against phishing attack is promising due to their effectiveness with respect to classifying the phishing attack in the publically known literature.

For phishing attack against costumers, the Anti-Phishing Working Group APWG is *an international consortium of law enforcement, industry and academic researchers devoted to combating Internet scams and online fraud* [19]. This group stated that the peak was in 2010 where APWG identified more than 115,000 unique phishing sites worldwide [19, 85]. In spite of this, phishing costs varied widely in terms of the expected damage, ranging from \$61,000,000 per year to \$3,000,000,000 per year in the U.S. [60].

Security experts and phishing attackers are in a rat race nowadays. Since security experts make great efforts to develop and improve techniques to detect phishing and spam attacks, attackers are continually learning new techniques and consequently changing their strategies [20]. Although most scientific papers (e.g., [117]) emphasize that training and education overcome the human weakness to some extent regarding phishing attacks, updating the detection algorithms continually can improve human knowledge in the fight against phishing attacks.

Given these examples, attackers augment their knowledge of the rules of the algorithm, which is used by the system, by receiving feedback as a result of the interaction between them and the algorithm. In order to determine the new rules, the attacker must repeat this process of learning acquisition. Therefore, the learning acquisition process plays an important factor in this thesis. As stated previously in Chapter 1, the longer it takes for the attacker to acquire the necessary knowledge regarding the rules of the algorithm, the longer the system is protected from misuse. Accordingly, since the work in this thesis considers using a learning acquisition process as a rationale in Chapter 1, an overview of this process is presented in the next section.

2.3 An Overview of the Learning Acquisition Process

The first serious discussions and development of learning theories emerged during the 1970s with Rumelhart and Normans [124]. These theories argued that knowledge is structured in the form of schemata (i.e. a mental structure of preconceived ideas). The schemata can be modified in three ways: Accretion, Tuning and Restructuring. *Accretion* takes place when a learner has certain disjointed ideas about the material to be learned, with gaps that need to be filled in order to learn. This kind of learning has also been termed gap filling [37]. *Tuning* illustrates the evolutionary changes for interpreting information. Finally, *Restructuring* refers to modifications in knowledge that include the creation of new structures that are constructed either to account for new information or to reinterpret old information. Therefore, during the process of learning, an individual's understanding of the domain can change, usually resulting in a degree of restructuring of knowledge [145].

In light of the aforementioned theory, the following subsections discuss related concepts, which will be discussed with results of Chapters 3, 4 and 6. First of all is the learning curve theory. In addition, the importance of practical experience in learning is presented in this section by Problem-Based Learning (PBL) as one of the best known self-directed learning approaches. Furthermore, the effectiveness of information order on the intentional and incidental learning concepts is discussed.

2.3.1 Learning Curve Theory

Learning Curve Theory is an early investigation of learning concentrated on the performance of individual subjects. This investigation shows that the time required to execute a task reduces at a decreasing rate as experience of the task improves. Therefore, the learning curve signifies a graphical illustration of increasing the learning with experience.

In particular, the term 'learning curve' is exploited in two significant ways: where a body of knowledge is increased over time, or where an identical task is repeated in a number of trials [118]. There is a large volume of published studies describing the role of the learning curve in two main fields: psychology and economics. These are outlined in the following.

Learning Curve in Psychology and Cognitive Fields

In 1885, Wozniak was the first person to explain the learning curve [148]. He conducted an experiment that involved memorising a series of nonsense syllables, then recording the success over a number of trials. Accordingly, the yield results of this experiment were represented by a diagram of learning against trial numbers. Accordingly, learning curve theory is involved explicitly or implicitly in most of the studies discussed previously.

Several learning curve models have been investigated in the cognitive science field. For instance, Card et al. [34] investigated the learning which occurred while using cursor positioning devices by testing the performance on the continuous movement devices against the predictions of Fitts's Law. Four devices were evaluated for this study. As a result, the mouse was found to be fastest on all counts and also to have the lowest error rate. Another study by Anderson proposed in [16] a framework for skills acquisition. The proposed framework includes two main stages in the improvement of cognitive skills. The first stage is procedural in that domain knowledge is directly personified in procedures for performing the skill, whereas the other stage is a declarative one, in which the facts about the skill domain are interpreted. This framework is based on the Adaptive Control of Thought-Rational (ACT-R) production system [17] in which the distinction between procedural and declarative knowledge is essential. The declarative knowledge represents a propositional network, while procedural knowledge represents the productions.

Learning Curve in the economic Field

In 1936, Theodore Paul Wright [149] was the first to attempt to formulate relations between learning variables in quantitative form. He explored the impact of learning on production costs in the aircraft industry. That is, the relationship between the amount of time it takes an organization with a learning rate percentage of r to produce the n^{th} item can be expressed by an operation manager as an equation:

$$T_n = T_1 (n^b)$$

where T_n indicates the time required to complete the n^{th} task, and b indicates $\ln(r)/\ln(2)$, where r indicates the learning rate percentage.

In general, the majority of learning rates r range between 70% and 90%. Usually, this learning percent is determined by statistical analysis of actual cost data for similar products. Stewart et al. proposed in [137] guidelines for estimating learning rates in different situations, as shown in Table 2.3.

Table 2.3: Guidelines for estimating the learning rates in different fields [137].

Field	Learning rate
Aerospace	85%
Shipbuilding	80-85%
Complex machine tools for new models	75-85%
Repetitive electronics manufacturing	90-95%
Repetitive machining or punch-press operations	90-95%
Repetitive electronic operations	75-85%
Repetitive welding operations	90%
Raw materials	93-96%
Purchased parts	85-88%

In consideration of mathematical models, Hackett compared in [57] the efficiency of a selection of models of learning. One of these models was an accumulative learning model proposed by Restle and Greeno [120]. They posited that all information on the activity being learnt is accumulated. Furthermore, a number of methods have already been suggested to model and assess the software development process. For example, Hanakawa et al. proposed in [58] a simulation model for software development that takes into account the developer's learning curve; thus, it can be used to compute a developer's productivity. Particularly, they proposed a knowledge model that shows quantity of gain to a developer's knowledge by executing an activity. This quantity of gain to the developer's knowledge is derived from the relationship between b_{ij} , which is the developer's experience level i while performing the activity j , and θ , which is the required knowledge level to execute this activity. This model is based on the following assumptions:

- If b_{ij} is higher than θ , the developer i does not achieve any new knowledge by executing activity j . This means that the developer's knowledge level is unchanged.

- If b_{ij} is lower than θ , the developer i gains a quantity of new knowledge by executing the activity j . This means the developer's knowledge level is increased. The amount gained depends on the gap between the current knowledge and the required knowledge level.

The simulation knowledge model proposed in [58] is defined as follows:

$$L_{ij}(t) = \begin{cases} K_{ij}e^{-E(\theta-b_{ij}(t))} & (b_{ij} \leq \theta) \\ 0 & (b_{ij} > \theta) \end{cases} \quad 1$$

where $L_{ij}(t)$ is the quantity of gain to knowledge of developer i by executing the activity j , which has knowledge level θ , at time t ; K_{ij} is the maximum quantity of gain to knowledge of the developer i by executing activity j ; b_{ij} is the developer i 's knowledge level about activity j ; E is the developer's efficiency of gain to knowledge by executing activity j ; and θ is the required knowledge level to execute the primitive activity of activity j .

The knowledge level is reset to the developer's new knowledge level b_{ij} at each step:

$$b_{ij}(t+1) = b_{ij}(t) + L_{ij}(t) \quad 2$$

Therefore, by plotting the level of the developer's knowledge in time sequence, the developer's learning curve during the execution of an activity can be obtained. In the simulation of this model, the growth of the developer's knowledge level b_{ij} during the execution of activity j shows the developer's learning curve, as shown in Figure 2.7. In this figure, Line (1) shows the learning curve in the simulation in which the growth of the developer's knowledge level b_{ij} has a great impact on the development progress. Additionally, when the activity is chosen in ascending order of the required knowledge level, then the shape of the learning curve will be flat, as shown in Line (2).

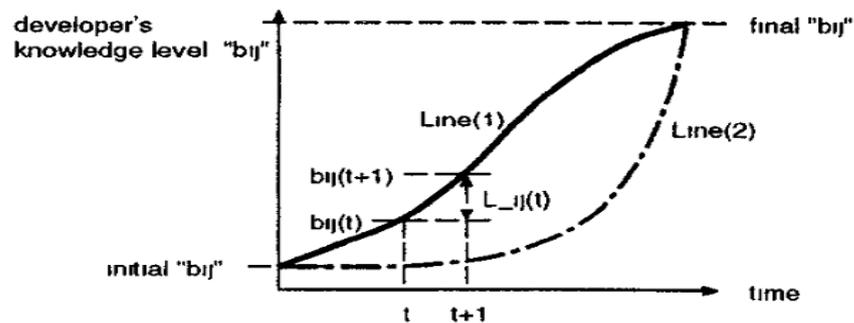


Figure 2.7: Learning Curve in the simulation [58].

Broader Interpretation

The learning curve notion was introduced in the educational and psychology fields, but this notion has gained a broader interpretation over time. For example, “Efficiency Curve”, “Experience Curve”, “Improvement Curve”, “Cost Improvement Curve”, “Learning Curve” and “Progress Curve” are often utilised interchangeably. Generally speaking, all learning displays incremental changes over time. Furthermore, in the economic field, in view of the fact that the development indicates a whole system learning progress with varying rates of progression, the subject is rates of development. In this thesis, Attacker Learning Curve concept is developed that will be detailed in Chapter 6.

Since the attackers deal with the released algorithm as a problem needs to be solved, a Problem-Based Learning approach, which is a widely known self-directed learning skill curriculum approach in the education field, is highlighted in the following subsection. The correlation between this approach and our work is discussed in Chapter 6.

2.3.2 Problem-Based Learning Approach

There is also a large volume of published studies describing the role of Problem-Based Learning (PBL) approach and its significance in improving learning skills. Barrows and Tamblyn [23] are two of the main theorists behind PBL. They have made a key contribution to the development of the next historical stage of medical education. For this, they presented in [23] the scientific basis of the PBL approach in medical education. In addition, they described the methods of problem-based medical learning that have been developed over the years at McMaster University.

A study by Norman analyzed in [107] three concepts: Problem-solving skills, solving a problem and the problem-based learning concept. This study referred to problem-solving skills, as described and measured in medical education, possessing a number of characteristics. Firstly, a skill should be a general strategy. Secondly, it is applicable in a variety of situations. Finally, it is independent of the specific knowledge of situations. Furthermore, this study, supported later by Neville [105], pointed out that despite the heterogeneity of the extensive literature on PBL, there is certainly sufficient cognitive psychological evidence to validate this approach of learning. Not only this, but there is also a significant amount of empirical evidence of effective learner outcomes.

Brown et al. proposed in [31] the improvement of a new cognitive apprenticeship to train students' thinking and problem solving skills to be included in school subjects such as reading. This study introduces a framework that offers a critical lens for evaluating both the advantages and the disadvantages of different learning environments and teaching approaches. Furthermore, this study and others in [77, 29] stated that PBL has dual importance in helping learners not only to construct knowledge, but also to develop strategies.

In another major study, Norman and Schmidt examined in [108] the psychological basis for PBL using theoretical perspectives. As there has been no assessment of the experimental evidence underneath the possible differences in students' learning that can be attributed to PBL, these theoretical perspectives are primarily from cognitive psychology. They found that there is a strong basis for the idea of the PBL and there is an evidence to support its effect on the learning process.

In consideration of enhancing the value of PBL, Kolodner et al. showed in [77] how the suggestions of Case-Based Reasoning (CBR) can improve the PBL approach. In particular, CBR is proposed in [78] as a method for implementing software that can solve problems based on past experience. This method provides a cognitive theory that situates learning in reasoning regarding real-world situations. Also, it has several principals, for example the computational accounts it provides of reasoning activities, particularly of knowledge access, access to old experience (Cases), and use of old experience in reasoning. These principals have been utilized to inform the design of stand-alone learning environments [132]. However, CBR was not able to describe the teacher's role and other issues of classroom practice. As a result, this study found that PBL methodology could offer principals of practice to go along with CBR's educational principals. To achieve the goal of this study by combining PBL and CBR, it is necessary, for example, to analyze why PBL works can contribute to understanding how to move PBL to a new environment; as well as to consider the design of computer programs to facilitate the learning from a problem-solving activities point of view.

Hmelo and Ferrari investigated in [62] how the tutorial process in PBL can be used to develop higher order thinking skills. This investigation takes into account the role of the problem, the facilitator role, collaboration among the students, and the importance of students' reflection. For example, facilitators are responsible for encouraging all stu-

dents to be actively involved, encouraging them to express their thoughts and critically respond to comments of other students. Two forms of guidance are recommended to assist learners in engaging in meaningful independent learning. The first is that prior independent learning activities with discussion should be held. The second is that the location of learning resources may prove useful in learning additional concerning assigned learning issues. This study concluded that because PBL places the learning ability in real-world problems, it is well suited to help students not only to become active learners, but also to develop strategies and construct knowledge.

In the following subsection, the effectiveness of representing information order in the learning acquisition process is reviewed.

2.3.3 Effects of Information Order in Learning Process

Following Rumelhart and Norman's learning theory [124], a number of studies have investigated whether category learning is influenced by the order in which examples are presented.

Since this thesis evaluates whether the order in which different defensive mechanisms are released will impact the time an attacker needs to break each one of them, effects of information order in the learning process is reviewed in this section in order to be compared with our results in Chapters 3 and 4.

Elio and Anderson investigated in [47] the difference between two models of schema abstraction⁹: the Generalization model and the Instance-only model. This investigation was done by conducting three experiments. In the first experiment, the aim was to manipulate the likelihood of forming category generalization in two different sets of study exemplars, keeping the similarity of transfer items between two study sets as constant as possible. In the second experiment, two generalized conditions were constructed: the first condition, in which forming generalization may be facilitated by blocking, and the second condition, in which forming generalizations was hindered by random presentation of instances. In the third experiment, a generalized study set and a control study set were designed, each with its own transfer item set and the relationship between the transfer set and the study set was set to be as equivalent as possible for both generalized and control materials. The results of these experiments are as follows. In the experi-

⁹ The schema abstraction refers to a mental structure of preconceived idea [47].

ments 1 and 3, accuracy and confidence on transfer items were better in generalized condition than in the control condition, whereas in experiment 2, study items were learned faster and transfer performance was better with blocked presentation than with random presentation. In all experiments, there was an effect on the similarity of transfer items to study materials. Furthermore, when training is blocked into groups of mutually similar examples, then categories are learned faster.

The same authors, Elio and Anderson, evaluated in [48] the effects of information order and variance on schema abstraction. This evaluation was accomplished by measuring the transfer performance after different numbers and different types of category exemplars had been studied. The stimuli were descriptions of people belonging to one of two clubs and all the categories were constructed using numerical notation. For example, an item such as 1113 may translate as “Works for the government, is college educated, is single, plays chess, like jazz.” To test the effect of category variance on the schema abstraction, a large variable category in which different number types both shared many overlapping features and had unique feature patterns was needed. The results of this evaluation revealed that transfer performance was better if subjects began with a low-variance sample and were gradually introduced to the allowable variation on subsequent samples than if they consistently saw representative samples. On the other hand, this information order effect may interact with a teach model subject to be more analytical about the material performed better if their initial and subsequent samples were representatives of the category variation.

In 1994, Medin and Bettger examined in [98] the influence of order of examples on old-new recognition memory¹⁰. Specifically, there were two groups; each can see exactly the same set of examples but in two different orders. The first group’s order maximizes the similarity of successive examples, while the second group’s order minimizes the similarity. When the sequence of examples has been presented, the participants are given an old-new recognition test. If shared properties of successively presented examples are selectively strengthened, the two orders should produce both main effect and interactions in recognition. This study concludes that a strong learning advantage can be achieved when training objects are presented in an order that tends to maximize the similarities between successive examples. Related work on the effect of semantic or-

¹⁰ Old-new recognition memory refers to measure item-specific memory, devoid of inter-item associations [98].

ganization on word recognition is consistent with this prediction. Particularly, when word lists are compared with words from categories, recognition is better if examples are blocked by the category than if they are overlapping [102].

Another study by Clapper and Bower investigated in [40] the difference between two general methods of an unsupervised category learning concept. The first method is based on learning explicit correlation rules or associations within a stimulus domain. The second method is based on inventing separate categories to capture the correlation structure of the domain. The unsupervised category learning is not arbitrarily predefined by the experimenter; rather, subjects should discover categories or explore a given stimulus domain concerning a sequence effect. This investigation was accomplished by experimental studies. There were three conditions: blocked condition, mixed condition and control condition. In the blocked condition, the stimuli were partitioned into two categories based on patterns of correlated attribute values. In the mixed condition, instances of both categories were randomly interspersed in the training sequence rather than being grouped into a separate block. Finally, in the control condition, as none of the attributes were distinct categories, all attributes of the stimuli varied independently. The results showed that the overall score of learning was higher in the two correlated conditions (blocked and mixed) than in the controlled condition. Also, the learning was higher in the contrast condition than in the practice condition throughout the experiment. Additionally, the learning increased while the number of instances increased. Moreover, it was clear from the subjects' performance in this study that diagnostic features of many of the fuzzy categories used in standard supervised learning experiments are often highly unreliable.

To maximize comparison and understanding how memory affects category learning, Sandhofer and Doumas examined in [131] sequencing training instances experimentally. The results of this experiment indicated that the learning process significantly increases when learning begins by interacting with a limited set of highly similar exemplars. However, the process increases more slowly when the instances are distributed and dissimilar. The results of this study showed that the information order effects were examined with a symbolic connectionist model of general learning and representation discovery. In short, this study suggested that when a presentation of examples is ordered in such a way that discrete instances of a category could be more readily connected in memory, category learning and discovery are more likely to occur. In addition, begin-

ning learning by interacting with a limited set of highly similar exemplars leads to more learning than when instances are distributed and dissimilar.

It is important to recognize that the presentation order effects are especially interesting in the light of categorization models that emphasize incremental learning from trial to trial. For instance, Sakamoto et al. examined in [126] how people learn about the variance of categories based on two basic approaches of explaining the nature of the mind: the mechanistic and rational approaches. The mechanistic models attempt to simulate human behaviour using processes analogous to those used by humans, while the rational analyses attempt to characterize the environment and the behavioural effects that humans look to optimize. In this study, the authors' argument was that mechanistic models are best suited for driving surprising behaviour predictions. Their argument was validated empirically. The overall results demonstrated that the participants showed sensitivity to category variability and assigned transfer items which lay between two categories to the higher variability category. This assumes that people can learn the mean and the variability of each category and use this information to classify new items. Furthermore, although one category was ordered semi-regularly, both ordered and random fashions had equal variance. Therefore, the difference between the stimuli on successive trials was small. In short, this study concluded that people assess variability by building incremental adjustments to memory representation of the source of local comparisons.

Mathy and Feldman recently investigated in [91] the mechanism in which concepts are learned from examples by manipulating the presentation order. In particular, they introduced the idea of a rule-based presentation order. This idea is a sequence that respects the internal organization of the examples within a category. In this investigation, the performance of subjects with the rule-based presentation order was compared with both the similarity-based and dissimilarity-based orders. The hypothesis behind this was that the rule-based presentation order would better facilitate the learning process compared to other approaches, especially in highly structured concepts. As a result, their study yields a better learning approach compared to the similarity-based order approach that maximizes the adjacency of the training examples previously found to be most advantageous in artificial classification tasks [47, 48, 98]. Furthermore, the proposed order in this study was better in terms of learning than the dissimilarity-based order. To show the results in a meaningful way, the learning curves of all presentation orders, as shown in Figure 2.8, illustrate the influence of a presentation order on learning. Specifically,

ranking of effectiveness of the three presentation orders was visible in learning curves and can be expressed as follows: rule-based > similarity-based > dissimilarity-based. More importantly, the three learning curves were statistically distinct.

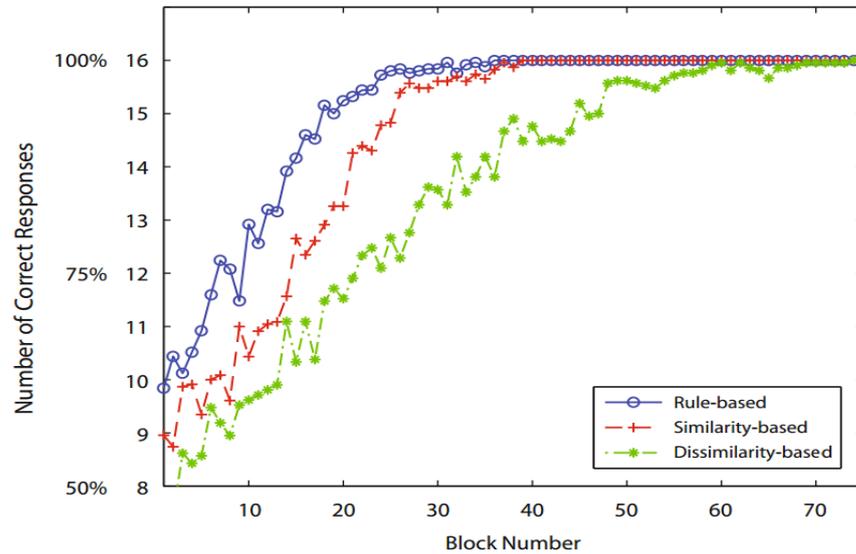


Figure 2.8: Performance over time for all three presentation order types [91].

The differences between the presentation orders depend on the nature of the concepts learned. Particularly, the rule-based presentation order gives a substantial benefit in case the category is highly structured. As such, it contains salient sub-categories around which the presentation order can be organized. It is more likely that a rule-based presentation is, in effect, a random presentation. However, a similarity order might encourage a sequence of short-term over specific hypotheses (blind alleys) based on accidentally contiguous examples, which would impede learning.

2.4 Security Quantification and Attack Modelling

As the work in this thesis considers utilizing a quantitative measure as a target approach to the release order of defensive mechanisms, this quantitative measure is reviewed in this section under the attack modelling umbrella.

In the dependability community, there are several well-known and efficient approaches for quantifying reliability, availability and safety. In the security community, there is a motivation behind quantified security that is a variation of the following thought: since we cannot measure it, we cannot control it. In trying to ascertain how well security requirements are met, a significant challenge is to provide an accurate knowledge of secu-

rity properties in relevant operational settings. To address this problem, the quantification of security can be a solution to such needs. Mostly during the past decade, therefore, numerous studies have been accomplished on applying the dependability paradigm to security. Furthermore, it has been claimed in scholarly literature and by leading standards organisations that such quantification is not only possible, but also beneficial and can even be necessary for good security management [15, 27, 36, 64, 113]. In this section, the importance of attacker behaviour in quantifying security, and the estimation of the time to compromise a system component that is visible to an attacker and game theory are highlighted.

2.4.1 Security Quantification and Attacker Behavior

The relationship between attack modelling, security quantification and the attacker behaviour has been widely investigated [5, 30, 70, 80, 89, 110, 127]. A groundbreaking paper was proposed by Brocklehurst et al. in [30] as a first attempt towards operational measures of computer security. They developed a quantitative theory of operational security by conducting an experiment in which attackers would be allowed to break a system under controlled conditions. The *Mean Effort to Security Breach* as a quantitative measure is defined and discussed. The study pointed out two significant results: the lack of quantitative measures for determining operational security and relative security assessment to the reliability domain. In addition, the study stated that quantifying the contribution that is made by different resources, as well as the ability and experiences of the attacker need more investigation.

Based on empirical data collected from an intrusion experiment, Jonsson and Olovsson [70] devised a hypothesis on typical attacker behaviour. The hypothesis suggests that the attacking process can be divided into three phases: the learning phase, the standard attack phase and the innovative phase, as shown in Figure 2.9. The probability for successful attacks in the standard attack phase is expected to be considerable, while in the learning and innovative phases it is expected to be small. That is, the inexperienced attacker spends more time in the learning phase before actually crossing the attacking skill threshold, as indicated in Figure 2.10a. On the other hand, the experienced attacker progresses much faster in the following phases, as indicated in Figure 2.10b.

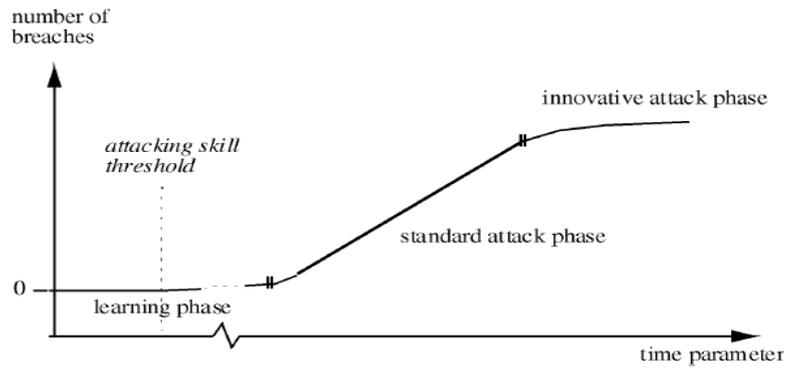


Figure 2.9: A typical attacking process [70].

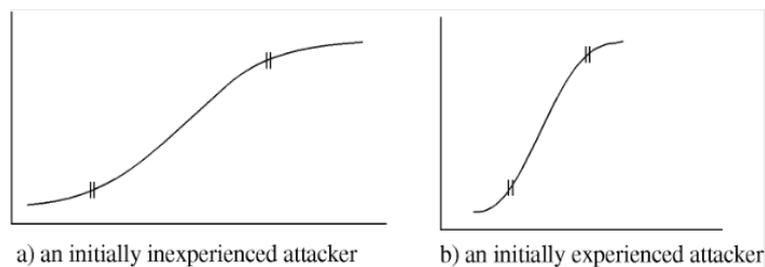


Figure 2.10: The standard attack phase [70].

In order to model the attack behaviour, a study by Sallhammar et al. investigated in [127] the attacker behaviour that can be integrated in the transition rate matrix of a stochastic model for operational security evaluation. It aims to provide a realistic measure of operational security and an approach was recommended by the authors to compute the expected behaviour for rational attackers. In this approach, it is considered that the attacker rewards the possible costs - if the actions are detected by the system - and the probabilities of succeeding with particular attack actions.

Almasizadeh and Azgomi developed in [5] a method for quantifying security based on the assumption that a typical attacker needs time to perform the attack phase. During the attack, the attacker can be detected by the system; therefore, the overall attacking process is interrupted. In addition, a generic model which focuses on evaluating security and allowing analysis of the security of systems capable of detecting and responding to attacks has been developed in [89].

In 2013, Krautsevich et al. described in [80] their initial ideas on modelling the behaviour of an attacker that has uncertain knowledge about a computer system. Their model is based on Markov Decision Processes (MDP) theory, which will be explained later in Section 2.6.2, for predicting possible attacker's decisions.

Ortalo et al. developed in [110] a tool to evaluate the security of a system based on attacker behaviour. That is, in order to evaluate the quantitative measure characterizing the operational security based on a privilege graph, it is necessary to recognize the scenario of attacks that can be attempted by a possible attacker to achieve the objective. Therefore, this study describes a technique for transforming a privilege graph into a Markov chain. Thus, as a result of a series of atomic attacks on a system, the resulting Markov chain signifies enhanced privileges gained by an attacker.

Sallhammar et al. presented in [128] a new approach to integrate security and dependability evaluation. Their approach is inspired by all the previous studies and is based on the underlying assumption that an attacker has a complete knowledge of the system states, the possible transitions between states and existing vulnerabilities. However, this assumption might not be accurate in a real life scenario.

Aghajani and Azgomi presented in [2] a high-level stochastic model to evaluate several significant security measures of a multi-layer. Their model is based on stochastic activity networks that are a stochastic extension of Petri nets. This model can capture attacker behaviour and the system responses of intrusion tolerant web services architecture. Similarly, Sahner et al. used quantitative modelling techniques in [125] to evaluate security properties. Additionally, Goseva-Popstojanova et al. proposed in [54] a state transition model depicting the dynamic behaviour of an intrusion tolerant system. This model contains a framework defining the vulnerability and the threat set. Several pieces of work use the concept of the MDP in the context of security. For instance, Kreidl introduced a simple MDP in [81] with only three states (normal, under attack and failure) and three decisions (wait, defend and reset), which analyses the cost of defence countermeasures against the cost of an intrusion.

Likewise, Roy et al. proposed in [122] an Attack Countermeasure Tree (ACT) to consider both attacks and countermeasures in an attack tree structure. They devised several objective functions based on greedy, branch and bound techniques. The aim of these functions is to reduce the number of countermeasures, reduce investment cost, and exploit the benefit from implementing a certain countermeasure set. In the design of this study, each countermeasure optimization problem can be solved not only with probability assignments to the model, but also without this probability. However, their solution focuses on a static attack scenario and predefined countermeasure for each attack.

2.4.2 Time-To-Compromise System

In recent years, there has been an increasing amount of literature on the time taken for an attacker to compromise a system and misuse its resources. McQueen et al. proposed in [96] a new model for estimating the time to compromise a system component that is visible to an attack. This study provides a function of known and visible vulnerabilities and attacker skill level that can estimate the expected value of the time-to-compromise. The proposed model suggests a number of strategies for reducing the risk of cyber threats. For example, the time necessary for an attacker to compromise components can be increased theoretically by government restrictions on the publication of valid exploits. Moreover, the authors emphasized that unless there is a constant effort to disable services as soon as a new vulnerability is discovered, the dynamic nature of cyber security is decreasing over time. However, there were a number of downsides in the model, such as the estimation of available exploits to various skill levels of the attackers was not validated.

Kadota et al. considered the constrained regret-optimization problem for a semi-Markov decision process in [71]. As such, the expected regret-utility of the total reward earned until the reaching time to a given absorbing subset is minimized subject to multiple expected regret-utility constraints. Therefore, a saddle point theorem is achieved and the existence of a constrained optimal policy proved by introducing a corresponding Lagrange function.

In 2006, McQueen et al. suggested in [97] a methodology for obtaining a quantitative measurement of the risk reduction. This risk reduction is achieved when a control system is modified with a view to enhancing cyber security defence mechanisms against attackers. Furthermore, in this study as well as in [96], it was recommended that the attackers' skill levels should be considered when determining the mean time to compromise a system. In addition, these studies pointed out that a number of techniques proposed for estimating cyber security are likely to require considerable details about the target system. This can make these techniques unmanageable as a comparative tool for multiple systems. Therefore, this was addressed by Leversage and Byres in [83] by offering a model which focuses on being a comparative tool and becomes a more generally applicable methodology, though still allowing meaningful comparisons. They stated that the selection of time as the unit of measurement is fundamental to the model's

strength. Similarly, Nguyen and Sood proposed in [106] a quantitative analysis of Self-Cleansing Intrusion Tolerance (*SCIT*) that is time-based intrusion tolerance architecture. By utilizing *SCIT* to systems, engineers are offered the capability of adjusting the system of intrusion tolerance specified by *Mean Time To Security Failure* (MTTSF).

Paulauskas and Garsva provided simulation results in [112] to evaluate the computer system security by using Mean Time-to-Compromise criteria. They highlighted that the effect of the attack depends largely on the attacker's skill level. Thus, their study suggests a normal skill level distribution in the skill group.

2.4.3 Game Theoretic Security Approach

Game theory is a mainstream research topic in the economic community. Economic concepts have been applied to computer security to address the analysis of strategic choices between attackers and defenders at an assumed cost. As game theory views the interactions between attacker and defender as two players, it can provide a mathematical framework for analyzing and modelling network security problems. In traditional network security solutions, McInerney et al. discussed in [94] one of the first approaches to applying game theory to network security. Feedback Reasoning for Information Assurance Response System is proposed (FRIARS). A FRIAR is an automatic Information Assurance technique and is based on a MDP.

Lye and Wing modelled in [88] the interaction between an attacker and a defender as a two-player stochastic game. By using their model, the best-response strategies for the players (attacker and defender) can be computed and then the results can be used by the administrator to enhance the security of the network. Generally, their study showed how a very general game-theoretic formalism can be applied to the security concept.

Alpcan et al. investigated in [6] the problem of Nash Equilibrium Design for a general class of games from an optimization and control theoretic perspective. Specifically, this study considered how long it took the game to approach a Nash equilibrium when many players were trying to solve it in a distributed way. A feedback system approach is suggested as a control input to make the system robust and to control the system's progress.

A study by Jiang et al. examined in [67] an active defence using an approach to attack prediction. In particular, the study systematically identifies cost factors of a cost-sensitive model and introduces the attack strategy prediction and optimal active defence

strategy decision algorithm. In addition to this study, Alpcan and Baser utilized in [7] the Min-max Q learning approach in order to gradually improve of the defender's quality. This work can handle reactive defence actions, while Shive et al. proposed in [134] proactive defence measures based on a game theory inspired defence architecture.

A game theory is also recommended in [129] as a method for modelling the probabilities of expected attacker behaviour in a quantitative model. This method models the penetration attempt as a series of intentional state changes. These changes lead the secure system from an assumed secure state to a state in which one or more of the systems aspects are attacked. At each intermediate stage of the attack, the attempt might be detected by the system to bring the system back to the first secure state.

Sallhammar et al. proposed a game theory approach in [130] to attack modelling as a means for computing and therefore predicting the expected attacker strategy. In their study, the possible use of the Nash Equilibrium as a part of the transition probabilities in a state transition model is defined.

The interaction of an attacker and the network administrator as a repeated game is modelled in [8] with finite steps or infinite steps. In particular, the study is focused on establishing a quantitative approach with a practical degree of abstraction in order to analyse the underlying principles for the development of Intrusion Detection Systems (IDSs). By utilizing a modified version of this study, Bloem et al. introduced in [28] an Automatic or Administrator Response (AOAR) algorithm for allocating the time that a system administrator has available to respond to attacks. Although it lays out a practical implementation of an algorithm and demonstrates its utility, the study lacks a formal theoretical framework.

Roy et al. recently conducted a comprehensive survey [123], concluding that most of the current research in game theory are based on static games, games with perfect information, or games with complete information.

There are a number of detection techniques that can be used to improve the protection of a system, such as Anomaly Detection Techniques. Due to the fact that a type of anomaly detection technique is part of the work undertaken in this thesis, as shown in Chapter 7, it is reviewed in the next section.

2.5 Anomaly Detection Techniques

There exist a considerable number of studies that consider a hybrid¹¹ design approach based on anomaly detection techniques, as shown in [84], but these formulations do not fit exactly with the approach in this thesis that shown in Chapter 7. Thus, this section mainly highlights the related works on hybrid machine learning based classifiers in which an unsupervised machine learning¹² algorithm is applied as a first layer for observing an attack attempt, as summarised in Table 2.4.

Table 2.4: Hybrid approaches in which an unsupervised machine learning algorithm is applied as a first layer.

Author	Method 1	Method 2	Evaluation	Methodology
Khan et al. [73]	Unsupervised	Supervised	Simulation	Clustering, SVM
Liu et al. [86]	Unsupervised	Supervised	*	Clustering, SOM
Liu and Yi [87]	Unsupervised	Supervised	Simulation	SOM, Neural
Gunes et al. [55]	Unsupervised	Unsupervised	**	SOM
Horng et al. [65]	Unsupervised	Supervised	**	Clustering, SVM
Zhang et al. [153]	Unsupervised	Supervised	**	SOM, KAA
Ours [14]	Unsupervised	Supervised	Real data	Density-based, DLDA

* 1998 DARPA Intrusion Detection Evaluation Data and TCP dump raw data.

** KDDCUP99 training data.

A Dynamically Growing Self-Organizing Tree (DGSOT) algorithm is used in [73] to train Support Vector Machines (SVM) for classification and reducing both false negatives and false positives. Similarly, Liu et al. proposed in [86] an IDBGC algorithm (Intrusion Detection Based in Genetic Clustering). This algorithm includes two stages: a nearest neighbour clustering and a genetic optimization. The main purpose of the first stage is to decrease the size of data objects to a reasonable one; therefore, it can be suit-

¹¹ A hybrid approach typically consists of two functional components. The first one takes raw data as input and generates intermediate results. The second one will then take the intermediate results as the input and produce the final results [84].

¹² The unsupervised algorithm seeks out similarities between pieces of data in order to characterize them, whereas the supervised algorithm builds a concise model of the distribution of class labels in terms of predictor features.

able for genetic algorithms in the second stage. Their experiment is evaluated by using 1998 DARPA Intrusion Detection Evaluation Data and TCP dump raw data. In addition, Liu and Yi proposed in [87] a modified, unsupervised learning algorithm PCASOM (Principal Components Analysis and Self-Organizing Map) and neural networks as a detection method. Their simulation is carried out to illustrate the performance of the proposed method by using DARPA 1998 evaluation data sets. Therefore, the detection rate is 94.286% (660 out of 700).

Gunes et al. investigated in [55] a hierarchy of Self-Organizing Feature Maps as an ID approach. They demonstrated that using a two-layer self-organizing map (SOM) hierarchy, based on all 41-features from the KDDCUP99 training set, can achieve 90.4% a detection rate under test conditions; interestingly, this represents the best performance based on an unsupervised learning algorithm.

Moreover, Horng et al. proposed in [65] an SVM-based network intrusion detection system with algorithm clustering for data pre-processing. This algorithm could provide highly qualified, abstracted and reduced datasets. According to their experiment on the KDD Cup 1999, the proposed system could reach an accuracy of 95.72%.

Recently, Zhang et al. proposed in [153] a novel approach by combining the SOM and the kernel auto-associator (KAA). The SOM organizes the prototypes of samples while the KAA provides data description of normal connection patterns. Using the KDD CUP, 1999 dataset, the performance of the proposed scheme was compared with some state-of-the-art novelty detection methods in terms of separating normal connection patterns from intrusive connection patterns. This showed marked improvements in terms of the high intrusion detection accuracy and low false positives.

As a stochastic model is a part of the work carried out in this thesis as a target formalism when modelling the release order of defensive mechanisms, an overview of this stochastic model is presented in the next section.

2.6 Stochastic Modelling Formalisms

Since the likelihood of the occurrence of large various daily events is probabilistic making them described as stochastic processes, these events are modelled using stochastic models which rely on probabilistic theory. The stochastic process is a mathematical rep-

resentation of the system with probabilistic or random characteristics. Furthermore, it forms the system behaviour as a function of times which can be continuous or discrete [90].

In order to identify the stochastic process formally, numerous definitions require introduction first. The first definition is of the *Random Experiment*. This is an experiment that can have one or more potential results (e.g., student's marks). Secondly, the *Sample Space* of this experiment is a set of all potential results which can be finite or infinite (e.g., a positive integer between 0 and 100). When a single result is achieved from the *Sample Space* (e.g., a student's mark is 62), then it is called a *Sample Point*. Another definition is the *Random Variable* that is a function known over the *Sample Space* of an experiment. It provides a real number to each result from the *Sample Space*. An example of this is the random variable *Pass* that gives fail (i.e. 0) results to students whose marks are under 50, while it gives passes (i.e. 1) otherwise [99].

Given these aforementioned definitions, the Stochastic Process $\{X_t, t \in T\}$ is defined as a set of random variables sorted by a parameter, from an indexed set, T that mostly represents a time t . Therefore, X_t is supposed to be the current state of the system at time t , and the *State Space* S of this process is the set of all the random variable values [99].

The State Space S of a stochastic process can be either *Discrete* or *Continuous*. The *Discrete* is when the states can be counted by positive integers, while the *Continuous* is the opposite. Consequently, the stochastic process is supposed to be a *Continuous-State Stochastic Process* (or a Chain) if its state space is continuous (e.g., the waiting time of jobs to be served), or a *Discrete-State Stochastic Process* if its state space is discrete (e.g., the number of job arrivals). Additionally, the index set T can be continuous if the process is examined during an interval of time, or discrete if the process is examined in specific time instants. As a result, the stochastic process can be a Continuous-Time Stochastic Process if its time parameter is continuous (e.g., during the whole week), or a Discrete-Time Stochastic Process if its time parameter is discrete (e.g., every day of the week). Bearing these in mind, the stochastic process can be one of four types depending on the combination between state types and time types. These types are as follows: (1) *Continuous-Time Continuous State Space Stochastic* (e.g., the waiting time of customers arriving at any time of the week); (2) *Continuous-Time Discrete State Space Stochastic Process* (e.g., the number of customers waiting in a shop at any time of the

week); (3) *Discrete-Time Discrete State Space Stochastic Process* (e.g., the number of customers waiting in a shop every day of the week); and finally (4) *Discrete-Time Continuous State Space Stochastic Process* (e.g., the waiting time of customers arriving every day of the week) [25, 59, 99].

Most existing stochastic processes offer several kinds of dependence between the states that have previously occurred, the current state and the future state. As an example, the total gain of a person after n coin flips depends on the gain at the end of the $(n-1)^{\text{th}}$ flip. However, as this dependence becomes more complicated, the analysis of such systems becomes more difficult. For this reason, a process with dependence of the first-order is desirable [25]. There is a set of stochastic processes that exploits this particular kind of dependency of the system states. This is called the *Markov Property* or the *Memoryless Property* that considers the future state. This consideration relies only on the present state, and this property is independent of the previous states or the time spent in the current state. In other words, the firing rate of system activities is exponentially distributed [99]. The *Markov Process* is a stochastic process that satisfies the *Markov Property*, whereas in the *Semi Markov Process*, the sojourn time of the current state impacts the next state (i.e. the *Memoryless* property of the state's sojourn time is not valid) [59].

As stated previously, the stochastic process illustrates the behaviour of the system with states and activities that change them [119]. In order to assist the application of an analytical and numerical solution, these stochastic models often utilise a Markov or Semi-Markov chain. Although this describes the system at a low level, given all the states and transitions that the system may go through, high-level modelling formalisms, such as *Stochastic Petri Nets* (SPNs), are often used because of the complexity of giving a full representation of every state and transition in a concrete system. These formalisms are then automatically transformed into the core *Markov* or *Semi-Markov chain* [140].

In the next sub-sections, three modelling paradigms that are used to describe stochastic systems are reviewed. Since *Semi-Markov* mode and *Stochastic Discrete Event System* are not utilised in this thesis, they will not be described. Only the paradigms that have been used in this thesis are described; these are the *Markov Chain*, the *Markov Decision Process* model and the *Stochastic Petri Net* (SPN). Furthermore, the software tools for building and solving models are also described.

2.6.1 Continuous-Time Markov Chain

The *Markov Chain* is a mathematical model that transitions from one state to another between a finite or countable state space [109]. The term “*Markov Chain*” indicates the sequence of random variables X_1, X_2, \dots, X_n with the Markov Property. Formally:

$$\Pr(X_{n+1} = x | X_1 = x_1, X_2 = x_2, \dots, X_n = x_n) = \Pr(X_{n+1} = x | X_n = x_n)$$

The stochastic process $\{X(t), t \geq 0\}$ is a continuous-time Markov chain if for all $s, t \geq 0$ and nonnegative integers $i, j, x(u), 0 \leq u < s$ [121]:

$$\begin{aligned} P\{X(t+s) = j | X(s) = i, X(u) = x(u), 0 \leq u < s\} \\ = P\{X(t+s) = j | X(s) = i\} \end{aligned}$$

Particularly, a continuous-time Markov chain is a stochastic process which has Markov property and in which the conditional distribution of the future $X(t+s)$ given the present $X(s)$ and the past $X(u), 0 \leq u < s$, relies merely on the present and is independent of the past.

Additionally, once $P\{X(t+s) = j | X(s) = i\}$ is independent of s , then the continuous-time Markov chain is supposed to have stationary or homogeneous transition probabilities. For example, a continuous-time Markov chain enters state i at time 0, and the process does not leave state i during the next 10 minutes. The probability that the process will not leave state i during the following 10 minutes can be described as follows. By the Markov chain property, the probability that the process remains in state i during the interval $[10, 20]$ is just the unconditional probability that it stays in state i for at least 10 minutes [121]. As such, if T_i indicates the amount of time that the process stays in state i before making a transition into a different state, then:

$$P\{T_i > 20 | T_i > 10\} = P\{T_i > 10\}$$

Or generally, by the same reasoning

$$P\{T_i > s+t | T_i > s\} = P\{T_i > t\}$$

for all $s, t \geq 0$. Therefore, the random variable T_i is *memoryless* and should thus be *exponentially* distributed.

Based on the above, a continuous-time Markov chain can be defined as follows: it is a stochastic process with properties which mean that each time it enters state i :

- The amount of time it spends in that state before making a transition into a different state is exponentially distributed with the mean, and
- If the process leaves state i , then it next enters state j with some probability P_{ij} that satisfies the following: $P_{ii} = 0$ for all i and $\sum_j P_{ij} = 1$ for all i .

Specifically, a continuous-time Markov chain is a stochastic process that moves from one state to another in accordance with a discrete-time. However, it is such that the amount of time it spends in each state is exponentially distributed. Moreover, the amount of time that the process consumes in state i and the next state visited should be an independent random variable. In the case that the next state visited was dependent on T_i , information as to how long the process has already been in state i would be relevant to the prediction of the next state.

As an optimisation problem is a part of the work undertaken in this thesis that is solved via a Markov Decision Process model, as shown in Chapter 5, this model is reviewed in the next subsection.

2.6.2 Continuous-Time Markov Decision Process

The Markov Decision Process (MDP), which is an extension of the Markov chain, provides a mathematical framework for modelling decision making in conditions where the results are slightly random and slightly under the control of a decision maker. Therefore, the MDP is a constructive tool for studying a wide range of optimisation problems [109]. Formally, the MDP is a 4-tuple $(S, A, P_a(s, s'), R_a(s, s'))$, where

- S is a finite set of states
- A is a finite set of actions
- $P_a(s, s') = Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$ is the probability that action a in state s at time t leads to state s' at time $t+1$
- $R_a(s, s')$ is the immediate reward received after the transition from state s to state s'

The core problem of the MDP is to find a *policy* for the decision maker. This policy is a rule π for choosing actions when in state s . Typically, a policy π is a set of numbers $\pi = \{\pi_i(a), a \in A, i = 1, \dots, M\}$ with the understanding that if the process is in state i , then action a is to be chosen with probability $\pi_i(a)$ as follows [109]:

$$\pi_i(a) = \begin{cases} 0 \leq \pi_i(a) \leq 1, & \text{for all } i, a \\ \sum_a \pi_i(a) = 1, & \text{for all } i \end{cases}$$

In the continuous-time Markov Decision Process, the decision can occur at any time chosen by the decision maker. Thus, one of the best ways to model the decision making process for a system that has continuous dynamics is by using the continuous-time Markov decision process. The continuous-time Markov Decision Process (CTMDP) is defined as follows. In the case that the state space and action are finite, the definition is [56]:

- S : State Space
- A : Action Space
- $Q(i/j, a): S \times A \rightarrow \Delta S$, a transition rate function
- $R(i, a): S \times A \rightarrow R$, a reward function

In the case that the state space and action are controlled [56]:

- X : state space
- U : state of possible control
- $f(x, u): X \times U \rightarrow \Delta X$, a transition rate function
- $r(x, u): X \times U \rightarrow R$

A reward rate function is such that $r(x(t), u(t))dt = dR(x(t), u(t))$, where, $R(x, u)$ is the reward function [56].

In this thesis, CTMDP is utilised to develop an optimisation algorithm that will be explained in Chapter 5. Moreover, as the release order of defensive algorithms is modelled by using *Stochastic Petri Nets* (SPNs), as shown in Chapter 4, the next sub-section reviews this SPN approach.

2.6.3 Stochastic Petri Nets

In order to achieve a qualitative measure, Petri Nets (PNs) are widely used for modelling systems with simultaneous and chronological transitions in order to obtain qualitative measures. They are also very effective in representing system concurrency and synchronisation [90]. The Petri net is a type of directed graph with an initial state called the initial marking. The basic graph of a Petri net is a directed, bipartite graph consisting of two kinds of nodes. The first are called places, and the second transitions. Arcs (or arrows) connect places and transitions. The arcs can only connect a place with a transition or a transition with a place. Connections between two nodes that are of the same kind are not allowed (e.g. that connect a place with a place).

In graphical Petri net representations, places are drawn as circles and transitions as bars or boxes. The initial distribution of tokens among places is called the initial marking of the Petri net. The marking of tokens over places determines the state of a Petri net. A transition is enabled if each place connected to the transition contains at least one token. The firing of transitions changes the distribution of tokens and produces new states. Transitions are generally assumed to be the active components of Petri nets. Transitions can stand for tasks, events, operations, transformations, transportation, and so on.

However, places are usually passive and they could represent a medium, phase, and condition. Tokens often indicate physical objects or represent information. The flow of these tokens and the firing of transitions are then used to model the dynamic behaviour of the system. Figure 2.11 illustrates a simple example of a graphical Petri net, where the circles represent places (P1, P2, P3 and P4), the bars correspond to transitions (t1 and t2) and tokens are represented by small dots (in P1).

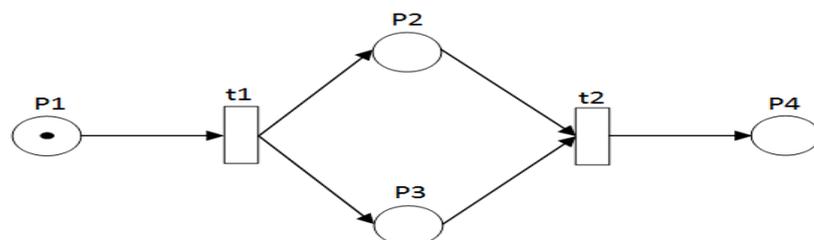


Figure 2.11: A Graphical Petri Net Example.

A Petri net model can use transitions to represent tasks and places which stand for the pre- and post-conditions of the tasks or resources involved in the system. A transition is

enabled if each input place contains at least a number of tokens which equal the weight of the flow relationship from the places to the transition; when a transition fires, it consumes a number of tokens from each input place and produces a number of tokens equal to the weight of the flow relation from the place to the transition. For example, t_1 in Figure 2.11 produces two tokens to P_2 and P_3 .

The numbers of tokens contained in the place are used to define the situation or condition of a Petri net state. When firing transitions, removing tokens from input places and adding them to output places identifies a change in state and describes the dynamic behaviour of a Petri net. The initial marking denotes the distribution of tokens over places.

The arcs connect places with transitions or vice versa. Arcs are grouped into the two types that are used in this thesis with respect to transitions. These types are as follows: *input arcs*, which go from places to transitions shown by arrows, and *output arcs*, which go from transitions to places shown by arrows.

In the interest of allowing the extraction of quantitative and time-related performance results, Stochastic Petri Nets (SPNs) were introduced by assigning exponentially distributed random functions to the delay of the Petri net transitions. By means of this exponential distribution, the state of the modelled system can be changed in a probabilistic way. Not only this, but it also allows the estimation of more cumulative performance results from the steady state distribution like the average delay. Moreover, since SPNs with the exponential distribution and Continuous-time Markov chain both use the *memoryless* property of transition firing, the exponential distribution allows SPNs to resemble Continuous-time Markov chains [90].

One of the SPN's features is that it provides an integration of graph modelling and probabilistic modelling. This feature allows a system's behaviour to be analysed. As such, while an SPN is capable of giving a visual description of a system process, it is automatically transformed into the underlying Markov Chain model for performance analysis [3]. In order to achieve performance results, firstly the defined reward variables should be converted to their equivalent essential state-level stochastic processes with the corresponding rewards specified at the state level [3].

There are several limitations of an SPN. Firstly, as the SPN is mostly able to model small-sized systems, SPN graphs become extremely complicated when the system size

increases. This leads to a significant increase in the number of Markov states. Furthermore, transitions with low impact on the model might not require being associated with an exponential distribution, since the SPN needs to associate an exponential distribution with each transition. That is, removing the time from the transition with low impact might result in a smaller number of Markov states. Therefore, the performance extraction is simplified.

On the other hand, *Generalised Stochastic Petri Nets* (GSPNs) were constructed in [3] to overcome the aforementioned shortcomings of SPNs. GSPNs introduce two types of transitions: *Timed* and *Immediate*. The former transition is included with an exponential-distributed delay function, while the latter is included with zero time delay. Accordingly, a delay function is merely related to timed transitions and is dependent on a place marking. In immediate transitions, a firing priority is declared over timed transitions if they are both enabled. If multiple immediate transitions are enabled, they fire based on a probability distribution function [3].

Although GSPNs exploit all the SPN's characteristics, the smaller reachability set of a GSPN reduces the confusion of performance analysis much more than SPNs do [3].

There is an additional arc called an *inhibitor* that can be used by GSPNs alongside the places, timed/immediate transitions and directed arcs. This inhibitor arc enables a transition to fire in such a way that is opposite to that of the normal arc, while the input place that is connected to the inhibitor does not contain tokens. This additional arc offers a more flexible description of the system graph and also decreases its size [3].

The SPN is used to model the release order of defensive algorithms, as shown in Chapter 4. In the next subsection, software tools that are used to build and solve an SPN model are reviewed.

2.6.4 Software Tools for Building and Solving SPN Models

Numerous software tools have been introduced in order to assist in accomplishing all the steps of modelling, starting from building the model and ending with solving the model. As a part of this thesis focuses on the stochastic mode, and SPN specifically, the software tools that will be reviewed in this section relate to the tools that solve SPN models. Nevertheless, the largest proportion of this section is devoted to the first tool under review, because it is the only used tool for implementation in Chapter 4.

SPNP

The *Stochastic Petri Net Package* (SPNP) is a modelling tool that is utilised for analysing the performance, dependability and performability of a system model. Furthermore, SPNP is exploited for building and solving *Stochastic Petri Net* (SPN) Reward Models, specifically *Stochastic Reward Net* (SRN) with the foundation of *Markov Reward Models* (MRM) [38].

An SPNP allows a number of results to be obtained, for instance transient, steady-state, cumulative, and time-averaged measures using an analytic model or discrete simulation. The reward rates can be defined by the SPNP at the net level. Furthermore, although non-Markovian SPN models can be also defined using an SPNP, and can be solved by using discrete event simulation.

The SPNP has a textual and a graphical input method. In the textual input method, the *CSPL* is applied, which is a subset of the C programming language with additional constructs for defining the model parameters [38]. In terms of the graphical input, the *iSPN* interface has a set of graphical user interfaces (GUIs) to aid creating and solving the model. Several of these GUIs are explained as follows.

- *Petri Net editor*: this assists construction of the SRN model graphically.
- *Function definition GUI*: this assists creation of the reward, guard, distribution, arc cardinality, and probability functions.
- *Environment GUI*: this assists options for setting up the environment. For example, the solver type (i.e. numerical or simulation) and, from the same GUI, the analysis option (i.e. steady state or transient) can be specified.
- *Animation GUI*: this assists visualising how the tokens move from one place to another in the model.
- *Analysis frame*: this assists in defining the time used to solve the reward variables. In addition to this, from the same frame, the model can be run and then the results represented.

Since the following tools are not utilised in this thesis, only a brief description for each tool will be given.

Möbius

The Möbius is used in a wide range of discrete state computing systems for performance modelling. As such, it is a framework that involves both multiple formalisms, such as SAN, and multiple solution approaches, such as simulation. Several of these approaches are independent of the modelling formalism being used. Thus, these may be utilized in combination with each other [45].

In addition, by using multiple modelling formalisms, the Möbius tool allows a single model to be built. When a model has been built, it is converted into a model which is specified by using Möbius framework components. In addition, communication between different parts can use an Abstract Functional Interface (AFI). This AFI is a group of C++ functions which enable interaction between different models and solvers [43].

PIPE

The *Platform Independent Petri net Editor* (PIPE) is an open source Petri net modelling tool. It allows users to identify queries on the modelled system and solve them. Once the model is created and the performance query of interest is identified using the PIPE front-end user interface, they are both converted into XML files and then sent to the Analysis Server for assessment. Note that because a single query can include a number of sub-queries that need to be evaluated before the main query is assessed, the query is decomposed into its sub-queries based on their dependencies. The analysing server manages several distributed analysing tools such as SMARTA and MOMA. That is, it allocates each derived query to an appropriate analysing tool, after transforming the XML files into an input type that is suitable for that tool [46].

GreatSPN

The Graphical Editor and Analyser for Timed and Stochastic Petri Nets (GreatSPN) is a software tool for creating, validating and analysing the system model. There are two main techniques that can be used to build a model: a Generalised Stochastic Petri Net (GSPN) (and its extensions) and a Stochastic Well-formed Net (SWN). Moreover, as GreatSPN has no common rate and impulse reward definition, it can define performance results that have limited expressive power [115].

2.7 Summary

This chapter has provided background information regarding two important aspects related to the contribution accomplished by this thesis. Firstly, this chapter presented the importance of defensive mechanisms in the Internet against attackers who attempt to abuse the services. The discussion included essential definitions of both an attacker and a defender. This was followed by interactive defensive mechanism types, which revealed a number of security algorithm examples that are expected to be broken over time. The discussion also included the fundamental idea behind these algorithms, and the developing circle life of an algorithm being released and eventually broken. The learning curve theory in psychology, and cognitive and economic fields that apply to the attacker learning process were highlighted, and a rationale for this choice was also included. Furthermore, this was followed by more discussion on several relevant theories related to the learning acquisition process. The discussion included relevant existing work on the effectiveness of information order in the learning process.

Secondly, this chapter examined several related works in the literature to support the proposed methodology which will be evaluated and discussed thoroughly in the following chapter including a review of quantitative security by modelling attacks, modelling attacker behaviour and estimating the time to compromise a system components; a game theory framework to identify the best attack and defence strategies; and an exploration of anomaly detection techniques in which unsupervised machine algorithms were applied as a first layer in hybrid approaches in order to detect an attack attempt.

Additionally, the discussion in this chapter also elaborated on the background of stochastic modelling formalisms; in particular, the focus was directed towards the modelling of the release order of defence algorithms, which are used to evaluate an attacker's performance. The continuous-time Markov Decision Process was selected as a tool to develop the optimisation algorithm, while stochastic Petri nets and their tools were selected as an instrument to represent the interaction between defence algorithms and an attacker.

Chapter 3. EXPERIMENTAL STUDY

This chapter reports a controlled-laboratory experiment that has been carried out as a method to evaluate the proposed idea. In this experiment, investigating whether the order of the release algorithms will matter or not, the influence of the release order on independent algorithms and the importance of the strategies used in the knowledge acquisition process are evaluated.

The effect of the presentation order on the learning mechanism is not new. That is, previous research in the field of education and psychology provide several insights into the effect of presentation order [47, 48, 91, 98, 131]. However, to the best of our knowledge, we are the first to address this particular issue of the release order strategy in the information security concept. There exists a considerable amount of related work that considers the attack and defence interaction as a game-theoretic problem such as in [67, 88, 134], but these formulations do not fit exactly with our approach. Chapter 2 discusses this further.

Since the content-based type of spam filters¹³ have been found to offer a very good model for our experimental requirements, it is important to point out that it is not the objective of the experiment to say anything definitive about spam filter algorithms. Thus, a bespoke spam filter was designed that would fit the purposes of the experiment, although this has created an intuitively appealing experiment that has various elements in common with traditional spam filters. The designed spam filter includes several algorithms that act as a defensive mechanism against attackers.

¹³ This is an electronic mail (e-mail) service feature that is designed to block unwanted e-mail messages sent by unethical senders. This feature has different types such as Bayesian filter, Blacklist-White-list and Content-Based filter, which is the most common type [41]. The developed content-based filter will be discussed in Sections 3.3.2 and 3.3.4. Moreover, the content-based filter is highlighted in Chapter 2 (Section 2.2.3).

Furthermore, an automated program can be used to break the algorithms of the designed spam filter. However, a form of understanding of a human learning process can be seen clearly by sending e-mails to evade a spam filter, as automated approaches are abstractions of this human learning process that require encoding by humans. Any automated approach would need to know the parameters to try, and the range within which these parameters may fall.

The remainder of this chapter is structured as follows. The experiment scope is presented in Section 3.1. Section 3.2 defines the main hypothesis under test. Section 3.3 outlines the experiment setup. The experiment procedure is presented in Section 3.4. Section 3.5 reports the results. Section 3.6 provides the discussion. Finally, Section 3.7 summarizes this chapter. Moreover, an early version of this experimental study was published in [11].

3.1 Experiment Scope

This experiment covers the evaluation of a set of algorithms approach, investigating specifically the importance of a set of algorithms from the release order perspective, the significance of breaking up a secure algorithm into a set of algorithms, the broadening of understanding of the knowledge acquisition process in the information security, and the impact of the strategies used in breaking an algorithm in the knowledge acquisition process.

3.2 Hypotheses

A controlled laboratory experiment study was carried out in order to investigate the research questions of this thesis. This study focuses on the impact of the release order of defensive algorithms on the time needed to break them. Therefore, the main hypothesis under test is:

Hypothesis H₁: *The time it takes to break a series of algorithms is dependent on the order in which the algorithms are released.*

Since the rationale included in improving the interactive defensive mechanism approach was framed using a set of algorithms, it is interesting to see whether there is any significant difference in the release order of dependent algorithms on further independent al-

gorithms. Hence, a hypothesis¹⁴, which is a rather subtle variant of the previous one, is that:

Hypothesis H₂: The time taken to defeat a *future* algorithm does not depend on the order in which *earlier* algorithms were broken.

The next section explains the experimental setup of this study.

3.3 Experiment Setup

The experiment involves subjects acting as potential attackers, carrying out attacks on a test system, within which a number of different defence algorithms have been deployed. The experimental design, the system, attackers, algorithms, material, variables, measurement units, generalisation and threat validity, avoiding bias and control measures are discussed in the following subsections.

3.3.1 The Experimental Design

The experiment used the between-subject design, in which each participant is exposed to only one of the experimental conditions. That is, two sets of subjects break a series of defence algorithms where the order is different between the groups. This type of design ensures that the exact same algorithms are used in each experiment condition, and that there is no unnecessary confounding factor biasing the results (at the cost of recruiting relatively many participants).

The participants are randomly assigned to one of the following two experimental groups:

- **Group 1 (G1):** The order of algorithms for this group was: Algorithm 1 (A1), Algorithm 2 (A2) then Algorithm 3 (A3).
- **Group 2 (G2):** The order of algorithms for this group was: Algorithm 2 (A2), Algorithm 1 (A1) then Algorithm 3 (A3).

Specifics about the algorithms will be given in Section 3.3.4.

¹⁴ There is an additional reason to consider this particular hypothesis, namely whether the use of a Markov decision model with a state space that simply keeps track of which algorithms are broken can be justified. The hypothesis corresponds to demonstrating the memoryless property of the Markov model with that state space. The Markov decision model itself is outside the scope of this chapter, and more details will be given in Chapter 5.

In order to gather more information to improve the design of the experiment, a pilot study was conducted. Four PhD students and Four MSc students from the school of computing science at Newcastle University participated in this study. The pilot study emulated a similar environment, conditions and measures of the planned experiment and its running scenario, and gathered related information in the shape of errors, problems, comments, observations, suggestions, required time and task implementation flow. Based on this information, modifications and improvements were made to the experiment design and the developed algorithms. The modifications included changing the thresholds of similarity that are utilised in the defence algorithms developed in order to manage the difficulty and adjust the system to be more usable for the participants. Other modifications were related to the arrangement of the experiment instructions. The pilot study also helped in estimating the required time for each subject to perform the experiment; thus, 30-40 minutes was found to be suitable for the participants. The required improvements to the experiment material and measurement were made and then the real experiment was started.

It was made clear to the subjects that the data collected in the experiment are strictly confidential to the experimenter and his supervisor. They are only used for research purposes and not for any other intention. The subjects' contact details were only used for announcing the winners (i.e. the first and the second winner).

3.3.2 System

A challenge in designing the experiment was to design a system that could be breached by ordinary people in a matter of minutes. It was found that a content-based spam filter could offer a very good model for the experiment requirements. Although as mentioned there is no attempt to study and derive results for spam filters themselves, it is believed that the simple spam filters have enough similarities with reality to act as an example of the class of systems introduced in Chapter 1 (Figure 1.1).

Thus, a web-based system on which to perform the experiment was developed. A Web application was also developed, which enabled each participant to perform a registration process (e.g. choosing a username, password and educational background), sign a consent form, and read a brief introductory page with necessary information (e.g. description of the experiment, experiment factors, the participant goal, and applied method on how to defeat a content-based spam-filter). The participants could then begin the ex-

perimental process, interacting with the spam-filter algorithms. The main idea was that the participants tried to send e-mails that pass through the spam filters, as described in more detail in Section 3.4. The system recorded all attempts and the time taken by each participant to break each algorithm in each session. The interfaces of the web application developed are included in Appendix A.

The settings of this developed system are flexible for further investigation. For instance, it can be possible to add more algorithms with varying robustness. Moreover, the similarity threshold of each algorithm and the number of attackers for each group can be increased/decreased. Also, it can be possible to change the e-mail text.

3.3.3 Attackers

A nontrivial problem was to find potential attackers. The aim was to find attackers that could be considered to be non-specialists. Whilst specialist attackers or security experts could have been recruited, they would have provided information mostly about where and how particular algorithms needed to be improved and less about learning. Forty students were recruited for this experiment (34 male and 6 female, something that was not considered relevant for this experiment). The typical age range of subjects was 24-33 with 4 participants in the group 40+. The subjects of this experiment were 40 Master's and PhD students from the School of Computing Science and other schools in Newcastle University. The subjects were recruited by email; emails were sent to all MSc students and PhD students at the school and this returned a positive response from 40 subjects.

37 subjects have technical backgrounds (majoring in computer science and engineering), and the remaining 3 subjects non-technical (in linguistics). It is important to note that because our aim was to observe the learning acquisition, the learning process is accomplished despite the backgrounds, as stated in [72].

Each participant was offered £5 for participation. To motivate participants to do their best, like real attackers, an additional incentive to increase their motivation was offered. The participant who got the highest score in each group was awarded £40 while the second ranked subject was awarded £20. The highest score is based on the time and number of trials to complete the task.

3.3.4 Algorithms

The rationale behind the defensive spam-filter algorithms constructed is as follows. A simple algorithm A1 acts as a base algorithm and a more complicated algorithm A2 extends the rules used by A1. In other words, the rules in A1 are a subset of A2, which is the first case to consider when one wants to test a hypothesis. The third algorithm A3 does not share any rule with A1 or A2, and acts as a simple independent algorithm. Hence, the first hypothesis H_1 is challenged by means of the following question:

Q₁ – Do G1 and G2 break A1 and A2 within a similar amount of time?

Furthermore, the second hypothesis H_2 is challenged by means of the following question:

Q₂ – Do G1 and G2 break A3 within a similar amount of time?

A negative answer will be provided in Section 3.5 to Q₁ along with a positive answer to Q₂. The specific algorithms A1, A2 and A3 will now be described, as well as pseudo code describing their operation. Modern density-based spam filters [152] form the basis for implementing the algorithms. These spam filters detect a spam e-mail message by utilising *an unsupervised learning engine* that plays an increasingly important role in this type of filter.

Note that some of the symbol names being used in the pseudo code of the algorithms are given in Table 3.1. Briefly, the symbols in Table 3.1 are explained as follows. As in [152], a hash-based vector representation is used. That is, for each e-mail, hash values of each length 9 substring extracted from the e-mail are calculated¹⁵, and then the first N of them is used as a vector representation of the e-mail. To check a single e-mail, in order to find similar previous e-mails which share S% of the same hash values, the algorithm checks the database. As a result, an e-mail which is transferred more than D times is marked as spam. Therefore, the algorithm does evaluate whether an e-mail message is spam or not based on the similarity with previously submitted e-mail. This similarity will be exploited later in Chapter 6 for constructing the Attacker Learning Curve notion.

¹⁵ The standard hash function provided in the Java library is used.

Algorithm 1 (A1). This algorithm is a simple implementation of the proposal of [152] where only the first part of the message is checked for similar hashes. Furthermore, the similarity threshold is 75. The pseudo code of this is shown in Figure 3.1.

Table 3.1: Symbols used in pseudo code and their values in the experiments.

Symbol	Meaning	Value
D	Spam threshold	100
N	Number of hash values for each email	100
S ₁	Similarity threshold “Algorithm 1&3”	75%
S ₂	Similarity threshold “Algorithm 2”	65%

```

Input:  T: Text of Mail
        Var h: Hash value

Output: R: result of detection
New-Hash-DB-Candidate ← Make N Hash values from T
For h in New-Hash-DB-Candidate do
  For each first 25 hash in New-Hash-DB-Candidate do
    If hi in H1 is similar to hj in H2
      Then increment similarity, increment j and i=j
      Else increment j
      If H1 and H2 share S1 same hash value
        Then R= detected;
        Update-Similar-Mail (Mail in Hash-DB pointed by h)
        If No. of Similar Mail > D
          Then Mark Hash-DB as “spam”
      Else R= no similarity
      // If No Similar Entry exists in Hash DB
      Store-New-Mail (New-Hash-DB-Candidate)
Return R;

```

Figure 3.1: Pseudo code of A1.

Algorithm 2 (A2). This algorithm is similar to A1 except that, before any calculation of the hash values, the message would go through word transformation that would delete all redundant letters and white spaces, and would unify letter case, and transform common number shortcuts to their equivalent letters (e.g. 4 would become for). Those transformations would create a harder algorithm since it would detect any attempt of the attacker to trick the spam filter by using such word transformations. Furthermore, the similarity threshold is 65. In other words, this algorithm has more rules to increase the robustness level. The pseudo code of this is shown in Figure 3.2.

Algorithm 3 (A3). This algorithm is a simple implementation of the proposal of [152]; however, the last part of the message is checked for similar hashes, and the similarity threshold is 75. The pseudo code of this is shown in Figure 3.3.

It is worthwhile to note that the reason behind using a different threshold is that it was empirically found that the similarity threshold can play an important role in terms of determining the difficulty of an algorithm. That is, in A1 and A3, the attacker needs to modify only 25% of the part that the algorithm is checking, while it is 35% in A2.

```

Input:  T: Text of Mail
        Var h: Hash value

Output: R: result of detection
// Remove all white spaces; make the whole text lowercase
T' = Normalise (T)
//Remove triple letters; convert some numbers to letters (like
4 to for)
New-Hash-DB-Candidate ← Make N Hash values from T
For h in New-Hash-DB-Candidate do
  For each first 25 hash in New-Hash-DB-Candidate do
    If  $h_i$  in H1 is similar to  $h_j$  in H2
      Then increment similarity, increment j and  $i=j$ 
      Else increment j
    If H1 and H2 share  $S_2$  same hash value
      Then R= detected;
      Update-Similar-Mail (Mail in Hash-DB pointed by h)
      If No. of Similar Mail > D
        Then Mark Hash-DB as "spam"
        Else R= no similarity
// If No Similar Entry exists in Hash DB
Store-New-Mail (New-Hash-DB-Candidate)
Return R;

```

Figure 3.2: Pseudo code of A2.

```

Input:  T: Text of Mail
        Var h: Hash value

Output: R: result of detection
For read T until the last part
New-Hash-DB-Candidate ← Make N Hash values from the last part of T
For h in New-Hash-DB-Candidate do
  For each first 25 hash in New-Hash-DB-Candidate do
    If  $h_i$  in H1 is similar to  $h_j$  in H2
      Then increment similarity, increment j and  $i=j$ 
      Else increment j
    If H1 and H2 share  $S_1$  same hash value
      Then R= detected;
      Update-Similar-Mail (Mail in Hash-DB pointed by h)
      If No. of Similar Mail > D
        Then Mark Hash-DB as "spam"
        Else R= no similarity
// If No Similar Entry exists in Hash DB
Store-New-Mail (New-Hash-DB-Candidate)
Return R;

```

Figure 3.3: Pseudo code of A3.

3.3.5 Materials: stimulus and rationale

The stimulus material provided to participants consisted of some default e-mail text. The subjects were asked to send this text to the server, as if it was a typical e-mail. The e-mail text was chosen to be 512 characters in length. Although real-life spammers may send messages that are shorter than this, the length of messages provides the subjects with sufficient text to utilize a range of different strategies to breach the spam-filter.

The same e-mail text was assigned to all subjects, rather than allowing each subject to write his own e-mail. There were several reasons for this. First, self-written e-mails may be of different lengths, making the measurement and comparison of participant's learning a difficult task. Second, self-written e-mails might be chosen because they are easy to type (or, in perverse cases, particularly hard to type). This would again introduce biases that are difficult to control. Third, the use of the same e-mail template across all subjects means that each subject can be treated as an impostor for all the other subjects, putting testing on a firm foundation. Finally, using the same e-mail for everyone affected experimental control over unanticipated biases.

3.3.6 Variables

In this experiment, the main independent variable is the algorithm order. The time consumed to break each algorithm and the numbers of trials are the dependent variables.

3.3.7 Measurement Units

Table 3.2 defines the measurement units (e.g. time taken, effort), which are used in this experiment.

Table 3.2: The Experiment Measurement Units.

The Unit	Definition	How is it measured?
Time taken	The duration of time that is spent in order to break a specific algorithm.	Difference between start time and end time of breaking the algorithm.
Effort	The exertion necessary to break a specific algorithm.	Number of trials and the manipulation accomplished on the email text to break the algorithm.

3.3.8 Generalisation and Threats Validity

Generalisation

Since the content-based spam-filter was chosen as the form to evaluate the proposed approach for this experiment, other interactive defensive mechanisms such as CAPTCHA can be applied. However, it would be costly (time consuming and impossible) to cover all interactive defensive mechanisms in this thesis. Thus, the objective of this research is to show the validity of the claim that release order matters. That in itself is challenging. Showing it holds for another interactive defensive mechanism is an additional difficulty, beyond the scope of this research.

Validation

The experiment validity is an important issue in ensuring the quality and generalisability of findings. Two types of validity are involved in this experiment: internal validity, which is concerned with how the study supports the findings; and external validity, which is concerned with generalisability of the results. The threats to internal and external validity are addressed and taken into account as follows:

Internal Validity

For internal validation, the following factors are taken into account:

Selection: The subjects were randomly assigned to the control and experimental groups.

History: Most of the subjects were selected from the same place; therefore, in order to reduce any influential effects, they were recruited and contacted individually and performed the tasks individually at different times.

Motivation: Since the subjects were volunteers and the tasks did not take a long time, there was little concern about boredom or loss of enthusiasm during the experiment.

Time: The time required for the experiment was estimated after conducting a pilot study and the subjects were informed of this when they were recruited. During the experiment, they were told to take enough time to perform the task and that they could stop if they were not willing to continue.

Training: A brief description was given to all of the subjects and the necessary clarification and training was provided before starting the task. Moreover, the subjects were told that they had the right to ask any questions.

External Validation

A number of measures were taken in order that the sample reflected the rationale for the learning acquisition process. Although the selected sample was mostly comprised of subjects with a technical background, some had no technical background as the aim was to observe learning acquisition, not to evaluate the algorithms themselves. Thus, this does not threaten the validity of the research.

3.3.9 Avoiding Bias and Control Measures

Experiments are very sensitive to errors. Many errors could arise due to bias in the experiment. The following measures were taken to avoid and reduce any bias in this experiment:

- The subjects were randomly divided into Group 1 and Group 2.
- Since the aim was to recruit non-specialist attackers in order to observe learning acquisition, the inconsistency-bias produced by attackers' backgrounds is reduced. In addition, as stated in [47, 72] the learning process is achieved regardless of the backgrounds of subjects.
- Self-written emails were not allowed because these may have been of different lengths, making the measurement and comparison of participants' learning a difficult task. Therefore, a default email text was prepared for all participants.
- The maximum number of changes (i.e., manipulations) they were allowed to introduce at each trial was 80. This made it impossible for participants to write a completely different e-mail message.
- The copy and paste functions were not activated to avoid sending completely different e-mails.
- The system recorded time consumption individually in order to avoid the subjects affecting each other or any other meaning, provide the same level of support, observe subjects' progress during the task implementation, and give the same amount of time and support.
- On all the data documents, only the subjects' reference numbers were used, rather than the names. This anonymity made the data analysis more reliable and maintained the subjects' privacy.

3.4 Experiment Procedure

In this section, the way the experiment was run is explained, i.e., instructions to subjects, procedures and the data collected.

3.4.1 Instructions to subjects

Subjects were instructed to act as attackers whose goal was to defeat the spam-filter algorithms by successfully passing through the spam filter algorithms with 3 e-mails (where each e-mail is interpreted as a batch of 100¹⁶). The subjects were instructed that, to defeat an algorithm, they should introduce enough changes to the message template provided to trick the spam filter into thinking that the message being sent was genuine.

Subjects were also informed that there are a number of candidate attacks that spammers can enact to fool spam filter algorithms. For example [51]: *Random addition*, *Thesaurus substitution*, *Perceptive substitution* and *Aimed addition*. The aim of this was creating a level playing field for all participants.

Subjects were told that if they needed a break, they were to do so *after* they had defeated all the algorithms. Subjects were able to gauge their progress by looking at a counter at the right of the screen which showed how many e-mails had been sent successfully so far and how many yet remained. Subjects were advised to focus on the task and to avoid distractions, such as talking with the experimenter, while the task was in progress.

3.4.2 Procedures

The experiment was begun with a brief description about the experiment aim, phases and the assigned tasks to make the subjects ready for the task implementation. Since a web application was developed on which to perform the experiment, each participant was able to firstly perform a registration process (e.g. choosing a username, password, educational background, age, and gender). Then, each participant was asked by the system to read and sign the consent form and the participant was informed of the right to stop at any time. After this, each participant was asked to read a brief introductory page

¹⁶ In the original algorithm [152], when an e-mail message transferred more than 100 times, then it is marked as spam. Therefore, due to the time limitation of the experiment, each sent e-mail message is interpreted as a batch of 100.

with necessary information (e.g. description of the experiment, experiment factors, the participant goal, applied method on how to defeat a content-based spam-filter). The participant could then begin the experimental process, interacting with the spam-filter algorithms. The main idea was for the participants to try to send e-mails that would pass through the spam filters. In particular, after every e-mail message sent by the participant, the system gave information about the progress made: whether the spam attempt passed or failed and, once the algorithms were considered defeated, a notice that the deployed algorithm of the system had been changed. Note that it was necessary to defeat each algorithm twice by an attacker in order to confirm the learning process, and more details regarding its usefulness will be given later in Chapter 6. At the end of the experiment, each participant was informed about the score achieved, the time taken and the number of trials. Finally, the participant was asked to fill in a short survey/questionnaire about his or her experience.

3.4.3 Collected Data

The time taken by each participant to defeat the algorithms in each session was recorded by the system. Furthermore, the number of trials and the e-mails sent for each session were recorded for later analysis. Thereafter, the questionnaires were collected.

3.5 Results and Analysis

In the experimental study, all the participants successfully completed their tasks. The following sub-sections will discuss the hypothesis with respect to the ordering of the algorithms A1 and A2, the hypothesis regarding the insensitivity of the order of A1 and A2 with respect to the time used to defeat A3, and the impact of order on defeating all algorithms.

3.5.1 Testing Hypothesis: Does Order Matter?

The average time needed to break each algorithm in the two groups is shown in Figure 3.4. From the totals (the rightmost bar), it can be seen that Group 1 took longer than Group 2. This indicates there are implications of the ordering of the algorithm, which will be discussed in more detail below. As expected, the ‘tougher’ algorithm A2 took more time to break than A1. In Group 2, it took on average 16.2 minutes and in Group 1, 14.1 minutes. The time needed to break A1 was far less in Group 2, possibly because

learning the techniques to break A2 first, which is effectively a superset of A1, is enough to break A1. The statistical significance of this will now be discussed.

Table 3.3 compares the algorithms A1 and A2 in the two groups, with respect to time needed (middle column) as well as trials made (rightmost column). For both, average (Avg.), standard deviation (SD) and maximum (Max) and minimum (Min) values are provided. With respect to average time, it was found that the time needed for breaking A1 and A2 in Group 1 was 25.0 minutes, while it was 20.1 minutes in Group 2. A t-test yields a result of $t=1.89$, $p<0.1$, indicating that the difference between Group 1 and Group 2 is indeed statistically significant.

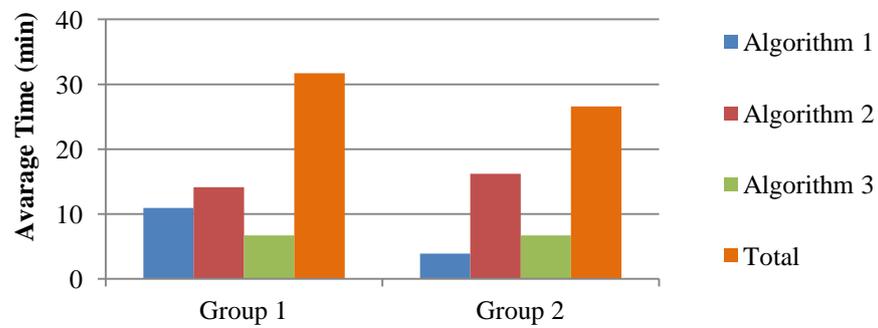


Figure 3.4: The average time (in minutes) for ‘attackers’ to break the algorithms.

Table 3.3: Order of two algorithms A1 and A2.

Group	Total time				Total trials			
	Avg.	SD	Max	Min	Avg.	SD	Max	Min
1	25.0	10.6	57.1	13.5	33.4	20.1	99.0	14.0
2	20.1	4.3	26.0	10.8	26.1	8.3	40.0	11.0

Therefore, the answer to Q_1 can be seen as negative, by observing that G2 broke A1+A2 in significantly less time than G1. This result validates the hypothesis H_1 , since it was the case that the order of release had an impact on the time required to break an algorithm.

With respect to the number of trials, there was found to be a less significant difference. The average number of trials was 33.4 for Group 1 and 26.1 for Group 2. However, this difference is not statistically significant ($t=1.55$, $p=0.139$). The discrepancy between

time and trials is interesting; it does not invalidate the claim that order matters but shows that it is not always apparent (and, of course, as always, possibly not true). This is discussed more at the end of this section. Additionally, Chapter 6 highlights the effect of the applied strategy on the knowledge gained, where a strategy with few trials increases the knowledge gained more effectively than others with many trials.

Looking then at A1 and A2 individually, it is evident that the difference in the total time/trials can be attributed particularly to the time/number of trials it takes to break A1. Comparing the time and trials to break A1 in the two groups in Table 3.4, a t-test yields a result of $t=6.33$, $p<0.001$, indicating that the time consumed to break A1 in Group 1 is significantly higher than that in Group 2. Also, a statistically significant difference is found in the number of trials ($t=6.62$, $p<0.005$).

Table 3.4: Breaking A1 for each group.

Group	Total time A1				Total trials A1			
	Avg.	SD	Max	Min	Avg.	SD	Max	Min
1	10.9	4.7	24.8	5.5	14.8	6.8	31	6
2	3.8	1.3	6.5	0.86	4.3	1.87	8	2

Likewise, Table 3.5 compares A2 in the two groups. A t-test yields a result of $t=-1.32$, $p=0.196$, indicating that the time consumed to break A2 in Group 2 is not significantly higher than that in Group 1. The difference found in number of trials was also not found to be a statistically significant ($t=-0.86$, $p=0.399$). In other words, the difference for A2 is less significant than that for A1. This suggests that attackers learn more from A2 than from A1, in terms of how much helps them to speed them up in attacking the other algorithm.

Table 3.5: Breaking A2 for each group.

Group	Total time A2				Total trials A2			
	Avg.	SD	Max	Min	Avg.	SD	Max	Min
1	14.1	6.2	32.3	7.8	18.6	14.7	74	8
2	16.2	3.4	21.3	8.9	21.7	7.2	34	9

It is worthwhile to note that previous research assumed, based on psychology studies, that interacting with a limited set of highly similar exemplars leads to more learning than when the instances are distributed and dissimilar [131]. The order results appeared to confirm this assumption. The average time of breaking A1 in Group 2 was 3.8 minutes, and the maximum time was 6.5 compared to 10.9 minutes and the maximum time was 24.8 minutes in Group 1. In contrast, the average time of breaking A2 in Group 2 was 16.2 minutes, as opposed to 14.1 minutes in Group 1.

3.5.2 The Influence of Order on Defeating Future Algorithms

The negative answer to Q1 validates the hypothesis H_1 ; however it is also necessary to verify that the difference between G1 and G2 indeed comes from the release order, and not from the fact that G2 contains more naturally talented attackers. To research this question, the third algorithm A3 was added at the end of each experiment group. This provided an opportunity to check if the order of the previous algorithms had any effect on the time needed to defeat the subsequent algorithm A3.

A3 is compared in the two groups in Table 3.6. A t-test yields a result of $t=0.14$, $p=0.891$, indicating that there is no statistically significant difference between the times needed to break A3 in Groups 1 and Group 2. Also, no statistically significant difference was found in the number of trials ($t=1.12$, $p=0.273$). Hence, this points to a positive answer to Q2, by observing that G1 and G2 take a similar amount of time to break the independent algorithm A3. This result validates hypothesis H_2 .

Table 3.6: Breaking A3 for each group.

Group	Total time A3				Total trials A3			
	Avg.	SD	Max	Min	Avg.	SD	Max	Min
1	6.70	4.93	17	0.97	8.8	10.3	49	2
2	6.5	4.4	16.3	0.56	6.05	4.4	20	2

One needs to be careful in generalizing this result: it is not claimed here that any independent algorithm would require the same amount of time, regardless of the history of the attackers. It may be that if A3 was more closely related to A1 and A2, the results would be different. In addition, one would expect that aspects which influence the memory of the attacker may matter, such as the absolute time it takes to break algo-

rithms. After all, it is not unlikely that attackers simply forget more of the knowledge gained from earlier attacks if the attack is more distantly in the past. Thus, this experiment is regarded as an initial look at this issue.

3.5.3 The Influence of Order on Defeating All Algorithms

To complete the discussion, it is necessary to revisit the influence of ordering on the time and effort it takes to break all three algorithms.

The effects of all algorithms in the two groups are compared in Table 3.7. A t-test yields a result of $t=1.76$, $p<0.1$, indicating that the time needed to break the series of algorithms in Group 1 is significantly higher than in Group 2. The average number of trials is 42.3 trials in Group 1 compared to 32.1 trials in Group 2, and a t-test yields a result of $t=1.78$, $p<0.1$, indicating that the number of trials in Group 1 is statistically significantly higher than in Group 2. This is somewhat surprising, since the time to break A3 differs little between groups, and it was shown in Section 3.5.1 that, without algorithm A3, the difference in the total numbers of trials of the two groups is not statistically different. This may suggest that, with respect to the number of trials needed, the validity of the hypothesis is at the edge of statistical significance.

Table 3.7: Breaking all algorithms for each group.

Group	Total time				Total trials			
	Avg.	SD	Max	Min	Avg.	SD	Max	Min
1	31.7	10.8	59.4	17.4	42.3	23	112	16
2	26.6	7.36	36.1	12.2	32.1	11	56	13

3.5.4 Attacking Process

Qualitative data were collected, in the form of surveys, to verify that the attacking process was accomplished by structured strategies that are based on the knowledge gained rather than complete randomness. In particular, an analysis was carried out of the strategies that were used to defeat the algorithms, the part of the e-mail that the participants believed that each algorithm was checking, and the algorithm which the participants thought was the toughest to defeat.

Most of the participants, (90% or 36 out of 40), used structured strategies to defeat the algorithms. In particular, 42% (15 out of 36) used Thesaurus substitution, Perceptive substitution and Delete spaces, 33% (12 out of 36) used Random addition, Thesaurus substitution, Perceptive substitution and Add spaces, and 25% (9 out of 36) used Perceptive substitution. A further investigation will be carried out for these strategies in Chapter 6.

In terms of identifying the correct part of the e-mail that each algorithm is checking, the results in Group 1 were 70% (14 out of 20), 80% (16 out of 20) and 75% (15 out of 20) for A1, A2 and A3, respectively. In Group 2, the results were 100% (20 out of 20), 100% (20 out of 20) and 70% (14 out of 20) for A1, A2 and A3, respectively. Furthermore, it was observed that 80% (16 out of 20) in Group 1 found that A2 was the hardest algorithm, whereas 95% (19 out of 20) found that A2 was the hardest algorithm in Group 2.

3.6 Discussion

For simplicity and consistency purposes, the hypotheses are discussed one by one hereafter, in the same order that they were introduced in the previous sections.

Hypothesis H₁: The time it takes to break a series of algorithms is dependent on the order in which the algorithms are released.

This experimental study provides statistically significant evidence that the release order for a set of algorithms can increase the time needed to break a system's security. In particular, as shown in Table 3.3, the time required by Group 1 to break the algorithms was significantly higher than Group 2. Thus, the main objective of this experiment, namely that 'order matters' is established. In other words, the result of the experiment does support hypothesis H₁.

It is important to point out that the value in testing the two conditions (i.e. A1 is deployed before A2, and A2 is deployed before A1) in which the defences are overlapping, were necessary to build the hypothesis on solid ground before conducting further experiments. Even though several studies in the psychology and education fields indicated that learning curves can be considerably increased by interacting with a limited set of highly similar exemplars [e.g. 47, 98, 131], these studies were only focused

on the effectiveness of the presentation order on the categorization models. For this reason, in this experiment, even the trivial assumption (i.e. A2 is deployed before A1) is tested to avoid surprises. Hence, the experimental results achieved from these conditions can lead to a further experiment in which the algorithms' order is, for instance, a subset, independent and superset or other orders.

There was not found to be a statistically significant difference in the number of trials when breaking A1 and A2 in Group 1. A possible explanation for this might be that the strategy used may impact the knowledge gained, regardless of the number of trials. In other words, in a single trial, some strategies may provide more feedback to the attacker than others. Since there were several strategies applied by the attackers in this study based on the results of the survey, it can be assumed that the released algorithm's rules can be disclosed by some strategies faster than other. A further investigation will be undertaken in Chapter 6.

Since A1 is a simplified version of A2, the experiments also indicate that the success of attacks can be delayed by breaking up an algorithm into parts that are released in sequence. It would be unwise to generalize that conclusion too quickly, but it is an interesting insight which implies that the reasoning that, by breaking up an algorithm into subsets the attacker is 'taught' how to attack, is less valid.

Hypothesis H₂: The time taken to defeat a *future* algorithm does not depend on the order in which *earlier* algorithms were broken.

The concatenation of A3 at the end of both Groups 1 and 2 yielded an interesting and important result. It showed that, despite the knowledge gained at any point of the release chain, injecting a non-subset algorithm would force the attacker back to the learning phase. Accordingly, we carried out an investigation in Chapter 4 into such orders, for example, A2, A3 and A1 and these led to more interesting results. Furthermore, it is of note that the insight that breaking A3 takes an equal amount of time for both groups was used as confirmation that a Markov model is an appropriate formalism for the problem at hand, as will be shown in Chapter 5. Thus, the result of the experiment does support hypothesis H₂.

Furthermore, based on the qualitative data, it was found that the participants performed the attacking process by employing strategies which used the skills gained. This result

seems to be consistent with other researchers in the education field which found that using such strategies can help to construct knowledge [62, 97]. Although applying such strategies can assist in increasing the learning acquisition process, it might, on the other hand, facilitate the detection of the attacker. This will be investigated in more detail in Chapter 7.

A recent study by Mathy and Feldman in the field of education stated in [91] that a dissimilarity-based presentation order was statistically less effective in terms of learning. Therefore, as the aim here in this thesis was to impede the learning process of the attackers, there can be similarities between the attitudes expressed by the order of Group 1 in the experiment and that described by Mathy and Feldman. Hence, it could conceivably hypothesise that a dissimilarity-based presentation order might be useful to the release order strategy of defensive mechanisms.

The findings in this chapter support previous research into this quantitative security area which link a time and an attack phase. For instance, Almasizadeh and Azgomi assumed in [5] that a typical attacker needs time to perform the attack phase. They used this assumption as a starting point to develop a model that focuses on evaluating security and allows the analysis of the security of systems capable of detecting and responding to attacks.

This combination of findings may provide some support for the conceptual premise of quantitative measures for determining operational security by proposing prolonging the time of breaking defensive algorithms as long as possible.

3.7 Summary

This chapter has presented the control-laboratory experiment conducted to validate the hypothesis that the order in which defensive algorithms are released impacts the success of the attacker. The rationale behind this hypothesis is based on the observation that attackers increase knowledge by learning from their attempted attacks, and on the intuition that the learning experience of attackers can be influenced by the order in which defensive algorithms are released. The experiment was undertaken at the School of Computing Science at Newcastle University. There were 40 recruited subjects (MSc and PhD students). This chapter has included the experimental design, system, measurements and procedures involved.

Through a between-subjects experiment with simplified but representative spam filter algorithms, it has been possible to show that the order in which defensive algorithms are released indeed influences the length of time attacks take. This is a very encouraging result for this line of research, indicating that the problem merits study. The experiment also provides an indication that breaking up a defensive algorithm can be a beneficial tool in prolonging the overall attack time, but this issue needs to be researched in much more detail before this conclusion can be drawn more widely. Furthermore, the experiment shows that success in breaking future algorithms does not depend on how that total amount of knowledge was gained.

Since the results of this chapter are encouraging, modelling the release order of defensive algorithms is worth to studying. With such a model, it can be possible to estimate the time needed to break a given set of algorithms without conducting a time-consuming experiment. Therefore, the following chapter will discuss and present the Stochastic Petri Net, to model the release order strategy.

Chapter 4. MODELLING OF RELEASE ORDER STRATEGIES

This chapter presents the proposed model of the utilization of several algorithms to act as a defence mechanism. As the attack evolves, the algorithm used by the system is changed to maintain the security property. The aim of changing the algorithm is to maximize the time before the next security breach. This time is based upon the knowledge the attacker gains from previous attempts to breach the system. Although the attacker is subject to limited time, knowledge and rationality, he/she may develop tools that aim to decrease the required time and rationality while still requiring the knowledge to encode. As attackers learn from their attempts, the order of the algorithms impacts on the time taken to break a system (i.e., an attacker needs to defeat a set of algorithms in order to break the system), as verified in Chapter 3.

The model in this chapter is developed based on Stochastic Petri Nets (SPN), which can describe the interaction between an attacker, the set of algorithms used by a system, and the knowledge gained by the attacker with each attack. This framework allows for a theoretical analysis of the release order of a set of algorithms, and for a better estimate of the time required to break a defensive algorithm in various algorithm orders. Based on the empirical results in Chapter 3, the model is parameterized, evaluated and allows us to draw a conclusion in terms of understanding the effectiveness of interleaving independent algorithms (i.e., with disjointed sets of rules) with dependent algorithms (i.e., with overlapping sets of rules), with respect to the time required for an attacker to break the algorithms. An early version of this model was published in [12].

The remainder of this chapter is organized as follows. Section 4.1 defines stochastic modelling formalisms, and illustrates briefly those that are relevant to this chapter. Section 4.2 describes Stochastic Petri Nets, presenting an overview of Stochastic Petri Nets, evaluation tools, model assumptions, and model metrics. In Section 4.3, model design and performance are described. Section 4.4 offers a case study in order to evaluate the performance of the proposed model. The results and analysis are discussed in Section 4.5. The discussion is presented in Section 4.6. Finally, Section 4.7 summarizes this Chapter.

4.1 Stochastic Model

The reason for the occurrence of a great many daily events is so complex that they are best defined as probabilistic, meaning they can be described as stochastic processes. For this reason, these events are modelled using stochastic models which rely on probabilistic theory. A Stochastic Process is a mathematical representation of the system with probabilistic or random characteristics. Within such a model, a stochastic process represents the behaviour of the system over time, given the occurrence of certain events. It models the system behaviour as a function of time which could be continuous or discrete [90].

In order to assess the interaction between the attackers and the defence mechanism, a stochastic model is constructed, which refines and formalizes the abstract model described in Chapter 1 (Figure 1.1). Broadly speaking, a stochastic model can be depicted as a state transition diagram, which describes all relevant operational system states and the possible transitions between these states. To describe time aspects between events, a rate matrix is specified. It is usually assumed that both the next event and the time before this event are random. Hence, the behaviour of the system is a stochastic process. The main advantage of this modelling approach is that it captures dynamic aspects of system behaviour, which is arguably an applicable approach for modelling the security of a system [138]. More details about stochastic modelling formalisms can be found in Chapter 2 (Section 2.6).

To allow the extraction of quantitative and time-related performance results, Stochastic Petri Nets (SPNs) are utilized for the model in this chapter and are explained in the following section.

4.2 Proposed Stochastic Petri Net (SPN) Model

In this section, an overview of SPNs, evaluation tools used in modelling the release order of defensive mechanisms, assumptions of the model, and the metrics of the model are discussed.

4.2.1 SPN: An Overview

Stochastic Petri Nets (SPNs) are built by assigning exponentially distributed random functions¹⁷ to the delays of Petri net transitions. This allows the extraction of quantitative and time-related performance results. The SPN is a technique used to model and analyse the dynamic behaviour of parallel and distributed systems such as business processes. It is a mathematical modelling technique used to describe a probabilistic nature as a function of a parameter that usually has the meaning of time. It has been used as a helpful modelling formalism and analytic tool in many applications. It is used for the performance evaluation of distributed and parallel computer systems and has been proposed for modelling the qualitative and quantitative analysis of current systems [90].

SPNs use timed transitions, and the delays of a transition firing are a random variable with an exponential distribution λ_i . This means that the transition is associated with a random firing delay, of which the probability density functions are negative exponentials with specific rates. In this case, the distribution of a random variable X_i of the firing time of a transition is given by the equation: $FX_i(X) = 1 - e^{-\lambda_i x}$, and the average time of firing the transition T_i is equal to $1/\lambda_i$. This allows the mapping of an SPN system onto continuous-time Markov chains (CTMC) [3] in order to analyse and compute interesting performance measures such as the probability of transition firing, the probability of being in a subset of markings, or the mean number of tokens.

The SPN is defined formally in [90] as a 4-tuple, $SPN = (P, T, F, \lambda)$ where:

1. $P = \{p_1, p_2, \dots, p_m\}$ is a finite set of places.
2. $T = \{t_1, t_2, \dots, t_n\}$ is a finite set of transitions.
3. $F \subseteq (P \times T) \cup (T \times P)$ is a set of flow relations.
4. $\lambda = \{\lambda_1, \lambda_2, \dots, \lambda_n\}$ is a finite delay associated with each transition.

¹⁷ The exponential distribution is very widely used in performance modelling and denotes the probability distribution that describes the time between events that occur continuously and independently at a constant average rate [138].

The relationships between places and transitions are $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$. A place p is called an input place of transition t if, and only if, there exists a direct arc from p to t . In contrast, place p is called an output place of transition t only if there exists a direct arc from t to p . A place p contains zero or more tokens at any time, which are drawn as black dots. The initial distribution of tokens among places is called the initial marking of the Petri net. This marking represents a state in the Petri net. Each transition has a specific firing delay, which specifies the amount of time that must elapse before the transition can fire. The firing rate λ_i is associated with the timed transition t_i of the Petri net.

Since, as mentioned previously, SPNs are a modelling formalism that account for the randomness of event occurrence times, competition for resources, simultaneous progress of independent processes and synchronization of multiple flows make them suitable for representing the security status of a system in terms of attacker behaviour. As an abstract modelling formalism, SPNs do not directly refer to any specific aspect of the security domain, but expect the modeller to provide the meaning of places, tokens and transitions. In the context of the security phenomena that are highlighted in this thesis, the classical interpretation of Petri Net elements is as follows:

- **Places** represent a set of algorithms that are not yet deployed, a deployed algorithm, a broken algorithm, an attacker and an attacker's knowledge.
- **Tokens** inside a place (the marking of the place) model the number of algorithms, the attacker, and the attacker's knowledge gained after breaking an algorithm. Tokens are anonymous entities that do not carry any qualifying information; thus, the security of the system or the algorithms entity they represent changes as they move from one place to another. Tokens are not always graphically depicted, apart from those cases in which they are numerous.
- **Transitions** represent the system status change. There are two fundamental transitions in the proposed model in this chapter: changing the broken algorithm with a new one by the system at rate " μ ", and making the attack by the attacker at rate " λ ". For the former, the rate of the transition μ represents the speed at which the system reacts. For the latter, the rate of the transition λ represents the speed of the attacks. Furthermore, there is an immediate tran-

sition (i.e. not a timed transition) for moving the token from the broken algorithm to both the attacker’s knowledge and the attacker.

- **Arcs** (arrows linking places to transitions and transitions to places) represent the flow of system transformations, from attacker to a new algorithm, from a new algorithm to a broken algorithm, from a broken algorithm to attacker’s knowledge, and from a broken algorithm to attacker.

In order to generate and evaluate the proposed SPN model, an appropriate stochastic analysis tool is chosen. More details about the chosen evaluation tool will be given in the following sub-section.

4.2.2 Evaluation Tools

Given the SPN model, a stochastic analysis tool is required to generate, solve and evaluate this model. Since there are several stochastic analysis tools, such as Stochastic Petri Net Package (SPNP) [38], Möbius [45] and Platform Independent Petri net Editor (PIPE) [46], an SPNP tool was chosen in this study. Moreover, although a further tool might be suitable to be applied to building, solving and evaluating the proposed SPN model in this study, the SPNP software tool was found to be more suitable due to its features such as textual and graphical input methods.

The SPNP tool creates files of a standard structure using a CSPL (C-based SPN language) specific to SPNP. This CSPL is a simple input language that is based on C programming language and which represents the model textually with its state variables, actions, reward variables and solving commands in a single file. For ease of reference, this is here called the SPNP file. This SPNP file should contain six basis functions [38], each of which is designed to carry out one or several tasks by using a number of relevant functions. These six functions, used in the mapping relations, are described as follows:

- ***option()***: This function is used to carry out certain tasks by calling on some relevant function which affects the way in which the Stochastic Reward Net (SRN) is described and solved.
- ***net()***: This function is used to define an SRN. The following functions are called on by this function:

- ***place()*** and ***init()***: The former is used to define a place with a name p ; and the latter defines the initial number of tokens in place p to be n .
 - ***imm()***: This function defines the time transition.
 - ***rateval()*** and ***probval()***: The first of these functions defines the firing rate of the timed transition t and the firing weight or probability of an immediate transition t as a constant value val . The function *probval()* needs to be used only if the value of the firing weight of the immediate transition is different from the default value 1.0.
 - ***priority()***: This defines the priority of transition.
- ***assert()***: This is a Boolean marking function used to check the validity of each newly found marking.
 - ***ac_init()***: This function is used to call a set of functions before starting the construction of the reach ability graph in order to output information about the model to the “.out” file of the *SPNP*.
 - ***ac_reach()***: This function is used to call a set of functions after the construction of the reach ability graph is completed in order to output information about it to the “.out” file of the *SPNP*.
 - ***ac_final()***: This function calls a set of functions designed for the user to flexibly define outputs; for example, a function to solve the Markov chain numerically at time t or for steady state analysis, and a function to output data about the Markov chain and its solution.

In the interests of clarifying and simplifying the application of measurements, schedule and functions of the chosen SPNP tool, the following assumptions have been made.

4.2.3 Model Assumptions

The design of the proposed model is based on a set of assumptions related to the system, attack, and attackers. These assumptions are as follows:

- The distribution times for replacing an algorithm (i.e. when the released algorithm is broken, it is replaced by a new algorithm) and attack attempt to break the algorithm are assumed to be exponentially distributed with the rates of μ and λ , respectively. The assumption of exponential distribution

can be easily relaxed by defining other time distributions and evaluating the model using *SPNP* version 6.0 [39].

- The system enters a security failure state when all algorithms are broken.
- There is only one attack against the system. The reason behind this is to focus on the modelling process of the interactions between the attacker and the system.
- Only scenarios of attacks that eventually lead to breaking all algorithms are considered, and not the scenarios which may be aborted during the attack process. Therefore, in the model, there is not a place that allows the attacker to interrupt an ongoing attack.

To represent key properties of the proposed model, a metric over the proposed model can be defined. This metric is discussed in the following sub-section.

4.2.4 Model Metrics

Since the main concern in this study is the security of the interactive defensive mechanisms, a metric is required that encapsulates this security concern and the relationships between this concern and the time (as the attacker evolves in the proposed SPN model).

Hence, a particularly meaningful metric which is most suitable for this study is the *Mean Time to Security Failure (MTTSF)* [2]. This metric represents the average time elapsed to reach a security failure state. Accordingly, a higher *MTTSF* is desirable. For this approach, a security failure occurs when all algorithms have been broken, and the attacker can misuse the system resources.

4.3 Model Design and Performance

The proposed SPN model is designed and evaluated by using *SPNP* [38] to describe the system behaviour under attack with the objective of assessing the *MTTSF* of the system. Using the graphical representation feature of the *SPNP* tool, as noted previously, it is possible to represent several algorithms, which vary in terms of rules, the order of the released algorithm, an attacker and the knowledge gained by the attacker. This graphical representation of all elements of the model helps in understanding the modelled system.

Therefore, Figure 4.1 presents a graphical illustration of the proposed SPN model, and Table 4.1 identifies the used symbols in the proposed SPN model.

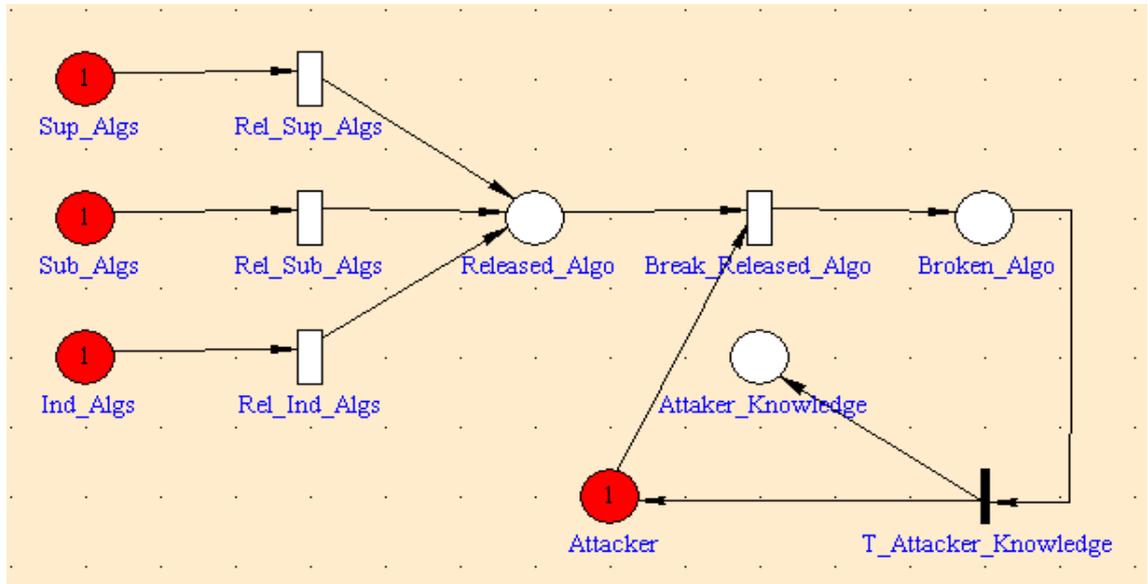


Figure 4.1: A graphical illustration of the proposed SPN model.

Table 4.1: Identifying the Symbols.

Symbol	Meaning
Sup_Algs	Superset algorithm
Sub_Algs	Subset algorithm
Ind_Algs	Independent algorithm
Rel_Sup_Algs	Transition to release a super algorithm
Rel_Sub_Algs	Transition to release a subset algorithm
Rel_Ind_Algs	Transition to release an independent algorithm
Released_Algo	Released algorithm
Break_Released_Algo	Transition to break the released algorithm
Broken_Algo	Broken algorithm
T_Attacker_Knowledge	Transition to increase attacker's knowledge
Attacker_Knowledge	Knowledge gained by the attacker

The explanation of places and transitions utilised in the proposed model is in the following. The places *Sup_Algs*, *Sub_Algs* and *Ind_Algs* represent a set of defence algorithms. Specifically, *Sup_Algs* and *Sub_Algs* represent the overlapping type (i.e. $Sup_Algs \supset Sub_Algs$ as described previously in Chapter 1 (Section 1.1)), while *Ind_Algs* represents a non-overlapping algorithm type (i.e. independent algorithm). This means that A1 is a simplified version of A2, whereas A3 does not share any rule with A1 or A2. In addition, there are three timed transitions: *Rel_Sup_Algs*, *Rel_Sub_Algs*, *Rel_Ind_Algs* and *Break_Released_Algo*. As usual, the transition time is non-zero and follows a probability distribution, e.g., the exponential distribution. The first three transitions are associated with an enabling function in each transition that guards the firing of the transition depending on the *Released_Algo* state. This enabling function avoids releasing a new algorithm while the current algorithm remains unbroken, and can control the releasing order of algorithms. When an algorithm released in *Released_Algo* place is broken, the selected algorithm will be enabled to deploy a new token (i.e. a new algorithm).

The construction of the model takes the system's lifecycle into account. Initially, a set of algorithms is placed in *Sup_Algs*, *Sub_Algs* and *Ind_Algs* places as tokens. Each algorithm has a weight reflecting the patterns that can be exploited when broken. In particular, there are overlapping rules between subset algorithms and superset algorithms, whereas independent algorithms have different rules, reflecting various weights between them and the dependent algorithms. The selection of a released algorithm is modelled by firing one of the transitions *Rel_Sup_Algs*, *Rel_Sub_Algs* or *Rel_Ind_Algs*. Once an algorithm is selected, a token at a time (if it exists) is moved from an associated place to the *Released_Algo* place.

The attacker now attempts to break the deployed algorithm at an initial rate. Once the deployed algorithm is broken, it is moved to the *Broken_Algo* place, which is modelled by the transition of *Break_Released_Algo*, the firing of which will move a token into place *Broken_Algo*. Then, the *T_Attacker_Knowledge* transition immediately translates this token to both the *Attacker_Knowledge* place, which increases the attacker's knowledge, and the *Attacker* place, which in turn will prepare to break the next deployed algorithm. Now, the attacker attempts to break the new deployed algorithm, which is released after breaking the previous algorithm by firing one of the transitions *Rel_Sup_Algs*, *Rel_Sub_Algs* or *Rel_Ind_Algs*. Once the released algorithm is defeated, the next one is released, and so on and so forth.

It is important to note that the firing rate λ of the attacking process is controlled by the *Break_Released_Algo* transition, in order to have an effect on the firing rate (i.e. Attack Rate λ) in defeating the released algorithm through a sequence of trials.

The time taken for an attacker to cause a transition from one state to another depends on two significant factors: the attacker's knowledge and the robustness of the system. Therefore, it is considered in this thesis that the time taken to break an algorithm is calculated by [12]:

$$T_i = (SL * RL_i) \quad 1$$

where T_i is the time taken to break algorithm i , SL is the skill level of the attacker that can be derived by normal distribution intervals suggested in [112] and RL is the robustness level of the algorithm. The attack rate λ is simply calculated by [12]:

$$\lambda = 1/T_i \quad 2$$

Finally, the system is considered as experiencing a security failure if the security failure condition described in Section 4.2.4 is met, which is when all algorithms have been broken. This is modelled by making the system enter an absorbing state when the condition is met. Specifically, the absorbing state is represented in Figure 4.1 when all algorithm tokens are transferred to the *Attacker_Knowledge* place.

As stated earlier, this model has been implemented into an SPNP tool [38], which supports the adopted modelling formalism and allows for graphical model definition and for solution via simulation.

4.3.1 Performance Metrics Calculation

As mentioned previously in Section 4.2.2, a *MTTSF* is chosen in this study in the interests of achieving the security metric. An *MTTSF* can be obtained using the concept of mean time to absorption (*MTTA*) in the *SPN* model. In particular, a reward assignment is used so that a reward of 1 is assigned to all states except absorbing states, which is done with the following reward assignment function: if $\text{mark}(\text{Released_Algo}) = 1$ (i.e. an algorithm is released) then return 1, else return 0. As also stated previously, the absorbing state occurs when all tokens of the algorithms are moved to the *Attacker_Knowledge* place. The *MTTSF* of the system is then simply the expected accumulated reward until absorption $E[Y(\infty)]$, defined as follows:

$$E[Y(\infty)] = \sum_{i \in S} r_i \int_0^{\infty} P_i(t) dx$$

where S denotes the set of all states except the absorbing ones, r_i (reward) is 1 for those states and $P_i(t)$ is the probability of state i at time t .

4.3.2 Further Details for the Used Functions

In the proposed SPN model, several tasks are mapped to various functions defined in the C functions of the CSPL file. These functions are Guard, Distribution and Reward functions. There are three guard functions in order to control the release order of the algorithms. Specifically, in the transition of each type of algorithm (i.e. subset algorithms, superset algorithms and independent algorithms) as shown in Figure 4.1, there is a guard function. Based on the order that is needed for evaluation, the guard function of each transition will be defined. For instance, if the order that will be evaluated is a *superset*, *subset* and *independent* algorithm, then the guard functions for the superset, subset and independent algorithm are shown in Figures 4.2, 4.3, and 4.4, respectively.

```
if (mark("Released_Algo")==1)
return (0);
else
return (1);
```

Figure 4.2: A Guard Function for Superset Algorithm.

```
if (mark("Sup_Algs")==1 || mark("Sup_Algs")==0 &&
mark("Released_Algo")==1 )
return (0);
else if (mark("Sup_Algs")==0 && mark("Released_Algo")==0)
return (1);
```

Figure 4.3: A Guard Function for Subset Algorithm.

```
if (mark("Released_Algo")==1 || mark("Released_Algo")==0 &&
mark("Sub_Algs")==1)
return (0);
else if (mark("Sup_Algs")==0 && mark("Released_Algo")==0 &&
mark("Sub_Algs")==0)
return (1);
```

Figure 4.4: A Guard Function for Independent Algorithm.

As shown in the above figures, the guard function of superset algorithms allows releasing a super algorithm if there is not yet a released algorithm. In order to release a subset algorithm, the guard of this algorithm verifies, if there is no released algorithm whether a super algorithm or other types, then a subset algorithm is allowed to be released; otherwise, it is not. Finally, while there is not a released superset or subset algorithm, an independent algorithm is allowed to be released.

Moreover, in order to control the attack rate λ for each released algorithm, a distribution function is developed in the *Break_Released_Algo* transition. The attack rate of an attacker is evaluated by using the distribution function that is presented in Figure 4.5. That is, while the number of tokens in the *Attaker_Knowledge* place is less than 1 or equal to 0, then the first released algorithm attack rate is returned; if not, then while the *Attaker_Knowledge* place is less than 2 or equal to 1, then the second released algorithm attack rate is returned; otherwise, if the *Attaker_Knowledge* place is less than 3 or equal to 2, then the third released algorithm attack rate is returned.

```

if(mark("Attaker_Knowledge")==0 || mark("Attaker_Knowledge")<1) return  $\lambda$ 
for the first released algorithm;

else if(mark("Attaker_Knowledge")==1 || mark("Attaker_Knowledge")<2) re-
turn  $\lambda$  for the second released algorithm;

else if(mark("Attaker_Knowledge")==2 || mark("Attaker_Knowledge")<3) re-
turn  $\lambda$  for the third released algorithm;

```

Figure 4.5: A Distribution Function for controlling the Attack Rate λ

In respect to reward functions, there are two reward functions in order to analyse both the security and the amount of knowledge that an attacker has achieved during the attacking process. The security of a system is evaluated by using the reward function that is depicted in Figure 4.6. As such, while there is a released defensive algorithm, 1 is returned, indicating that the algorithm is not yet broken. However, in case the released algorithm is broken, 0 is returned. With regards to the amount of knowledge achieved by the attacker, this is analysed by using a reward function that is shown in Figure 4.7. In particular, *Expected reward rate at time t* function, which its experiment parameters are start value, stop value and increment value, is used to analyse the reward amount at time t . This time represents the attacker's trial in this model. Therefore, with each attack

attempt, the reward function returns the amount of knowledge that the attacker has gained. For example, when analysing the reward amount for 10 trials, then the experiment parameters of *Expected reward rate at time t* function are 1 for the start value, 10 for the stop value and 1 for the increment value.

```
if (mark("Released_Algo")==1) {return 1;}  
else {return 0;}
```

Figure 4.6: A Reward Function for Evaluating the Security of a System.

```
return (mark("Attaker_Knowledge"));
```

Figure 4.7: A Reward Function for Evaluating Knowledge Gained by the Attacker.

Given that the developed model is evaluated by means of a case study that is based on the previous experiment presented in Chapter 3, this case study will be discussed in the following section.

4.4 Case Study

This section presents a case study to evaluate the performance of the proposed model discussed in the previous section. The purpose of this evaluation is to investigate the degree to which the proposed model can achieve its objectives. Furthermore, it aims to explore and discuss any areas that need to be enhanced. The chosen case study is a spam-filter system. Since the previous experiment shown in Chapter 3 was conducted using a form of spam-filter system, the results of that experiment are utilised in this case study. More importantly, the model can be parameterised based on empirical results. Therefore, all the following numerical values are the results achieved from that experiment.

The following is a brief explanation of the experiment, which involved subjects acting as potential attackers carrying out attacks on a spam filter system. By using simplified but representative spam filter algorithms, the study was able to show that the order in which defensive algorithms are released influences the length of time attacks take. There were three algorithms A1, A2 and A3 acting as a set of defence mechanisms, such that A1 was a subset of A2, while A3 was independent of both A1 and A2. The attackers were divided into two groups, Group 1 and Group 2; the algorithms in Group

1 were released in the order A1, A2 and A3, while the release order in Group 2 was A2, A1 and A3. Each attacker was asked to break the algorithms sequentially, and the time taken to break each algorithm was recorded, as well as the trials. The average time to break all algorithms for each group is shown previously in Table 3.7. The best order to maximize the time taken to break all algorithms by the attackers was in Group 1 (i.e. A1, A2 and A3).

Since the time taken to break each algorithm is known, the fire rate λ of *Brak_Released_Algo* transition for each algorithm is derived from Equation (2) as shown in Table 4.2. Furthermore, the default fire rate of releasing defence algorithms (i.e. *Rel_Sup_Algs*, *Rel_Sub_Algs* and *Rel_Ind_Algs* transitions) is $\mu_1 = \mu_2 = \mu_3 = 1$, respectively. This means that the system reacts through 1 time unite in order to replace the broken algorithm with another algorithm.

Table 4.2: Fire rate of Break_Released_Algo transition.

Order of the algorithms	Fire rate		
	A1	A2	A3
A1, A2 and A3	0.0917	0.0709	0.1492
A2, A1 and A3	0.2631	0.0617	0.1538

In addition to the aim of the previous experiment – investigating the influence of the release order of defensive algorithms – this case study aims to analyse the interleaving of independent algorithms with dependent ones. Hence, the release process of algorithms is thus: firstly, the *Rel_Sub_Algs* transition is fired (i.e. for releasing A1); once this algorithm is broken, the *Rel_Ind_Algs* transition is fired (i.e. for releasing A3); finally, after breaking this algorithm, the *Rel_Sup_Algs* transition is fired (i.e. for releasing A2). So, once this algorithm is broken, the *MTTSF* is calculated.

More importantly, the fire rate of algorithm A2 is taken from Group 2 (as shown in Table 4.2), since the release order of the algorithm in Group 2 was A2, A1 and A3. Therefore, it is assumed that releasing the independent algorithm among dependent algorithms can impair the learning process of the attackers. Even though A1 was broken and the attacker’s knowledge was increased, breaking A2 after A3 forced the attacker back to the beginning of the learning phase. This additional release order allows a comparison with that of the best order in the previous experiment (i.e. A1, A2 and A3).

This order of interleaving of independent algorithms with dependent ones can be applied also for this order: A2, A3 and A1. In this case study, however, the order namely A1, A3 and A2 is only analysed.

4.5 Results and Analysis

This section presents the numerical data obtained from evaluating the SPN model and discusses the physical meaning. Firstly, the results achieved in the previous experiment are replicated based on the *MTTSF* metric. Moreover, the effect of the algorithm orders on the *MTTSF* is examined for both the first order of the previous experiment (i.e. A1, A2, and A3) and the proposed order in this section (i.e. A1, A3 and A2). Finally, the progress of the attacker’s knowledge acquisition process is shown for both orders.

In the following figures, “E. Order” represents the experiment-based order, while “M. Order” represents the model-based order (i.e. A1, A3 and A2). Moreover, “E. Attacker Knowledge Progress” represents the attacker’s knowledge progress based on the experiment order, while “M. Attacker Knowledge Progress” represents the attacker’s knowledge progress based on the proposed model order.

4.5.1 Replicating the Results of Release Order

The results achieved in the previous chapter are replicated by the proposed SPN model based on the fire rates that are presented in Table 4.2, with the purpose of demonstrating the performance of the proposed model. Figure 4.8 demonstrates the influence of the algorithm order on the time taken to break all algorithms by mode-based generated results.

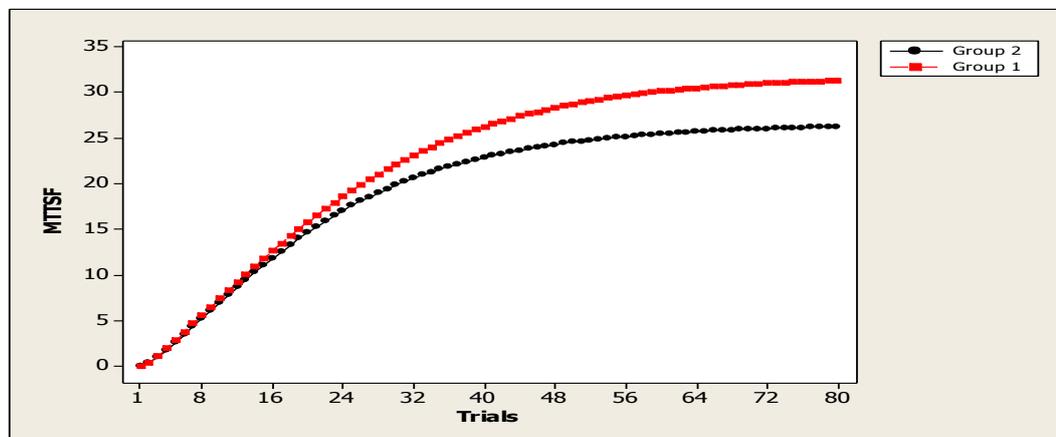


Figure 4.8: Replicating the results of release order

A comparison of the results of the proposed SPN model solution with those obtained via the real experiment reveals that in the proposed model the *MTTSF* for Group 1 is about 31.1 minutes, while it is about 25.8 minutes for Group 2. These results indicate that the time taken to break all algorithms in Group 1 is extended longer than in Group 2, as reported from experimental results in the previous chapter. It is noticeable that the estimation of 80 as the number of trials for the *Expected reward rate at time t* function is found to be suitable empirically.

4.5.2 Algorithms Order vs. *MTTSF*

The implication of the algorithm order, which is the E. Order and M. Order, on the *MTTSF* is shown in Figure 4.9, where it can be observed that the proposed order (i.e. model-based order) achieves a higher *MTTSF* than the experiment order. In particular, the amount of time needed for breaking the proposed order is almost 35 minutes. On the other hand, the time needed for breaking the experiment-based order is approximately 31 minutes.

Although the order of Group 1 (i.e. A1, A2 and A3) in the previous experiment introduces an approach to, for instance, break one algorithm into a number of algorithms while keeping a level of security, the proposed order in this thesis can improve the security of this order, as shown in Figure 4.9. That is, each subset algorithm obtained from the main algorithm can be followed by an independent algorithm. Even though an attacker may observe this release strategy, he needs to return to the learning phase with each independent algorithm.

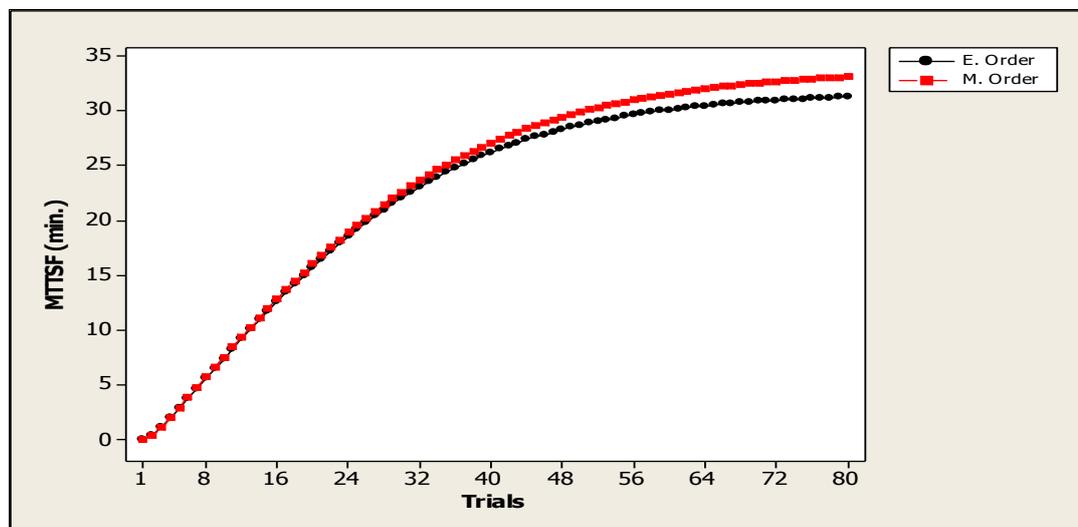


Figure 4.9: *MTTSF* vs. Algorithm Orders.

On the other hand, the order of Group 2 in the experiment (i.e. A2, A1 and A3) might not be improved by the proposed order, as A2 is the superset and A1 the subset algorithm. However, interleaving independent algorithms might lead to increasing the time taken to break the subset algorithm, as in E. Order.

4.5.3 Attacker’s Knowledge Acquisition Process

The estimation of the attacker’s knowledge acquisition progress for breaking all algorithms is shown in Figure 4.10, which is obtained by estimating the time for the token of the broken algorithm to arrive at the *Attacker_Knowledge* place (i.e. the absorbing state). This estimation has been obtained by using the *Expected reward rate at time t* function that uses the reward function that is shown in Figure 4.7 and 80 as the stop value for its experiment parameters.

As shown in Figure 4.10, the process of knowledge acquisition in the proposed order (i.e. M. Attacker Knowledge Progress) seems extended in comparison to the experiment order (i.e. E. Attacker Knowledge Progress). Although the Group 1 order in the previous experiment proved empirically that breaking up an algorithm in subsets does not necessarily ‘teach’ the attacker how to attack, impairing the learning process with an independent algorithm among dependent algorithms can be a strategy to extend the breaking time of a system. Moreover, the knowledge acquisition progress of the attackers with the Group 2 order of the previous experiment might be affected by the proposed order.

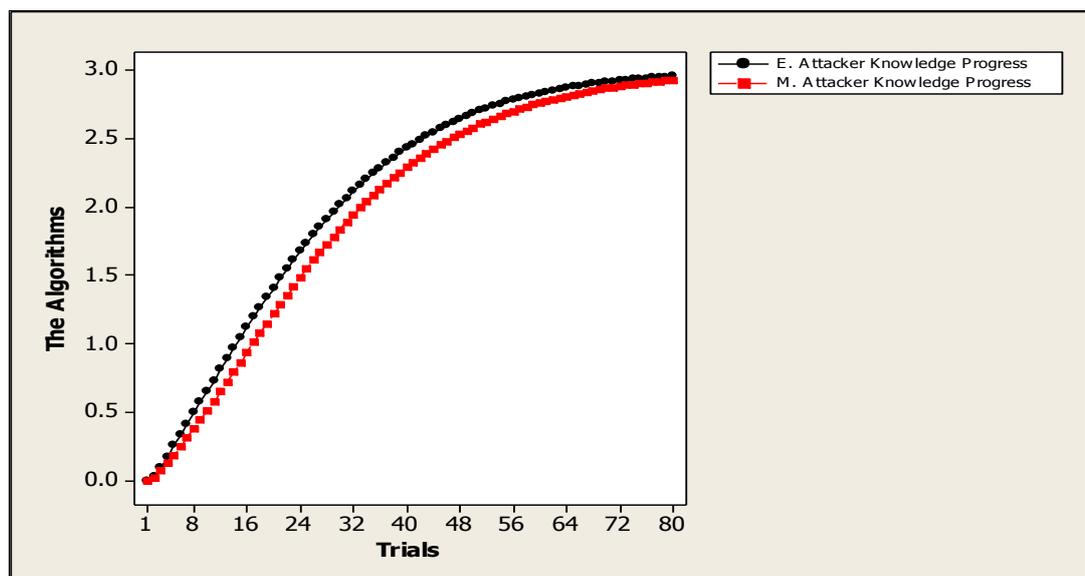


Figure 4.10: Attacker’s Knowledge Acquisition Process.

4.6 Discussion

The present study was designed to model the application of set of algorithms to act as a defensive mechanism. The Stochastic Petri Nets were used for the formalization. Using the developed model, in addition to the proposed release order of algorithms in the previous experiment, the investigation of the effect of algorithms with disjointed rule sets interleaved with algorithms which have overlapping rules is accomplished.

The results of this developed model replicate the time taken to break all algorithms for both groups by using a *MTTSF* metric. This indicates that the metric used is appropriate for the proposed model, which is defined by the reward function illustrated in Figure 4.6. The results obtained by evaluating the proposed model through empirical results are very encouraging.

Another important finding was that such an interleaving of independent algorithms with dependent ones can maximise the time required to break a defence mechanism. In particular, as shown in Figure 4.9, the time that was required to break the system was extended by the model-based order compared to the order of Group 1 in the previous experiment. Although the previous experiment highlighted the interesting insight that the success of attacks can be delayed by breaking up an algorithm in parts that are released in sequence, this insight can be improved by the proposed order in this case study.

In this study, the methodology of presenting the attacker's knowledge acquisition process, as shown in Figure 4.10, yielded an interesting prediction of the importance of the release order of the algorithm. Specifically, the learning progress of the attackers in the Group 1's order in the previous study was affected by the proposed order in this case study. A study in the field of psychology, based on empirical results, stated that when the instances are distributed and dissimilar, this leads to a reduction in the learning process [131]. The results of the proposed order can confirm this assumption. Therefore, a possible explanation for this study's results may be the enforcement of an attacker to return to the learning phase with each independent algorithm. These findings suggest that the proposed order appears promising in terms of extending the length of the learning acquisition process as much as possible.

Nevertheless, the proposed order (i.e. model-based order) might not be recommended to such systems as Group 1's order due to the following facts: 1) Deploying/releasing a

defensive mechanism has a cost. For instance, recently, a study by Caliendo et al. calculated in [33] the cost of deploying a spam filter within a particular organisation, and found that the cost is circa fifteen thousand Euros for the first year. 2) Most companies have limited resources, and therefore a limited number of defensive mechanisms they can deploy over a given period of time. Therefore, in this context, Group 1's order seems economically better than the proposed order, where given a "complex" mechanism, it can generate (for a low cost) many more subsets of this mechanism than can be released according to Group 1's order.

Furthermore, Jonsson and Olovsson suggested a hypothesis in [70] that the attacking process can be split into three phases: the learning phase, the standard attack phase, and the innovative attack phase. The probability for successful attacks during the learning and innovative phases is expected to be small. The proposed order provides further support for this hypothesis in terms of the learning phase, as the learning process is assumed in this study to be impaired with an independent algorithm among dependent algorithms.

It is worthy to note that the interaction between the attacker and the defender in the proposed SPN model appears like a game between two players. For instance, Lye and Wing modelled in [88] the interaction between an attacker and a defender as a two-player stochastic game. Their model could compute the best-response strategies for the players and then use the results by the administrator to enhance the security of a system. However, it typically does not consider maximizing the duration of the game as the objective, but aim for Nash or other equilibria.

4.7 Summary

This chapter has proposed a model of a set of defensive algorithms approach, based on the assumption that a single defence algorithm has a number of drawbacks due to the attacker learning as time progresses. The proposed model is constructed by using Stochastic Petri Nets (SPN), which can describe the interaction between an attacker, the set of algorithms used by a system, and the knowledge gained by an attacker with each attack. The purpose of this framework was to facilitate theoretical analysis of the release order of a set of algorithms approach.

Due to several important features, a Stochastic Petri Net Package (SPNP) is utilised as a software tool to implement the proposed model. Based on the empirical results achieved from the previous experiment conducted in Chapter 3, the proposed model is parameterized and evaluated. The model allows the results obtained previously in Chapter 3 to be replicated. Not only this, but it also demonstrates and predicts the consequences of introducing algorithms that are not a subset of a set of used algorithms, which forces the attacker back to the beginning of the learning phase.

The study has gone some way towards enhancing our understanding of the interleaving of independent algorithms with dependent ones, which was a model-based order. Despite this model's apparent cost efficiency, this study offers some insight into impeding the learning acquisition process of the attacker.

Studying and developing an optimisation algorithm is important. As shown in Chapter 3, a statistically significant impact on the time attackers take to break all algorithms, while breaking A3 took an equal amount of time for both groups. This can be used as confirmation that a Markov model is an appropriate formalism for the problem at hand. Therefore, the following chapter will entail discussion of the use of a Markov model and optimisation of the release order of a set of defensive algorithms.

Chapter 5. OPTIMAL RELEASE ORDER STRATEGIES

This chapter defines an optimal strategy approach that has been accomplished for the release order of defensive algorithms. Specifically, since the order in which spam filter algorithms are released has a statistically significant impact on the time attackers take to break all algorithms, as shown in Chapter 3, this problem is modelled as an optimisation problem¹⁸ using a stochastic model in this chapter. In order to get a model that has, on one hand, enough complexity to capture the complexity of the phenomenon in the problem and, on the other hand, enough structure and simplicity, a continuous-time Markov Decision Process (CTMDP) is found to be a suitable tool for this optimisation problem. This CTMDP allows determining an optimal strategy for the model. The objective of this optimisation algorithm will be the release order of algorithms so as to maximise the time until the attackers break through all algorithms available. For this, when maximising the mean time to break algorithms, well-known iterative algorithms [141] can be applied. Therefore, this chapter provides a version of such an iterative algorithm that exploits the absorbing nature of the underlying Markov chain and avoids generating (and storing) the whole Markov chain. An early version of this optimisation algorithm was published in [13].

The remainder of this chapter is structured as follows. Section 5.1 provides a preliminary outline of the approach to the derivation of optimal release strategies and establishes a design for how the continuous-time Markov Decision Process is applied. Section 5.2 describes the proposed optimisation algorithm. An application example is pro-

¹⁸ Generally, an optimisation problem seeks values of the variables that lead to an optimal value of the function that is to be optimised.

vided in Section 5.3. The discussion is presented in Section 5.4. Section 5.5 summarises the chapter with an overall discussion.

5.1 Deriving Optimal Release Order Strategy

The release order of defensive mechanisms has indeed influenced the time attackers take to break them, as verified in Chapter 3. Particularly, the time required by Group 1 to break the algorithms was significantly higher than Group 2. Moreover, the concatenation of Algorithm 3 (i.e. A3) at the end of both Group 1 and Group 2 yielded an interesting result, namely that the time taken to break A3 is not significant; rather, the significance lies with which algorithms were broken. This result therefore can indicate that injecting a significantly different algorithm forces the attackers back to the learning phase. Whether this can be extended to more similar algorithms remains to be seen.

Thus, optimising the release order of defensive algorithms is a problem worthy of study. This optimisation is a procedure used to make a system as effective as possible in terms of maximising the time taken by attackers to break the system. To study and determine the optimal release strategies, a stochastic model that takes into account the important aspects of the problem is used in this study.

Broadly, a *stochastic model* is a model that involves probability, or randomness, associated with time and events. When using such a model, a stochastic process represents the behaviour of the system over time, given the occurrence of certain events. A stochastic model can be depicted as a state transition diagram, which describes all relevant operational system states and the possible transitions between these states. To describe time aspects between events, a rate matrix is specified. One usually assumes that the event that will occur next in the system, as well as the time before this event, is random. Hence, the behaviour of the system is a stochastic process. The main advantage of this modelling approach is that it captures dynamic aspects of system behaviour, which it can be argued is an applicable approach for modelling the security of a system [138]. Such a stochastic process is known as a *Markov process*.

A *Markov process* is a special type of *stochastic process* in which the conditional probability distribution function satisfies the *Markov property* or the *Memoryless property* of a Markov chain. This *Memoryless property* means that the past history of a random variable that is exponentially distributed plays no role in predicting its future. For in-

stance, when X is the random variable that denotes the length of time that a person spends in a service and X is exponentially distributed, then the probability that the person in service finishes at some future time t is independent of how long that person has already been in service. Accordingly, as observed in Chapter 3, the concatenation of Algorithm 3 at the end of both groups (i.e. Groups 1 and 2) demonstrated the memory-less property of the Markov model.

Therefore, the optimisation problem is modelled in this study as a *Markov Decision Process* [24] with a state space S that simply keeps track of which algorithms are broken. This allows determining an optimal strategy from the model. More details regarding the utilisation of Markov Decision Process are given in the next section.

5.1.1 Modelling the Release Order Strategy

A Markov Decision Process (MDP) is a widely unified framework for modelling and describing sequential decision making problems that arise in engineering, economics, operations research and computer science [24]. MDP is also useful for studying a wide range of optimisation problems. There are four principal components in an MDP model: a state space, an action space, the effects of the actions and the immediate cost incurred by the actions. A decision process is characterized by the fact that in each state there is a choice to be made between possible actions. Each action takes the process to a new state.

Since time is an essential factor in the problem at hand, a continuous-time Markov Decision Process (CTMDP) is introduced. As the released algorithm is broken and knowledge regarding its rules is gained by the attacker, the system responds to this by replacing the defeated algorithm with another one. The system does not make the decision about replacing actions blindly, but takes into account past, current, and possible future states of the attackers and also possible rewards that are connected with the actions. The goal of the system in this study is to maximise the time taken by the attacker to break all algorithms.

Formally, CTMDP is a set $P = \{S, A, \lambda, R\}$, where:

1. S is a set of system states $s_i \in S$.
2. A is a set of possible actions $a_i \in A$ in any state $s_i \in S$.

3. λ is a transition delay λ_a associated with any actions $a_i \in A$, and is also used if the action results in a transition from state i to j .
4. R is a set of rewards functions r_{ij} dependent on the state s_i and the action a_{ij} .

States in the MDP must reflect the amount of knowledge gained by the attacker. Here the very natural assumption is made that if a given set of algorithms has been broken, the time it takes to break future algorithms is the same regardless of the order in which the earlier algorithms were broken. (The experiments showed this a valid assumption, since the time to break A3 was not influenced by the order in which the earlier ones were broken). Then, the state is completely specified by maintaining which algorithms are broken. If G is the set of all algorithms (with $|G|$ elements), then a state $s \in S$ is a tuple $s = (g_1, g_2, \dots, g_{|G|})$, where $g_i = 0$ if the i -th algorithm has not been broken yet, $g_i = 1$ if the i -th algorithm has been broken.

The actions in a state represent the selection of a next algorithm to be released. Thus, there is an action corresponding to any algorithm that is not yet broken; that is, there are as many possible actions in state $s_i \in S$ as there are 0 elements.

The delays signify the time it takes for an attacker to break the algorithm associated with action a_i . This time depends on the knowledge gained from breaking earlier algorithms, which is maintained in the state.

This formulation immediately shows that there exist, at most, $2^{|G|}$ states. The possible order in which the $|G|$ algorithms can be released is $|G|!$. To determine which release order is optimal, it is first necessary to define the optimization criterion. For that optimization criterion, a reasonably efficient algorithm is required to search through the many options.

The particular metric of interest (and, hence, the optimization criterion) in this study, as mentioned previously, is for maximizing the time it takes to break all algorithms. Therefore, let the stochastic process $R(t)$, defined for $t \geq 0$, indicates if all algorithms have been broken at time t : $R(t) = 1$ if $s = (1, 1, \dots, 1)$ and otherwise $R(t) = 0$. It is important to note that $R(t)$ turns 1 only once, and then stays 1. The probability that all algorithms are broken at time t is $P(R(t) = 1)$, where P indicates the probability, as usual. $R(t)$ also pro-

vides the *Mean Time to Security Failure*¹⁹ (MTTSF) [2]: $E[R(t)] = \int_0^\infty (1 - R(s))ds$, and with higher moments similarly. In what follows, $R(t)$ is referred to as the time to security failure.

Finding the best strategy corresponds to a standard Markov Decision Process optimization problem with a finite horizon only for the first moment²⁰ $E[R(t)]$, but not for higher moments or its distribution. In the following section, a specific backward algorithm is presented that efficiently generates all paths ‘backwards’ from the state in which all algorithms have been broken.

5.2 Optimisation Algorithm

The optimization term in this chapter, and in the thesis, refers to the selection of a best defensive algorithm to be released from a set of available alternative algorithms. The selection of best defensive algorithm to be released is calculated by a probability distribution. In order to describe this probability distribution, the exponential distribution needs to be defined. As mentioned previously in Chapter 4, the exponential distribution is the probability distribution that describes the time between events that occur continuously and independently at a constant average rate. Despite the exceptional mathematical tractability that flows from the memoryless property of the exponential distribution, mathematical tractability sometimes is not sufficient to overcome the need to model processes for which the exponential distribution is simply not adequate. Thus, Phase-type distributions [138] permit the modelling of more general distributions, while maintaining some of the tractability of exponential distribution.

To calculate the optimal strategy of the release order in the proposed optimisation algorithm, it is useful to realize that any selected sequence of algorithms corresponds to a hypo-exponential distribution, which in turn is a special case of a Phase-type distribution. The hypo-exponential is a series of k exponential distributions, each of which has its own rate λ_i , the rate of the i^{th} exponential distribution. If there are k independently distributed exponential random variables x_i , then the random variable:

¹⁹ For quantifying the security of a system, MTTSF refers to the length of time to reach such absorbing states [2].

²⁰ The first moment refers to a mathematical quantity that is defined in relation to random mathematical objects known as a point process. This point process seeks to represent a collection of points randomly on some underlying mathematical space. Therefore, the s^{th} moment of a set of data with a total of n discrete points X_1, X_2, \dots, X_n is given by the formula: $(X_1^s + X_2^s + \dots + X_n^s)/n$. For instance, the first moment was set as $s = 1$ [104].

$$X = \sum_{i=1}^k x_i$$

is hypo-exponentially distributed [138]. Therefore, the following result for hypo-exponential distributions is needed: If H_1 is hypo-exponential with rates $\lambda_1, \dots, \lambda_K$ and *MTTSF* $E[R_1(t)]$, and H_0 is hypo-exponential with rates $\lambda_0, \lambda_1, \dots, \lambda_K$, and *MTTSF* $E[R_0(t)]$, then $E[R_0(t)] = 1/\lambda_0 + E[R_1(t)]$.

Although this is an obvious result, it is important to note that the same does not hold for higher moments. In other words, lower moments are utilised here in this optimisation. The above implies that a *backward algorithm* can be executed that optimizes for hypo-exponential distributions of increasing length. The backward algorithm is an inference algorithm which computes the posterior marginal of all state variables given a sequence of observations. The algorithm represents the principle of dynamic programming to efficiently compute the values that are required to obtain the posterior marginal distribution [121]. Thus, it implies that known MDP theory can be used, since reward $r_{i,j} = 1/\lambda_{i,j}$ can be associated with each transition from state i to j . Because of the specific structure of the proposed model, it makes sense to provide a bespoke algorithm that avoids generating the complete state space S as shown in Figure 5.1.

```

start = (0, 0, ..., 0);
end = (1, 1, ..., 1);
For All s ∈ S set ETs = 0;
ToDoSet = {end};
While( ToDoSet ≠ {start} ) Do {
    ToDoSet = {s | s → i, for any i ∈ ToDoSet}
    For All s ∈ ToDoSet Do {
        For All i ∈ S such that s → i Do {
            If( 1/λs,i + ETi > ETs ) Then {
                ETs = 1/λs,i + ETi;
                BestNexts = i;
            }
        }
    }
}

```

Figure 5.1: The Backward Optimization Algorithm.

In the proposed backward optimisation algorithm, it is noticeable that $(1, 1, \dots, 1)$ is the absorbing state with all algorithms broken. The algorithm starts from that absorbing

state and explores all possible previous states (stored in `ToDoSet`). For each previous state, it selects the action that maximizes the time to reach the absorbing state (stored in the `BestNext` variable associated with each state). This continues until the state with no broken algorithms is reached.

Given the backward optimisation algorithm, the optimal order of releasing the algorithms is then obtained as follows, in the tuple `Optimal`:

```
s = start;
Optimal = (s);
While( s ≠ end ) Do {
Optimal = (Optimal, BestNexts);
s = BestNexts;
}
```

It is noted that the above algorithm neither generates the complete state space S , nor all possible sequences of algorithms. The storage required is approximately $N!/[(N/2)!(N/2)!]$ real-valued variables, which occurs halfway through the backward algorithm (which starts with a single state (`end`) and ends with a single state (`start`)). That still limits the size of the model one will be able to solve, but with modern day computing equipment this implies that the problem can be solved for up to several tens of algorithms.

It is important to remark that the above algorithm does not work if higher moments are considered. Moreover, it is also straightforward to find release strategies that optimize the *MTTSF*, but which do not optimize the second moment of the time until security failure.

5.3 Application to the Example

This section presents an example in which the best strategy can be achieved. The example presented, with three algorithms, is of course a simple case, in that it has only a few states, and the best release strategy can therefore be easily computed. Nevertheless, it is useful to provide the MDP for this case that is discussed in the following subsection.

5.3.1 Markov Decision Process for Example

In the experimental study conducted in Chapter 3, the release order of the three developed algorithms (A1, A2, A3) was: A1, A2 then A3 for Group 1, while it was A2, A1 then A3 for Group 2. The MDP for this is provided in Figure 5.2.

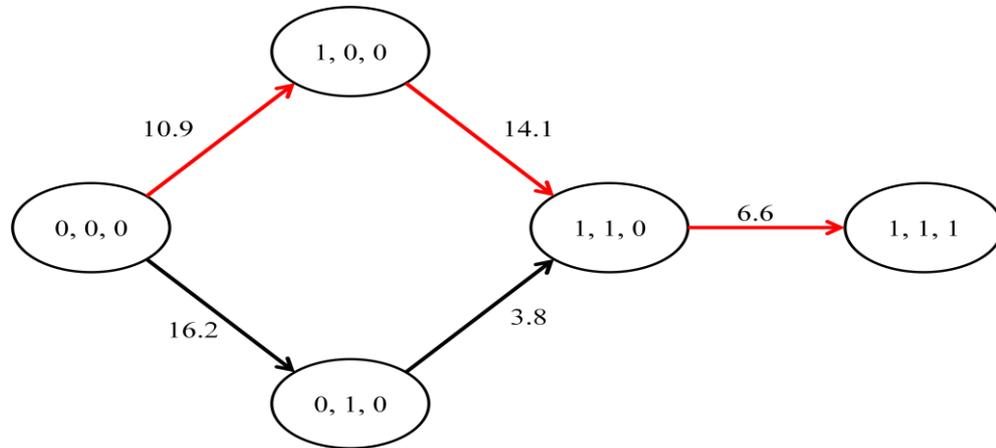


Figure 5.2: Markov Decision Process for Example.

As mentioned, there are three algorithms in this example, (A1, A2, A3), leading to 8 theoretically possible states (denoted by the circles in Figure 5.2). Since the release orders for both groups in the experimental study put A3 at the end, the possible order is restricted in this example and always puts algorithm A3 last. The actions in each state are given by the arcs. Only in state (0,0,0) there is a choice between actions, namely to first release algorithm A1 (leading to (1,0,0)) or algorithm A2 (leading to (0,1,0)). The arcs are labelled according to the time it takes to completely break the algorithm, as seen from the experiment carried out in Chapter 3. Referring back to Chapter 3, Group 1 followed the trajectory at the top of Figure 5.2, using 10.9 minutes to break A1 and 14.1 minutes to break A2. Group 2 followed the trajectory at the bottom of Figure 5.2, using 16.2 minutes to break A2 and 3.8 minutes to break A1. Then all participants broke A3, in an average of 6.6 minutes.

The backward optimization algorithm of Figure 5.1 traverses backwards and picks the best action. Before getting to state (0,0,0), it obtains intermediate results of $14.1 + 6.6 = 20.7$ for state (1,0,0) and $3.8 + 6.6 = 10.4$ for state (0,1,0). For state (0,0,0), it then selects the action that maximizes the time to security failure, so it releases algorithm A1 first (the trajectory at the top of Figure 5.2), because $10.9 + 20.7 > 16.2 + 10.4$. Thus, the optimal release strategy becomes A1 followed by A2 followed by A3.

5.4 Discussion

The experimental study accomplished in Chapter 3 confirms that optimising the release order for a set of algorithms can increase the time needed to break a system's security in a statistically significant manner. This optimisation problem led to investigating an appropriate formalism to optimise the release order strategy. Accordingly, the optimisation problem is modelled mathematically as a Markov Decision Process and a tailored optimisation algorithm is provided, as shown in Figure 5.1 using efficient quantitative methods.

By applying the optimisation algorithm, the optimal release order of the defensive algorithms indicates that releasing the subset algorithms before the superset algorithms can maximise the length of time taken to break the algorithms. This is because, according to the empirical results accomplished in Chapter 3, the rules of superset algorithms need more time to be disclosed, even though the rules of its subset algorithm have been exploited. Although gaining knowledge of the rules for subset algorithms by the attackers could offer a number of indications of how the rules of superset algorithms act, releasing the superset algorithms after the subset algorithms keeps the attackers in the learning phase for a considerable length of time.

The results of applying the proposed optimisation algorithm to dependent algorithms (i.e. super and subset algorithms) suggest that it may be a suitable tool for optimising the release order of a set of algorithms which was basically one algorithm broken up into superset and subset parts. Furthermore, when a system has a limited type of defensive algorithm, this optimisation algorithm may be adapted to optimise the release order of these algorithms.

Moreover, the consequence of using the model-based order proposed in Chapter 4 is that the proposed optimisation algorithm may be a suitable tool to release a set of algorithms for a system that has unrestricted types of defensive algorithms (i.e. using mixed dependent and independent algorithms). Although establishing this kind of system may be a challenge in terms of cost, the security level of the system can be increased significantly by releasing each independent algorithm. As a result, a tradeoff can be recognised clearly between the security level and the financial cost in addition to the third factor, which is the *usability* that complements them, as reported in [68].

It is interesting to note the relative correlation between the release order of the defensive algorithms strategy and the game theory approach, with regards to the demand of the optimisation problem. That is, game theory can provide a mathematical framework for analysing and modelling network security problems. This framework has been targeted by various studies such as in [6, 7, 67, 129, 130]. The optimisation algorithm is applied in these studies in order to provide a level of security from different angles. For instance, Alpcan et al. investigated in [6] how long it took the game to approach a Nash equilibrium when many players tried to solve it in a distributed way. A feedback system approach is suggested as a control input to make the system robust and to control the system's progress. In addition to this study, Alpcan and Baser utilized in [7] an optimal reactive defensive action through the Min-max Q learning approach in order to gradually improve the defender's quality. Moreover, Jiang et al. developed in [67] an optimal active defensive strategy decision algorithm. Despite these studies which involve dynamic games apply optimisation algorithms in order to find the best solutions among a set of candidate solutions, none of them consider the attacker's learning and/or maximizing the duration of the game as the game's objective.

Likewise, an optimisation algorithm was used in [122] in order to gain scalable optimal countermeasure selection using implicit enumeration on Attack Countermeasure Trees (ACT). However, this solution focuses on a static attack scenario and predefined countermeasure for each attack.

5.5 Summary

This chapter has given an account of and the reasons for the demand of optimising the release order of a set of algorithms approach. This demand for optimisation is introduced in Chapter 3, which schedules the release of defensive algorithms so as to prolong the time attackers need to successfully defeat all algorithms.

In this chapter, the aim was to develop and provide an optimisation algorithm for the release order of defensive algorithms. Therefore, this chapter has provided a tailored optimisation algorithm using a Markov Decision Process to obtain efficiently the optimal release strategies for any given model.

Based on the empirical results achieved in Chapter 3, an application example has been demonstrated. The results of this application indicate that the proposed optimisation

algorithm may be a useful tool to optimise a set of similar algorithms with variation in their rules. Not only this, but the proposed optimisation algorithm can also be a practical tool to optimise a set of mixed dependent and independent algorithms. Nonetheless, this type of several algorithms for a system can be costly in terms of money, though the security level of the system could be increased, and a tradeoff between the security and expense of the system could be recognised intuitively. In general, therefore, the proposed model solution should scale without problems to optimise the release order of tens of defensive algorithms.

As the learning acquisition process of the attacker plays an important role in this thesis, the next chapter sheds light on this topic by investigating an Attacker Learning Curve notion based on the data collected in the controlled experimental study carried out in Chapter 3.

Chapter 6. ATTACKER LEARNING CURVE

This chapter explains the proposed Attacker Learning Curve (ALC) notion, while analysing the data collected from the experimental study reported in Chapter 3. Since the defence algorithms used in the experiment evaluate the attackers' attempts based on a similarity mechanism, this leads to similarity-based quantitative data. Therefore, the idea of the ALC lies in collecting the attempts data of an attacker when several manipulated attempts to break an algorithm are carried out by the attacker. As such, accumulative manipulation, which is the attacker's aggregated amount of knowledge, can gradually create the ALC. The ALC effectively represents how close an attacker is to breaking a defence algorithm.

This chapter also outlines several strategies which were utilised to break the algorithms, and which were discovered when analysing the attackers' attempts. By applying the proposed ALC, the impact of all used strategies used in breaking the algorithms is demonstrated. Furthermore, an ALC is formalised as an Attacker Learning Curve Model (ALCM) that allows estimation of the learning curve of an attacker to break an algorithm. An early version of this proposed ALC notion and its model were published in [14].

The remainder of this chapter is organised as follows. Section 6.1 begins by the theoretical dimensions of the proposed idea of the ALC. Section 6.2 describes the design, synthesis, characterisation and evaluation by an illustrative example of the ALC. In Section 6.3, the impact of all observed strategies in breaking an algorithm is discussed. Section 6.4 describes the inspired ALCM. Section 6.5 presents the discussion. Finally, Section 6.6 summarises the chapter.

6.1 An overview of Attack Scenario

In order to define the principle behind this chapter, an overview of the attacker scenario is provided, involving the attacker and the defensive mechanism, as shown in Figure 6.1. This figure describes the interaction scenario between an attacker and a security defensive mechanism.

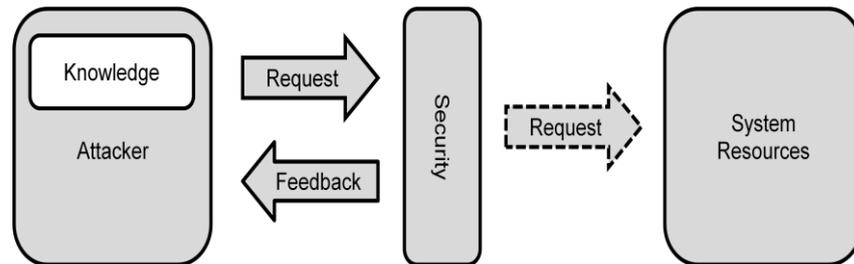


Figure 6.1: Attack Scenario.

The scenario starts with an attacker who has prior knowledge regarding the rules used by the defensive algorithms to classify a request. Then, the attacker attempts to structure requests systematically in order for the requests to be classified by the security layer as accepted, since the security layer classifies requests as acceptable or not based on a set of rules. An acceptable request can proceed through the security layer and use system resources, while an unacceptable request is blocked. On each failed attempt (i.e. unacceptable request), the attacker receives feedback from the system with regards to the failed attempt, as shown in Figure 6.1. This feedback may be a simple Boolean response, or may include reasons for the failure, as mentioned previously in Chapter 1.

Indeed, this feedback can be a fundamental aspect that motivates attacks on the interactive defensive mechanisms, as detailed in Chapter 1. In particular, the attacker learns from the feedback, and uses it for subsequent requests. By repeatedly performing this knowledge acquisition process, the attacker can gradually derive the rules used by the algorithms to classify requests, including both the parameters used, and the value of these parameters. For example, attackers executed an attack on several CAPTCHA schemes by gradually deriving the parameters used and the value of these parameters, until this scheme was broken [150]. The attacker can then misuse the system resources by sending requests that are structured in such a way that they are classified as acceptable by the algorithm that is in the security layer, in which case, the algorithm is considered broken.

The focus of this scenario is mainly on a similarity assessment mechanism, where any request from each attacker is evaluated by comparing this request with all previous requests. In this light, a similarity threshold, which is a lower limit for the similarity of two data records that belong to the same cluster, is defined. Additionally, it is assumed in this scenario that an attacker sends a set of messages in order to break the released algorithm that is in the security layer. Intuitively speaking, a rule R consists of a set of previous messages that are known to be attacks. This set is updated with each attempt, and so $R_n = \{p_1, p_2, \dots, p_n\}$ is written for the rule R at step n , where each p_i denotes the previous attempt at step i . An attacker submits a message m with some manipulations against a rule R , and either this message is considered different enough and is thus accepted, or it is not, in which case m is added to R . The difference between m and R is calculated with the functions:

$$\delta(p, m) = |\{i \in [0, MH] \mid h_i(m) \neq h_i(p)\}|$$

$$\Delta(R, m) = \min \{\delta(p, m) \mid p \in R\}$$

where $h_i(m)$ indicates the i^{th} hash value of the message m , and MH is the maximal number of hashes. Given a rule R and a message m , if m has a lower similarity threshold ST , then m is accepted and added to R (i.e. the rule of the algorithm is broken); otherwise, m is rejected.

For the purpose of this scenario, a detection approach adapted from that utilised in Chapter 3 (i.e., using a spam filtering detection approach) is used to detect attempts of attackers. This is the experimental scenario considered here, but it is also a good fit for the detection of abnormal database queries, or detection of information leakage. More details of how this spam filtering detection approach is exploited to develop the ALC are discussed in the following section.

6.2 Attacker Learning Curve

The Learning Curve phenomenon is widely known, especially in the psychology and economic fields. As such, this phenomenon is exploited in two significant ways: where a body of knowledge is increased overtime or where an identical task is repeated in a number of trials [118]. For instance, since organisations gain experience with production, productivity and quality improve at a decreasing rate. In other words, accumulating experience leads to improved performance [21].

Accordingly, the notion of the Attacker Learning Curve (ALC) is that, as attackers gain knowledge with regards to the rules used in the defensive mechanism, accumulating experience regarding the applied rules in the defensive mechanism leads to improved performance in breaking the defensive mechanism. In view of this, the accumulation of experience can be anticipated quantitatively based on the detection approach of attacker's attempts. That is, the attacker's attempt is evaluated by Similarity-Based evaluation method which is explained briefly in the next subsection. This produces quantitative data (i.e., Similarity-Based data) which is the distance between the attempt and the similarity threshold, as shown in Figure 6.2. Since breaking a defensive algorithm typically requires several attempts by attackers, each attempt is manipulated by attackers in order to avoid detection. Thus, the *accumulative manipulation*, which is the attacker's aggregated amount of knowledge, can gradually build the ALC that effectively represents how close an attacker is to breaking a defensive algorithm. Figure 6.2 depicts the structure of the ALC, starting with the detection of the attacker's attempt through the *accumulative manipulation* process, and then achieving the ALC. Based on the ALC, both quantitative data (i.e., Accumulative-Based data) and qualitative data (i.e., Strategy) are organized as an input in the Training data. The qualitative data will be highlighted in Section 6.3. In addition, although Training data are beyond the scope of this chapter, they will be defined and used as phase 2, and the components of the structure of the ALC as phase 1 in the proposed detection approach that will be described in the next chapter (Section 7.1.3).

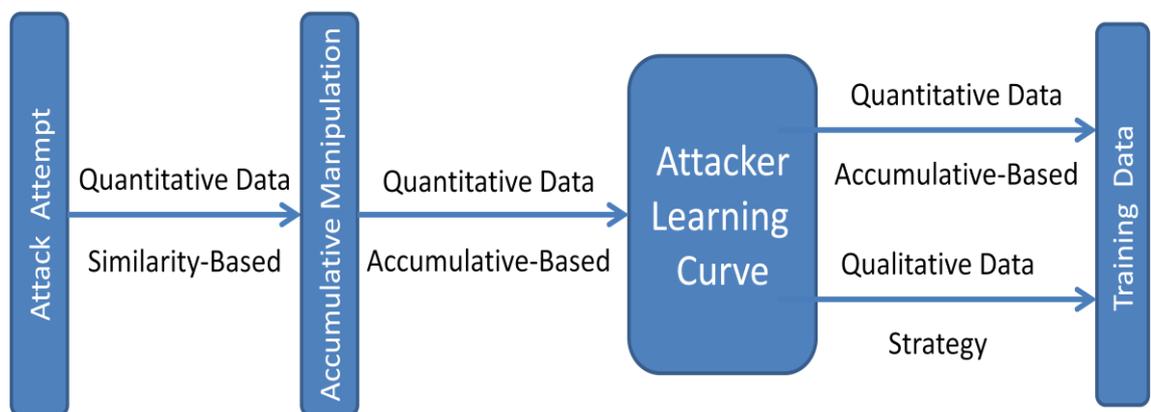


Figure 6.2: The structure of the ALC.

Based on the data analysed from the previous experiment, the following subsection shows examples on how the attacker performance is extracted from the similarity evaluation results of the defence algorithm.

6.2.1 Extracting Similarity-Based Data

The similarity evaluation approach is one of the common approaches in interactive defensive mechanisms to determine an attack attempt [152]. Since the previous controlled experiment study uses a similarity evaluation approach, each similarity result of each attempt is recorded and collected²¹. Based on the data collected from this experiment, similarity-based data are developed in this study to demonstrate the attacker’s progress during the attacking process. Therefore, Figure 6.3 shows an example of one attacker’s performance, which is extracted from the experiment data to break all algorithms (i.e. Algorithm 1 (A1), Algorithm 2 (A2) and Algorithm 3 (A3)). The example shown in Figure 6.3 uses *structured strategies* to break the algorithms. More details about strategies applied in the attack process are in Section 6.3.

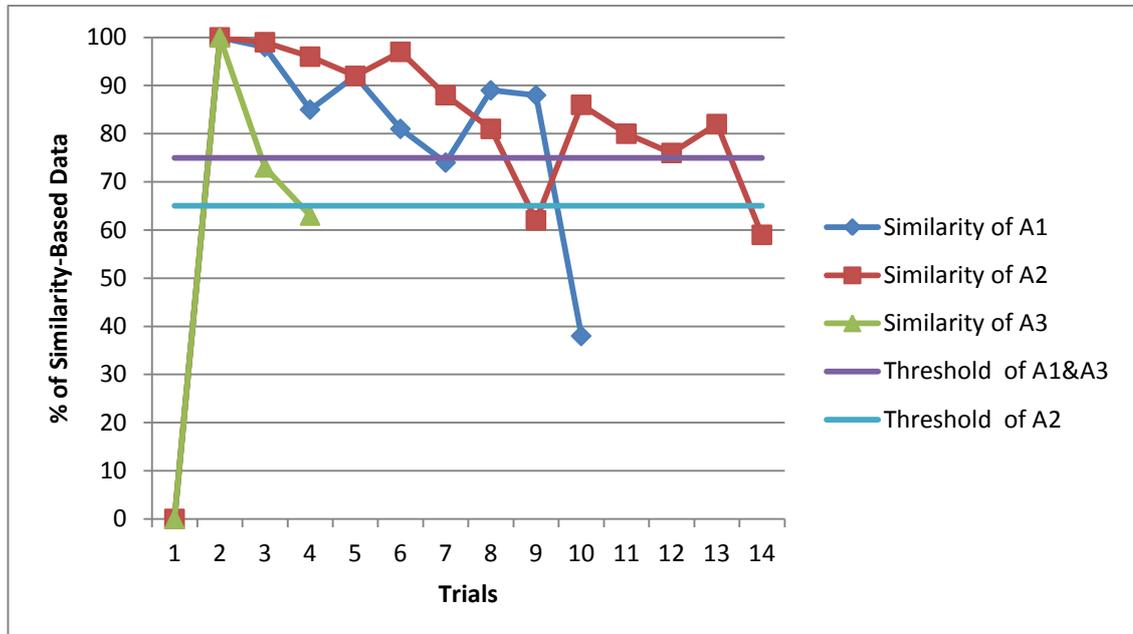


Figure 6.3: Example of Structured Attacker Performance.

In contrast, Figure 6.4 illustrates an example of one of the attackers’ performances to break all algorithms by using *random strategies*. Hence, the impact of the strategies on the attack process can be observed in terms of time taken. Furthermore, these similarity-based data encourage investigation into the *accumulative manipulation* concept.

As the *accumulative manipulation* term has an important implication for developing the ALC, the formalism of this term is presented in the following subsection.

²¹ For each algorithm, an attack attempt is evaluated against a similarity threshold. While the similarity of an attempt is above the similarity threshold, the attacker needs to try again.

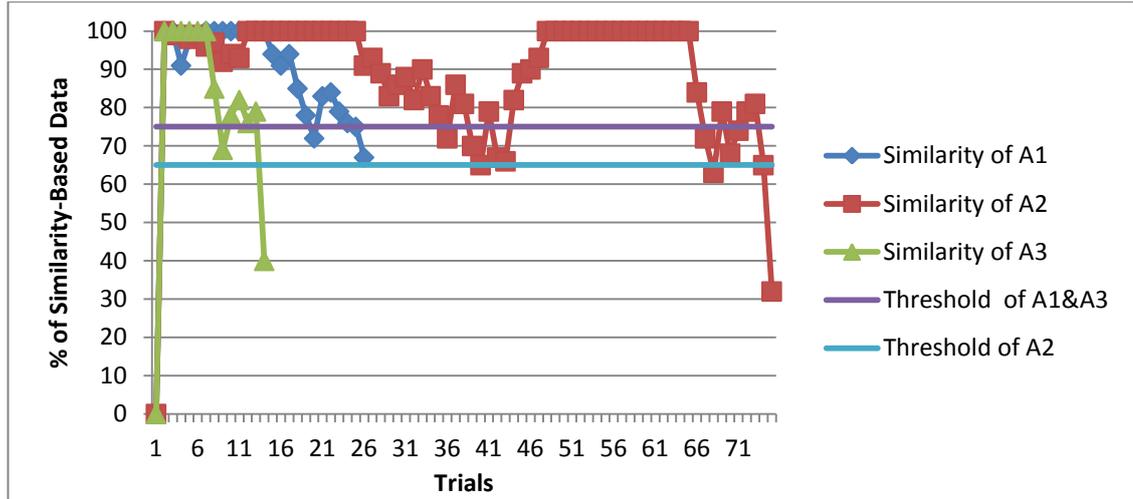


Figure 6.4: Example of Unstructured Attacker Performance.

6.2.2 Accumulative Manipulation

The manipulation of an attacker denotes the distance between one attempt from an attacker and his previous attempts. The *accumulative manipulation* of an attacker A_k is therefore given by the function $AcMan$, defined as:

$$AcMan(A_k, R, i) = \begin{cases} AcMan(A_k, R, i-1) + \Delta(R_i, m_{ki}) & (i \geq 1) \\ 0 & (i = 0) \end{cases}$$

where R is a rule and m_{ki} denotes the i^{th} message sent by A_k . For example, consider the simple case where the length of a message is limited to 7, i.e., $MH=7$, where the rule R is initialised with “Message” and where the similarity threshold is set to $ST=2$, meaning that at least two characters need to be different in order for a message to be accepted. If A_k attempts to submit “Message”, then all hash values are identical, and this attempt is therefore rejected, and $AcMan(A_1, R, 1) = 0$.

If the next message sent by A_k is “Messoge”, then $h_5(\text{“Messoge”}) \neq h_5(\text{“Message”})$, and it follows that $\Delta(R, \text{“Messoge”}) = 1$. Since this value is still below ST , the new message is added to the rule, i.e., $R = \{\text{“Message”}, \text{“Messoge”}\}$ and $AcMan(A_k, R, 2) = 1$. Now, if the next message is “Messo9e”, it remains the case that $\Delta(R, \text{“Messo9e”}) = 1$, since Δ considers the minimum of δ . However, $AcMan(A_k, R, 3) = 1 + 1 = 2$, thus denoting that the attacker has made some progress. It is worth noting here that the accumulative manipulation can continuously increase, even if the attacker never breaks the rule. More details will be given in the following subsection by presenting an illustrative example based on empirical results.

Next, an ALC example derived from the empirical results of the experimental study in Chapter 3 is described. This example demonstrates the efficiency of accumulative manipulation.

6.2.3 Illustrative Example to ALC

In order to link the data collected in Chapter 3 with the derivation of the ALC, the background of the attack attempt detection mechanism is briefly outlined. In the experiment, attackers sent a manipulated e-mail to the system for breaking its security algorithms. Each e-mail submitted was evaluated by a similarity approach to quantitatively determine an attack attempt based on similarity with previously submitted e-mails. The level of similarity is exploited in this study by calculating the accumulative manipulation. In particular, the similarity of submitted emails, which is determined by the similarity threshold, is subtracted from the whole hash values (i.e. 100 hash values²²), and this difference is called the amount of effective manipulation (e.g. the third column in Table 6.1). This amount is then increased accumulatively while the attacker is attempting to break an algorithm, as shown in the fourth column in Table 6.1.

Table 6.1 presents the calculation of the accumulative manipulation of one of the attackers, who performed the attack task successfully in the previous experiment in Chapter 3. The similarity threshold of this algorithm was 75%. Based on this accumulative manipulation, Figure 6.5 illustrates the ALC.

Table 6.1: Calculating the Accumulative Manipulation.

Trials	Similarity	Manipulation=100-Similarity	Accumulative Manipulation
1	100	0.00	0.00
2	95	5.00	5.00
3	91	9.00	14.00
4	99	1.00	15.00
5	92	8.00	23.00
6	87	13.00	36.00
7	97	3.00	39.00
8	84	16.00	55.00
9	73	27.00	82.00
10	88	12.00	94.00
11	76	24.00	118.00
12	69	31.00	149.00

²² More details regarding the value of parameters are presented in Chapter 3, specifically in Section 3.3.4.

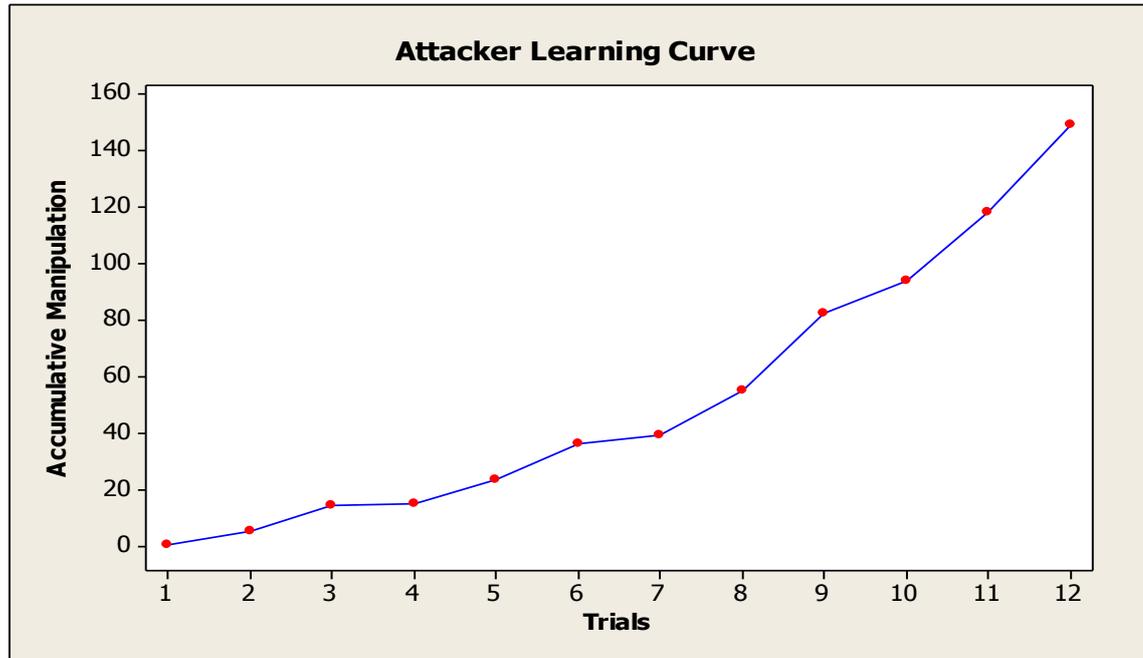


Figure 6.5: The Attacker Learning Curve Based on the Accumulative Manipulation.

It is important to note that the attackers in the experiment were asked to break the algorithm twice (i.e. pass through the spam filter algorithm with 3 e-mails, where the first email was for training the database and the remaining two are for breaking the algorithm). The reason behind this was to make sure that the attacker was learning during the attack process. For this, in Table 6.1, the row highlighted in orange corresponds to the first break, while the one highlighted in red corresponds to the second break. Hence, as shown in Table 6.1, once the attacker had defeated the algorithm, the second break required only two trials.

6.3 Strategies Applied in Attack Process

Since attackers' tactics and techniques are constantly evolving, attackers are continually developing new attack tools and strategies to enable the possibility of a successful attack process against a secure system. In the interests of understanding the strategies applied in an attack process of the previous experiment, qualitative data in the form of a survey were collected from the experimental study, as mentioned in Chapter 3, to verify that the attacking process was accomplished by structured strategies based on the knowledge gained rather than complete randomness. In particular, each attacker was asked to detail the strategy (or strategies) that were used to defeat the algorithms, the part of the email that the participants believed that each algorithm was checking, and the algorithm which the participants thought was the toughest to defeat, as shown in Ap-

pendix A. The results of this survey indicate that 90% of the attackers (36 out of 40) used structured strategies to defeat the algorithms.

These results led to further investigation into the text of the e-mails sent by each attacker. The main purpose of this investigation was not only to identify the most effective strategy in terms of consuming time in the attacking process, but also to demonstrate the influence of the observed strategies on the ALC. Thus, the following subsections describe the observed strategies and their impact on breaking the algorithms by utilising the ALC based on the results of the investigation.

6.3.1 Observed Strategies

There were three main strategies that attackers used which were discovered by means of the investigation procedures. These strategies are as follows:

- Thesaurus substitution, Perceptive substitution, Delete spaces.
- Random addition, Thesaurus substitution, Perceptive substitution, Add spaces.
- Perceptive substitution.

where Random Addition adds random characters to the original email text; Thesaurus Substitution substitutes some words of the original text with synonyms defined in a thesaurus; Perceptive Substitution substitutes some characters of words in the original text without changing the aim of how the words are to be perceived by the reader, for example, “security” could become “s3curity”; Add Spaces randomly adds spaces; and Delete Spaces randomly deletes spaces. The effectiveness of each of these strategies in manipulating a given e-mail text is described as follows.

Suppose that a spam e-mail is sent by an attacker to users, as shown in Figure 6.6. This e-mail can be then manipulated by the attacker using a Random Addition strategy, Thesaurus Substitution strategy, Perceptive Substitution strategy, Add Spaces strategy or Delete Spaces strategy, as shown in Figures 6.7, 6.8, 6.9, 6.10 and 6.11, respectively.

Dear student,
 We are dictate2us, the UK's number one transcription service for academics! We have a great deal of experience in typing: Dissertations.
 Contact us today for more information, or for a free no obligation quote and let d2u take the stress out of your studies.
 Kind Regards,
 Dictate2us
 Office: 0161 762 1100
 Fax: 0161 762 9694

Figure 6.6: Original Spam E-mail.

Dear studentttt
 We are dictate2uss, the UK's number one transcription service for academics!!!! We have a great deal of experience in typing::: Dissertations.rrrrrrrrrrrr
 Contact us today for more information, or for a free no obligation quote and let d2u take the stress out of your studies.xxxxxxxxxxxx
 Kind Regards,,,,,
 Dictate2us
 Office: 0161 762 1100
 Fax: 0161 762 9694

Figure 6.7: Using Random Addition.

Hello Student,
 We are dictate2us, the UK's number one transcription service for academics! We have a large deal of knowledge in writing academic papers such as dissertations.
 Contact us today for more information, or for without charge no obligation quote and let d2u take the stress out of your studies.
 Kind Regards,
 Dictate2us
 Agency: 0161 762 1100
 Fax: 0161 762 9694

Figure 6.8: Using Thesaurus Substitution.

D3ar Student,
 We are dictate2us, the UK's #1 transcriptiOn service 4 academics! We have a great dea1 of exp3rience in typing: Dissertation5.
 Contact us t0day 4 more information, or 4 free no ob1igation quote and let d2u take the stress out of y0ur studies.
 Kind Regards,
 Dictate2us
 Office: 0|6| 762 |100
 Fax: 0|6| 762 9694

Figure 6.9: Using Perceptive Substitution.

Dear Student,
 We are dictate2us, the UK 's number one transcription
 service for academics! We have a great deal of
 experience in typing : Dissertations.
 Contact us today for more information, or for free no
 obligation quote and let d2u take the stress out
 of
 your studies.
 Kind Regards,
 Dictate2us
 Office: 0161 762 1100
 Fax: 0161 762 9694

Figure 6.10: Using Add Spaces

Dear Student, We are dictate2us, the UK's number one
 transcription service for academics! We have a great deal of
 experience in typing: Dissertations. Contact us today for more
 information, or for free no obligation quote and let d2u take
 the stress out of your studies.
 Kind Regards, Dictate2us
 Office: 01617621100 Fax: 01617629694

Figure 6.11: Using Delete Spaces.

The influence of each strategy on the ALC is discussed in the following subsection.

6.3.2 Impact of All Strategies in Breaking Algorithms

Since the ALC effectively represents how close an attacker is to breaking a defensive algorithm, it is interesting to demonstrate the impact of each observed strategy, which is described in the previous section, in breaking all algorithms outlined in Chapter 3 by the proposed ALC. In addition, because of the experimental groups (i.e. Group 1 and Group 2 as stated in Chapter 3) took a similar amount of time to break algorithm 3 (A3), a demonstration of the impact of all observed strategies in breaking algorithms is only based on the data collected for breaking A3 in both groups. The reason for this is that it allows demonstration of the ALC in a comparable model. In other words, the results of the ALC based on Group 1's data can be compared with those of the ALC based on Group 2's data. For this, the average accumulative manipulation of each strategy in both groups is calculated, as depicted in Figure 6.12.

It can be observed that although the groups are disjointed, each strategy behaves in a comparable way in each group. For example, in both groups, strategy 2 was the most

effective in terms of breaking the algorithm, whereas strategy 3 was the least effective. In other words, the time taken to break an algorithm by using strategy 3 was longer than that for strategy 2.

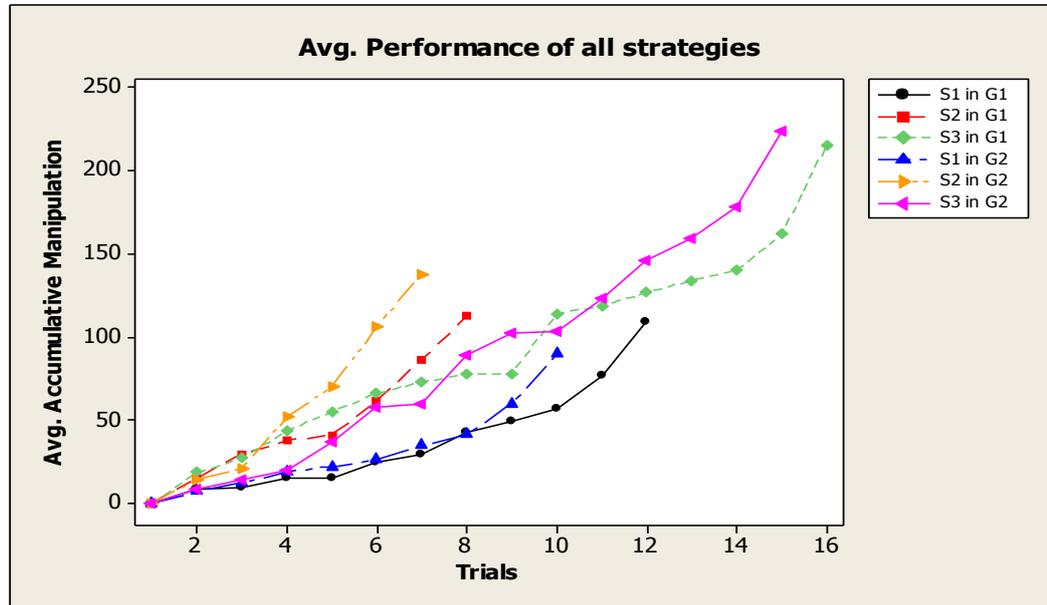


Figure 6.12: The average Accumulative Manipulation vs. all strategies.

Based on these results, the ALC is modelled as an Attacker Learning Curve Model (ALCM). The developed ALCM is inspired by a previous model used for describing the development learning curve during software development. More details are presented in the following section.

6.4 Attacker Learning Curve Model (ALCM)

This section describes the proposed ALCM that has the potential to estimate the learning curve of an attacker in breaking an algorithm. Specifically, since several models have been proposed for software development to estimate the progress of software development, one of these models [58] is used here in this chapter as a starting point for constructing the proposed ALCM. In the following, the knowledge model of the previous model of software development, the proposed ALCM, and the performance of the proposed model are presented.

6.4.1 Knowledge Model

Hanakawa et al. proposed in [58] a simulation model for software development that takes into account the developer's learning curve (more details are in Section 2.3.1).

This model shows quantity of gain to a developer's knowledge by executing an activity. This quantity of gain to the developer's knowledge is derived from the relationship between b_{ij} , which is the developer's experience level i while performing the activity j , and θ , which is the required knowledge level to execute this activity. This model is defined as follows [58]:

$$L_{ij}(t) = \begin{cases} K_{ij}e^{-E(\theta-b_{ij}(t))} & (b_{ij} \leq \theta) \\ 0 & (b_{ij} > \theta) \end{cases} \quad 1$$

where $L_{ij}(t)$ is the quantity of gain to knowledge of developer i by executing the activity j , which has knowledge level θ , at time t ; K_{ij} is the maximum quantity of gain to knowledge of the developer i by executing activity j ; b_{ij} is the developer i 's knowledge level about activity j ; E is the developer's efficiency of gain to knowledge by executing activity j ; and θ is the required knowledge level to execute the primitive activity of activity j .

The knowledge level is reset to the developer's new knowledge level b_{ij} at each step:

$$b_{ij}(t+1) = b_{ij}(t) + L_{ij}(t) \quad 2$$

Therefore, by plotting the level of the developer's knowledge in time sequence, the developer's learning curve can be determined during the execution of an activity. In the simulation of this model, the growth of the developer's knowledge level b_{ij} during the execution of activity j shows the developer's learning curve, as shown previously in Chapter 2 (Figure 2.7). In that figure, Line (1) shows the learning curve in the simulation in which the growth of the developer's knowledge level b_{ij} has a great impact on the development progress. Additionally, when the activity is chosen in ascending order of the required knowledge level, then the shape of the learning curve will be flat, as shown in Line (2). In light of this, Equation (1) can be exploited to form the proposed ALCM, and this is described in the subsequent section.

6.4.2 Proposed ALCM

As the proposed ALCM is based on Equation (1), each factor in this equation is developed for the purposes of the proposed model as follows. The required knowledge level θ is defined by the Algorithm Robustness Level (ARL) [14]:

$$ARL=T/SL$$

3

where T is the time required to break the algorithm and SL is the skill level of the attacker. This skill level is divided into three main categories, as suggested by [112]: Beginner, Intermediate and Expert. Since the previous experiment presented in Chapter 3 focuses on beginner attackers²³, a single value is considered for SL , experimentally defined as 0.3, and this value is also used for the parameter E , characterizing the efficiency of the attacker. The quantity of knowledge gained of the developer L_{ij} is defined with regard to Attacker Knowledge AK_{ij} . The maximum quantity of knowledge that can be gained K_{ij} is defined with respect to the used strategy S_{ij} by the attacker i in order to break an algorithm j , and has empirically established values 7.5, 10.5 and 5 for strategies 1, 2 and 3, respectively. Finally, the developer's knowledge level b_{ij} is replaced by Accumulative Attacker Knowledge $AccAK_{ij}$ that defines the attacker's knowledge level i about an algorithm j . The proposed ALCM is then defined as follows:

$$AK_{ij}(t) = \begin{cases} S_{ij}e^{-SL(ARL-AccAK_{ij}(t))} & (AccAK_{ij} \leq \theta) \\ 0 & (AccAK_{ij} > \theta) \end{cases} \quad 4$$

The knowledge level is reset to the attacker's knowledge level $AccAK_{ij}$

$$AccAK(t+1) = AccAK(t) + AK(t) \quad 5$$

By parameterising the proposed ALCM based on the empirical results achieved in the previous experiment, estimating the performance of an attacker is possible. The following section presents the results of evaluating the proposed model.

6.4.3 Performance of the Proposed Model

In order to evaluate the performance of the ALCM, the parameters of the model are fitted with the empirical data from Groups 1 and 2, presented in Table 6.2 and Table 6.3, respectively. Figure 6.13 presents the results of fitting the model with both groups, obtained with the Java *jmathplot* library²⁴. For the sake of clarity, the curves are presented by using *percentage-based grading*; that is, each curve reaches 100% when the corresponding strategy in the corresponding group breaks the algorithm's rules.

²³ In particular, the results of the survey indicate that most of the participants had a beginner skill level.

²⁴ <https://code.google.com/p/jmathplot/>

The results of running the model based on the parameters derived from both Groups 1 and 2, as shown in Figure 6.13, indicate that strategy 2 is more efficient in both groups in terms of increasing the learning acquisition process of the attacker, while strategy 3 is less efficient. By using the proposed model, therefore, it is possible to estimate the learning curve of an attacker in breaking an algorithm. For example, when the strategy S , the skill level of the attacker SL , and the robustness of the algorithm ARL are given, the progress of the attacker can be predicted. Furthermore, the accumulative output of the model can also be used for the proposed detection approach which will be shown later in the next chapter.

Table 6.2: Fitted Parameters for Group 1.

Strategy #	Strategy (S)	Avg. Braking Time (T) minutes	Skill level (SL)	Algorithm Robustness Level (ARL)	Attacker Knowledge (AK)
1	7.5	7.32	0.3	24.4	0.0
2	10.5	6.12	0.3	20.4	0.0
3	5	12.7	0.3	42.3	0.0

Table 6.3: Fitted Parameters for Group 2.

Strategy #	Strategy (S)	Avg. Braking Time (T) minutes	Skill level (SL)	Algorithm Robustness Level (ARL)	Attacker Knowledge (AK)
1	7.5	6.24	0.3	20.8	0.0
2	10.5	5.8	0.3	19.3	0.0
3	5	11.9	0.3	39.7	0.0

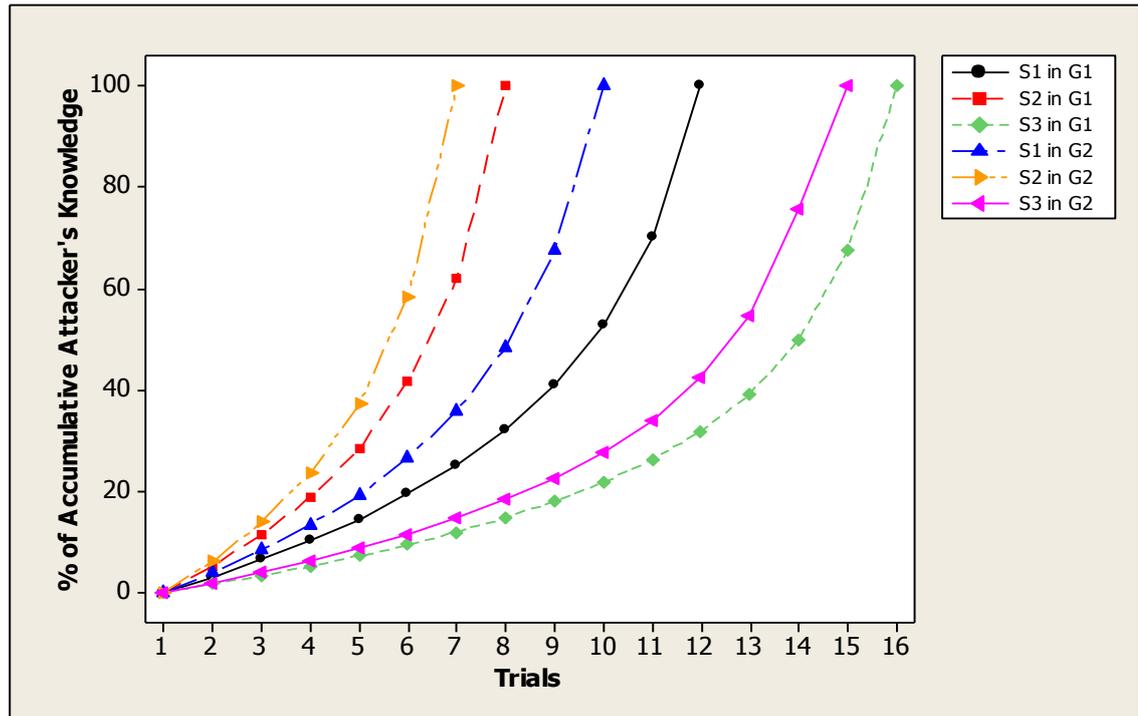


Figure 6.13: Result of running the proposed model on Group 1 and Group 2.

6.5 Discussion

This chapter set out to quantitatively represent the knowledge gained by attackers. The correlation between various manipulated attempts in order to break the interactive defensive algorithms and the accumulated manipulation could gradually build the Attacker Learning Curve (ALC). This ALC represents quantitatively how close an attacker is to breaking a defensive algorithm.

Based on the empirical results achieved in Chapter 3, it could be possible to obtain the manipulation amount of each attacker's attempt. As a consequence, the ALC was represented as shown in Figure 6.5. Although the learning curve concept in the fields of psychology and education has been signified by a graphical illustration of increasing the knowledge with experience [126], to the best of our knowledge this is the first study that proposes the notion of accumulative manipulation, which forms the basis of the ALC, in the security context.

Furthermore, the strategies observed indicate that the attacking process was accomplished by means of structured strategies based on knowledge gained rather than complete randomness. This finding corroborates the idea of the Problem-Based Learning (PBL) approach [23], which is a widely known self-directed learning approach in the

education field that leads to gaining knowledge via thinking strategies. For instance, Hmelo and Ferrari concluded in [62] that PBL is used to help students not only to become active learners, but also to develop strategies and construct knowledge. In general, it may thus be that such connections exist between conducting an attack repetitively against a defensive algorithm and a PBL approach, where an attacker needs to learn strategies to defeat a defensive algorithm and where the students are required to learn strategies to solve a given problem.

On the other hand, the feedback gained through unstructured attacks seems to have been unhelpful to the attackers, as shown in Figure 6.4. An implication of this is the possibility that an attacker who applies an unstructured attack strategy may interrupt ongoing attacks due to the huge number of trials and consequent time consumption. This implication might also apply to structured attacks when an ineffective strategy is adopted, since feedback achieved from this strategy could be unhelpful, as strategy 3 was shown in Figure 6.12.

Another important finding was the validation of the hypothesis concerning the influence of the observed strategies on the developed ALC, as shown in Figure 6.12. This influence reflects the impact of the strategy with regards to its feedback to the attacker. For example, the feedback of strategy 2 was more effective in helping to reveal the rules of the released algorithm than others. Although strategy 1 has overlapping strategies with strategy 2, the feedback of strategy 2 was more supportive in terms of the attacker learning process. Hence, it is possible to hypothesise that the strategy used can be identified by using such detection approaches. This hypothesis will be validated in the next chapter.

It is encouraging to compare this result of the impact of the strategy used with that found by Jonsson and Olovsson who found in [70] that the inexperienced attacker spends more time in the learning phase, while the experienced attacker spends less time. It can thus be suggested that the inexperienced attacker's progress might be improved by employing a more effective strategy. Conversely, the experienced attacker's progress might be reduced by ineffective strategies. Moreover, this finding on the importance of the strategy applied in the attack enhances previous researches into this area such as in [96, 112] which links attacker skill levels and determines the mean time used to compromise a system. Moreover, these findings regarding the manipulated attempts of attack-

ers support the idea of Hung et al. who stated in [66] that it was easy for attackers to search convex classifiers to find input that can avoid being classified as negative.

Additionally, if the ALCM inspired by a previous model that is utilised for explaining a developer's learning curve, the evaluation of the ALCM suggests the applicability of using it. It is important to note that the knowledge acquisition process of an attacker presented in Chapter 4 used the developed model, as shown in Figure 4.10. However, this derivation of the attacker knowledge acquisition process did not take into account the strategy applied in the attacking process. Therefore, the ALCM seems a typical model to estimate the learning curve of an attacker to break an algorithm, due to its recognition of the strategy applied in the attacking process.

This combination of findings provides support for the conceptual premise that the order of defensive algorithms matters due to the rationale that attackers learn from their attempts. Not only this, but it also bears in mind the possibility of improving the proposed release order strategy of a set of algorithms, which will be highlighted in Chapter 7.

6.6 Summary

This chapter has investigated the central importance of the Attacker Learning Curve (ALC) notion of breaking an algorithm. In this investigation, the aim was to represent the ALC quantitatively by observing the accumulative manipulation of an attacker for each attack. The aim was also to represent the ALC based on several applied strategies. This ALC was then modelled to estimate the learning acquisition process of an attacker.

The results of this investigation show that the developed ALC could represent the performance of an attacker quantitatively, depending on the detection approach applied to detect the attempts of attackers. Furthermore, by means of the ALC, it could be possible to distinguish between the strategies used of an attack.

The study has gone some way towards enhancing our understanding of the attacker's performance to break a system using quantitative data. Therefore, this work contributes to existing knowledge on the acquisition process of the attacker by providing the notion of accumulative manipulation, which forms the basis of the ALC, then modelling this as the Attacker Learning Curve Model (ALCM).

Since the recognisability of the ALC for the applied strategies could affect the probability of detecting them, the next chapter highlights a proposed approach that enables detection of the strategies used based on the defined ALC notion given in this chapter.

Chapter 7. DETECTION OF ATTACK STRATEGIES

This chapter describes the proposed simple but novel attack strategy detection approach that builds upon the Attacker Learning Curve (ALC) concept which was explained previously in Chapter 6. That is, as stated in Chapter 1, defensive mechanisms should face attackers who interact with the system by mostly applying a strategy. This strategy plays an important role in receiving feedback²⁵ on effectiveness to the attackers. It also augments the knowledge of the attackers regarding the rules used by defensive mechanisms to characterize misuse. They are then able to adapt their future interactions accordingly, increasing their ability to break the defensive mechanisms, until eventually reaching the point where the defensive mechanism is broken.

A number of interesting solutions have emerged such as Anomaly Intrusion Detection System (AIDS) that utilizes normal usage behaviour patterns to recognize the intrusion. The detection techniques of the AIDS can be classified into three main categories [84]: Statistic-based, Knowledge-based and Machine Learning-based. The machine learning-based category has several advantages such as flexibility and adaptability, and can be generally classified as either Unsupervised or Supervised learning²⁶. Several studies have investigated hybrid²⁷ schemes from different angles (e.g., [55][65]), and although

²⁵ As defined in Chapter 1, the feedback may be a simple Boolean response, or may include reasons for the failure.

²⁶ As defined in Chapter 3, the unsupervised algorithm seeks out similarities between pieces of data in order to characterize them, whereas the supervised algorithm builds a concise model of the distribution of class labels in terms of predictor features [84].

²⁷ A hybrid approach typically consists of two functional components. The first one takes raw data as input and generates intermediate results. The second one will then take the intermediate results as the input and produce the final results [84].

most of these studies have focused on classifying records as either normal or abnormal behaviour, detecting the type of attack strategy has not yet been investigated. Thus, knowing which strategy an attacker is using can provide an advantage for the security mechanism, for instance by using an attack-defence tree [79], or by optimizing the release order of algorithms [12, 13], which was previously discussed in Chapters 3, 4 and 5.

Hence, an attack strategy detection approach is proposed in this chapter. Based on the collected data in Chapter 3, each abnormal attempt of an attacker is detected using an unsupervised learning algorithm, which leads to the construction of the ALC. Since the ALC differs from one attack strategy to another, as demonstrated in the previous chapter, the following question is asked: *Can the applied attack strategy be detected quantitatively by using the accumulative manipulation of attackers?*

To explore this question, the previous experimental study's groups are divided into two sets: a training set, which is Group 1, and a testing set, which is Group 2. Therefore, the corresponding ALC of each attacker belonging to a training set for each strategy is generated. Then, a *Diagonal Linear Discriminant Analysis* (DLDA) classification method is applied to detect the strategy used by attackers belonging to the testing set. This detection mechanism achieved a detection success rate higher than 70% on experimental data. An early version of this proposed detection approach was published in [14].

The rest of this chapter is organised as follows. Section 7.1 outlines types of attack strategies, the applied detection approach and the workflow of this detection approach. Section 7.2 reports the experimental evaluation. The results of this evaluation are presented in Section 7.3. Section 7.4 presents the discussion. Finally, Section 7.5 summarises this chapter.

7.1 Strategy-Based Detection Approach: An overview

The underlying principle of the proposed attack strategy detection approach is that the *accumulative manipulation*, which was developed in the previous chapter, is characteristic for each observed strategy. In this section, a brief outline regarding the observed strategies, the methodology of the detection approach and the workflow of the detection approach are presented.

7.1.1 Types of Attack Strategies

Despite the fact that strategies observed in the previous experimental study have been provided in detail previously in Chapter 6, a brief outline of these strategies is given in this section in the interests of reminding the reader. Thus, the observed strategies are divided into three main strategies²⁸ as follows:

1. Thesaurus substitution, Perceptive substitution, Delete spaces.
2. Random addition, Thesaurus substitution, Perceptive substitution, Add spaces.
3. Perceptive substitution.

The first two strategies contain mixed types of strategies. In contrast, the third strategy contains only one strategy. The influence of these strategies on the performance of the attackers is obviously demonstrated by the proposed ALC, as shown in the previous chapter (Figure 6.12). Thus, the implication of this finding in relation to making features and heuristics leads to investigate a strategy detection method. In order to develop the detection approach, these features and heuristics can be used as a training set for a supervised learning machine that classifies the strategy used based on the training set. In light of this, a *Diagonal Linear Discriminant Analysis* (DLDA) is chosen to be a detection approach in this chapter for classifying the strategy used based on the *accumulated manipulation* notion that was developed from collected data in Chapter 3. The following subsection defines this chosen classification methodology.

7.1.2 Detection Approach: DLDA

Among many possible prediction techniques, a *Discriminant* analysis approach is utilised in this chapter due to several advantages such as the powerful but computer-intensive bootstrap methodology [95]. This advantage is now computationally feasible with the relatively easy access to high-speed computers.

There are different discriminant methods for classifying the data. These methods include traditional ones such as *Nearest Neighbours* and *Linear Discriminant Analysis*, as well as more modern ones such as *Classification Trees*. Since each observed strategy has small samples, *Diagonal Linear Discriminant Analysis* (DLDA) is applied to clas-

²⁸ The definition of each strategy is given in Chapter 6 (Section 6.3.1).

sify the strategy used, as a common technique for data classification. This method is a variation of *Linear Discriminant Analysis* (LDA), which is used to fit the linear combination of features that best separate two or more classes of object or event.

In DLDA, however, the common within-group covariance matrix is assumed to be diagonal. The resulting combinations may be used as a linear classifier, or more commonly in dimensionality reduction before later classification. This method is the simplest case of the maximum likelihood discriminant rule, in which the class densities are supposed to have the same diagonal covariance matrix. The most important advantage of the DLDA algorithm lies in its computational efficiency [95].

Furthermore, one of the features that motivated the choice of this method is that it does not require a large sample compared with others such as the *Quadratic Linear Discriminant Analysis* (QLDA) type. Finally, many researchers have pointed out that the naive Bayes classifier of high-dimensional data with small sample sizes sometimes known as DLDA. More details about DLDA can be found in [95].

Given the approach used for the detection mechanism, the subsection below describes the workflow of the proposed strategy detection approach starting from detecting an attack attempt to detecting the applied strategy.

7.1.3 The Workflow of the Detection Approach

This section explains the workflow of the proposed strategy detection approach as a preparation for the implementation stage of this approach, which will be discussed in the next section. The workflow is depicted in Figure 7.1.

As shown in Figure 7.1, the workflow is divided into two phases: Phase 1 and Phase 2. Phase 1, which was shown previously in Chapter 6 (Section 6.2), includes the detection of attack attempts that produce quantitative data (similarity-based) and the ALC that is represented by accumulative manipulation of each attempt with the corresponding strategy. Phase 2 starts with a training data set that includes the accumulative manipulation (i.e., Quantitative data) and the strategy (i.e., Qualitative data). Moreover, Phase 2 contains Attack Detection based on Qualitative data that is trained by both the training data set and the output of the proposed Attacker Learning Curve Model (ALCM), which was developed in the previous chapter (Section 6.4).

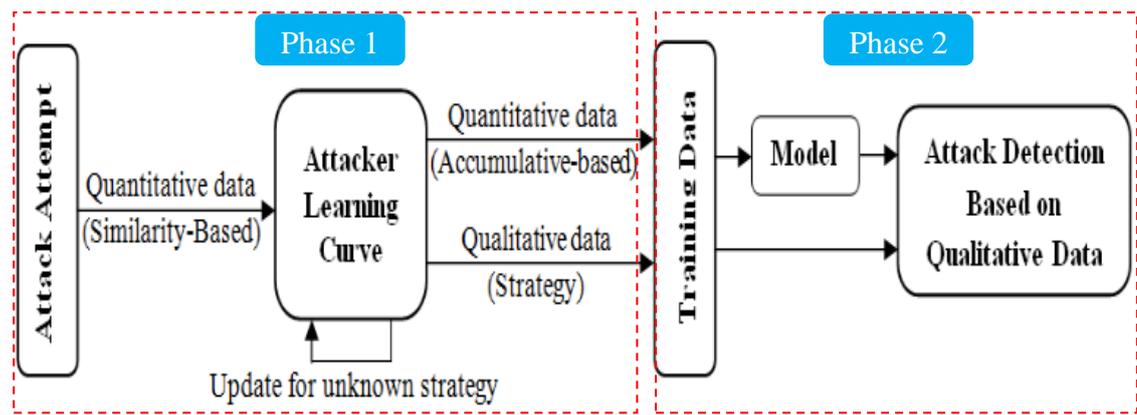


Figure 7.1: The workflow of the Detection Approach.

Each component of both phases is discussed as follows:

- *Attack Attempt*: Initially, each attack attempt is observed by using an unsupervised learning algorithm. Using a similarity evaluation function, it decides that an attempt is abnormal when the similarity between the attempt and a set of known abnormal attempts is higher than an empirically defined similarity threshold. Thus, the next step exploits this Similarity-Based quantitative data.
- *Attacker Learning Curve (ALC)*: The notion behind the *Learning Curve* is that accumulating experience leads to improved performance [21]. For the ALC, since breaking a defence algorithm typically requires several attempts by attackers, each attempt, which is signified by Similarity-Based data, is manipulated in order to avoid detection. Thus, the *accumulative manipulation*, which is the attacker's aggregated amount of knowledge, can gradually build the ALC that represents effectively how close an attacker is to breaking a defence algorithm.
- *Training Data*: A training set is a set of data used in different areas of information science to discover potentially predictive relationships. Since the proposed detection approach relies on a supervised learning machine, the accumulated manipulation data (Quantitative data) are associated with the corresponding strategy (Qualitative data) in this training set data, known from manually analyzing attacker attempts.
- *Attack Detection Based on Qualitative Data*: The proposed detection approach depends on a supervised machine learning that uses a DLDA type of

discriminant function. This supervised machine learning takes place using a training set that is prepared in the previous stage. The main purpose of this detection approach is to investigate the possibility of detecting the attack strategy used by means of a given accumulated manipulation. Therefore, a strategy could be detected unless it is not recognised in stage 2. In this case, the strategy is new and its accumulated manipulation should be updated in the ALC stage.

As the workflow of the detection approach is depicted and explained in this section, the next section presents an evaluation experiment that will not only show the detection results of ALC-based accumulative manipulation, but also the results of the ALCM-based accumulative manipulation.

7.2 Experimental Evaluation

In order to test the question of whether the strategy used in an attack can be detected based on the accumulative manipulation of the attacker, an evaluation experiment is conducted. The main question under investigation is:

- *Can the applied attack strategy be detected quantitatively based on the accumulative manipulation of attackers?*

This section presents an experimental evaluation of the above question by firstly describing the setup of and then the procedure of this experiment.

7.2.1 Experiment Setup

The experiment study reported in Chapter 3 involves subjects acting as potential attackers carrying out attacks on a test system, within which a number of different security algorithms have been developed. One aspect of the collected data is the similarity amount of each submitted e-mail. Since Groups 1 and 2 took a similar amount of time to break Algorithm 3 (A3), as stated in Chapter 3, the training set as well as the testing set is based upon the data collected from breaking A3. Therefore, the data collected from Group 1 for breaking A3 are used as a training set, while a test set utilises the data collected from Group 2 for breaking A3. Furthermore, the software and computing used to conduct the experiment are presented.

Training Set

The training set is divided into two main sets: ALC-Based and ALCM-Based. As such, the former is based on the accumulative manipulation of attacks accomplished by the experiment, whereas the latter is based on the accumulative manipulation carried out by the proposed model. For both sets, the training set is built on the average accumulative manipulation of Group 1 for each observed strategy. As mentioned in Chapter 6, 90% of the attackers (i.e. 36 out of 40) used structured strategies to defeat the algorithms. Specifically, in Group 1, there were 17 attackers who used structured strategies, and the distribution of the number of attackers on each observed strategy is shown in Table 7.1.

Table 7.1: The Number of Samples of each Strategy for Group 1.

Strategy	1	2	3	Total
Number of Samples	7	6	4	17/36

The average ALC-Based accumulative manipulation of each strategy for this training set is illustrated in Figure 7.2, while the average ALCM-Based accumulative manipulation of each strategy for the training set is shown in Figure 7.3.

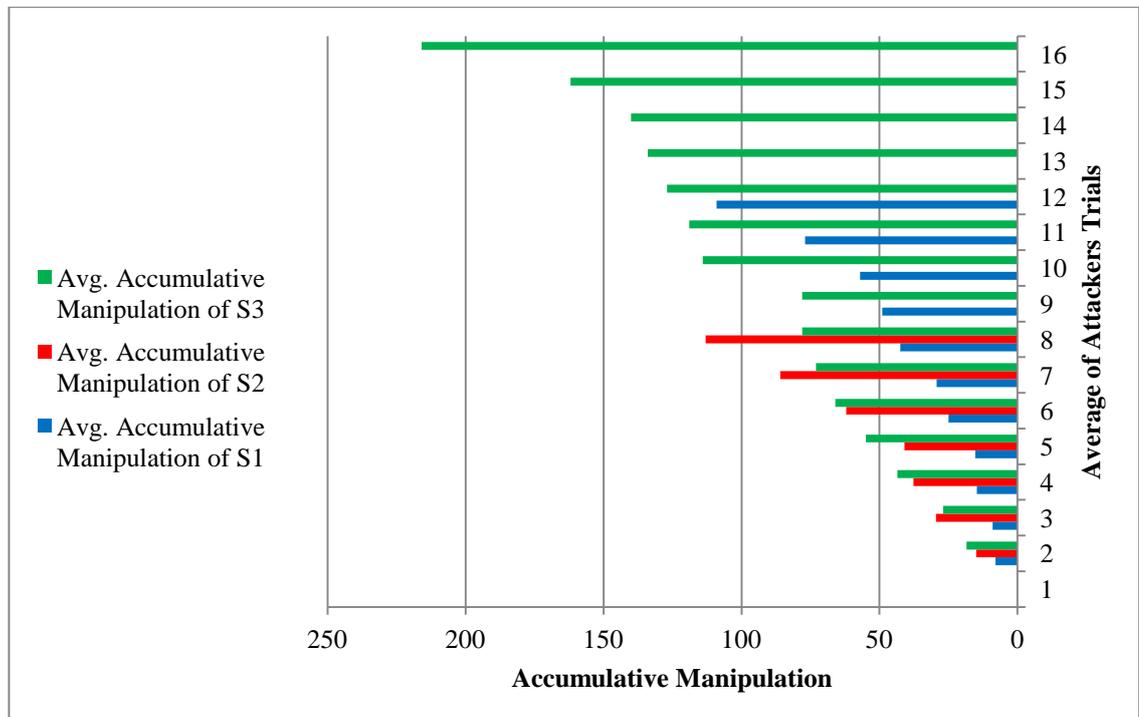


Figure 7.2: The average of ALC-Based accumulative manipulation for each strategy.

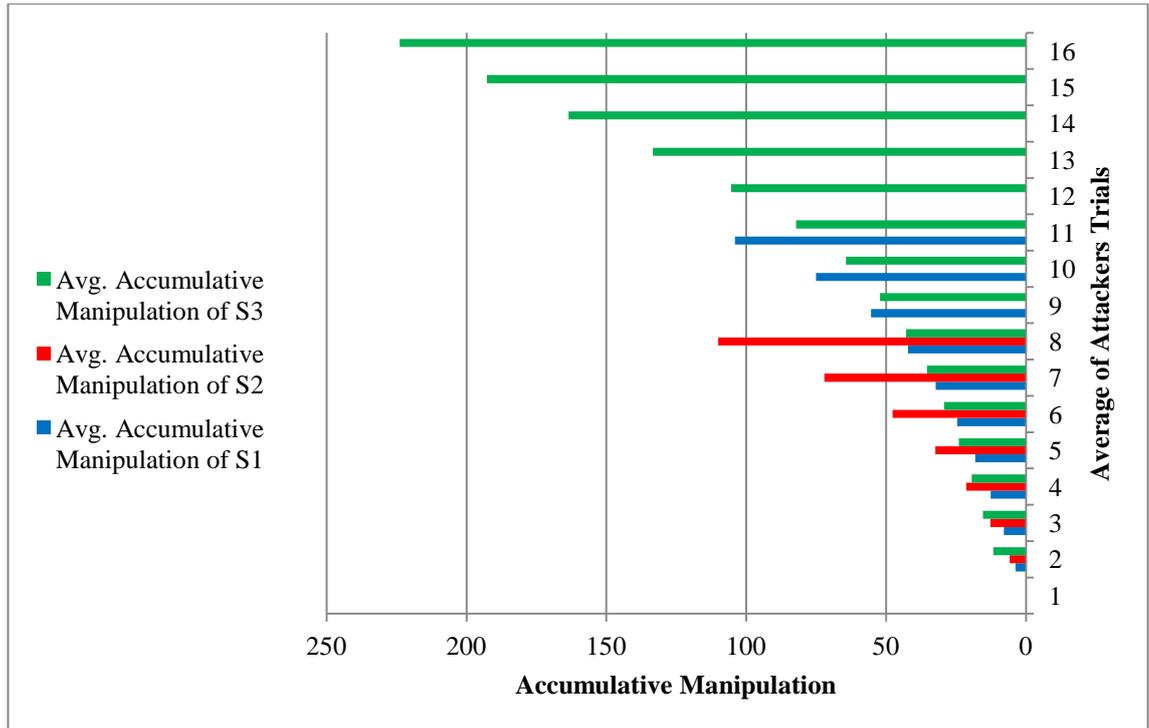


Figure 7.3: The average of ALCM-Based accumulative manipulation for each strategy.

Test Set

The test set is divided into two test sets: the ALC-Based and ALCM-Based test sets. For both sets, the accumulative manipulation of each attacker in Group 2 is used in the test set. Since the number of attackers who used structured strategies in Group 1 was 17, the number of attackers who used structured strategies in Group 2 was 19. It is important to note that even though the difference in attacker numbers in the groups is small, it seems useful to utilise Group 2 as a test set in terms of evaluating the effectiveness of the proposed detection approach. The number of attackers who employed each strategy for Group 2 is shown in Table 7.2.

Table 7.2: The Number of Samples of each Strategy for Group 2.

Strategy	1	2	3	Total
Number of Samples	8	5	6	19/36

The accumulative manipulation of each of the 19 attackers in Group 2 used in the test set is compared against both the ALC and the ALCM for the corresponding strategy built from Group 1. The ALC-Based accumulative manipulation of each attacker in

Group 2 is shown in Figure 7.4, whereas the ALCM-Based accumulative manipulation of each attacker in Group 2 is shown in Figure 7.5.

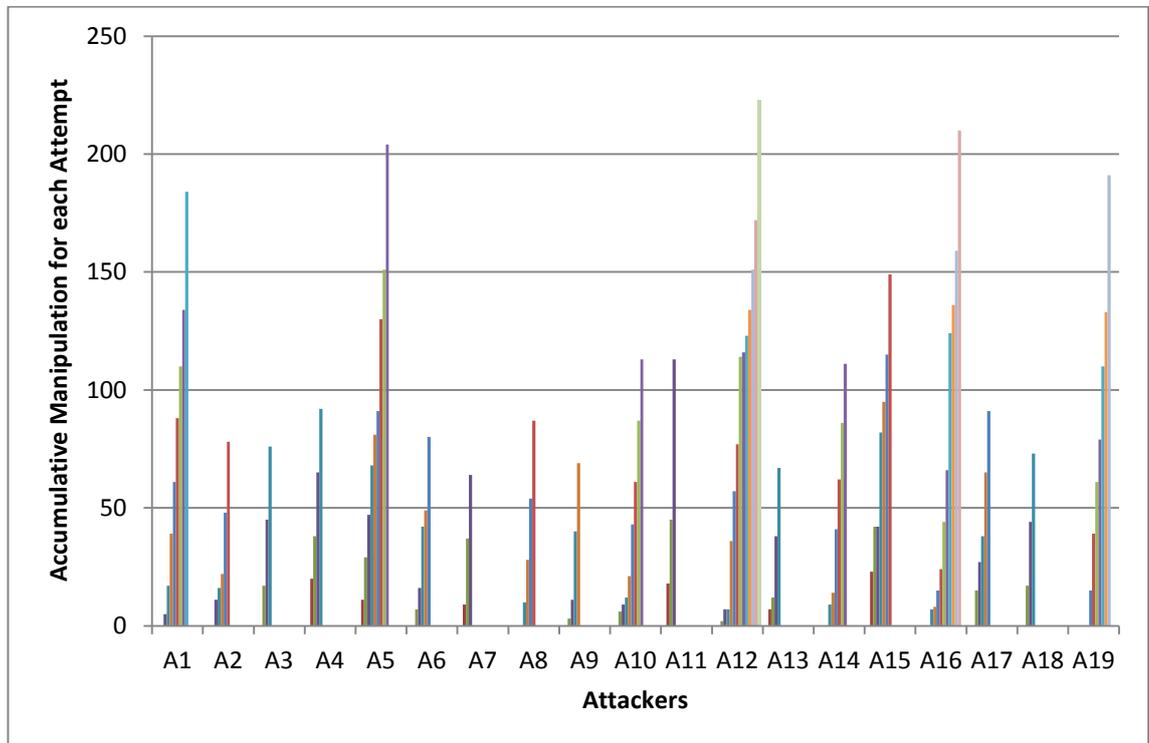


Figure 7.4: The ALC-Based accumulative manipulation of attacker’s attempts.

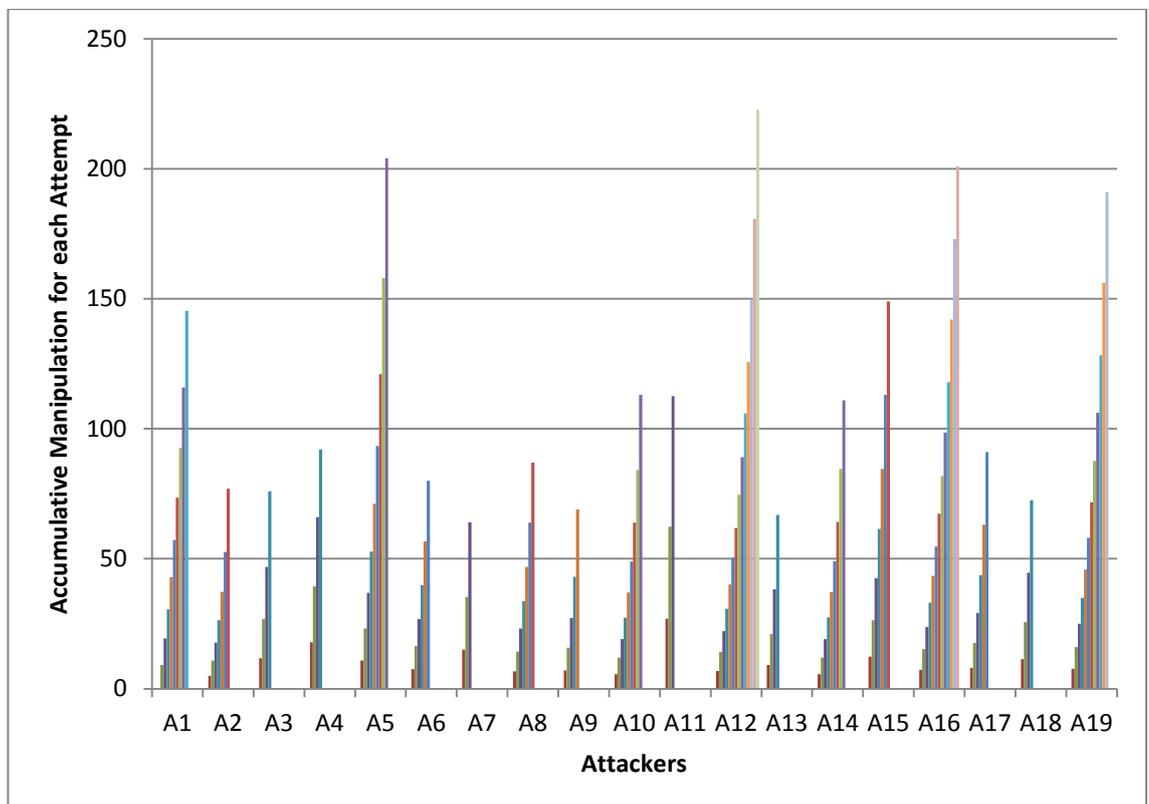


Figure 7.5: The ALCM-Based accumulative manipulation of attacker’s attempts.

Software and Computing

MATLAB²⁹ (version 7.6) is used as standard software for classification method realization, with a Statistics Toolbox [93]. This Statistics Toolbox provides statistical and machine learning algorithms and tools for organising, analysing and modelling data. Among these algorithms and tools, the “*classify*” function is selected, which performs the classification by using different types of Discriminant analysis. The syntax and description of this function will be highlighted in the following section. For the purposes of this experiment, the type “*Diaglinear*”, which was mentioned previously as *DLDA*, is used to carry out the classification results.

Classification Function Used

As stated above, the *classify* function is selected to perform the classification task. The syntax and the description of this function are as follows. The syntax of the *classify* function is [93]:

$$[class, POSTERIOR] = \text{classify}(sample, training, group, 'type')$$

The description of the above syntax is that *sample*, *training*, *group* and *type* refer to the *test set*, *training set*, *strategy* and *Diaglinear*, respectively. That is, this function classifies each row of the data in the test set into one of the strategies in the training set. The test and training sets should be matrices with the same number of columns. Strategy is a grouping variable for the training set. It is a unique value defined strategy; each element defines the strategy to which the corresponding row of the training set belongs. In this experiment, the strategy is 1, 2, and 3. The training set and strategy should have the same number of rows. The output class indicates the strategy to which each row of the test set has been assigned, and is of the same type as *group*. Furthermore, the output also returns a matrix *POSTERIOR* – estimates of the posterior probabilities that the j^{th} training set was the source of the i^{th} test set observation. The results of this matrix *POSTERIOR* are detailed in Section 7.3. In addition, more details regarding the *classify* function are in [93].

7.2.2 Experiment Procedure

In this section, the procedures involved in the experiment are explained. As such, the detection approach is trained separately by both training sets, ALC-Based and ALCM-

²⁹ MATLAP refers to **matrix laboratory** which is a high-level mathematical language [92].

Based accumulative manipulation, through the *classify* function. In other words, the ALC-Based accumulative manipulation is used with the corresponding strategy firstly to train the detection approach using the *classify* function. Meanwhile, the test set of ALC-Based accumulative manipulation, which comprises only the accumulative manipulation of each attacker, is fed to the *classify* function. Since this test set includes only the accumulative manipulation of each strategy, the detection approach classifies the predictability of each attacker's accumulative manipulation for a specific strategy. Once the results of this classification (i.e. detection) stage are achieved, the next stage of training for ALCM-Based accumulative manipulation is prepared. Secondly, the same procedure is followed for the training and test sets of ALCM-Based accumulative manipulation in the detection approach through the *classify* function.

7.3 Results of the Evaluation

In order to assess the proposed detection approach, the two ALC-Based and ALCM-Based test sets were successfully fed to the detection approach. In this section, the overall success detection rate and probabilities of the classification are presented. Note that, because the standard performance, including precision and recall, is beyond the scope of this thesis, it is not presented in this chapter.

7.3.1 Overall Rates of Success Detection

The overall detection rates on both the ALC-Based and ALCM-Based test sets are 73.68% and 68.42%, respectively. In other words, given the accumulative manipulation of an attacker from Group 2, there are 14 chances out of 19 to find the correct strategy of this attacker when training the detection approach on the average ALC-Based accumulative manipulation for each strategy in Group 1, as shown in Table 7.3. In contrast, there are 13 chances in 19 when training the detection approach on the average ALCM-Based accumulative manipulation built from Group 1, as shown in Table 7.3. Additionally, Table 7.3 shows the details of detecting each strategy.

It is important to point out that these detection results are based on the correct detection of each sample's trials until the first breaking attempt, since the attacker is asked to break each algorithm twice. As noted before in Chapter 6 (Section 6.2.3), the reason for asking the attackers to break an algorithm twice is to ascertain the learning process.

Table 7.3: The Results of Classifying each Strategy.

	ALC-Based test set				ALCM-Based test set			
	Strategy			Total	Strategy			Total
	1	2	3		1	2	3	
Number of Samples	8	5	6	19	8	5	6	19
Correct Classification	5	4	5	14	5	3	5	13
Incorrect Classification	3	1	1	5	3	2	1	6

It is apparent from this table that the DLDA can be an effective method for detecting the strategy of an attacker quantitatively. As shown in Table 7.3, strategies 2 and 3 were detected with a high success rate, while most of the undetected attackers were using strategy 1. Moreover, the detection results using the ALCM-Based test set are quite encouraging, and attacks using strategies 2 and 3 were detected with a high success rate, whereas most of the undetected attackers used strategy 1, as observed in the ALC-Based test set. More details regarding each attacker’s accumulative manipulation, classified results and labelled results for both ALC-Based and ALCM-Based test sets are presented in Appendix B.

Bearing in mind the overall success detection rate, the probability aspect of classifying each attempt with a specific strategy, which is provided by the classify function as shown in Section 7.2.1, can be useful in terms of tracking the classification process of an attacker’s attempt. Therefore, the following section sheds light on this aspect.

7.3.2 Probabilities of Classification

Interestingly, the results of the matrix *POSTERIOR*, which is a returned output with the detection results, can allow tracking of the classification process until the attacker breaks the algorithm. That is, a positive correlation was found between the applied strategy in the attack and the increase of the probability with each attempt. For example, Table 7.4 shows the successive probabilities of the classification for a subject using *strategy 3* (i.e. *attacker number 5* in Appendix C for ALC-based and ALCM-based test sets). From the data in Table 7.4, it is worth observing that the probability of classification using strategy 3 almost always increases both for the ALC and the ALCM, which tends to indicate that the confidence in the classification increases with the number of trials.

This observation might allow us to conjecture that if the number of trials were increased significantly, the accuracy of the detection would also increase. Not only this, but it might add an improvement to the proposed release order strategy by introducing a *Proactive Defence Approach*. That is, instead of waiting until an algorithm is broken to replace it, such a moment could be anticipated and a new algorithm deployed in time. However, due to the time limitation, this will be for future work.

Table 7.4: The probability of classifying one of the correctly classified samples.

Trial	Probability of classifying each strategy (ALC-Based)			Probability of classifying each strategy (ALCM-Based)		
	1	2	3	1	2	3
1	0.528	0.338	0.132	0.224	0.351	0.424
2	0.258	0.361	0.379	0.261	0.362	0.375
3	0.229	0.352	0.418	0.240	0.356	0.402
4	0.203	0.365	0.431	0.234	0.354	0.410
5	0.261	0.365	0.372	0.222	0.350	0.427
6	0.229	0.352	0.418	0.163	0.321	0.515
7	0.133	0.298	0.568	0.104	0.271	0.623

One the other hand, Table 7.5 shows the probabilities of the classification for a subject using *strategy 1* who was wrongly classified as using *strategy 2* (i.e. *attacker number 6* in Appendix C for the ALC and ALCM).

Table 7.5: The probability of classifying one of the incorrectly classified samples.

Trial	Probability of classifying each strategy (ALC-Based)			Probability of classifying each strategy (ALCM-Based)		
	1	2	3	1	2	3
1	0.528	0.338	0.132	0.528	0.338	0.132
2	0.528	0.338	0.132	0.335	0.372	0.292
3	0.362	0.375	0.261	0.282	0.366	0.351
4	0.371	0.379	0.248	0.350	0.372	0.277
5	0.371	0.386	0.242	0.270	0.365	0.364

From the data in Table 7.5 on the ALC-Based test set case, it is worth observing that the probability of the classification as strategy 1 is quite close to that of strategy 2, which could be interpreted as of rather low confidence in the final result. On the other hand, no

relative convergence was found in the ALCM-Based test set case compared to the ALC-Based test set, since the probability of strategy 1 is the lowest of all, as shown in Table 7.5. From the data in this table for the ALCM-Based test set, it can be seen that the approach is inaccurate, which underlines the fact that this approach might not be perfect.

More details on the probability of detecting each attempt using the predicted strategy are presented in Appendix C.

7.4 Discussion

The present study was designed to determine the effect of the developed ALC on the possibility of detecting the attack strategy used experimentally. The results of this experiment show that the correct detection rate on the testing set of the ALC-Based accumulative manipulation was 73.68%. Since this is, to the best of our knowledge, the first attempt to detect the attack strategy used, the accuracy obtained with this experiment indicates that the notion of accumulative manipulation, which forms the ALC, can be successfully used as an input feature for a supervised detection algorithm.

It is interesting to note that although the results of the matrix of *POSTERIOR* could allow tracking by the detection approach, the percentage of incorrect classification was 31.58% (i.e., 6 out of 19 attackers). However, a possible explanation for this might be that overlapping between the types of strategies used could affect the accumulative manipulation produced. Accordingly, this may cause a misclassification in the detection approach, as the detection approach relies essentially on the ALC, which is formed by the accumulative manipulation of each attacker. For instance, Table 7.5 shows the probabilities of the classification for a subject that was using *strategy 1*, and was wrongly classified as using *strategy 2*. Since there is an overlap between two types of strategies: *Thesaurus substitution* and *Perceptive substitution*, this may cause misclassification.

Hence, a security mechanism that can monitor and log the manipulation performed by an attacker can leverage that information to detect the strategy used by the attacker. Such knowledge can be particularly useful in adapting the defensive mechanism to that particular attacker, in terms of efficiency (e.g., deploying the best way to block that attacker) or cost (e.g., only deploying countermeasures for that particular kind of attacker). Furthermore, this finding has important implications for developing an ap-

proach that considers the evolution of the classification instead of the final classification only.

In the current study, the results obtained from the ALCM-Based test set show that a learning curve model can be utilised to detect the strategy used by an attacker. Although this approach is less accurate compared to the ALC-Based test set, it does not require training the classifier with effective previous attempts, but can directly build the learning curves from the possible attacker skill levels, the strategies used and the robustness of the algorithms, as noted previously in Chapter 6.

There are several limitations to the work presented here in this chapter. Firstly, the number of participants for each strategy is quite low. However, this work is considered as proof of the concept, showing that using accumulative manipulation makes sense in some contexts. Clearly, further work is required in order to understand which contexts are suitable and which are not. Secondly, only attempts that are known to be attacks are used. In other words, the data are not cluttered with data coming from normal usage. In a practical setting, it would probably be necessary to first detect whether a particular user is attacking the system, and only then try to detect which strategy is being employed. Finally, in the design of the experiment, the attackers do not care about being detected or not, whereas an actual inside-attacker would try to hide as much as possible. It is however difficult to design an experiment that can cover all possible kinds of attack, and the goal of the work presented here is not to provide a tool ready to use in any possible context, but rather to identify the features that can be useful when using machine learning in the context of security.

Indeed, the results presented in this chapter corroborate the ideas of Liao et al., who pointed out in [84] the need for a better understanding of the different types of features and heuristics for specific goals in network intrusion detection. In other words, selecting and understanding an effective set of features is a challenging and labour-intensive task. Therefore, the results in this chapter have identified a proper set of features for the detection of attack strategies.

These findings will doubtless be much scrutinized, but there are some immediately dependable conclusions for the likelihood of detecting the applied attack strategy quantitatively.

7.5 Summary

This chapter has proposed an attack strategy detection approach using the original concept of an attacker accumulative manipulation developed in the previous chapter, which is abstracted as the Attacker Learning Curve (ALC). Based on the results of the developed ALC and its model, the proposed detection approach is evaluated. The result of this evaluation shows that the overall detection success rate is higher than 70%. Returning to the question posed at the beginning of this chapter: “*Can the applied attack strategy be detected quantitatively by using the accumulative manipulation of attackers?*”, it is now possible to state that the attack strategy applied can be detected quantitatively. Moreover, the empirical findings in this study provide a new understanding of not only detecting the attack strategy used quantitatively, but also tracking the attack strategy used through the probabilities of the classification.

The findings in this chapter may be subject to at least two limitations. First, data used in the experiment are not cluttered with data coming from normal usage because the attempts are known to be attacks. In other words, it would probably be necessary to first detect whether a particular user is attacking the system, and only then to try to detect which strategy is being adopted. Furthermore, a limitation of this study may be that the number of samples in each strategy was relatively small. However, we consider this work as a proof of concept, showing that using accumulative manipulation makes sense in some contexts.

In the next chapter, a summary of the research contributions and several potential future works are presented.

Chapter 8. CONCLUSION AND FUTURE WORK

This chapter concludes the thesis with a discussion of the findings of this research and with a future work. In particular, this thesis has given an account of and the reasons for interactive defensive mechanisms from the perspective of the knowledge acquisition process of attackers and proposed a set of algorithms approach, of which the rationale was to prolong the protection of a system as far as possible. A cornerstone of this thesis was the investigation into whether the order of releasing a set of defensive algorithms has an effect on the time taken to break all algorithms or not. Through an experimental study, it was possible to demonstrate that the order in which defensive algorithms are released does indeed influence the time attacks take. Based on the empirical results achieved by this experiment, a Stochastic Petri Net model was developed, which can describe the interaction between an attacker and a set of algorithms, in order to estimate the time required to defeat a defensive algorithm with various algorithm orders. Furthermore, an optimisation algorithm was proposed to obtain efficiently the optimal release strategy by using a Markov Decision Process model. To contribute an advantage for the interactive defensive mechanisms, a detection of attack strategy approach, which relies on the developed Attacker Learning Curve (ALC) notion, was proposed and evaluated.

The remainder of this chapter is organised as follows. Section 8.1 outlines the contributions made by this thesis. Section 8.2 then provides reflections on the research conducted in this thesis in order to answer the research questions. Finally, Section 8.3 offers a discussion on potential future works which can be derived from the research works conducted in this thesis.

8.1 Summary of Contributions

The thesis has made several contributions as follows:

- **A Classification scheme of defensive mechanisms (Addressed in Chapter 1).** Based on the insight into the confusion matrix in for instance in machine learning, a classification scheme that consists of different dimensions such as assertive and predictive defensive mechanisms has been proposed. Furthermore, the interactive and non-interactive defensive mechanisms are introduced under the predictive defensive mechanism umbrella. To our knowledge, the proposed classification scheme has not been introduced by other researchers. The value of such classification is in providing interested parties such as researchers, defensive mechanism designers and developers with a tool to accurately classify defensive mechanisms. Moreover, this classification has allowed the identification of a correlation between a defensive mechanism, such as an interactive defensive mechanism, and other possible factors such as the knowledge acquisition process of attackers. This provides a consistent and clear understanding of the problem of interactive defensive mechanisms.
- **A novel experimental study for evaluating the proposed approach (Addressed in Chapter 3).** One of the challenges in this thesis was designing an experiment in order to evaluate whether the release order of a set of defensive algorithms matters. More precisely, the dilemma was to design a system that could be reached by non-specialists in a matter of minutes, since the rationale behind the proposed approach was that attackers learn from their attempts. Hence, we decided a spam-filter would offer a very good model for the experimental requirements. Based on a chosen content-based spam filter, several simplified but representative algorithms were developed. Accordingly, a web-based system was developed as well, which allows the participants to interact with the algorithms of the spam filter in order to break them. Using this developed system, the evaluation of the release order of defensive algorithms in terms of time taken to break them was carried out. Not only this, but also the attackers' learning progress was observed and analysed quantitatively.
- **A model for the release order of a set of defensive algorithms (Addressed in Chapter 4).** The proposed model represents a generic application level blueprint

for the underlying principle of the developed experimental study. A Stochastic Petri Net model was used to construct the proposed model. As such, this model could allow for a theoretical analysis of the release order of a set of algorithms, and for a better estimation of the time required to break a defensive algorithm with various algorithm orders. This approach is unique and important since there has been no such attempt to provide a model that addresses the issue of the release order of the defensive algorithms. Moreover, this model could be a valuable tool to interested parties such as interactive defensive algorithm designers and developers of interactive defensive algorithms.

- **An optimisation algorithm to obtain the optimal release strategies (Addressed in Chapter 5).** Based on the empirical results, which are presented in Chapter 3 and demonstrate that the release order of defensive algorithms has a statistically significant impact on the time attackers take to break all algorithms, an optimisation algorithm has been proposed. The metric of interest (and, hence, the optimisation criterion) in this algorithm was to maximise the time it takes to break a set of algorithms. The approach to the proposed optimisation algorithm was to mathematically model the optimisation problem, and to present a bespoke and efficient solution algorithm that derives the optimal release strategy for any model. The mathematical model used on this algorithm was the Markov Decision Process model, with a specific state space that is utilised to derive the efficient optimisation algorithm. To the best of our knowledge, this is the first to address this particular issue of optimising the release strategy to delay a successful attack success for as long as possible.
- **An approach to demonstrate an attacker's progress (Addressed in Chapter 6).** Since the feedback achieved from a system while an attacker attempts to break it plays an important role in gradually weakening the security level of interactive defensive algorithms, a quantitative approach to show the effectiveness of this feedback from the attacker perspective has been proposed. This approach is the *accumulative manipulation* amount of an attacker's attempts that led to developing the Attacker Learning Curve (ALC) concept. The value of this concept is not only in demonstrating the performance of an attacker during the attack process, but also in distinguishing the applied strategy in the attack. Furthermore, the ALC concept represents the importance of the feedback to the at-

tacker in terms of disclosing rules of the defensive algorithms. Therefore, this concept can reduce the gap of knowledge by showing quantitatively both the progress level of an attacker and the strategy used in an attack.

- **A mechanism to detect the attack strategies (Addressed in Chapter 7).** Based on the ALC concept developed, a mechanism to detect attack strategies has been proposed. This mechanism exploits the features and heuristics that can be provided by the proposed *accumulative manipulation*, which forms the ALC concept, in order to detect attack strategies. The performance of the proposed mechanism shows the practicality and efficiency of this novel detection mechanism. The value of this detection mechanism is in providing an advantage for the security mechanism, for instance by using an attack-defence tree, or even by optimising the release order of algorithms, which has been proposed in Chapter 5.

8.2 Reflections on Research Outcomes

This section provides reflections on the research conducted in this thesis in light of the research questions addressed in Section 1.2. In particular, each research question is answered first, and then a reflection on the overall thesis is provided.

8.2.1 The first research question

The first research question was as follows: *Does the order in which different defensive mechanisms are released impact the time an attacker needs to break each one of them?*

This is indeed a practical question to investigate, particularly when a set of algorithms scheme proposed is new to the defensive mechanism realm. The majority of interactive defensive mechanisms can be considered from a qualitative point of view by releasing a single defensive mechanism. In practice, using an interactive defensive mechanism, such as a CAPTCHA or spam filter, the attacker and defender exchange ‘blows’, each celebrating (temporary) success in breaking and defending. However, the feedback given by the system during the attack process by attackers allows for the gradual disclosure of the rules included in the defensive algorithms over time. The issue of feedback seems evident in the interactive defensive mechanisms, as discussed in Chapter 1. Therefore, this has led to propose a set of defensive mechanisms approach in order to prolong the time needed to break a system. Intuitively, releasing a set of algorithms one by one se-

quentially extends the required time to break a system, rather than releasing only one algorithm.

In order to evaluate whether the order will affect the security level of a system in terms of maximizing the time taken to break all algorithms, a controlled laboratory experiment study was conducted. The results of this experimental study revealed that the order in which interactive defensive mechanisms are released has a statistically significant impact on the time attackers take to break all algorithms (Figure 3.4, Tables 3.3 and 3.7). It is important to note that the effect of the presentation order on the learning mechanism is not new in the fields of education and psychology. As such, previous research provides several insights and experiments into the effect of presentation order [48, 91, 98, 131]. However, to the best of our knowledge, this is the first experiment to address this particular issue of the release order strategy in the security field. This allows the question to be answered as follows:

“The order in which different defensive mechanisms are released can impact the time an attacker needs to break each one of them.”

8.2.2 The second research question

The second research question was as follows: *Could we optimize the order in which defensive mechanisms are released?*

The release order of defensive mechanisms has indeed influenced the time attackers take to break them, as answered in question 1. Therefore, optimizing the release order of defensive mechanisms is a problem worthy of study. The aim of this optimization is to make a system as effective as possible in terms of maximizing the time taken by attackers to break the system. Thus, we have provided in Chapter 5 a tailored optimization algorithm using a Markov Decision Process to obtain efficiently the optimal release strategies for any given model. Moreover, the proposed model solution should scale without problems to optimize the release order of tens of defensive mechanisms. This allows the question to be answered as follows:

“We could optimize the order in which defensive mechanisms are released using a Markov Decision Process.”

8.2.3 The third research question

The third research question was as follows: *How does dependency between algorithms impact on ability to answer question 2?*

The proposed set of defensive algorithms approach leads to an investigation of not only independent and dependent defensive algorithms, but also of the defensive algorithm of which these algorithms were originally a part. Since Algorithm 1 was a simplified version of Algorithm 2 in the controlled experimental study that was carried out in Chapter 3, the results of this study showed that the success of attacks can be delayed (i.e. extending the time of attack) by breaking up an algorithm into parts, when these parts are released in a specific order. This specific order was determined by the results of the experiment, in which a subset defensive algorithm was released before the superset defensive algorithm, as shown in the order of Group 1. On the other hand, the success of the attack could be also delayed by breaking up an algorithm into parts when the parts are released in the reverse order, as shown in the order of Group 2, but not as much as the order of Group 1. Thus, it is an interesting insight that implies the intuitive reasoning that by breaking up a defensive algorithm into parts the attacker is not ‘taught’ how to attack.

Furthermore, in the setup of the controlled experimental study, Algorithm 3 was a relatively independent (i.e. non-subset) defensive algorithm using a quite different defensive approach to that of Algorithms 1 and 2 was applied. In light of this, the concatenation of Algorithm 3 at the end of the release order of Group 1 and Group 2 yielded interesting and important results. These results showed that, despite the knowledge gain at any point of the release chain, injecting a non-subset algorithm would force the attacker back to the learning phase. More importantly, the time taken to break Algorithm 3 in both groups was equal. This finding has important implications for developing an optimization algorithm for the order release strategy, which has been accomplished by means of a Markov Decision Process model, as reported in Chapter 5. This allows the question to be answered as follows:

“It could be useful to break up a defensive algorithm into multiple algorithms, and release them one by one if the order of these multiple algorithms is that the subset defensive algorithm is released before the superset defensive algorithm.” Also:

“The time taken to defeat a future independent defensive algorithm does not depend on the order in which earlier algorithms were broken.”

8.2.4 The fourth research question

The fourth research question was as follows: *Could we model the learning acquisition process of attackers?*

Since an exploration was carried out to find out the answer to all of the previous questions, the common denominator between them was the attacker’s aggregated amount of knowledge. Using quantitative data that collected from the experiment, an *accumulative manipulation* notion that represents effectively how close an attacker is to breaking a defensive algorithm was developed. As noticed previously, this can be observed through a number of attempts accomplished by the attackers, which forms the basis of the ALC (i.e. the attacker’s performance by accumulative experience). Using this approach, it was possible to not only demonstrate the attacker’s performance, but also to distinguish between the attack strategies applied by means of different amounts of effective manipulation for each attacker’s attempts.

Additionally, we also developed an Attacker Learning Curve Model (ALCM) that is inspired by a previous model that is utilised for explaining a developer’s learning curve, as presented in Chapter 6. The evaluation of the ALCM suggests the applicability of using it. Although the knowledge acquisition process of an attacker presented in Chapter 4 derived through the developed model, as shown in Figure 4.10, this derivation of the attacker knowledge acquisition process did not take into account the strategy applied in the attacking process. Therefore, the ALCM seems a typical model to estimate the learning curve of an attacker to break an algorithm, due to its recognition of the strategy applied in the attacking process. This allows the question to be answered as follows:

“We could empirically model the learning acquisition process of attackers.”

8.2.5 The fifth research question

The fifth research question was as follows: *Based on understanding the learning acquisition, can we devise an attacker detection approach?*

The results of the ALC have attracting an interest by us to devise an attacker detection approach that can add an enhance security level to the release order strategies. Thus, a

detection of attack strategies approach has investigated and evaluated empirically based on the data of ALC that collected from the controlled experimental study. The empirical findings from this method provide an additional advantage with respect to supporting the proposed approach for prolonging the time taken to break a set of defensive mechanisms, that is, by considering the evolution of the attacker's performance. This allows the question to be answered as follows:

“A detection methodology for attack strategies has been developed which could be used to prolong the time taken to break a system as far as possible.”

8.2.6 Overall Reflection

This section gives an overall opinion regarding the viability of the proposed methodology.

The methodology proposed in this thesis starts from a set of defensive mechanisms as a holistic defensive approach and maps it to the release order strategy. Automated approaches to breaking defensive mechanisms, such as bots, were purposely not used, as they are beyond the scope of this research. The reason for this is that any automated approach would need to know the parameters to try, and the range in which these parameters may fall. Instead, the significant challenges in answering the aforementioned questions should be stressed, in terms of designing a representative experiment, system implementation of the experiment, and conducting, and analyzing the experiment. In particular, as it is assumed that an attacker is human, the problem of human learning would be seen clearly by sending e-mails to evade a content-based spam filter, as it requires a low degree of technical proficiency (i.e. it is possible to show the objective to non-specialists people). Consequently, a content-based spam filter was chosen because it allows some understanding of a human learning process, as automated approaches are abstractions of this human learning process that require encoding by humans.

It is significant to note that the testing of the two conditions, in which the defenses are overlapping (i.e. Algorithm 1 and Algorithm 2), was necessary to build a solid hypothesis before further experiments were conducted. It is hard to predict how our brains process knowledge and, hence, even the trivial assumptions should be tested to avoid surprises. In addition, adding Algorithm 3 at the end of the two experimental conditions is harmless to the integrity of the original experiment results, which pertained to the re-

lease order. Rather, the addition of Algorithm 3 has provided an insight into the effect of non-wholly overlapping algorithms on overlapping algorithms.

Furthermore, an abstract model of human learning was considered as a starting point. It is assumed that a simple feedback loop exists, in which a human acts and then learns, then acts, and so on. More complex learning processes would certainly be interesting to investigate, but it is believed that the approach in this thesis provides a valid starting point. Accordingly, an accumulative manipulation of an attacker notion could be developed that effectively represents the learning curve of the attacker. Moreover, this notion allows a distinction between the applied attack strategies to be measured quantitatively. Based on the features and heuristics that are provided by this notion, a detection of attack strategies approach is proposed and evaluated. Although a number of important limitations in the evaluation of this proposed detection approach need to be considered, the aim behind this approach is not to provide a tool ready to use in any possible context, but rather to identify the features that can be useful when using machine learning in the context of security.

This research is exploratory in nature, not directly aiming to support or refute any existing theories or practice. However, it does open up a new platform for more research to be conducted in the near future. This research also does not attempt to provide an ultimate solution to the existing issues, but instead opens possible avenues to be investigated. This research does not provide a silver bullet to an issue but instead creates more opportunity for interested parties to collaborate in an effort to improve the existing proposed countermeasures.

8.2.7 Applicability to Other Security Scenarios

This section speculates how the concepts that are presented in this thesis would relate to other security scenarios e.g. CAPTCHA. In general, due to the utilization of a set of algorithms rather than a single algorithm as a defensive mechanism, it is assumed that the algorithm with more security rules than another is already known. For example, if Algorithm 1 has a, b and c rules and Algorithm 2 has the same rules plus an advanced one i.e. a, b, c and d, that is, if the similarity between the proposed defensive algorithms is known, then their deployment can be ordered appropriately. As a result, the optimization order increased the time needed to break the defensive algorithm from 16.2 minutes, in the case of applying Algorithm 2 *alone*, to 25 minutes in the case of applying

the optimized set of defensive algorithms (i.e. releasing Algorithm 1, which was the subset, took 10.9 minutes, then releasing Algorithm 2, which was the superset, took 14.1 minutes). By generalizing the optimization approach, it could be applied to any security system that currently depends on a single defensive algorithm. It is important to emphasise that the objective of this research is to show the validity of the claim that the release order of defensive algorithms matters. That in itself is challenging. Showing that it holds true for another system, such as a CAPTCHA, is an additional and difficult question beyond the scope of this thesis.

However, this section describes how the experiment would be different and how differences between algorithms and attackers would be classified in case, for instance, CAPTCHA. Therefore, the following highlights the algorithms, attackers and the experiment.

Algorithms

It could be possible to generate three types of CAPTCHAs: overlapping rules, non-overlapping rules or mixed. For the first type, the rules of generating such text-based CAPTCHAs have, for example, the following characteristics:

- Eight characters are used in each sample;
- Only upper case letters and digits are used, and
- Foreground (i.e. sample text) is dark red whereas background is light gray.

While such samples have, in addition to the previous characteristics, the following characteristics:

- Warping (both local and global) is used for character distortion;
- Nine and seven characters are used in each sample;
- Small case letters are used, and
- Foreground is blue, green and black whereas background is white.

Thus, based on the characteristics, a generator can produce a set of overlapping CAPTCHAs with a view to not only extend the time taken to break the system, but also to break up a CAPTCHA into parts.

For the second type, the rule of generating the samples are based on text-based scheme as shown for instance in Figure 2.2 and image-based scheme as shown for example in Figure 2.3. Thus, a generator can produce a set of non-overlapping CAPTCHAs in order to prolong the time to break the system as this is the cornerstone of this thesis.

Finally, for the mixed type, it can generate for example overlapping text-based schemes and image-based scheme which represents non-overlapping rules scheme. Furthermore, it can also generate text-based scheme by generating independent text-based schemes.

Attackers

According to the state of the art of CPATCHAs, all attackers against CAPTCHAs are automated programs (i.e. bots) [100, 150, 32]. The automated programs are abstracted of humans learning process that require encoding by human. Therefore, any automated programs would need to know the parameters to try, and the range within which these parameters can fall. Depending on the targeted CAPTCHA scheme, the automated program should be encoded by a set of heuristics that are observed by human after acquiring such feedbacks³⁰.

For example, an automated program can be encoded by the following heuristics that control the movement of the automated program in order to break a text-based sample scheme:

- Whenever feasible, an automated program moves down vertically as much as possible. As such, down movement is the direction that has the highest priority.
- The automated program moves down from its starting point until it is immediately above a foreground pixel.
- When the automated program moves left and up only, it move left one pixel, and then moves down as much as possible.
- When the automated program moves right and up, it moves right one pixel, and then moves down as much as possible.
- A vertical slicing line could be a legitimate segmentation line.
- Distance control, when the automated program reaches the bottom line, it is done. On the other hand, when the automated program cannot reach the bottom, it is aborted and all its trace is deleted.

³⁰ The feedback concept has been explained in Chapter 1 (Section 1.1)

Once the automated done these heuristics successfully, a recognition approach can be used with a view to identify a specific letter of the targeted CAPTCHA. It is important to note that the automated program that is utilised to break a set of CAPTCHAs needs to be learned by sophisticated heuristics, since all of attacks, to the best of our knowledge, are developed against a system that has only a single algorithm to generate CAPTCHAs.

Experiment

The common methodology that is used in order to observe and analyse CAPTCHA schemes follows the practical in the fields such as computer vision and machine learning. In particular, an attack is built on observing and analysing a random number of samples. These samples are called a “Sample set”. To show the effectiveness of a developed attack, a large “Test set” of random samples are tested by the developed attack, which is no prior knowledge about any sample in this set.

So, the time taken by the attacker to analyse the released schemes and testing the attack on the observed schemes could be calculated. Since the time was playing an important role in the previous experiment with a view to demonstrate the impact of the release order of a set of algorithms on the time taken to break them, this impact could be shown to CAPTCHAs. Furthermore, this could be shown also to other security scenarios.

Moreover, as we mentioned previously, the proposed optimisation algorithm is increased the time needed to break the defensive algorithms. By generalising this optimisation algorithm, it could be applied to any security system that currently depends on a single defensive algorithm. The next section provides some discussion on possible future work relevant to this research.

8.3 Future Work

The research work presented in this thesis provides a basis for a number of potential related future works as follows:

- It would be useful to conduct a controlled experimental study in which the defensive algorithms are not wholly overlapping and in fact are qualitatively different, and in which defensive algorithms are returned to be released again rather than removed (i.e. not released again) when the next defensive

algorithm is deployed. This would provide an insight into the effectiveness of the knowledge gained on the attacker's performance when the broken algorithm is released after breaking a different one. Although it might be inconvenient to re-release a broken algorithm against attacks, it would be interesting to investigate the amount of remaining knowledge regarding this algorithm after breaking a relatively different one.

- It would be worth investigating a proactive defensive approach based on either related algorithms (i.e. algorithms which have overlapping rules) or non-related algorithms. The idea behind this approach is that the released algorithm is replaced with another one before it is broken, based on the proposed optimization algorithm which applies a set of related algorithms, while the proposed optimization algorithm might not be applied in the case where a set of non-related algorithms is used. This would provide an insight into the length of the time needed to break an algorithm with knowledge gained intermittently.
- It would be interesting to investigate whether we can extend the developed Petri Net model, which is in Chapter 4, in order to prolong the time needed to break a system. That is, based on the probability of attacker's knowledge evolution to break the released algorithm, it could replace the released algorithm by another algorithm from the pool of algorithms before breaking the released one.
- It would be of value to conduct more research to study further the accuracy of the proposed detection attack strategy approach before a defensive algorithm's rules are broken and to understand how to establish better confidence in the classification results, for instance by considering the evolution of the classification instead of the final classification only.
- It would be interesting to develop an approach that considers qualitative reasoning, in light of what has been considered here in this research, based on formal logic, to model the knowledge gained by the attackers by combining both a qualitative and quantitative approach. This would provide an opportunity to get more precise detection results.

- Since the proposed detection of attack strategies was useful for insider attacks particularly, and evaluated based on attempts that are known to be attacks, it would be interesting to first detect whether a particular user is attacking the system, and only then to try to detect which strategy is being used by the attacker. This could be carried out in a controlled experimental study in which random attackers are involved among ordinary users.
- As the main goal of the detection of attack strategies presented here in this thesis is to identify the features that can be useful when using machine learning in the context of security, it would also be interesting to try the proposed detection approach in a different context in which, for instance, an attacker types commands into a terminal looking for misconfiguration.

APPENDIX A: EXPERIMENT MATERIALS

Contents

- Participants Recruiting Email
- Screenshots for the System Website
 - o Registration Page
 - o Participant Consent Form Page
 - o Experiment instructions
 - o An Example to an attacker's attempts to break the system
 - o Survey Page
 - o Logout Page

Participants Recruiting Email

Hello,

My name is Suliman and I am PhD student at School of Computing Science. As part of my research work, I am planning to evaluate a novel defence mechanism by conducting an experiment. Therefore, I need participants to take place in this experiment. The experiment takes only about 30-40 minutes.

The experiment will be as a game, the higher score you get, the higher reward you will achieve. Specifically, the first winner will get £40 and the second winner will get £20. Additionally, just for the participation, each participant will get £5.

The experiment would take place at Cybercrime lab, room 702, Claremont Tower on 13th April to 17th April 2012. I will be there from 9:00am till 6:30pm and you can choose to come between these hours to try out the experiment at any date of the above dates.

If you are interested in helping out, please e-mail me at (suliman.alsuhibany@ncl.ac.uk) and I will allocate a slot for you.

For any further information, please do not hesitate to contact me.

Thank you and I am waiting for your participation in this experiment.

Suliman Alsuhibany
Cybercrime lab
Room 702, Claremont Tower
School of Computing Science
Newcastle University
suliman.alsuhibany@ncl.ac.uk

Screenshots for the System Website - Participant Consent Form Page (1)

8/29/13
Register to the SpamDefender

Main
Register
Login



Register Now !

SCHOOL OF COMPUTING SCIENCE FACULTY OF SCIENCE, AGRICULTURAL & ENGINEERING

NEWCASTLE UNIVERSITY

Participant Consent Form

Name of Study:

An experiment to investigate whether the order in which security algorithms are deployed in a computer system has a significant effect on the security of that system.

Purpose:

The purpose of this study is to investigate the participants' performance and the behaviour in sending emails as an attacker. This study would also investigate the learning ability of the participant.

Procedure:

- a. Login the system and fill demographic information.
- b. A brief introduction about the experiment.
- c. Doing the session that includes *three* algorithms and s/he is asked to send more than 300 emails to pass each algorithm. For passing each level, some techniques should be followed to modify the email text.
- d. Fill a short overview about the session.
- e. The total time required to complete the study should be approximately 30 - 40 minutes.

Benefits/Risks to Participant:

Participants will be informed about the importance of the proposed approach in terms of security. Further, they will be motivated to discover the parts that the algorithms are checking for detecting the spam email.

Voluntary Nature of the Study/Confidentiality:

Your participation in this study is entirely voluntary and you may refuse to complete the study at any point during the experiment, or refuse to complete any task which you are uncomfortable. You may also stop at any time and ask the researcher any questions you may have. Your student number will never be connected to your results instead; it will only be used for providing compensation for your participation. We will use serial numbers instead, for identification purposes. Information that would make it possible to identify you or any other participants will never be included in any sort of report. The data will be accessible only to those working on the project.

Contacts and Questions:

At this time you may ask any questions you may have regarding this study. If you have questions later, you may contact the person conducting the study, Suliman Alsuhibany and Ahmad Alonaizi via email at suliman.alsuhibany@ncl.ac.uk and a.alonaizi@ncl.ac.uk respectively. Questions or concerns about institutional approval should be directed to Ms Jo Mayne, Deputy Head of Administration at the Faculty of Science, Agricultural & Engineering, Newcastle University via email joanne.mayne@ncl.ac.uk or call her at 0191 222 5923.

localhost:8080/SpamDefender/pages/signup.jsp
1/3

Screenshots for the System Website - Participant Consent Form Page (2)

8/29/13 Register to the SpamDefender

Statement of Consent:

I have read the above information. I have asked any questions I had regarding the experimental procedure and they have been answered to my satisfaction. I consent to participate in this study.

NAME OF PARTICIPANT suliman

DATE 29-08-13

Accept

Reject

localhost:8080/SpamDefender/pages/signup.jsp 2/3

Screenshots for the System Website – Experiment instructions

8/29/13

The Battle Field: Show me what you can!!

Main Logout



The Battle Field: Show me what you can!

So you accepted the challenge? Dare you are! Read the instructions first and then try to defeat me, if you can!

The experiment instructions:

- You are supposed to act as an attacker and your target is to defeat the SpamDefenders' algorithms by sending a specific number of Emails successfully.
- To defeat an algorithm you have to investigate:
 - a. The email part that the algorithm is checking to compare the similarity with previous sent messages.
- There are a number of disguising attacks that Spammers can enact to fool Spam filter algorithms. For example:
 - a. Random addition: random characters are added to the text.
 - b. Thesaurus substitution: words of the original text are substituted by synonyms ("seem" could be substituted by "appear")
 - c. Perceptive substitution: characters of the original text are substituted without changing the text perception ("security" could be substituted by "s3curity")
 - d. Aimed addition: sequences of characters are added at the original text.
 - e. You can vary between these methods or find out new methods if they do not work out for you!
- By defeating an algorithm, you will be notified that you have passed to the next algorithm.

Always Remember: All your messages should be understandable and deliver the same meaning to all the targeted recipients.

localhost:8080/SpamDefender/pages/theBattleField.jsp

1/2

Screenshots for the System Website - An Example to an attacker's attempts to break the system

- A number of successful/failed trials for Breaking Algorithm 1

[Main](#) [Logout](#)

[Quick Hints](#)

" Trial: 1 "



Algorithm: 1

Number of Recipients (Do not change)

Remaining Letters

Start Typing [Reset Message](#)

Dear Students,
We are dictate2us, the UK?s number one transcription service for academics! We have a great deal of experience in typing: Dissertations

We charge per dictated minute, so you only pay for your audio time - not our typing time. All students qualify for a 15% discount on all multiple speaker audio files,

Contact us today for more information, or for a free no obligation quote and let d2u take the stress out of your studies.
Kind Regards,
dictate2us
Office:0161 762 1100
Fax: 0161 762 9694

Messages sent successfully so far: 0 / 900

[Main](#) [Logout](#)

[Quick Hints](#)

" Trial: 2 "



Algorithm: 1

Number of Recipients (Do not change)

Remaining Letters

Start Typing [Reset Message](#)

Dear Students,
We are dictate2us, the UK?s number one transcription service for academics! We have a great deal of experience in typing: Dissertations

We charge per dictated minute, so you only pay for your audio time - not our typing time. All students qualify for a 15% discount on all multiple speaker audio files,

Contact us today for more information, or for a free no obligation quote and let d2u take the stress out of your studies.
Kind Regards,
dictate2us
Office:0161 762 1100
Fax: 0161 762 9694

Messages sent successfully so far: 100 / 900

SPAM attempt Succeeded !!

Total score is: 41 %

Main Logout

Quick Hints

" Trial: 3 "



Algorithm: 1

Number of Recipients (Do not change) 100

Remaining Letters 80

Start Typing Reset Message

Dear Students,
 We are dictate2us, the UK?s number one transcription service for academics! We have a great deal of experience in typing: Dissertations

We charge per dictated minute, so you only pay for your audio time - not our typing time. All students qualify for a 15% discount on all multiple speaker audio files,

Contact us today for more information, or for a free no obligation quote and let d2u take the stress out of your studies.

Kind Regards,
 dictate2us
 Office:0161 762 1100
 Fax: 0161 762 9694

Submit

Messages sent successfully so far: 100 / 900

Your SPAM attempt failed !!

Main Logout

Quick Hints

" Trial: 5 "



Algorithm: 1

Number of Recipients (Do not change) 100

Remaining Letters 80

Start Typing Reset Message

Dear Students,
 We are dictate2us, the UK?s number one transcription service for academics! We have a great deal of experience in typing: Dissertations

We charge per dictated minute, so you only pay for your audio time - not our typing time. All students qualify for a 15% discount on all multiple speaker audio files,

Contact us today for more information, or for a free no obligation quote and let d2u take the stress out of your studies.

Kind Regards,
 dictate2us
 Office:0161 762 1100
 Fax: 0161 762 9694

Submit

Messages sent successfully so far: 200 / 900

Your SPAM attempt failed !!

- **A trial for Breaking Algorithm 2**

Main Logout

" Trial: 12 "

Number of Recipients (Do not change)

Remaining Letters

Start Typing [Reset Message](#)

Algorithm: 2

Dear Students,
 We are dictate2us, the UK?s number one transcription service for academics! We have a great deal of experience in typing: Dissertations

We charge per dictated minute, so you only pay for your audio time - not our typing time. All students qualify for a 15% discount on all multiple speaker audio files,

Contact us today for more information, or for a free no obligation quote and let d2u take the stress out of your studies.
 Kind Regards,
 dictate2us
 Office:0161 762 1100
 Fax: 0161 762 9694

Quick Hints

Messages sent successfully so far: 400 / 900

SPAM attempt Succeeded !!

Total score is: 98 %

- **A trial for Breaking algorithm 3**

Main Logout

" Trial: 17 "

Number of Recipients (Do not change)

Remaining Letters

Start Typing [Reset Message](#)

Algorithm: 3

Dear Students,
 We are dictate2us, the UK?s number one transcription service for academics! We have a great deal of experience in typing: Dissertations

We charge per dictated minute, so you only pay for your audio time - not our typing time. All students qualify for a 15% discount on all multiple speaker audio files,

Contact us today for more information, or for a free no obligation quote and let d2u take the stress out of your studies.
 Kind Regards,
 dictate2us
 Office:0161 762 1100
 Fax: 0161 762 9694

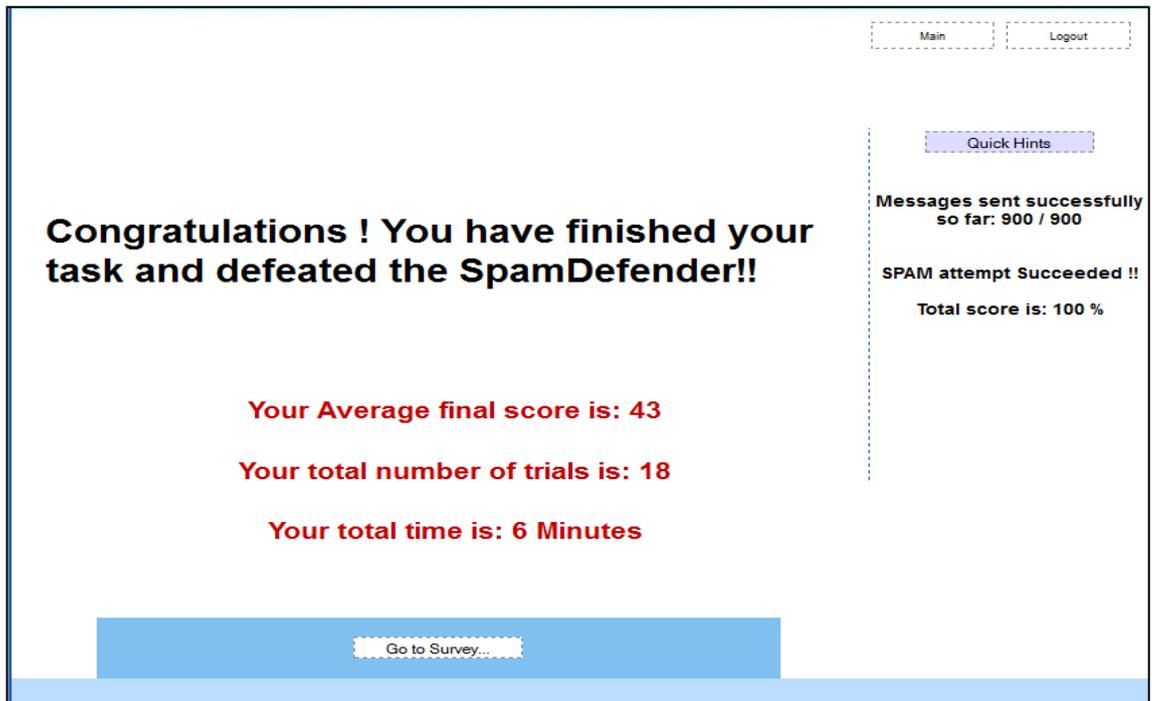
Quick Hints

Messages sent successfully so far: 800 / 900

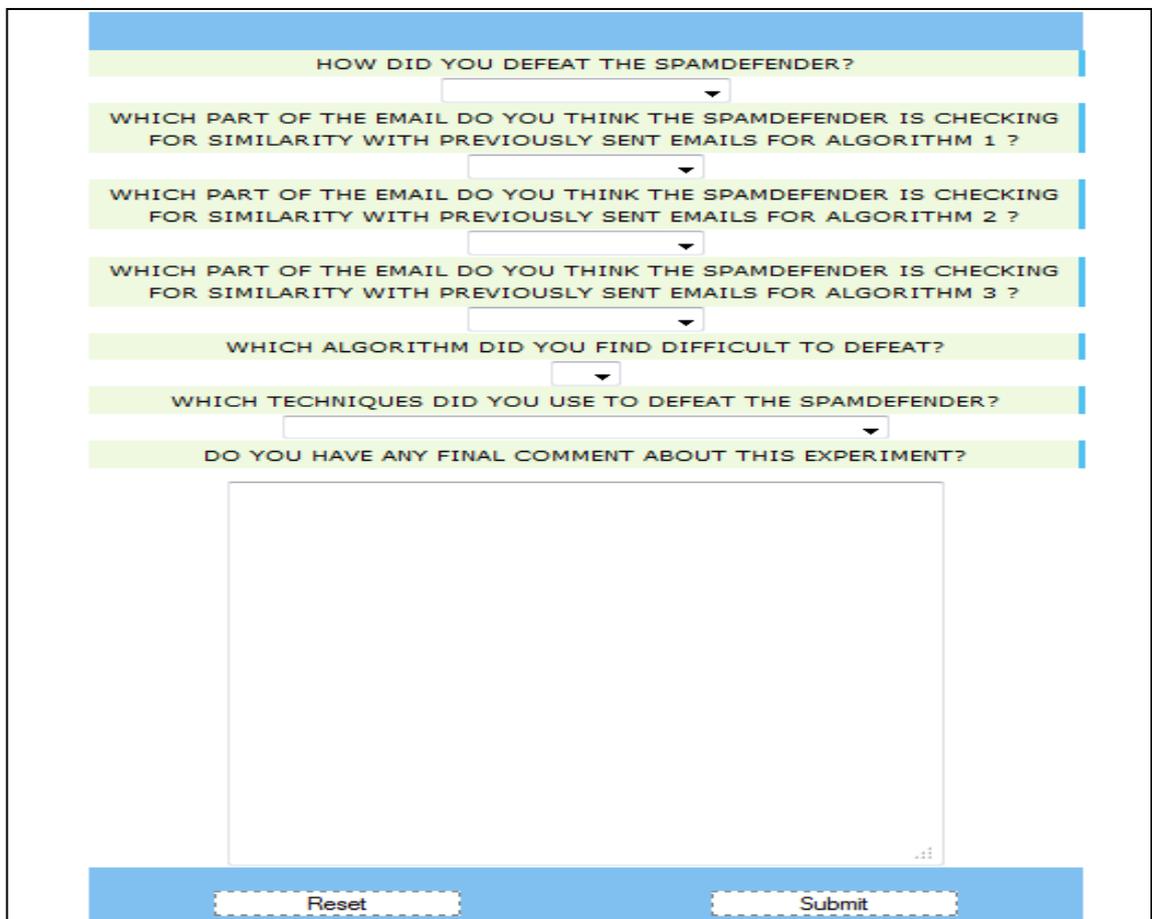
SPAM attempt Succeeded !!

Total score is: 95 %

- **Breaking the system**



Screenshots for the System Website – Survey Page



HOW DID YOU DEFEAT THE SPAMDEFENDER?

WHICH PART OF THE EMAIL DO YOU THINK THE SPAMDEFENDER IS CHECKING FOR SIMILARITY WITH PREVIOUSLY SENT EMAILS FOR ALGORITHM 1 ?

WHICH PART OF THE EMAIL DO YOU THINK THE SPAMDEFENDER IS CHECKING FOR SIMILARITY WITH PREVIOUSLY SENT EMAILS FOR ALGORITHM 2 ?

WHICH PART OF THE EMAIL DO YOU THINK THE SPAMDEFENDER IS CHECKING FOR SIMILARITY WITH PREVIOUSLY SENT EMAILS FOR ALGORITHM 3 ?

WHICH ALGORITHM DID YOU FIND DIFFICULT TO DEFEAT?

WHICH TECHNIQUES DID YOU USE TO DEFEAT THE SPAMDEFENDER?

DO YOU HAVE ANY COMMENTS?

- Random Additions (letters, numbers, spaces, ...)
- Thesaurus Substitutions (letters, numbers, spaces, ...)
- Perceptive substitution (e.g. o to be 0)
- Two of the above
- All of the above
- Other Techniques

Screenshots for the System Website – Logout Page

Main Logout



The Battle Field: Show me what you can!

So you accepted the challenge? Dare you are! Read the instructions first and then try to defeat me, if you can!

Thanks for your Time and Participation!

Click here to logout

 Copyrights © SpamDefender: A project of the Centre of Cybercrime and Computer Security - Newcastle University - 2012

APPENDIX B: ACCUMULATIVE MANIPULATION OF TEST SETS

Contents

- ALC-Based Accumulative Manipulation
- ALCM-Based Accumulative Manipulation

ALC-Based Accumulative Manipulation

Note: Green colour: classified successfully, while Red colour: misclassified. An L. result refers to labelled data and C. result refers to classified result.

ALC-based Accumulative manipulation																			
Attacker																			
Trial	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	20	11	0	9	0	0	0	18	0	7	0	23	0	0	0	0
3	0	0	17	38	29	7	37	0	3	6	45	2	12	0	42	0	15	17	0
4	5	11	45	65	47	16	64	0	11	9	113	7	38	0	42	0	27	44	0
5	17	16	76	92	68	42		10	40	12		7	67	9	82	7	38	73	0
6	39	22			81	49		28	69	21		36		14	95	8	65		0
7	61	48			91	80		54		43		57		41	115	15	91		15
8	88	78			130			87		61		77		62	149	24			39
9	110				151					87		114		86		44			61
10	134				204					113		116		111		66			79
11	184											123				124			110
12												134				136			133
13												151				159			191
14												172				210			
15												223							
L. results	S3	S1	S1	S2	S3	S1	S1	S2	S1	S3	S2	S3	S1	S1	S2	S3	S1	S2	S3
C. results	S3	S1	S2	S2	S3	S2	S1	S1	S1	S3	S2	S3	S1	S1	S2	S3	S2	S2	S2

ALCM-Based Accumulative Manipulation

ALCM-based Accumulative manipulation																			
Attacker																			
Trial	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	5.009	11.8	17.93	10.9	7.571	15	6.67	7.07	5.635	26.97	6.882	9.111	5.65	12.4	7.35	8.1	11.38	7.7
3	9.211	10.83	26.8	39.39	23.1	16.38	35.3	14.29	15.8	11.94	62.3	14.25	21.09	12	26.4	15.3	17.6	25.67	16.02
4	19.31	17.762	46.9	65.98	36.9	26.89	64	23.17	27.2	19.1	112.6	22.19	38.24	19.1	42.6	23.8	29.2	44.68	25.05
5	30.49	26.297	76	92	52.7	39.85		33.77	43.1	27.35		30.78	66.93	27.4	61.5	33.1	43.7	72.49	34.95
6	42.98	37.322			71.2	56.65		46.87	68.9	37.09		40.14		37.2	84.5	43.4	63.1		45.87
7	57.14	52.668			93.4	80		63.9		48.92		50.43		49.1	113	54.7	91		58.05
8	73.45	76.988			121			87		63.91		61.82		64.2	149	67.4			71.81
9	92.65				158					84.15		74.59		84.5		81.8			87.6
10	115.9				204					113		89.1		111		98.5			106.1
11	145.3											105.9				118			128.3
12												125.7				142			156.1
13												149.9				173			191
14												180.7				201			
15												222.6							
L. results	S3	S1	S1	S2	S3	S1	S1	S2	S1	S3	S2	S3	S1	S1	S2	S3	S1	S2	S3
C. results	S3	S2	S1	S2	S3	S2	S1	S1	S2	S3	S2	S3	S1	S1	S3	S3	S1	S2	S2

APPENDIX C: PROBABILITY OF CLASSIFICATION RESULTS

Contents

- ALC-Based Accumulative Manipulation
- ALCM-Based Accumulative Manipulation

Appendix C: Probability of Classification Results

		ALC-based Probability of detecting each attempt to the predicted strategy										
		Trials										
Attacker	strategy	1	2	3	4	5	6	7	8	9	10	11
1	1	0.528	0.528	0.171	0.233	0.322	0.237	0.178	0.125			
	2	0.338	0.338	0.356	0.370	0.305	0.355	0.329	0.291			
	3	0.132	0.132	0.471	0.396	0.371	0.406	0.492	0.583			
2	1	0.528	0.528	0.528	0.528	0.498	0.482	0.389				
	2	0.338	0.338	0.338	0.338	0.348	0.353	0.370				
	3	0.132	0.13	0.132	0.132	0.152	0.164	0.239				
3	1	0.528	0.528	0.365	0.363							
	2	0.338	0.338	0.372	0.368							
	3	0.132	0.132	0.261	0.267							
4	1	0.528	0.355	0.360	0.365							
	2	0.338	0.475	0.454	0.372							
	3	0.132	0.169	0.184	0.261							
5	1	0.528	0.258	0.229	0.203	0.261	0.229	0.133	0.095			
	2	0.338	0.361	0.352	0.365	0.365	0.352	0.298	0.261			
	3	0.132	0.379	0.418	0.431	0.372	0.418	0.568	0.643			
6	1	0.528	0.528	0.362	0.371	0.371						
	2	0.338	0.338	0.375	0.379	0.386						
	3	0.132	0.132	0.261	0.248	0.242						
7	1	0.528	0.471	0.434								
	2	0.338	0.356	0.364								
	3	0.132	0.171	0.200								
8	1	0.528	0.528	0.528	0.528	0.495	0.434	0.372	0.370			
	2	0.338	0.338	0.338	0.338	0.349	0.364	0.345	0.323			
	3	0.132	0.132	0.132	0.132	0.154	0.200	0.281	0.305			
9	1	0.528	0.528	0.498	0.488	0.458						
	2	0.338	0.338	0.348	0.351	0.359						
	3	0.132	0.132	0.152	0.159	0.181						
10	1	0.528	0.528	0.223	0.291	0.270	0.302	0.242	0.305	0.189		
	2	0.338	0.338	0.369	0.335	0.364	0.327	0.371	0.323	0.361		
	3	0.132	0.132	0.406	0.372	0.364	0.369	0.386	0.370	0.448		
11	1	0.528	0.371	0.328								
	2	0.338	0.379	0.372								
	3	0.132	0.248	0.298								
12	1	0.528	0.528	0.233	0.322	0.166	0.264	0.178	0.127	0.047		
	2	0.338	0.338	0.370	0.305	0.354	0.363	0.329	0.293	0.190		
	3	0.132	0.132	0.396	0.371	0.478	0.371	0.492	0.579	0.762		
13	1	0.528	0.478	0.448	0.379							
	2	0.338	0.354	0.361	0.371							

Appendix C: Probability of Classification Results

		ALC-based Probability of detecting each attempt to the predicted strategy										
14	3	0.132	0.166	0.189	0.248							
	1	0.528	0.528	0.528	0.528	0.478	0.448	0.379				
	2	0.338	0.338	0.338	0.338	0.354	0.361	0.371				
15	3	0.132	0.132	0.132	0.132	0.166	0.189	0.248				
	1	0.528	0.322	0.370	0.319	0.357						
	2	0.338	0.371	0.389	0.371	0.398						
16	3	0.132	0.305	0.239	0.309	0.243						
	1	0.528	0.528	0.528	0.528	0.233	0.322	0.322	0.312	0.369	0.331	0.203
	2	0.338	0.338	0.338	0.338	0.171	0.305	0.305	0.316	0.258	0.299	0.365
17	3	0.132	0.132	0.132	0.132	0.471	0.371	0.371	0.371	0.372	0.369	0.431
	1	0.528	0.528	0.298	0.369	0.331	0.327					
	2	0.338	0.338	0.568	0.372	0.369	0.369					
18	3	0.132	0.132	0.133	0.258	0.299	0.302					
	1	0.528	0.528	0.291	0.261							
	2	0.338	0.338	0.583	0.643							
19	3	0.132	0.132	0.125	0.095							
	1	0.528	0.528	0.528	0.528	0.528	0.528	0.371	0.370	0.322	0.298	0.264
	2	0.338	0.338	0.338	0.338	0.338	0.338	0.379	0.396	0.371	0.372	0.371
	3	0.132	0.132	0.132	0.132	0.132	0.132	0.248	0.233	0.305	0.328	0.363

		ALCM-based Probability of detecting each attempt to the predicted strategy										
		Trials										
Attacker	strategy	1	2	3	4	5	6	7	8	9	10	11
1	1	0.528	0.528	0.224	0.222	0.240	0.151	0.084	0.037			
	2	0.338	0.338	0.351	0.350	0.356	0.312	0.249	0.169			
	3	0.132	0.132	0.424	0.427	0.402	0.535	0.666	0.793			
2	1	0.528	0.312	0.369	0.312	0.296	0.362	0.312				
	2	0.338	0.371	0.372	0.371	0.369	0.372	0.371				
	3	0.132	0.316	0.257	0.316	0.334	0.264	0.316				
3	1	0.528	0.318	0.357	0.371							
	2	0.338	0.371	0.372	0.319							
	3	0.132	0.310	0.269	0.308							
4	1	0.278	0.362	0.311	0.320							
	2	0.366	0.372	0.370	0.371							
	3	0.355	0.265	0.317	0.307							
5	1	0.224	0.261	0.240	0.234	0.222	0.163	0.104	0.097			
	2	0.351	0.362	0.356	0.354	0.350	0.321	0.271	0.263			
	3	0.424	0.375	0.402	0.410	0.427	0.515	0.623	0.639			
6	1	0.528	0.335	0.282	0.350	0.270						
	2	0.338	0.372	0.366	0.372	0.364						

Appendix C: Probability of Classification Results

	3	0.132	0.292	0.351	0.277	0.365						
7	1	0.528	0.382	0.447								
	2	0.338	0.361	0.344								
	3	0.132	0.256	0.208								
8	1	0.512	0.492	0.469	0.440	0.414	0.402	0.382				
	2	0.344	0.350	0.357	0.363	0.367	0.369	0.371				
	3	0.143	0.156	0.173	0.196	0.217	0.227	0.245				
9	1	0.512	0.335	0.351	0.369	0.360						
	2	0.344	0.372	0.366	0.372	0.365						
	3	0.143	0.292	0.282	0.258	0.274						
10	1	0.528	0.047	0.087	0.136	0.188	0.238	0.248	0.287	0.301		
	2	0.338	0.190	0.252	0.301	0.334	0.356	0.358	0.345	0.328		
	3	0.132	0.762	0.660	0.562	0.476	0.404	0.392	0.367	0.369		
11	1	0.528	0.350	0.289								
	2	0.338	0.372	0.368								
	3	0.132	0.277	0.342								
12	1	0.528	0.256	0.222	0.208	0.158	0.110	0.065	0.039	0.047		
	2	0.338	0.361	0.350	0.344	0.317	0.277	0.222	0.174	0.190		
	3	0.132	0.382	0.427	0.447	0.523	0.612	0.711	0.786	0.762		
13	1	0.528	0.510	0.489	0.464							
	2	0.338	0.345	0.351	0.358							
	3	0.132	0.144	0.159	0.176							
14	1	0.528	0.512	0.492	0.469	0.440	0.402	0.468				
	2	0.338	0.344	0.350	0.357	0.363	0.369	0.357				
	3	0.132	0.143	0.156	0.173	0.196	0.227	0.174				
15	1	0.528	0.226	0.224	0.163	0.104						
	2	0.338	0.351	0.351	0.321	0.271						
	3	0.132	0.421	0.424	0.515	0.623						
16	1	0.528	0.357	0.319	0.278	0.234	0.188	0.142	0.0970	0.056	0.047	0.025
	2	0.338	0.269	0.308	0.366	0.354	0.335	0.305	0.263	0.208	0.190	0.141
	3	0.132	0.372	0.371	0.355	0.410	0.476	0.552	0.639	0.734	0.762	0.832
17	1	0.528	0.372	0.351	0.370	0.372	0.379					
	2	0.338	0.335	0.366	0.323	0.350	0.371					
	3	0.132	0.292	0.282	0.305	0.277	0.248					
18	1	0.528	0.312	0.350	0.369							
	2	0.338	0.371	0.372	0.372							
	3	0.132	0.316	0.277	0.258							
19	1	0.528	0.370	0.357	0.319	0.278	0.234	0.229	0.208	0.188	0.185	0.175
	2	0.338	0.392	0.372	0.371	0.366	0.410	0.418	0.424	0.476	0.454	0.496
	3	0.132	0.236	0.269	0.308	0.355	0.354	0.352	0.366	0.335	0.360	0.328

BIBLIOGRAPHY

- [1] Abu-Nimeh, S., Nappa, D., Wang, X., and Nair, S. 2007. A comparison of Machine Learning Techniques for Phishing Detection. In *Proceedings of The AntiPhishing Working Group's Second Annual eCrime Researchers Summit* (Pittsburgh, Pa, USA, Oct. 4–5, 2007), 60–69.
- [2] Aghajani, Z., and Azgomi, M. A. 2009. Security Evaluation of an Intrusion Tolerant Web Service Architecture Using Stochastic Activity Networks. *Lectures on Advances in Information Security and Assurance*, edited by J. Park, H-H. Chen, M. Atiquzzaman, C. Lee, T-h. Kim, and S-S. Yeo, *Springer Lecture Notes in Computer Science*, 260–269.
- [3] Ajmone Marsan, M., Conte, G., and Balbo, G. 1984. A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. *ACM Transactions on Computer Systems*, 2(2), 93–122.
- [4] Alkahtani, H. S., Gardner-Stephen, P. A. U. L., and GOODWIN, R. 2011. A Taxonomy of Email SPAM Filters. In *Proceedings of the 12th International Arab Conference on Information Technology (ACIT' 11)*, (Riyadh, Saudi Arabia), 351–356.
- [5] Almasizadeh, J., and Azgomi, M. A. 2009. Intrusion Process Modelling for Security Quantification. In *Proceedings of the International Conference on Availability, Reliability and Security (ARES' 09)*. IEEE Computer Society, Los Alamitos, 114–121.
- [6] Alpcan, T., and Pavel, L. 2009. Nash Equilibrium Design and Optimization. In *Proceedings of the International Conference on Game Theory for Networks*, Istanbul, 13-15 May, 164–170.
- [7] Alpcan, T., and Baser, T. An Intrusion Detection Game with Limited Observations. 2006. In *Proceedings of the 12th International Symp on Dynamic Games and Applications*, Sophia Antipolis, France, July, 3–6.

-
- [8] Alpcan, T., and Baser. T. 2004. A Game Theoretic Analysis of Intrusion Detection in Access Control Systems. In *Proceedings of the 43rd IEEE Conference on Decision and Control*, 1568–1573.
- [9] Alpcan, T., and Basar, T. *Network Security: A Decision and Game-Theoretic Approach*. Cambridge University Press. 2011.
- [10] Alsuhibany, S. A. 2011. Optimising CAPTCHA Generation. In *Proceedings of the Sixth International Conference on Availability, Reliability and Security (ARES' 11)*, IEEE Computer Society, Austria, Vienna, 740–745.
- [11] Alsuhibany, S. A., Alonaizi, A., Morisset, C., Smith C., and van Moorsel, A. 2013. Experimental Investigation in the Impact on Security of the Release Order of Defensive Algorithms. In *Processing in 3rd IFIP International Workshop on Security and Cognitive Informatics for Homeland Defence (SeCIHD'13)*, volume 8128 of Lecture Notes in Computer Science (LNCS), Springer, September 2–6, 321–336.
- [12] Alsuhibany, S. A., and van Moorsel, A. 2013. Modelling and Analysis of Release Order of Security Algorithms Using Stochastic Petri Nets. In *Processing in 8th International Conference on Availability, Reliability and Security (ARES'13)*, IEEE Computer Society, September 2-6, 437–445.
- [13] Alsuhibany, S. A., Alonaizi, A., Morisset, C., and van Moorsel, A. 2013. Optimizing the Release Order of Defensive Mechanisms. In *Processing in 29th Annual UK Performance Engineering Workshop (UKPEW'13)*, 4th Jul, 34–41.
- [14] Alsuhibany, S. A., Morisset C., and van Moorsel, A. Detection of Attack Strategies. 2013. In *Processing in 8th International Conference on Risks and Security of Internet and Systems (CRiSIS'13)*, IEEE Computer Society, October 23–25, to appear.
- [15] American National Standards Institute (ANSI). 2008. The Financial Impact of Cyber Risk. *Internet Security Alliance (ISA)*. Retrieved Jun 14, 2013 from: http://www.ansi.org/meetings_events/events/Cyber_Risk08.aspx

-
- [16] Anderson, J. R. *Cognitive Skills and Their Acquisition*, Psychology Press, 1981.
- [17] Anderson, J. R. *Language, Memory, and Thought*, Psychology Press, 1976.
- [18] Anderson, R. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley Publishing, Inc. 2nd edition. 2008.
- [19] Anti-Phishing Working Group. Phishing Activity Trends Report: Third Quarter Report, Jan. 2010 [online]: http://apwg.org/reports/apwg_report_Q3_2009.pdf
- [20] Arachchilage, N. A. G. Gaming for Security. 2013. *ITNOW*, 55(1), 32–33.
- [21] Argote, L. 2013. Organizational Learning Curves: An Overview. *Organizational Learning*. Springer, Pittsburgh, US, 1–29.
- [22] Baird, H. S., and Popat, K. 2002. Human Interactive Proofs and Document Image Analysis. In *Proceedings of 5th IAPR Int. Workshop on Document Analysis Systems (DAS' 02)*, vol. 2423 of *LNCS*, 507–518.
- [23] Barrows, H. S., and Tamblyn, R. M. *Problem-Based Learning: An Approach to Medical Education*, Springer Series on Medical Education, New York, 1980, Edition: 1.
- [24] Bellman, R. 1957. A Markovian Decision Process. *Indiana University: Journal of Mathematics and Mechanics*, 6(5), 679–684.
- [25] Bhat, U. N., and Miller, G. K. *Elements of Applied Stochastic Processes*, Wiley-Interscience, Edition 3, 2002.
- [26] Bishop, M. *Computer Security: Art and Science*. Addison-Wesley, 2003.
- [27] Blakley, B., Mcdermott, E., and Geer, D. 2001. Information Security is Information Risk Management. In *Proceedings of the 2001 Workshop on New Security*, (Cloudcroft, New Mexico, USA), 97–104.

-
- [28] Bloem, M., Alpcan, T., and Basar, T. 2006. Intrusion Response as a Resource Allocation Problem. In *Proceedings of the 45th IEEE Conference on Decision and Control*, San Diego, CA, 6283–6288.
- [29] Bransford, J. D. *The Jasper Project: Lessons in Curriculum, Instruction, Assessment, and Professional Development*, New York: Routledge, 1997.
- [30] Brocklehurst, S., Littlewood, B., Olovsson, T., and Jonsson, E. 1994. On Measurement of Operational Security. In *Proceedings of the Ninth Annual IEEE Conference Computer Assurance (COMPASS '94)*, IEEE Computer Society, 257–266.
- [31] Brown, J., Collins, A., and Newman, S. 1989. Cognitive Apprenticeship: Teaching the Crafts of Reading, Writing, and Mathematics. In Resnick, L. B. (ed.), *Knowing, Learning, and Instruction: Essays in Honor of Robert Glaser*, Erlbaum, Hillsdale, NJ, 453–494.
- [32] Bursztein, E., Martin, M., and Mitchell, J. 2011. Text-Based CAPTCHA Strengths and Weaknesses. In *Proceedings of the 18th ACM conference on Computer and communications security (CCS '11)*, New York, NY, USA, ACM, 125–138.
- [33] Caliendo, M., Clement, M., Papiés, D., and Scheel-Kopeinig, S. 2012. The Cost Impact of Spam Filters: Measuring the Effect of Information System Technologies in Organizations. *Information Systems Research*, 32(3), 1–13.
- [34] Card, S. K., English, W. K., and Burr, B. J. 1978. Evaluation of Mouse, Rate Controlled Isometric Joystick, Step Keys, and Text Keys for Text Selection on a CRT. *Ergonomics*, 21(8), 601–613.
- [35] Chan, T. Y. 2003. Using a Text-to-Speech Synthesizer to Generate a Reverse Turing Test. In *Proceedings of 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 03)*, 226–232.

-
- [36] Chew, E., Marianne, S., Kevin, S., Nadya, B., Anthony, B., and Will, R. 2007. Nist Performance Measurement Guide for Information Security. *Technical report, NIST Institute*. Retrieved July 3, 2013 from: <http://csrc.nist.gov/publications/nistpubs/800-55-Rev1/SP800-55-rev1.pdf>
- [37] Chi, M. T. 2008. Three Kinds of Conceptual Change: Belief Revision, Mental Model Transformation, and Ontological Shift. In S., Vosniadou, A., Baltas and X., Vamvakoussi, (Eds.), *Re-Framing the Conceptual Change Approach in Learning and Instruction*. Advances in Learning and Instruction Series, Elsevier Press, 61–82.
- [38] Ciardo, G., Muppala, J., and Trivedi, K. 1989. SPNP: Stochastic Petri Net Package. In *Proceedings of the Third International Workshop on Petri Nets and Performance Models*, IEEE Press, Kyoto 11–13 Dec, 142–150.
- [39] Ciardo, G., Fricks, R. M., Muppala, J. K., and Trivedi, K. S. SPNP Users Manual Version 6.0, *Department Electrical Engineering*, Duke University, 1999.
- [40] Clapper, J. P., and Bower, G. H. 1994. Category Invention in Unsupervised Learning. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 20(2), 443–460.
- [41] Clark, K. P. 2008. *A Survey of Content-based Spam Classifiers*. [Online]. Available at: http://pdf.aminer.org/000/223/637/collaborative_detection_of_spam_in_peer_to_peer_paradigm_based.pdf
- [42] Cook, D., Hartnett, J., Manderson, K., and Scanlan, J. 2006. Catching Spam Before it Arrives: Domain Specific Dynamic Blacklists. In *Proceedings of the Australasian Information Security Workshop (Network Security) (AISWNet-Sec'06)*, (Hobart, Australia), 193–202.
- [43] Daly, D., Deavours, D. D., Doyle, J. M., Webster, P. G., and Sanders, W. H. 2000. Möbius: An Extensible Tool for Performance and Dependability Modeling. In *Proceedings of the 11th International Conference on Computer Per-*

-
- formance Evaluation: Modelling Techniques and Tools* (TOOLS '00), Springer-Verlag Berlin, Heidelberg, 332–336.
- [44] Damiani, E., di Vimercati, S. D. C., Paraboschi, S., and Samarati, P. 2004. An Open Digest-based Technique for Spam Detection. In *Proceedings of the 2004 International Workshop on Security in Parallel and Distributed Systems* (San Francisco, CA USA, September, 2004), 15–17.
- [45] Deavours, D. D., Clark, G., Courtney, T., Daly, D., Derisavi, S., Doyle, J. M., Sanders, W. H., and Webster, P. G. 2002. The Möbius Framework and its Implementation. *IEEE Transactions on Software Engineering*, 28(10), 956–969.
- [46] Dingle, N. J., Knottenbelt, W. J., and Suto, T. 2009. PIPE2: a Tool for the Performance Evaluation of Generalised Stochastic Petri Nets. *ACM SIGMETRICS Performance Evaluation Review*, 36(4), 34–39.
- [47] Elio, R., and Anderson, J. R. 1981. The Effects of Category Generalizations and Instance Similarity on Schema Abstraction. *Journal of Experimental Psychology: Human Learning and Memory*, 7(6), 397–417.
- [48] Elio, R., and Anderson, J. R. 1984. The Effects of Information Order and Learning Mode on Schema Abstraction. *Memory and Cognition*, 12(1), 20–30.
- [49] Fette, I., Sadeh, N., and Tomasic, A. 2007. Learning to Detect Phishing Emails. In *Proceedings of the 16th International World Wide Web Conference* (Banff, Canada, May 8–12, 2007), 649–656.
- [50] Freschi, V., Seraghiti, A., and Bogliolo, A. 2006. Filtering Obfuscated Email Spam by means of Phonetic String Matching. *Advances in Information Retrieval: 28th European Conference on IR Research ECIR' 06* (London, UK, 2006), Springer-Verlag Berlin, Heidelberg, 505–509.
- [51] Garcia, F. D., Hoepman, J. H., and Van Nieuwenhuizen, J. 2004. Spam Filter Analysis. In *Proceedings of 19th IFIP International Information Security Conference*, (Toulouse, France), Springer-Verlag Berlin, Heidelberg, 395–410.

-
- [52] Golbeck, J., and Hendler, J. A. 2004. Reputation Network Analysis for Email Filtering. In *Proceedings of Conference on Email and Anti-Spam (CEAS' 04)*, 1-8.
- [53] Gollmann, D. *Computer Security*, 3rd Edition, Wiley Publishing, 2011.
- [54] Goseva-Popstojanova, K., Wang, F., Wang, R., Gong, F., Vaidyanathan, K., Trivedi, K., and Muthusamy, B. 2001. Characterizing Intrusion Tolerant Systems Using a State Transition Model. In *Proceedings of the Information Survivability Conference and Exposition, (DISCEX '01)*, 211-221.
- [55] Gunes Kayacik, H., Nur Zincir-Heywood, A., and Heywood, M. I. 2007. A Hierarchical SOM-Based Intrusion Detection System. *Engineering Applications of Artificial Intelligence*, 20(4), 439-451.
- [56] Guo, X., and Hernandez-Lerma, O. *Continuous-Time Markov Decision Processes: Theory and Applications*. Springer, 2009.
- [57] Hackett, E. A. 1983. Application of A set of Learning Curve Models to Repetitive Tasks. *The Radio and Electronic Engineer*, 53(1), 25-32.
- [58] Hanakawa, N., Morisaki, S., and Matsumoto, K. I. 1998. A Learning Curve Based Simulation Model for Software Development. In *Proceedings of the 20th International Conference on Software Engineering*, (Kyoto, Japan, April 19-25), 350-359.
- [59] Haverkort, B. R. *Performance of Computer Communication Systems: A Model-Based Approach*, John Wiley & Sons, Inc., New York, NY, USA, 1998.
- [60] Herley, C., and Florêncio, D. 2008. A Profitless Endeavour: Phishing as a Tragedy of the Commons. In *Proceedings of the New Security Paradigms Workshop (Lake Tahoe, Ca)*, 22-25.
- [61] Heron, S. Technologies for Spam Detection. 2009. *Network Security*, 1(1), 11-15.

-
- [62] Hmelo, C. E., and Ferrari, M. 1997. The Problem-Based Learning Tutorial: Cultivating Higher Order Thinking Skills. *Journal for the Education of the Gifted*, 20(4), 401–422.
- [63] Hong, J. 2012. The State of Phishing Attacks. *Communications of the ACM*, 55(1), 74–81.
- [64] Hoo, K. J. S. 2000. How Much Is Enough? A Risk-Management Approach to Computer Security. Technical report, Consortium for Research on Information Security and Policy (CRISP) [Online]: http://cisac.stanford.edu/publications/how_much_is_enough_a_riskmanagement_approach_to_computer_security/
- [65] Horng, S. J., Su, M. Y., Chen, Y. H., Kao, T. W., Chen, R. J., Lai, J. L., and Perkasa, C. D. 2011. A Novel Intrusion Detection System Based on Hierarchical Clustering and Support Vector Machines. *Expert Systems with Applications*, 38(1), 306–313.
- [66] Huang, L., Joseph, A. D., Nelson, B., Rubinstein, B. I. P., and Tygar, J. D. 2011. Adversarial Machine Learning. In *Proceedings of the 4th ACM workshop on Artificial Intelligence and Security (AISec'11)*, 43–58.
- [67] Jiang, W., Tian, Z. H., Zhang, H. L., and Song, X. F. 2008. A Stochastic Game Theoretic Approach to Attack Prediction and Optimal Active Defence Strategy Decision. In *Proceedings of the 2008 IEEE International Conference on Networking, Sensing and Control (ICNSC'08)*, Hainan, China, 6–8 April, 648–653.
- [68] Johansson, J. M. The Fundamental Tradeoffs. Retrieved September 11, 2013 from, <http://technet.microsoft.com/en-us/library/cc512573.aspx>
- [69] Jones, A. K., and Lipton, R. J. 1975. The Enforcement of Security Policies for Computation. In *Proceedings of the 5th ACM symposium on Operating systems principles*. Austin, Texas, United States, 197–206.

-
- [70] Jonsson, E., and Olovsson, T. 1997. A Quantitative Model of the Security Intrusion Process Based on Attacker Behaviour. *IEEE Transactions on Software Engineering*, 23(4), 235–245.
- [71] Kadota, Y., Kurano, M., and Yasuda, M. 2006. Regret-Optimal Policies in Absorbing Semi-Markov Decision Processes with Multiple Constraints. In *Proceedings of the Development of Information and Decision Processes*, 87–94.
- [72] Kahn, P., and O’Rourke, K. *Guide to Curriculum Design: Enquiry-Based Learning*, Higher Education Academy, York, 2004. [Online]. Available: http://www.heacademy.ac.uk/resources.asp?id=359&process=full_record§ion=generic
- [73] Khan, L., Awad M., and Thuraisingham, B. 2006. A New Intrusion Detection System Using Support Vector Machines and Hierarchical Clustering. *The International Journal on Very Large Data Bases*, 16(4), 507–521.
- [74] Khonji, M., Iraqi, Y., and Jones, A. 2013. Phishing Detection: A Literature Survey. *Communications Surveys & Tutorials*, IEEE Communications Society, (99), 1–31.
- [75] Khorsi, A. 2007. An Overview of Content-Based Spam Filtering Techniques. *Informatica (Slovenia)*, 31(3), 269–277.
- [76] Kim, J. W., Chung, W. K., and Cho, H. G. 2010. A new Image-Based CAPTCHA Using the Orientation of the Polygonally Cropped Sub-images. *The Visual Computer*, vol. 26(8), 1135–1143.
- [77] Kolodner, J. L., Hmelo, C. E., and Narayanan, N. H. 1996. Problem-Based Learning Meets Case-Based Reasoning. In Edelson, D. C. and Domeshek, E. A. (eds.), *Proceedings of ICLS 96*, AACE, (Charlottesville, VA), 188–195.
- [78] Kolodner, J. *Case-Based Reasoning*, San Francisco, CA, USA: *Morgan Kaufmann*, 1993.

-
- [79] Kordy, B., Mauw, S., Radomirović, S., and Schweitzer, P. 2011. Foundations of Attack-Defence Trees. (Edits,) Pierpaolo Degano, Sandro Etalle, and Joshua D. Guttman, *Formal Aspects in Security and Trust*, Springer in Lecture Notes in Computer Science, 80–95.
- [80] Krautsevich, L., Martinelli, F., and Yautsiukhin, A. 2013. Towards Modelling Adaptive Attacker’s Behaviour. *Foundations and Practice of Security*. Springer Berlin Heidelberg, 357–364.
- [81] Kreidl, O. P. 2010. Analysis of a Markov Decision Process Model for Intrusion Tolerance. In *Proceedings of the International Conference on Dependable Systems and Networks Workshops (DSN-W 10)*, IEEE Press, 156–161.
- [82] Lam, H. Y., and Yeung, D. Y. 2007. A Learning Approach to Spam Detection based on Social Networks. In *Proceedings of Conference on Email and Anti-Spam (CEAS’ 07)*, (Mountain View, California USA), 1–9.
- [83] Leversage, D. J., and Byres, E. J. 2007. Comparing Electronic Battlefields: Using Mean Time-to-Compromise as a Comparative Security Metric. In *Proceedings of the 4th international Conference of Mathematical Methods, Models, and Architectures for Computer Network Security*, Computer Network Security. Springer Berlin Heidelberg, 213–227.
- [84] Liao, H. J., Tung, K. Y., Richard Lin, C. H., and Lin, Y. C. 2013. Intrusion Detection System: A Comprehensive Review. *Network and Computer Applications*, 36(1), 16–24.
- [85] Litan, A. Phishing Attack Victims Likely Targets for Identity Theft. *Gartner Group*, 4 May 2004 [online]: http://www.social-engineer.org/wiki/archives/IdTheif/IdTheif-phishing_attack.pdf
- [86] Liu, Y., Chen, K., Liao, X., and Zhang, W. 2004. A Genetic Clustering Method for Intrusion Detection. *Pattern Recognition*, 37(5), 927–942.

-
- [87] Liu, G., and Yi, Z. 2006. Intrusion Detection Using PCASOM Neural Networks. In *Proceedings of the Third International Symposium on Neural Networks (ISNN06)*, Chengdu, China, Springer-Verlag Berlin, Heidelberg, 240–245.
- [88] Lye, K. W., and Wing, J. M. 2005. Game Strategies in Network Security. *International Journal of Information Security*, 4(2), 71–86.
- [89] Madan, B. B., Gogeva-Popstojanova, K., Vaidyanathan, K., and Trivedi, K. S. 2002. Modelling and Quantification of Security Attributes of Software Systems. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks (DSN 02)*, IEEE Press, 505–514.
- [90] Marsan, M. A. 1990. Stochastic Petri Nets: An elementary introduction. In *Advances in Petri Nets 1989*, Springer: Lecture Notes in Computer Science pages 1–29.
- [91] Mathy, F., and Feldman, J. 2009. A Rule-Based Presentation Order Facilitates Category Learning. *Psychonomic Bulletin & Review*, 16(6), 1050–1057.
- [92] Matlab. The Language of Technical Computing. 2012. Retrieved September 15, 2013 from: <http://www.mathworks.com/products/matlab/>.
- [93] MATLAB online support: www.mathworks.com/access/helpdesk/help/techdoc/matlab.shtml.
- [94] McInerney, J., Stubberud, S., Anwar, S., and Hamilton, S. 2001. FRIARS: a Feedback Control System for Information Assurance using a Markov Decision Process. In *Proceedings of the IEEE 35th International Carnahan Conference*, IEEE Press, 223–228.
- [95] McLachlan, G. *Discriminant Analysis and Statistical Pattern Recognition*. Wiley, New York, 1992.

-
- [96] McQueen, M. A., Boyer, W. F., Flynn, M. A., and Beitel, G. A. 2005. Time-to-Compromise Model for Cyber Risk Reduction Estimation. In *Proceedings of the First Workshop on Quality of Protection* (Milan, Italy – September 15), Springer US, 49–64.
- [97] McQueen, M. A., Boyer, W. F., Flynn, M. A., and Beitel, G. A. 2006. Quantitative Cyber Risk Reduction Estimation Methodology for a Small SCADA Control System. In *Proceedings of the 39th Annual Hawaii Conference on System Science* (HICSS 06), 226–237.
- [98] Medin, D. L., and Bettger, J. G. 1994. Presentation Order and Recognition of Categorically Related Examples. *Psychonomic Bulletin and Review*, 1(2), 250–254.
- [99] Mitrani, I. *Probabilistic Modelling*. Cambridge University Press, 1998.
- [100] Moy, G., Jones, N., Harkless, C., and Potter, R. 2004. Distortion Estimation Techniques in Solving Visual CAPTCHAs. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 23–28.
- [101] Naor, M. 1996. *Verification of a Human in the Loop, or Identification via the Turing Test* [online]. Available: http://www.wisdom.weizmann.ac.il/~naor/PAPERS/human_abs.html
- [102] Neely, J. H., and Balota, D. A. 1981. Test-Expectancy and Semantic Organization Effects in Recall and Recognition. *Memory & Cognition*, 9(3), 283–300.
- [103] Nelson, B., Barreno, M., Chi, F. J., Joseph, A. D., Rubinstein, B. I., Saini, U., Sutton, C. A., Tygar, J. D., and Xia, K. 2008. Exploiting Machine Learning to Subvert your Spam Filter. In *Proceedings of the first USENIX Workshop on Large-Scale Exploits and Emergent Threats*, April 2008.
- [104] Neuts, M. F. *Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach*, John Hopkins University Press, 1981.

-
- [105] Neville, A. J. 2009. Problem-Based Learning and Medical Education Forty Years on: A Review of its Effects on Knowledge and Clinical Performance. *Med Princ Pract*, 18(1), 1–9.
- [106] Nguyen, Q., and Sood, A. 2009. Quantitative Approach to Tuning of a Time-Based Intrusion-Tolerant System Architecture. In *Proceedings of the 3rd Workshop Recent Advances on Intrusion-Tolerant Systems*, [Online]: <http://wraits09.di.fc.ul.pt/wraits09paper2.pdf>
- [107] Norman, G. 1988. Problem-Solving Skills, Solving Problems, and Problem-Based Learning. *Medical Education*, 22(4), 279-286.
- [108] Norman, G. R., and Schmidt, H. G. 1992. The Psychological Basis of Problem-Based Learning: A Review of the Evidence. *Academic Medicine*, 67(9), 557-565.
- [109] Norris, J. R. *Markov Chains*, Cambridge University Press, 1998.
- [110] Ortalo, R., Deswarte, Y., and Kaâniche, M. 1999. Experimenting with Quantitative Evaluation Tools for Monitoring Operational Security. *IEEE Trans Software Engineering*, 25(5), 633–650.
- [111] Oxford. *Dictionary of Computing*, Fourth Ed. Oxford University Press, 1996.
- [112] Paulauskas, N., and Garsva, E. 2008. Attacker Skill Level Distribution Estimation in the System Mean Time-to-Compromise. In *Proceedings of the 1st IEEE International Conference on Information Technology (IT 2008)*, (Gdansk, Poland, May 19-21), 1–4.
- [113] Payne, S. C. 2006. A Guide to Security Metrics. *Technical report*, SANS Institute. Retrieved July 20, 2013 from: <http://www.sans.org/reading-room/whitepapers/auditing/guide-security-metrics-55>
- [114] Peng, W. 2012. Analysis and Exploration of Related Issues on the Computer Network Security Based on Firewall and Anti-Virus Software. In *Processing in*

-
- Advanced Technology in Teaching-Proceedings of the 2009 3rd International Conference on Teaching and Computational Science (WTCS 2009)*, Springer Berlin Heidelberg, 45–49.
- [115] Performance Evaluation Group. GreatSPN Users Manual, version 2.0.2. Department of Computer Science, University of Torino, 2001 [Online]: <http://www.di.unito.it/~susi/DIDATTICA/SPC04-05/manual.pdf>
- [116] Pfleeger, S. L., and Bloom, G. 2005. Canning Spam: Proposed Solutions to Unwanted Email. *IEEE Security and Privacy*, 3(2), 40–47.
- [117] Purkait, S. 2012. Phishing Counter Measures and their Effectiveness – Literature Review. *Information Management & Computer Security*, 20(5), 382–420.
- [118] QFINANCE. 2013. Definition of Learning Curve. Retrieved July 17, 2013 from: <http://www.qfinance.com/dictionary/learning-curve>
- [119] Qureshi, M. A., and Sanders, W. H. 1994. Reward Model Solution Methods with Impulse and Rate Rewards: An algorithm and numerical results. *Performance Evaluation*, 20(4), 413–436.
- [120] Restle, F., and Greeno, J. G. *Introduction to Mathematical Psychology*. Addison-Wesley, Oxford, England, 1970.
- [121] Ross, S. M. *Introduction to Probability Models*, Academic Press, 1997.
- [122] Roy, A., Kim, D. S., and Trivedi, K. S. 2012. Scalable Optimal Countermeasure Selection Using Implicit Enumeration on Attack Countermeasure Trees. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN 12)*, IEEE Press, 1–12.
- [123] Roy, S., Ellis, C., Shiva, S., Dasgupta, D., Shandilya, V., and Wu, Q. 2010. A Survey of Game Theory as Applied to Network Security. In *Proceedings of the 43rd Hawaii International Conference on System Sciences*, Honolulu, HI, 1–10.

-
- [124] Rumelhart, D.E., and Norman, D.A. 1976. Accretion, Tuning and Restructuring: Three Modes of Learning. *ERIC Document Reproduction Service No. ED134902*, (Report no. 7602).
- [125] Sahner, R. A., Trivedi, K. S., and Puliafito, A. *Performance and Reliability Analysis of Computer Systems*. Kluwer Academic Publishers, Boston, 1995.
- [126] Sakamoto, Y., Jones, M., and Love, B. C. 2008. Putting the Psychology Back into Psychological Models: Mechanistic versus Rational Approaches. *Memory & Cognition*, 36(6), 1057–1065.
- [127] Sallhammar, K., Helvik, B. E., and Knapskog, S. J. 2005. Incorporating Attacker Behaviour in Stochastic Models of Security. In *Proceedings of the 2005 International Conference on Security and Management (SAM'05)*. Las Vegas, Nevada, USA, 79–85.
- [128] Sallhammar, K., Helvik, B. E., and Knapskog, S. J. 2006. On Stochastic Modelling for Integrated Security and Dependability Evaluation. *The Journal of Networks*, 1(5), 31–42.
- [129] Sallhammar, K., and Knapskog, S. J. 2004. Using Game Theory in Stochastic Models for Quantifying Security. In *Proceedings of the 9th Nordic Workshop on Secure IT Systems, November*, Espoo, Finland.
- [130] Sallhammar, K., Knapskog, S. J., and Helvik, B. E. 2005. Using Stochastic Game Theory to Compute the Expected Behaviour of Attackers. In *Proceedings of the 2005 International Symposium on Applications and the Internet Workshops. (Saint2005)*, 102–105
- [131] Sandhofer, C. M., and Dumas, L. A. A. 2008. Order of Presentation Effects in Learning Colour Categories. *Journal of Cognition and Development*, 9(2), 194–221.
- [132] Schank, R. C., Fano, A., Bell, B., and Jona, M. 1994. The Design of Goal-Based Scenarios. *The Journal of the Learning Sciences*, 3(4), 305–346.

-
- [133] Sheng, S., Wardman, B., Warner, G., Cranor, L., Hong, J., and Zhang, C. 2009. An Empirical Analysis of Phishing Blacklists. In *Proceedings of the Sixth Conference on Email and Anti-Spam* (Mountain View, California, USA, July 16–17, 2009).
- [134] Shiva, S., Roy, S., and Dasgupta, D. 2010. Game Theory for Cyber Security. In *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*.
- [135] Simmonds, A., Sandilands, P., and van Ekert, L. 2004. An Ontology for Network Security Attacks. In *Proceedings of the 2nd Asian Applied Computing Conference (AACC)*, Kathmandu, Nepal, Springer Lecture Notes in Computer Science, 317–323.
- [136] Stehman, S. V. 1997. Selecting and Interpreting Measures of Thematic Classification Accuracy. *Remote Sensing of Environment*, 62(1), 77–89.
- [137] Stewart, R. D., Wyskida, R. M., and Johannes, J. D. *Cost Estimator's Reference Manual*. New York, NY, Wiley, 1987
- [138] Stewart, W. J. *Probability, Markov Chains, Queues, and Simulation: The mathematical Basis of Performance Modelling*, Princeton University Press, 2009.
- [139] Strebe, M. *Network Security Jumpstart*, SYBEX, 2002.
- [140] Suto, T., Bradley, J. T., and Knottenbelt, W. J. Performance Trees: Expressiveness and Quantitative Semantics. In *Proceedings of the 4th International Conference on the Quantitative Evaluation of Systems (QEST '07)*, IEEE Computer Society, 41–50.
- [141] Tijms, H. C. *Stochastic Models: An Algorithmic Approach*, John Wiley and Sons, 1994.

-
- [142] Tim Anderson's ITWriting. Retrieved August 10, 2013 from: <http://www.itwriting.com/>
- [143] Von Ahn, L., Blum, M., and Langford, J. 2004. Telling Humans and Computer Apart Automatically. *Communication of the ACM*, 47(2), 56–60.
- [144] Von Ahn, L. 2005. Human Computation. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA.
- [145] Vosniadou, S., and Brewer, W. F. 1987. Theories of Knowledge Restructuring in Development. *Review of Educational Research*, 57(1), 51-67.
- [146] Wilson, B. *Systems: concepts, methodologies, and applications*. New York, NY, USA: John Wiley & Sons (2nd ed.), Inc., 1990.
- [147] Wittel, G. L., and Wu, S. F. On Attacking Statistical Spam Filters. 2004. In *Proceedings of the First Conference on Email and Anti-Spam. (CEAS 2004)* [Online]. Available: http://pdf.aminer.org/000/085/123/on_attacking_statistical_spam_filters.pdf
- [148] Wozniak, R. H. 1999. Introduction to Memory: Hermann Ebbinghaus (1885/1913). *Classics in the history of psychology*, Retrieved July 20, 2013 from: <http://psychclassics.yorku.ca/Ebbinghaus/wozniak.htm>
- [149] Wright, T. 1936. Factors Affecting the Cost of Airplanes. *Journal of Aeronautical Science*, 3(4), 122–128
- [150] Yan, J., and El Ahmad, A. S. 2007. Breaking Visual CAPTCHAs with Naïve Pattern Recognition Algorithms. In *Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC'07)*, FL, USA, IEEE computer society, 279–291.
- [151] Yeh, C. C., Wang, T. Y., and Fu, H. Y. 2011. Observation and Analysis on Spam Sending Behaviour. In *Proceedings of the Third International Confer-*

- ence on Communications and Mobile Computing (CMC)*. IEEE computer society, 19–22.
- [152] Yoshida, K., Adachi, F., Washio, T., Motoda, H., Homma, T., Nakashima, A., Fujikawa H., and Yamazaki, K. 2004. Density Based Spam Detector. In *Proceedings of the 2004 ACM international conference on Knowledge discovery and data mining (SIGKDD' 04)*, ACM, 486–493.
- [153] Zhang, B., Zhang, Y., and Lu, W. 2012. Hybrid Model of Self-Organizing Map and Kernel Auto-Associator for Internet Intrusion Detection. *International Journal of Intelligent Computing and Cybernetics*, 5(4), 566–581.