

UNIVERSITÀ DELLA CALABRIA

DIPARTIMENTO DI INGEGNERIA INFORMATICA, MODELLISTICA,
ELETTRONICA E SISTEMISTICA



Dottorato di Ricerca in
Information and Communication Technologies
XXX Ciclo

Tesi di Dottorato

**Malevolent Activities Detection
and
Cyber Range Scenarios Orchestration**

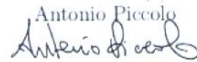
Antonio Piccolo

UNIVERSITÀ DELLA CALABRIA


Dottorato di Ricerca in
Information and Communication Technologies
XXX Ciclo

Tesi di Dottorato

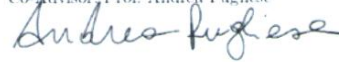
**Malevolent Activities Detection
and
Cyber Range Scenarios Orchestration**

Antonio Piccolo


Advisor: Prof. Domenico Sacca



Co-Advisor: Prof. Andrea Pugliese



Coordinatore: Prof. Felice Crupi



DIPARTIMENTO DI INGEGNERIA INFORMATICA, MODELLISTICA,

ELETTRONICA E SISTEMISTICA

Novembre 2017

Settore Scientifico Disciplinare: ING-INF/05

Acknowledgments

Ci tengo innanzitutto a ringraziare il Prof. Domenico Saccà, mio tutor di dottorato, ma anche la persona pi intellettualmente stimolante che abbia mai conosciuto. Mi ha sempre supportato in questi tre anni dispensando centinaia di idee interessanti, saggi consigli, attività appassionanti. Questo lavoro di tesi non sarebbe mai stato possibile senza il suo aiuto, supporto e pazienza. Ha fatto s che valesse veramente la pena di arrivare fino in fondo a questo viaggio. Grazie. Grazie al mio co-tutor, il Prof. Andrea Pugliese, per la grande fiducia riposta in me e la completa libertà che mi ha concesso nel seguire i miei temi di ricerca preferiti. Grazie a mia madre, che per tutto il tempo mi ha tenuto lontano dai problemi di salute di mio padre e dalle difficoltà di ogni giorno. Grazie per avermi incoraggiato e per aver creduto in me, senza riserve, fino ad oggi. Grazie al mio amore, Federica, che ha supportato la mia scelta di intraprendere il dottorato anche a costo di vivere lontani per qualche anno. Grazie ai miei amici, soprattutto a Luigi e Leonardo, per aver costantemente ispirato il mio lavoro e per avermi aiutato a crescere. Infine, ma non per importanza, grazie ad Angelo Furfaro, la sua preziosa collaborazione ha avuto un impatto significativo sul mio percorso di ricerca e sui risultati raggiunti.

Abstract

In the last few years, cyber-security has become a hot topic because of the ever-increasing availability of Internet accessible services driven by the diffusion of connected devices. The consequent exposition to cyber-threats demands for suitable methodologies, techniques and tools allowing to adequately handle issues arising in such a complex domain.

Most *Intrusion Detection Systems* are capable of detecting many attacks, but cannot provide a clear idea to the analyst because of the huge number of false alerts generated by these systems. This weakness in the IDS has led to the emergence of many methods in which to deal with these alerts, minimize them and highlight the real attacks. Furthermore, experience shows that the interpretation of the alerts usually requires more than the single messages provided by the sensors, so there is a need for techniques that can analyse the alerts within the context in which they have been generated. This might require the ability to correlate them with some other contextual information provided by other devices. Using synthetic data to design, implement and test these techniques is not fair and reliable because of the variety and unpredictability of the real world data. On the other hand, retrieving this information from real world networks is not easy (and sometimes impossible) due to privacy and confidential restrictions.

Virtual Environments, Software Defined Systems and *Software Defined Network* will play a critical role in many cyber-security related aspects like the assessment of newly devised intrusion detection techniques, the generation of real world like logs, the evaluation of skills of cyber-defence team members and the evaluation of the disruptive effects caused by the diffusion of new malware.

This thesis proposes, among other things, a novel domain-specific platform, named *SmallWorld*, aimed to easily design, build and deploy realistic computer network scenarios achieved by the immersion of real systems into a software defined virtual environment, enriched by Software Defined Agents put in charge of reproducing users or bot behaviours. Additionally, to provide validation and performance evaluation of the proposed platform, a number of

Scenarios (including penetration testing laboratories, IoT and domotics networks and a reproduction of the most common services on Internet like a DNS server, a MAIL server, a booking service and a payment gateway) have been developed inside *SmallWorld*. Over time the platform has been rewritten and radically improved leading to the birth of *Hacking Square*. This new version is currently available on-line and freely accessible from anyone. The impact of this research prototype has been demonstrated, above all, during the course of "*Metodi e Strumenti per la Sicurezza Informatica*" for the master degree in Cyber Security at DIMES, University of Calabria. In fact, the platform has been employed to build the laboratory of the course as an in cloud service for students (including all the material to conduct exercises and assignments) and to organize a, practical, *Capture the Flag (CTF)* like final test. Finally, the platform is under the attention of *Consorzio Interuniversitario per l'Informatica (CINI)*, as it could be used to manage and deploy training content for the CyberChallenge 2018.

Abstract

Negli ultimi anni si è sentito parlare sempre più di cyber-security a causa degli attacchi informatici condotti da criminali informatici contro infrastrutture critiche, personal computer, dispositivi mobili e contro l'Internet-of-Things. L'abbassamento dei costi di tali dispositivi e la diffusione della banda larga hanno contribuito significativamente ad accrescere i target vulnerabili e di conseguenza il numero di attaccanti e il loro profitto. Pertanto si richiedono metodologie, tecniche e strumenti per gestire adeguatamente le complesse problematiche del mondo della cyber-security.

Molti *Intrusion Detection Systems* sono capaci di rilevare diversi attacchi, ma non sono in grado di fornire agli analisti una chiara visione di cosa sta accadendo nei sistemi sotto attacco a causa di un'enorme quantità di falsi positivi generati dal sistema. Ci ha reso necessario lo studio di tecniche per minimizzare i falsi allarmi ed evidenziare i veri attacchi. Purtroppo per interpretare al meglio un alert è necessario integrare il messaggio proveniente dalla sonda con tutta una serie di informazioni estraibili dal sistema di riferimento che lo ha generato. Si palesa, quindi, che utilizzare dati sintetici per progettare, implementare e testare nuove tecniche non è un approccio corretto e affidabile a causa della varietà e imprevedibilità dei dati appartenenti a sistemi reali. D'altra parte, richiede o prelevare dati da sistemi reali è un'operazione molto delicata e talvolta impossibile a causa di leggi sulla privacy e restrizioni su dati potenzialmente confidenziali.

Tecnologie come *Virtual Environments*, *Software Defined Systems* e *Software Defined Network* giocheranno un ruolo chiave nella realizzazione di nuove tecniche di intrusion detection, generazione di log verosimili, la valutazione delle competenze di team di esperti di sicurezza e lo studio degli effetti potenzialmente distruttivi dovuti alla diffusione di nuovo malware.

Questa tesi propone una piattaforma innovativa, denominata *Hacking Square* (nome in codice *SmallWorld*), atta a semplificare e automatizzare la progettazione, realizzazione e dispiegamento di Cyber-Range, laboratori di penetration testing o in generale reti di computer virtualizzate quanto più realistiche possibili. A tale scopo sono stati sviluppati dei moduli, che creano all'interno

di ogni nuovo scenari, in modo completamente trasparente un DOMAIN NAME SYSTEM, un MAIL SERVER, un servizio di pagamento e un sistema di booking on-line. Per validare e valutare le performace della piattaforma proposta, sono stati creati un certo numero di Scenari di penetration testing, IoT e domotica. L'impatto di questo prototipo di ricerca stato dimostrato durante il corso di "Metodi e Strumenti per la Sicurezza Informatica" della laurea Magistrale in Ingegneria Informatica, indirizzo Cyber Security del DIMES, University of Calabria. Infatti, la piattaforma stata utilizzata per la realizzazione dei laboratori usati durante le esercitazioni del corso, fornendo agli studenti un accesso attraverso VPN, rendendolo accessibile ovunque e in qualsiasi momento, senza dover appesantire i loro personal computer. La prova pratica finale del corso, una *Capture the Flag (CTF)* stata organizzata anche grazie all'ausilio di Hacking Square. Infine, il *Consorzio Interuniversitario per l'Informatica (CINI)* sta valutando la piattaforma per gestire ed erogare le esercitazioni del corso formativo su cyber-security denominato CyberChallenge 2018.

Contents

List of Figures	x
List of Tables	xiii
1 Introduction	1
1.1 Contributions	2
1.2 Publications Related with this Thesis	3
1.3 Structure of the Thesis	5
2 Related Works	7
2.1 Introduction	7
2.2 Intrusion Detection and Prevention System	7
2.3 Next Generation Firewall	9
2.4 Malevolent Activity Detection	10
2.5 Graph-based Alert Correlation Frameworks	12
2.6 Virtual Environments for the Cyber-Security	13
3 MALICIOUS URL DETECTION	15
3.1 Introduction	15
3.2 The classification model	16
3.3 Spherical separation	18
3.4 Computational experience	20
4 The AC-Index: Fast Online Detection of Correlated Alerts	27
4.1 Introduction	27
4.2 Preliminaries and Problem Formalization	29
4.3 The AC-Index	32
4.4 Experimental Results	35
4.5 Summary	39

5	Malevolent Activity Detection with Hypergraph-Based Models	41
5.1	Introduction	41
5.1.1	Running Example	42
5.2	Modeling Malevolent Activities	44
5.3	Discussion: Features of the Hypergraph-Based Model	47
5.3.1	Synchronization Independency	47
5.3.2	Compactness	48
5.3.3	Modeling XORs and Contiguous Sequences	49
5.3.4	“Weak” Dependency	50
5.4	Consistency of Activity Models	50
5.5	Equivalence and Minimality of Activity Models	53
5.6	The Malevolent Activity Detection Problem	55
5.7	Indexing and Detecting Activity Instances	56
5.7.1	Insertion of Tuples and Retrieval of Instances	57
5.7.2	Performance of the AM-Index	62
5.7.3	Additional Information About the Experimental Setting	64
5.8	Summary	66
6	A Virtual Environment for realistic Cyber-Security Scenarios	67
6.1	Introduction	67
6.2	SmallWorld	69
6.2.1	Physical layer	69
6.2.2	Abstraction Layer	70
6.2.3	Core Service Layer	71
6.2.4	API Layer	71
6.2.5	Management and Control Layer	72
6.2.6	Scenario Development Tool	72
6.2.7	Virtual-System Development Tool	73
6.2.8	Agent Development Tool	73
6.2.9	Repositories	74
6.2.10	Management Dashboard	74
6.3	Case Studies and experimental results	75
6.3.1	Case study I : security assessment	75
6.3.2	Case study II: e-learning	77
6.4	Summary	81
7	Conclusions and Future Work	83
7.1	Ongoing and Future Work	83
7.2	Conclusions	85

A	Appendix	87
	A.1 Proof Sketch for Theorem 5.14	87
	A.2 Proof Sketch for Theorem 5.16	87
	A.3 Proof Sketch for Theorem 5.18	87
	References	89

List of Figures

4.1	Example patterns. Each stage is annotated with its associated severity value.	29
4.2	Example log (top left), network (top right), and sub-sequences (bottom).	31
4.3	Example initial index status (left) and Insert algorithm (right).	34
4.4	Example index status after indexing log tuples from 102 to 110 of the log of Figure 4.2 (top left).	34
4.5	Real-world patterns P_3 and P_4	36
4.6	Synthetic patterns P_5 and P_6	36
4.7	Parameter values used for each experimental run.	37
4.8	Tuple rates in the first round of experiments.	38
4.9	Tuple rates (top and center left), number of occurrences (center right), normalized indexing time per tuple (bottom left), and maximum size of the AC-Index (bottom right) in the second round of experiments.	39
4.10	Tuple rates (top left), number of occurrences (top right), normalized indexing time per tuple (bottom left), and maximum size of the AC-Index (bottom right) in the third round of experiments.	40
5.1	Example activity model (top) and log (bottom).	42
5.2	Log used in the example of Fig. 5.1.	46
5.3	Example scenario (top) and activity model (bottom). Every vertex of the hypergraph has cardinality constraint (1 : 1).	48
5.4	Graph-based model for the scenario of Fig. 5.3 (bottom).	49
5.5	Hypergraph- (top) and graph- (bottom) based models for the sequence of actions $\{a, b, c\} \rightarrow \{d, e, f\} \rightarrow \{g, h, i\}$. Every vertex of the hypergraph has cardinality constraint (1 : 1).	49
5.6	Graph-based model with XORs (left) and the corresponding hypergraph-based model (right).	50
5.7	Support graph associated with the path $h_1, h_2, h_3, h_4, h_5, h_6$	51

List of Figures

5.8	The graph reduction constructed from ϕ	52
5.9	Example activity model M_2	54
5.10	Status of the AM-Index after indexing the log of the running example up to tuple ℓ_{10}	57
5.11	AM.Insert (top) and AM.Retrieve (bottom) algorithms.	58
5.12	Evolution of the AM-Index when indexing tuples ℓ_{11} (top) and ℓ_{12} (bottom) of the log of the running example.	59
5.13	Alternative structure for the model of Fig. 5.5 (top).	61
5.14	Tuple rates (top) and maximum size of the AM-Index (bottom) in Rounds 1 and 2.	64
5.15	Tuple rates and maximum size of the AM-Index in Round 3.	65
5.16	Activity models used in Rounds 1 and 2.	65
5.17	Activity model for Function1 (some vertices, depicted in black, are represented twice for ease of presentation).	66
6.1	SmallWorld Architecture	69
6.2	Management Dashboard.	74
6.3	An example of scenario in SMALLWORLD.	76
6.4	Red Agent Activity Diagram.	77
6.5	Plugin Configuration types	78
6.6	An example of a scenario composed by more sub-scenarios	79

List of Tables

2.1	Features provided by previous approaches: support for order-independency (OR); support for temporal and/or cardinality constraints (TCC); rigorous characterization of consistency, equivalence, and minimality (CEM); availability of an efficient index structure (IS).....	12
3.1	First set of features	21
3.2	Most significant results with the first set of features.....	21
3.3	True Positive and True Negative Rate.....	22
3.4	Features combination using a bigger dataset	22
3.5	True Positive and True Negative Rate using a bigger dataset ...	23
3.6	Additional Features.....	23
3.7	Most significant results with two more features	23
3.8	True Positive and True Negative Rate for the best performing combination of features	24
3.9	Most significant results for dataset 2	24
3.10	True Positive and True Negative Rate for the best performing combination of features with dataset 2	25
5.1	Size of the log in Rounds 1 and 2 (default case in bold).....	63
6.1	Software and vulnerabilities.....	76

List of Abbreviations

API	Application Programming Interface
BS	Base-Station
CERT	Computer Emergency Response Team
CSRF	Cross Site Request Forgery
DDoS	Distributed Denial of Service
DNS	Domain Name Server
HW	Hardware
IDE	Integrated Development Environment
LFI	Local File Inclusion
MAC	Medium Access Control
P2P	Peer-to-Peer
RAM	Random Access Memory
RCE	remote Command Execution
RFI	Remote File Inclusion
ROM	Read Only Memory
RX	Reception
SDN	Software defined Networks
SDS	Software defined Systems
SNR	Signal-to-Noise Ratio
SOC	Security Operation Center
SQLi	SQL Injection
SSL	Secure Socket Layer
SW	SmallWorld
SW	Software
TX	Transmission
UML	Unified Modeling Language
VAPT	Vulnerability Assessment and Penetration Testing
VM	Virtual Machine
WPAN	Wireless Personal Area Network
XSS	Cross Site Scripting

Introduction

Internet and its underlying infrastructure are vulnerable to a wide range of risk resulting from both physical and software threats.

Sophisticated cyber actors and nation-states exploit vulnerabilities or organize phishing campaigns to steal information and money and are developing capabilities to denial, destroy, or threaten the delivery of essential services. Many of traditional crimes are now being perpetrated through cyberspace. These include banking and financial fraud, intellectual property violations, and other crimes, all of which have substantial human and economic consequences. In this scenario, a big issue is represented from IoT devices are becoming increasingly popular and due to their inherent vulnerabilities are taking part to the biggest DDoS network attacks. They are, also, becoming an attack vectors to our homes, garages and cars, while since some years ago it was required a physical tampering.

Cyberspace is particularly difficult to secure due to a number of factors: first of all, because security, by its very nature, is inconvenient, and the more robust the security mechanisms, the more inconvenient the process becomes. For example, one of the current trends in security is to add whole disk encryption to laptop computers. Although this is a highly recommended security process, it adds a second login step before a computer user can actually start working. Even if the step adds only one minute to the login process, over the course of a year this results in four hours of lost productivity. Some would argue that this lost productivity is balanced by the added level of security. But across a large organization, this lost productivity could prove significant. Other factors are the ability of malicious actors to operate from anywhere in the world, the linkages between cyberspace and physical systems, and the difficulty of reducing vulnerabilities and consequences in complex computer networks. Of increasing concern, is the cyber threat to critical infrastructure, which is increasingly subject to sophisticated intrusions that open to new risks. As information technology becomes more and more integrated with physical infrastructure operations, there is a growing risk for wide scale or high-consequence events that could cause harm or disrupt services upon which our economy and the

daily lives of millions of people depend. In light of the risk and potential consequences of cybercrime and cyberevents, strengthening the security and resilience of cyberspace in an effective way has become an important research topic.

1.1 Contributions

The first contribution of this thesis is the proposal of a binary classification method aimed at detecting malicious URL on the basis of some information both on the URL syntax and its domain properties. Our method belongs to the class of supervised Machine Learning models, where, in particular, classification is performed by using information coming from a set of URL's (samples in Machine Learning parlance) whose class membership is known in advance. The main novelty of our approach is in the use of a Spherical Separation-based algorithm, instead of SVM-type methods, which are based on hyperplanes as separation surfaces in the sample space. In particular we adopt a simplified Spherical Separation model which runs in $O(t \log t)$ time (t is the number of samples in the training set), and thus is suitable for large scale applications. We test our approach using different sets of features and report the results in terms of training correctness according to the well-established ten-fold cross validation paradigm.

The second contribution is an indexing technique for alert correlation that supports DFA-like patterns with user-defined correlation functions. Our *AC-Index* supports (i) the retrieval of the top- k (possibly non-contiguous) subsequences, ranked on the basis of an arbitrary user-provided severity function, (ii) the concurrent retrieval of sub-sequences that match any pattern in a given set, (iii) the retrieval of partial occurrences of the patterns, and (iv) the online processing of streaming logs. The experimental results confirm that, although the supported model is very expressive, the AC-Index is able to guarantee a very high efficiency of the retrieval process. This indexing technique has been employed in the third contribution, a hypergraph-based framework for modelling and detecting malevolent activities. The proposed model supports the specification of order-independent sets of action symbols along with temporal and cardinality constraints on the execution of actions. We study and characterize the problems of consistency checking, equivalence, and minimality of hypergraph-based models. In addition, we define and characterize the general activity detection problem, that amounts to finding all subsequences that represent a malevolent activity in a sequence of logged actions. Since the problem is intractable, we also develop an index data structure that allows the security expert to efficiently extract occurrences of activities of interest. The fourth, and last, contribution is the definition and implementation of a *Hypervisor Independent Platform for Orchestrating Cyber Range Scenarios*, named *Hacking Square* (code name *SmallWorld*), aimed both to easily man-

age courses, teaching materials, students and design, build, orchestrate and deploy realistic computer network scenarios achieved by exploiting state-of-the-art technologies like: *Software Defined Networks* (SDN), *Software Defined Systems* (SDS), virtualization and containers.

1.2 Publications Related with this Thesis

The research work related to this thesis has resulted in 6 publications, including 4 journal and 2 conference papers. In the following, a brief description of each publication is provided.

Malicious URL detection via spherical classification

This paper [1], published on Neural Computing and Applications journal, introduces a binary classification method aimed at detecting malicious URL on the basis of some information both on the URL syntax and its domain properties. The proposed method belongs to the class of supervised Machine Learning models, where, in particular, classification is performed by using information coming from a set of URL's whose class membership is known in advance. We adopt a simplified Spherical Separation model which runs in $O(t \log t)$ time (t is the number of samples in the training set), and thus is suitable for large scale applications.

The AC-Index: Fast Online Detection of Correlated Alerts

This paper [2], published in the International Workshop on Security and Trust Management conference, proposes an indexing technique for alert correlation that supports DFA-like patterns with user-defined correlation functions. Our *AC-Index* supports (i) the retrieval of the top- k (possibly non-contiguous) sub-sequences, ranked on the basis of an arbitrary user-provided severity function, (ii) the concurrent retrieval of sub-sequences that match any pattern in a given set, (iii) the retrieval of partial occurrences of the patterns, and (iv) the online processing of streaming logs. The experimental results confirm that the AC-Index is able to guarantee a very high efficiency of the retrieval process.

Malevolent Activity Detection with Hypergraph-Based Models

This paper [3], published in the IEEE Transactions on Knowledge and Data Engineering journal, is a significant extension of [2]. The index is used inside a hypergraph-based framework for modeling and detecting malevolent activities. The proposed model supports the specification of order-independent sets

of action symbols along with temporal and cardinality constraints on the execution of actions. We study and characterize the problems of consistency checking, equivalence, and minimality of hypergraph-based models. In addition, we define and characterize the general activity detection problem, that amounts to finding all subsequences that represent a malevolent activity in a sequence of logged actions.

SmallWorld: A Test and Training System for the Cyber-Security

This paper [4], published in the European Scientific Journal, introduces SmallWorld, a scalable software platform designed to reproduce realistic scenarios achieved by the immersion of real systems into a virtual environment with a fully integrated support for teaching with the aim to provide a venue for practical education in the learning and usage of all tools, techniques, and best practices employed to protect the confidentiality, integrity, authenticity, and availability of a designated information service.

A Virtual Environment for the Enactment of Realistic Cyber-Security Scenarios

This paper [5], published in the Cloud Computing Technologies and Applications (CloudTech) International Conference, is an extension of [4]. The paper presents an in depth description of the architecture and the technological aspects of SmallWorld, a scalable software platform designed to reproduce realistic scenarios achieved by the immersion of real systems into a software defined virtual environment.

Using virtual environments for the assessment of Cyber-Security issues in IoT scenarios

This paper [6], published in the Simulation Modelling Practice and Theory journal, is an extension of [4] and [5]. This paper describes an approach based on the exploitation of virtual environments and agent-based simulation for the evaluation of Cyber-Security solutions for the next generation of IoT applications in realistic scenarios. The effectiveness of the approach is shown by considering a concrete case study involving the cooperation of real and virtual smart devices inside a virtualized scenario, deployed inside SmallWorld, where security issues are first evaluated and then handled.

1.3 Structure of the Thesis

This thesis is organized as follows.

Chapter 2, contains an introduction to the most important Cyber-Security aspects, technologies and solutions related to the arguments discussed in this Thesis. It, also, provides the necessary background information and prerequisite material that are needed to understand the subsequent chapters.

In Chapter 3, a novel domain-specific technique for efficiently detect malicious URLs is presented. The most important contributions of this technique are described and analyzed. Furthermore, an in-depth performance evaluation has been carried out, and the main results reported.

Chapter 4, describes a technique whose objective is the fast retrieval of occurrences of given patterns in streams of events, where each event corresponds to a security alert. The *AC-Index*, allows the security expert to efficiently extract occurrences of activities of interest. Our *AC-Index* supports (i) the retrieval of the top- k (possibly non-contiguous) sub-sequences, ranked on the basis of an arbitrary user-provided severity function, (ii) the concurrent retrieval of sub-sequences that match any pattern in a given set, (iii) the retrieval of partial occurrences of the patterns, and (iv) the online processing of streaming logs. The experimental results confirm that, although the supported model is very expressive, the AC-Index is able to guarantee a very high efficiency of the retrieval process.

Chapter 5, describes a hypergraph-based framework for modeling and detecting malevolent activities. The proposed model supports the specification of order-independent sets of action symbols along with temporal and cardinality constraints on the execution of actions. We study and characterize the problems of consistency checking, equivalence, and minimality of hypergraph-based models. In addition, we define and characterize the general activity detection problem, that amounts to finding all subsequences that represent a malevolent activity in a sequence of logged actions. Since the problem is intractable, we also develop an index data structure, called AM-Index, and its associated maintenance and retrieval algorithms, that exploit temporal (and cardinality) constraints to prune partial instances of *hypergraph-based models* as soon as possible.

Chapter 6 includes the description of SmallWorld, a platform based on virtual environment for the enactment of realistic Cyber-Security scenarios, along with its architecture and technological aspect. It is reported some interesting research case studies of SmallWorld which include (i) the evaluation of Cyber-Security solutions for the next generation of IoT applications in realistic scenarios, (ii) the development and deploying of a penetration testing laboratory and (iii) improving education through the use of virtual environments.

Finally, Chapter 7 includes a summary of the main results of this thesis, along with some concluding remarks, and comments on possible future research directions that can derive from the work here presented.

Related Works

This chapter presents the state-of-the-art of hardware/software technologies and methodologies used everyday to contrast cyber-security threats, focusing, in particular, on the frameworks related to the contributions of this thesis.

2.1 Introduction

Retracing history of computers we can see that cyber-crime, as we know it today, was non-existent. Protection for users and security mechanisms would be built into the foundation of computer design, but this not happened because, at that time, designers had certainly no idea that computers might be connected together into an enormous worldwide web and exposed to intrusions or damages. The fundamental decisions in the early stages of computer design do not correspond to the needs of the present. To contrast these built-in vulnerabilities and consequent unattended behaviours, many mitigation solutions have been proposed, like Intrusion Detection or Prevention Systems, Firewall or Next Generation Firewall, Security Information and Event Management (SIEM), isolation through Virtualization, Network Analyser and Antivirus. This chapter presents a brief survey on hardware/software technologies and methodologies used everyday to contrast cyber-security threats, focusing, in particular, on the frameworks related to the contributions of this thesis.

2.2 Intrusion Detection and Prevention System

A firewall is the first line of defence, but it has limited visibility into the content while it makes traffic filtering decisions. Because a firewall is commonly deployed at the ingress and egress points of a network, all traffic paths will converge and traverse through the firewall. Therefore, the performance and scalability of a firewall affects the network as a whole. For this reason, although some firewalls may incorporate a DPI engine, a firewall is designed

to execute a limited set of actions against each packet, even when hardware acceleration is activated in the firewall. When an attack circumvents the firewall, an IDS extends the security coverage by inspecting the network and the end systems for evidence that corroborates whether some network events and security alerts were instigated by attacks or malicious infiltrations. An IDS generates alarms and reports to network management systems upon detecting abnormal or suspicious traffic. An IDS examines packets for signatures that are associated with known viruses, malware, and other malicious traffic. In addition to pattern scanning within the packets, an IDS analyses overall traffic patterns to detect anomalies and known attacks. Some examples of known attacks are denialofservice (DoS), port scanners that search for vulnerable network services, buffer overflow exploits, and selfpropagating worms. Examples of anomalies include malformed protocol packets and traffic patterns that deviate from the norm. An IDS is divided into two main categories: a networkbased intrusion detection system (NIDS) and a hostbased intrusion detection system (HIDS). NIDS and HIDS differ in where the IDS is deployed, which consequently dictates the types of data collected and analysed by that specific type of IDS. A NIDS monitors the activities of the entire network and examines both intranet traffic and Internetbound traffic. On the other hand, the firewall concentrates on traffic that flows into and out of the internal network to the Internet. The traditional NIDS scans packets against a database of signatures of known attacks. Similar to the open source IDS tool Snort, each signature in the data is often implemented as a matching rule. This signaturebased IDS runs the packets through these matching rules or signatures to detect attacks. Another approach is the statisticalbased or anomalybased NIDS, which is also known as the behaviorbased NIDS. With a statisticalbased NIDS, a profile of the network under protection is built over time, based on evolving historical data, which represents the norm of the network. Some examples of data collected and compiled into a profile that represents the network operating under normal conditions include the following: the number of new applications that are discovered per day on the network and the average traffic volume generated by each type of application; the average number of DNS queries transmitted from a specific IP address at a given time interval; the average overall aggregate throughput of the network; and the average number of HTTP transactions issued per minute from a specific IP address. Any deviation observed by the NIDS may be interpreted as anomalies or misuse that instigates responses as defined by corresponding security directives.

The key to the success of a signaturebased NIDS is the richness in the collection of the attack signatures. Identifying a unique and effective signature for a new attack, especially a complex attack, takes time to develop and evolve. As new attacks propagate across the networks and infrastructures, the signaturebased NIDS is incapable of detecting these attacks while the new signatures are being implemented. The success of the statisticalbased NIDS depends on the knowledge or heuristics of the network characteristics that are

considered as normal and serve as the baseline. Establishing the boundaries of normal network behaviour is challenging as the network fosters a wide range of protocols and applications and hosts a user base with a diverse spectrum of on-line behaviours that can trigger sporadic traffic patterns. A statistical-based NIDS can be effective against new attacks because new attacks can incite network behaviours that alarm the NIDS. A hostbased IDS (HIDS) is purposefully built, either for an operating system or for a specific application, and operates in individual end systems. The HIDS analyses the operating system process identifier (PID), system calls, service listeners, I/O and file system operations, specific application runtime behaviour, and system and application logs to identify evidence of an attack. Firewalls are called active protection systems because a firewall is in the path of all traffic, known as in-line deployment. This enables the firewall to examine live traffic, and when the firewall identifies an attack, it is capable of blocking that attack while it is in progress. In other words, upon detection, a firewall can prevent malicious traffic from reaching a targeted system. Intrusion detection systems can be categorized as passive protection systems because an IDS is typically connected to a SPAN (Switched Port Analyser) port on a network switch or to a network tap that duplicates packets for an entire link. While an IDS can also examine every packet, however, the packets under analysis have successfully passed through a firewall and cannot be filtered by the IDS; those packets may also have already reached the intended targets and enacted malicious activities. In other words, an IDS identifies an attack that may have already taken place, at which point the IDS begins to repair the damage by executing countermeasures, for example, sending alerts and notifications to monitoring and management systems. Unlike the passive network monitoring of an IDS, an IPS takes the active role of performing mitigation actions in realtime once attacks are detected. An IPS possesses all of the capabilities of an IDS, but an IPS is deployed physically in-line in the network, which enables the IPS to drop attack packets, reset TCP connections, or activate filters to block the source of the attack. An IPS can perform other functions such as configuring dynamic policies in security devices, such as a firewall, to interrupt the malevolent activity and prevent further damage to the network.

2.3 Next Generation Firewall

The most significant limitations of the traditional firewall are its inability to perform payload inspection and to distinguish applications. The concept of Unified Threat Management (UTM) gained visibility and momentum in 2004 to address the security gaps in firewalls, and to offer a solution for the lack of unified policy management across the various security control technology products commonly deployed together in an enterprise network. The UTM strategy is to combine multiple security features such as a firewall, NIDS, IPS, gatewaybased antivirus, and content filtering into a single platform or

appliance to offer multiple layers of security protection with simplified management and ease of policy implementation. The security posture continued to increase its focus on users and their applications, as the transformation in UTM took place in parallel. Gartner [7], an information technology research and advisory company, claimed to be the first to define the NextGeneration Firewall (NGFW). In its definition, the three key attributes of an NGFW are its ability to detect applicationspecific attacks, to enforce security policies, to intercept and decrypt SSL traffic. The NGFW includes all the capabilities of traditional firewall and incorporates the full functionality of a signaturebased IPS. Another key characteristic is its in-line deployment as a bumpinthewire. In addition, the NGFW can collaborate with external services to incorporate additional securityrelevant data and feeds to enhance its enforcement capabilities. NGFW definition has a large overlap with that of the UTM. The articulated differences have limited technical merits, and the deviations are largely a result of verbiage manipulation. The concept seems to be a desired by product of combining the UTM with the unique features of the secure proxy. The conceptualization of NGFW, with such a rich set of security features, processing network traffic at multigigabit wire speed, without any performance degradation, would be the ultimate goal of security system design architects and developers. However, firewall and proxy are fundamentally incompatible with respect to the policies each is designed to interpret and enforce. The process and method of application classification collides with the operation of proxy interception.

2.4 Malevolent Activity Detection

A large number of approaches to the problem of modeling and detecting malevolent behavior have been proposed so far. A comparative study is beyond the scope of this thesis – in this section, we review the relevant literature that is the closest in spirit to our approach described in Chapter 5.

In [8], an automatically-learned finite automaton is used for identifying malicious execution traces. An approach for detecting anomalous program behaviors is proposed in [9], where each node in a deterministic finite automaton represents a state in the program under inspection. In [10], AND-OR tree structures and automata theory are combined to model complex attacks with time dependencies. The resulting “enhanced attack tree” model is used for supporting an intrusion detection engine. In [11, 12] the “Bayesian attack graph”, which combines bayesian inference procedures with cycle-free graph structures, is used to analyze network vulnerability scenarios. An automated technique is proposed in [13] for generating and analyzing attack graphs based on symbolic model checking algorithms. Attack graphs are generally constructed by analyzing the dependencies among vulnerabilities and security conditions that have been identified in the target network [14, 15, 16], or for correlating intrusion alerts [17, 18]. The authors of [19] propose a cor-

relation algorithm that is capable of detecting multiple attack scenarios for forensic analysis. In [20], attack graphs are used for correlating, hypothesizing, and predicting intrusion alerts. A representation of groups of alerts with graph structures is proposed in [21] along with a method that automatically identifies frequent groups of alerts and summarizes them into a “suspicious” sequence of actions. A framework for managing network attack graph complexity through interactive visualization, which includes hierarchical aggregation of graph elements, is proposed in [22]. Attack graphs are used in [23] in combination with Hidden Markov Models to explore the probabilistic connections among system observations and actual states. A formal foundation of SAND attack trees, which are a popular extension of the well-known attack trees, is provided in [24]. They also introduce the SP-graphs that support order-independency among events. An indexing technique for alert correlation is proposed in [25] that supports DFA-like patterns with user-defined correlation functions. The current state of the art on attack modeling approaches based on directed acyclic graphs is discussed in [26], whereas a comprehensive survey of the use of attack graphs in security is presented in [27]. All these works adopt graph-based models, and only few of them incorporate temporal aspects, such as dynamic time variations and dependencies among actions (e.g. order or priority).

When basic forms of graphs are used to represent activities, edges define the sequential execution of adjacent vertices/actions – hyperedges correspond instead to arbitrary sets of actions which can be accomplished in any order. Thus, using hypergraphs is advantageous especially in those cases where a graph-based representation of activities becomes too large as the complexity of the attacker’s behavior increases. There is limited work in the literature exploiting hypergraphs for security analysis. Most of them use hypergraphs to model network topologies and security properties [28, 29, 30] or correlate alerts [31] in intrusion detection systems. In [32], hypergraphs are used to model the infrastructural components and their dependencies in the context of risk analysis, while [33] addresses the problem of detecting anomalous multivariate co-occurrences in generic event data. In [34], a logic-based framework for reasoning on violations of the logging infrastructure is proposed that represents attacks as directed hypergraphs.

The work presented in Chapter 5 is the first attempt at building a *unified* framework that (i) supports modeling order-independent sets of action symbols, (ii) provides primitives to specify temporal and cardinality constraints on the execution of actions, (iii) is rigorously characterized through an in-depth formal analysis of the problems of consistency, equivalence, minimality, and detection, and (iv) is equipped with an efficient index structure with its associated maintenance and retrieval algorithms. Table 2.1 summarizes the features provided by previous approaches.

Table 2.1. Features provided by previous approaches: support for order-independency (OR); support for temporal and/or cardinality constraints (TCC); rigorous characterization of consistency, equivalence, and minimality (CEM); availability of an efficient index structure (IS).

	OR	TCC	CEM	IS
Michael and Ghosh [8]	-	-	-	-
Sekar et al. [9]	-	-	-	-
Camtepe and Yener [10]	-	✓	-	-
Frigault et al. [12]	-	-	-	-
Sheyner et al. [13]	-	-	✓	-
Amman et al. [14]	-	-	✓	-
Albanese et al. [35]	-	-	-	✓
Albanese et al. [16]	-	✓	-	✓
Noel et al. [17]	-	-	-	-
Roschke et al. [19]	-	-	-	-
Wang et al. [20]	-	-	✓	-
Mao et al. [21]	-	-	-	✓
Noel and Jajodia [22]	-	-	✓	-
Zhang et al. [23]	-	-	-	-
Jhavar et al. [24]	✓	✓	-	-
Pugliese et al. [25]	-	✓	-	✓
Silva and Willet [33]	✓	-	-	-
Johnson et al. [34]	✓	-	-	-
This work	✓	✓	✓	✓

2.5 Graph-based Alert Correlation Frameworks

A number of interesting graph-based alert correlation frameworks has been proposed in the past. Attack graphs and finite automata have often been used for this purpose. [8] proposed a technique for identifying malicious execution traces with automatically-learned finite automata. [9] created an automaton-based approach for detecting anomalous program behaviors. Each node in the DFA represents a state in the program under inspection which the algorithm utilizes to learn “normal” data and perform detection. [36] proposes to increase the accuracy of the N-gram learning algorithm by using a DFA representation for intrusion detection via system call traces. In [37] a technique is presented to automatically produce candidate interpretations of detected failures from anomalies identified by detection techniques that use inferred DFAs to represent the expected behavior of software systems. [38] proposes an approach for the real-time detection of denial of service attacks using time-dependent DFAs.

[19] proposes a correlation algorithm based on attack graphs that is capable of detecting multiple attack scenarios for forensic analysis. In [20] attack graphs are used for correlating, hypothesizing, and predicting intrusion alerts. [21] proposes to represent groups of alerts with graph structures, along with a method that automatically identifies frequent groups of alerts and summarizes them into a suspicious sequence of activity. [15, 39] construct attack scenarios that correlate critical events on the basis of prerequisites and consequences of attacks. [40] focuses on the online approach to alert correlation by employing a Bayesian network to automatically extract information about

the constraints and causal relationships among alerts. Finally, [41] introduces a host-based anomaly intrusion detection methodology using discontinuous system call patterns.

Fusion-based correlation techniques make use of correlation functions in order to store, map, cluster, merge, and correlate alerts. [42] proposes a multisensor data fusion approach for intrusion detection. [43] suggests to design functions which recognize alerts corresponding to the same occurrence of an attack and create a new alert that merges data contained in those alerts. [44] presents a probabilistic approach to alert correlation by extending ideas from multisensor data fusion. Their fusion algorithm only considers common features in the alerts to be correlated, and for each feature they define an appropriate similarity function.

[45, 46] propose an event processing query language that includes iterations and aggregates as possible parts of patterns. Non-deterministic automata are used for pattern detection. [47] proposes a similar language with a (limited) support to negation. Its implementation focuses on multi-query optimization. [48] supports patterns with Kleene closure and event selection strategies including partition contiguity and “skip till next match”, but not the output of complete matches.

2.6 Virtual Environments for the Cyber-Security

Virtual Environments are employed everyday to test software updates, patch or server configurations, to reduce maintenance and hardware costs. In the last few years, virtualization technologies have gained an important role also in the information security field and many solutions based on this technology have been proposed. For example, the DeterLab testbed [49], a security and education-enhanced version of Emulab [50]. It offers scientific computing facilities for cybersecurity researchers engaged in research, development, discovery, experimentation, and testing of cybersecurity technology. DeterLab [51] allows to configure user and group accounts with assorted permissions. Each group can have its own pre-configured experimental environments made of physical machines running Linux, BSD, Windows, or other operating systems. Users running DeterLab experiments have full control of real hardware and networks running pre-built software packages.

The DeterLab platform also offer a solution based on virtualization technologies, however it is not cloud based and this limits its ability to reproduce large and complex scenarios. It supports the QEMU hypervisor [52], View OS containers [53] and OpenVZ process containers [54] that cannot share physical hardware and, in addition, there is also a limit on the number of containers that can be deployed on a physical node, as specified on the official documentation.

Most of the existing Cyber-Security assessment tools act on real systems, incurring in high costs and risk, and virtual laboratories support only static,

i.e. not editable, scenarios pre-built by developers and/or domain experts and do not allow for inclusion of real entities and traffic generation. In the following some of the main active projects on this subject are reported.

The *eLearningSecurity* platform [55] offers certifications, virtual labs and courses on Cyber-Security. It allows to use pre-built scenarios on the cloud or to load vulnerable Web applications to be tested within a sandbox. The scenarios are accessible by means of a VPN and new ones can be requested only by contacting the development team. *PENTESTIT* [56] was devised to emulate IT infrastructures of real companies, created for legal penetration-testing and for empowering penetration skills and abilities. Laboratories are always unique, contain the most recent known vulnerabilities and are created exclusively by the development team, limiting the customization of the experiments. The “grey box” methodology was adopted to conduct penetration testing, i.e. the participants receive information on the network infrastructure, in the form of schema or text description, to be used for testing. *The Hacker Accademy* [57] has a web-based platform for experiencing, and teaching information security from the hackers perspective. Users can practice in a virtual lab environment by downloading a virtual machine image or by accessing a on-demand cloud-based lab environment. The platform provides skills assessment quizzes and a certificate is awarded upon completion of a lesson. *Pentest laboratory* [58] offers a testing lab environment that includes all of the hosts, network infrastructure, tools, and targets necessary to practice penetration testing. However this solution is limited to a single scenario with four hosts, two networks and a firewall. In addition it is tied to GNU/Linux platforms. At last, *Offensive Security* [59] offers the following features: 1) Security Training and Certification using their GNU/Linux distribution named Kali. The laboratories can contain a number of simulated clients that can be exploited for experimenting with client side attacks; 2) Virtual Penetration Testing Labs that provide a safe virtual network environment designed to be attacked and penetrated as a means of learning and sharpening penetration testing skills; 3) Dedicated Hosted Virtual Labs for Corporations, which provides a dedicated environment with a management account; 4) Advanced Attack Simulation: based on information about a target company and its systems, the Offensive-Security team builds a simulation model of the target environment and of the potential attacks.

MALICIOUS URL DETECTION

This chapter describes a novel domain-specific technique for efficiently detect malicious URLs.

3.1 Introduction

A useful resource to prevent risks in computer security is provided by the so called black lists, which are data bases containing a typically large number of IP addresses, domain names and related URL's for suspicious sites in terms of generation of threats. A rich literature is available on the creation and usage of such lists (see e.g. [60]). If a URL is comprised into a black list, it is convenient to deviate the network traffic from it and, in fact, many Internet Service Providers (ISP) simply block all messages coming from it. The users who detect anomalies in messages or activities they consider suspicious often transfer the related information to appropriate web sites devoted to risk analysis.

Other possible way to compile black lists is the use of certain spam trap addresses which are diffused in the aim of being contacted by crawler spiders, typically used by phishers. As soon as one of such site address is contacted, the calling site is included into the black list.

Although black lists are rather useful, we cannot expect that they are exhaustive of all possible threats, either because the number of potentially dangerous site is extremely high or because the system is highly dynamic and it is almost impossible to keep any black list sufficiently updated.

Every time there exists any suspect about the malicious nature of a site the Whois service is able to provide some useful information in terms of IP, domain name and other characteristics related to it. Whois registers are publicly available and there exist online services providing upon request such information, in an appropriate form.

The very basic idea of [61] is to use the information available about a given set of URL's, in connection to the related Whois, to design a classifier based

on some machine learning technique. In particular one of the tools adopted is the well known SVM paradigm which, being suitable for supervised classification, requires that a sufficiently rich training set is available in advance. Such training set is constituted by a list of URL's labeled in the form malicious-non malicious. A set of both qualitative and quantitative features is defined and each URL is associated to a string of possible values of the features.

Of course different sets of features can be adopted for classification purposes and in next section we will describe in details the ones we have considered.

The differences of our approach w.r.t. the one previously cited [61] are twofold. As we are aimed at providing a methodology suitable for very large datasets too, we have confined ourselves to a limited number of features (e.g. we have used no "bag of words", which in general requires an explosion in the size of the sample space) and, on the other hand, we have adopted a low complexity algorithm, accepting in advance the possibility of obtaining less accurate classification performance w.r.t. the SVM approach which requires solution of a structured quadratic programming problem [62].

Following [61] we take into account in our classification model both lexical features of the URL and host information, as those provided by the Whois. As a classification tool, we adopt the spherical separation paradigm ([63], [64]), which differs from SVM basically because separation in the feature space is not pursued by means of a hyperplane, instead by a spherical surface (application of ellipsoidal surfaces has been introduced too in [65]). Of course goodness of the classification tool depends on the geometry of the sets to be separated and cannot be easily predicted. Our choice for spherical separation has been dictated by the availability of a simplified approach to it [66], where the centre of the separating sphere is calculated in advance and only the radius is optimised. Such approach allows us to calculate the classifier in $O(t \log t)$, where t is the size of the training set and, consequently, appears suitable for dealing with large scale applications.

3.2 The classification model

We present now the list of seven features that we used in our work to detect malicious URLs. For our purposes, we decided to not analyze the URL's page structure or its content: all the features have been generated using information derived from the general URL syntax (every URL consists of the following three parts: `<protocol>://<hostname>/<path>`). Some of the features used are strictly related to the lexical properties of the URL (intended as a character string), other to the properties of the URL's hostname (available thanks to a Whois query).

The features taken into consideration are:

1. **Number of subdomains.** This is the count of the subdomains which are detectable in the text of the URL. We don't consider only the URL's

- hostname but also the URL's path which is often used to redirect users to other dangerous web sites.
2. **URL age.** We consider the age in days of the URL's hostname (how long its content has been on the web). This is one of the most important features in malicious URL detection problem because dangerous sites usually have a very short life and are recently registered compared to safe sites. The registration date of URL's hostname is needed to calculate the value of this feature.
 3. **URL expiration.** We count the number of days remaining before the expiration date of URL's hostname. Also this feature is potentially useful since dangerous sites usually are registered for shorter time than safe ones.
 4. **Hostname Length.** This is the simply count of the textual characters forming the URL's hostname.
 5. **Path Length.** This is the simply count of the textual characters forming the URL's path. Excessive length is often correlated to suspicious re-addressing.
 6. **IP Address geographical location.** IP Addresses related to dangerous sites are usually located in specific geographical areas so we use this feature to express the information about the country location of the IP Address related to the URL's hostname. The list of countries used to evaluate this parameter has been reduced to just ten nations: Canada, China, Finland, France, India, Italy, Spain, Turkey, United Kingdom and United States of America. All other countries have been bundled together into a unique term, for the IP Address located in countries not included in our list. This is a categorical feature.
 7. **Presence or not of the word "Login".** Dangerous sites usually contain in the text of their web address specific terms in order to cheat the web users. Thus we have included such binary feature to report possible presence of the word "Login" in the text of the URL.

According to their definition, all the features taken into consideration (but "IP Address geographical location") can assume integer values greater than or equal to zero. As the number of features we have adopted is small, we have performed several experiments using different sets of features. According to the number of features used, the URL is represented in a different vector format. Using different sets of features it is important to detect the relevant ones, and to discard those unable to provide any significant contribution to the classification process.

The malicious URLs detection problem has been modelled as a binary classification problem and the predefined classes of URLs have been two: the set of malicious URLs (including URLs dangerous for the web users) and the set of benign URLs (including URLs safe for the web users).

3.3 Spherical separation

In many supervised machine learning problems the objective is to assign elements to a finite set of classes or categories. For a given set of sample points coming from two classes, we want to construct a function for discriminating between the classes. The goal is to select a function that will efficiently and correctly classify future points. Classification techniques can be used for data mining or pattern recognition, where many applications require a categorization.

The classical binary classification problem is to discriminate between two finite sets of points in the n -dimensional space, by a separating surface. The problem consists in finding a separating surface minimizing an appropriate measure of the classification error.

Several mathematical programming-based approaches for binary classification have been historically proposed [67, 68, 69]. Among the more recent ones we recall the support vector machine (SVM) technique [70], where a classifier is constructed by generating a hyperplane far away from the points of the two sets. By adopting kernel transformations within the SVM approach, we can obtain general nonlinear separation surfaces. In this case the basic idea is to map the data into a higher dimensional space (the feature space) and to separate the two transformed sets by means of one hyperplane, that corresponds to a nonlinear surface in the original input space.

Parallel to the development of SVM methods, the use of nonlinear separating surfaces in the dataset space, instead of hyperplanes, has received in recent years some attention. In particular, in our work, we have considered the spherical separation approach, characterized by the fact that the separation process takes place in the original input space and does not require mapping to higher dimension spaces.

More formally, let

$$\mathcal{X} = \{x_1, \dots, x_p\}$$

be a set of samples (or points) $x_i \in \mathbb{R}^n$. In the supervised learning, we assume that, in correspondence to any point x_i of \mathcal{X} , a label y_i is given. The case $y_i \in \mathbb{R}$ is known as “regression”, while, when the label y_i takes values in a discrete finite set, the task is a “classification” process. A particular case of the latter is the binary classification, where, for each i , the label y_i can assume only two possible values. The objective of the supervised learning is to predict the label of any new sample only on the basis of the information of the labeled points (the training set).

In particular, in the binary classification problems, we consider the following partition of \mathcal{X} into two nonempty sets:

$$\mathcal{X}_+ := \{(x_i, y_i) \mid x_i \in \mathbb{R}^n, y_i = +1, \quad i = 1, \dots, m\}$$

and

$$\mathcal{X}_- := \{(x_i, y_i) \mid x_i \in \mathbb{R}^n, y_i = -1, \quad i = m + 1, \dots, p\}.$$

In the spherical separation approach we define the set \mathcal{X}_+ *spherically separable* from \mathcal{X}_- if and only if there exists a sphere

$$S(x_0, R) \triangleq \{x \in \mathbb{R}^n | (x - x_0)^T(x - x_0) \leq R^2\}$$

centered in $x_0 \in \mathbb{R}^n$ of radius $R \in \mathbb{R}$, such that

$$y_i(\|x_i - x_0\|^2 - R^2) \leq 0 \quad i = 1, \dots, p. \quad (3.1)$$

In addition, when inequalities (3.1) are strictly satisfied, then the sets \mathcal{X}_+ and \mathcal{X}_- are strictly spherically separated, i.e.

$$\begin{aligned} \|x_i - x_0\|^2 &\leq (R - M)^2 \quad i = 1, \dots, m \\ \|x_i - x_0\|^2 &\geq (R + M)^2 \quad i = m + 1, \dots, p, \end{aligned} \quad (3.2)$$

for some M , the margin, such that $0 < M \leq R$. Setting $q \triangleq 2RM$ and $r \triangleq R^2 + M^2$, inequalities (3.2) become:

$$q + y_i(\|x_i - x_0\|^2 - r) \leq 0 \quad i = 1, \dots, p.$$

In general it is not easy to know in advance whether the two sets are strictly spherically separable; then in [66, 63, 64, 71] a classification error function has been defined in order to find a minimal error separating sphere.

In particular, in [66, 63] a separating sphere has been obtained by minimizing the following objective function:

$$z_{S_1}(x_0, r) = r + C \sum_{i=1}^p \max\{0, y_i(\|x_i - x_0\|^2 - r)\}, \quad (3.3)$$

with $M = 0$ and $r \geq 0$. C is a positive parameter giving the tradeoff between the minimization of the radius and the minimization of the classification error.

More precisely in [66] an ad hoc algorithm that finds the optimal solution in $O(t \log t)$, with $t = \max\{m, p - m\}$, has been presented for the case where the center x_0 is fixed. It basically consists of two phases: the sorting phase and the cutting phase. In the first one the sample points are sorted according to their distance from the center, while in the second one an optimal cut is found. The adopted simplification is rather drastic, nevertheless a judicious choice of the center (e.g. the barycenter of the set \mathcal{X}_+) has allowed to obtain reasonably good separation results at a very low computational cost.

In [63] the spherical separation problem has been tackled without any constraint in the location of the center by means of DCA (DC Algorithm) [72, 73], based on a DC (Difference of Convex) decomposition of the objective function. A similar DCA approach has been proposed in [71] for minimizing the following error function:

$$z_{S_2}(x_0, r) = r^2 + C \sum_{i=1}^p \max\{0, y_i(\|x_i - x_0\|^2 - r)\}^2, \quad (3.4)$$

with $M = 0$. The choice of z_{S_2} instead of z_{S_1} is motivated by the fact that using z_{S_2} allows a DC decomposition where all the computations in DCA are explicit.

Finally the DCA scheme has been used also in [64], for minimizing the following error function:

$$z_{S_3}(x_0, q, r) = C \sum_{i=1}^P \max\{0, y_i(\|x_i - x_0\|^2 - r) + q\} - q, \quad (3.5)$$

with $0 \leq q \leq r$ and $M \geq 0$, where the term $-q$ is aimed at maximizing the margin. Moreover, similarly to [66], also in [64] an ad hoc algorithm that finds the optimal solution in $O(t \log t)$ has been designed when, in function z_{S_3} , the center x_0 is fixed.

3.4 Computational experience

We present now the results of our numerical experiments in malicious URLs detection. We have adopted the spherical classification approach described in section 3.3. In particular we have coded in Matlab (on a Pentium V 2.60 GHz Notebook) the algorithm introduced in [66], which is able to find the optimal solution in case the centre of the separating sphere is fixed in advance.

For evaluating the effectiveness of our approach we have used a dataset obtained as follows. The samples have been randomly selected using some sources available on the web; in particular, for malicious URLs we used the on-line black list called PhishTank ([74]), a free community site where anyone can submit, verify, track and share phishing data. As for benign URLs we have collected our samples by developing a fast web crawler (a program that visits web pages and collects information) capable of gathering unique URLs (not duplicated) specifying the minimum and maximum length of the URL. The crawling phase started from the DMOZ Open Directory Project ([75]), a human-edited directory of safe web pages. The size of our dataset is 245 malicious and 384 benign URLs (dataset 1).

We have applied the standard ten-fold cross-validation protocol. The dataset has been divided into ten subsets of equal size and each subset has been validated using the classifier obtained by training executed on the remaining nine subsets. The classification accuracy is defined as the average percentage of well classified URLs (of both classes).

We have run the program for different values of the positive weighting parameter C . We have implemented two possible choices of the center x_0 , by selecting, respectively, the barycenter of the set of malicious URLs ($x_0^{(1)}$) and the barycenter of the set of benign URLs ($x_0^{(2)}$). However, after our evaluation, we have noticed that setting the center x_0 equal to the barycenter of the set of malicious URLs was the most performing choice.

We have executed first an exhaustive feature selection process on five out of the seven features analyzed in section 3.2: number of subdomains, URL age, URL expiration, hostname length and path length. In Tables 3.1 and 3.2 we report the most significant results obtained and the corresponding set of features. In Table 3.3 the True Positive and the True Negative Rate for the best performing combination of features are reported, where the True Positive Rate is defined as the percentage of benign URLs correctly classified as benign URLs and the True Negative Rate as the percentage of malicious URLs correctly classified as malicious URLs.

Table 3.1. First set of features

Feature Number	Description
1	Number of subdomains
2	URL age
3	URL expiration
4	Hostname length
5	Path length

Table 3.2. Most significant results with the first set of features

Feature Combination	Parameters	Average Training Set Correctness	Average Testing Set Correctness
2, 3	$C = 0.1, x_0^{(1)}$	86.3%	86.3%
2	$C = 10, x_0^{(1)}$	85.6%	85.0%
1, 2, 3	$C = 0.1, x_0^{(1)}$	83.7%	83.5%
2, 3, 4	$C = 0.1, x_0^{(1)}$	82.8%	82.2%

Table 3.3. True Positive and True Negative Rate

	Average Training Set True Positive	Average Training Set True Negative	Average Testing Set True Positive	Average Testing Set True Negative
Features: 2,3 $C = 0.1, x_0^{(1)}$	90.3%	80.1%	90.3%	80.0%
Features: 2 $C = 10, x_0^{(1)}$	88.2%	81.5%	88.0%	80.4%
Features: 1,2,3 $C = 0.1, x_0^{(1)}$	88.1%	76.6%	87.7%	76.7%
Features: 2,3,4 $C = 0.1, x_0^{(1)}$	87.4%	75.5%	87.0%	74.7%

The results of the feature selection process indicate that the best classification accuracy is obtained by considering a quite small subset of the available features, in particular the combination of the URL age and URL expiration features. The remaining one are were redundant and, sometimes, even misleading.

We have extended the numerical experiments applying our methodology to a bigger dataset constituted by 11975 URLs, 5090 malicious and 6885 benign (dataset 2). It has been constructed by adopting the same methodology as for dataset 1. The results are shown in the following Tables 3.4 and 3.5. Also in this case we report the subsets of features which have provided better results.

Table 3.4. Features combination using a bigger dataset

Feature Combination	Parameters	Average Training Set Correctness	Average Testing Set Correctness
2, 3	$C = 0.1, x_0^{(1)}$	83.9%	83.9%
2, 5	$C = 10, x_0^{(1)}$	83.4%	83.3%
2	$C = 10, x_0^{(1)}$	83.6%	83.4%
2, 3, 5	$C = 10, x_0^{(1)}$	83.3%	83.2%

Table 3.5. True Positive and True Negative Rate using a bigger dataset

	Average Training Set True Positive	Average Training Set True Negative	Average Testing Set True Positive	Average Testing Set True Negative
Features: 2,3 $C = 0.1, x_0^{(1)}$	86.1%	80.9%	86.0%	81.0%
Features: 2,5 $C = 10, x_0^{(1)}$	85.6%	80.5%	85.6%	80.2%
Features: 2 $C = 10, x_0^{(1)}$	85.7%	80.7%	85.8%	80.3%
Features: 2,3,5 $C = 10, x_0^{(1)}$	85.5%	80.3%	85.5%	80.2%

We have performed some additional experiments on both datasets by considering two more features, Table 3.7: the IP Address geographical-location and the presence of the “suspicious” word *Login* in the text of the URL. Since such information was not available for all samples, the size of the datasets has been reduced to 370 benign URLs and 210 malicious ones for dataset 1 and to 6432 benign URLs and 4520 malicious ones for dataset 2. As far as dataset 1, in Table 3.6 we report the most significant results obtained for this series of experiments and in Table 3.8 the True Positive and True Negative Rate for the best performing combination of features.

Table 3.6. Additional Features

Feature Number	Description
6	IP Address geographical-location
7	Presence or not of the word "Login"

Table 3.7. Most significant results with two more features

Feature Combination	Parameters	Average Training Set Correctness	Average Testing Set Correctness
2,7	$C = 0.1, x_0^{(1)}$	79.3%	78.6%
2,5,7	$C = 10, x_0^{(1)}$	76.8%	77.1%
2,5,6	$C = 10, x_0^{(1)}$	75.7%	75.7%
2,3,6	$C = 0.1, x_0^{(1)}$	75.6%	74.5%

Table 3.8. True Positive and True Negative Rate for the best performing combination of features

	Average Training Set True Positive	Average Training Set True Negative	Average Testing Set True Positive	Average Testing Set True Negative
Features: 2,7 $C = 0.1, x_0^{(1)}$	85.5%	68.6%	84.3%	68.6%
Features: 2,5,7 $C = 10, x_0^{(1)}$	81.9%	67.7%	82.4%	67.6%
Features: 2,5,6 $C = 10, x_0^{(1)}$	81.1%	66.3%	83.0%	62.9%
Features: 2,3,6 $C = 0.1, x_0^{(1)}$	82.5%	63.6%	84.0%	57.6%

The above results show that the two additional features didn't provide any positive contribution in terms of classification accuracy. We remark the role played by the URL age in this setting too.

As far as dataset 2 is concerned, the results are reported in Tables 3.9 and 3.10. The introduction of the additional features has provided, in this case, a slight improvement.

We observe, finally, that our methodology has provided a rather satisfactory quality in the classification process. It appears worth noting that, even for the larger dataset 2, the computation time has always been of the order of few seconds, which suggests possible utilization of our method at least as a preliminary classification tool in view of future very large scale applications.

Table 3.9. Most significant results for dataset 2

Feature Combination	Parameters	Average Training Set Correctness	Average Testing Set Correctness
2, 6	$C = 0.1, x_0^{(1)}$	84.4%	84.4%
2, 7	$C = 0.1, x_0^{(1)}$	84.0%	84.0%
2, 3, 6	$C = 0.1, x_0^{(1)}$	84.1%	84.0%
2, 5, 6	$C = 10, x_0^{(1)}$	83.6%	83.6%

Table 3.10. True Positive and True Negative Rate for the best performing combination of features with dataset 2

	Average Training Set True Positive	Average Training Set True Negative	Average Testing Set True Positive	Average Testing Set True Negative
Features: 2,6 $C = 0.1, x_0^{(1)}$	86.8%	81.0%	86.8%	81.0%
Features: 2,7 $C = 0.1, x_0^{(1)}$	86.5%	80.5%	86.5%	80.6%
Features: 2,3,6 $C = 0.1, x_0^{(1)}$	86.6%	80.6%	86.5%	80.5%
Features: 2,5,6 $C = 10, x_0^{(1)}$	86.1%	80.2%	86.1%	80.2%

The AC-Index: Fast Online Detection of Correlated Alerts

This chapter describes a technique whose objective is the fast retrieval of occurrences of given patterns in streams of events, where each event corresponds to a security alert. Our specifically-designed AC-Index supports a very expressive DFA-based model for the patterns and arbitrary correlation functions.

4.1 Introduction

Intrusion Detection Systems (IDSs) usually generate logs whose tuples encode timestamped security-related alerts that are recorded from a monitored system. In general, the *alert correlation* process transforms groups of such alerts into *intrusion reports* of interest for the security expert. Alerts typically contain attributes like the type of event, the address of the source and destination hosts, etc. These attributes are matched against known vulnerabilities, in order to avoid reporting alerts with no actual associated risk (e.g., a Linux-oriented attack blindly launched on a Windows machine). However, applying this approach alone can lead to missing relevant alerts that do not match any vulnerability (e.g., ICMP PINGs) but that can be part of a more complex multi-step attack. Alerts must therefore also be correlated using the knowledge encoded in specific structures (e.g. *attack graphs* [15]) that describe logical connections of interest among correlated alerts. In *anomaly detection* systems [76, 77, 78, 79, 80, 81], historical data is used to build profiles of the “normal” user behaviors, so that sequences of actions that deviate from the profiles are classified as “anomalous”. *Misuse detection* systems [82, 83, 84, 85, 86] make instead use of sets of descriptions of suspicious activities that are matched against the log in order to identify ongoing activities.

In order to describe logical connections among alerts, *multi-step* and *fusion-based* correlation techniques have been used in the past [87]. Multi-step correlation [88, 89, 90] seeks to identify suspicious activities that consist of multiple “steps” by modeling activities through attack graphs [82, 21, 19, 20] or deterministic finite automata (DFAs). Any activity that complies with a

graph or a DFA description is considered suspicious. Fusion-based correlation [88, 91, 90] uses instead similarity functions that, when applied to the attributes of incoming alerts, establish whether they should be considered part of a same activity.

The framework presented in this chapter provides the following main features:

- The general objective is that of retrieving the *top-k sub-sequences* of a log that match some given DFA-based pattern.
- The log is *streamed* into the system, so the retrieval of correlated alerts is performed in an *online* fashion.
- The ranking of the sub-sequences is done on the basis of user-provided *severity functions*.
- The retrieved sub-sequences can be constrained through user-provided *correlation functions* and *maximum durations*.
- The correlation and severity functions and the maximum durations are *pattern-specific* – moreover, the functions can be *arbitrary*, as we only mandate their polynomial-time computability.
- We *do not mandate any specific schema for the alerts*: we simply regard each alert as a relational tuple with a user-provided schema.
- We aim at managing *multiple patterns concurrently*.
- The retrieved sub-sequences can possibly be *non-contiguous*.
- The reports built can be based on *partial occurrences* of patterns, i.e., sub-sequences that have not yet reached their terminal stages in the DFAs.

Figure 4.1 shows the two patterns we will use as our running example throughout the chapter. Edges are labeled with alert symbols and each stage is annotated with its associated severity value. The sequence {*access, service exploit, DoS*} represents a possible *Denial of Service* attack. A security expert may want to take security measures at a certain “depth” of this attack. To this end, the expert wants to receive a report every time a stage of the sequence is traversed. In other words, we must look at all sub-sequences of the log that match some prefix of any path in the pattern. Furthermore, in order to counter the intrusions more quickly, the expert may want to only look at the first k sub-sequences, based on their associated severity value – in the example, we assume that the severity of a sub-sequence only depends on the stage reached in the pattern. Moreover, the correlation function looks at the attributes of the alerts in order to decide which alerts are to be considered part of a same attack. Finally, for each pattern, only the sequences that fit in a time window of maximum length τ are considered.

The rest of the chapter is organized as follows. In Section 4.2 we formalize the alert correlation problem targeted by our proposed AC-Index. Finally, in Section 4.3, we describe the AC-Index, which is then experimentally validated in Section 5.7.2.

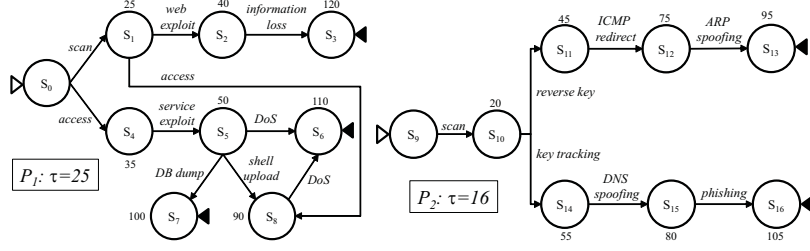


Fig. 4.1. Example patterns. Each stage is annotated with its associated severity value.

4.2 Preliminaries and Problem Formalization

In this section we introduce some preliminary notions and formalize the alert correlation problem targeted by our proposed index, which basically consists in finding the top- k sub-sequences of a log that represent an attack w.r.t. a given set of patterns.

We assume the existence of (i) a finite set \mathcal{A} of *alert symbols* and (ii) *attribute domains* ATT_1, \dots, ATT_w . A *log* is a set $L = \{\ell_1, \dots, \ell_n\}$ of tuples (each corresponding to an alert) of the form $\langle id, symbol, ts, att_1, \dots, att_w \rangle$ where id is an identifier, $symbol \in \mathcal{A}$, $ts \in \mathbb{N}$ is a timestamp, and $\forall i \in [1, w]$, $att_i \in ATT_i$. We assume that $\forall i \in [1, n-1]$, $\ell_i.ts < \ell_{i+1}.ts$. Moreover, we denote component c of log tuple ℓ as $\ell.c$.

The notion of a pattern is formalized by the following definition.

Definition 4.1 (Pattern). A pattern is a tuple $P = \langle S, s_s, S_t, \delta, \tau \rangle$ where:

- S is a set of stages;
- $s_s \in S$ is the start stage;
- $S_t \subseteq S$ is the set of terminal stages;
- $\delta : S \times \mathcal{A} \rightarrow S$ is the stage transition (partial) function;¹
- $\tau \in \mathbb{N}$ is the maximum duration of an occurrence of P .

We assume that $\forall s \in S_t, \forall sym \in \mathcal{A}$, $\delta(s, sym)$ is not defined, and that $\forall s \in S, \forall sym \in \mathcal{A}$, $\delta(s, sym) \neq s$.

In the following, when $\delta(s, sym) = s'$, we say that there is an edge from s to s' labeled with sym .

¹ Some past works assume acyclicity of the patterns because, in many practical cases, (i) the attacker’s control over the network increases *monotonically*, i.e., the attacker need not relinquish resources already gained during the attack, and (ii) the “criticality” associated with a sequence of alerts does not change when the sequence contains a portion that is repeated multiple times as it matches a cycle in the pattern. In such cases, the overall sequence is equivalent to the one obtained after removing the portion matching the cycle. We do not make this assumption as it would reduce the expressiveness of the model and it is not required by the AC-Index.

Example 4.2. Pattern P_1 of our running example is formalized as follows: $S = \{s_0, \dots, s_8\}$; $s_s = s_0$; $S_t = \{s_3, s_6, s_7\}$; $\delta(s_0, scan) = s_1$, $\delta(s_0, access) = s_4$, $\delta(s_1, access) = s_8$, $\delta(s_1, web\ exploit) = s_2$, $\delta(s_2, information\ loss) = s_3$, $\delta(s_4, service\ exploit) = s_5$, $\delta(s_5, DoS) = \delta(s_8, DoS) = s_6$, $\delta(s_5, DB\ dump) = s_7$, $\delta(s_5, shell\ upload) = s_8$; $\tau = 25$.

An occurrence of a given pattern is a possibly non-contiguous subsequence of the log whose associated alert symbols correspond to a path that begins in a start stage. In addition, the overall duration of the subsequence must comply with the maximum duration allowed by the pattern. The following definition formalizes this.

Definition 4.3 (Occurrence). *Given a pattern $P = \langle S, s_s, S_t, \delta, \tau \rangle$ and a log L , an occurrence of P in L is a set $O = \{\ell_1, \dots, \ell_m\} \subseteq L$ such that (i) $\forall i \in [1, m-1], \ell_i.ts < \ell_{i+1}.ts$; (ii) there exists a set $\{s_0, s_1, \dots, s_m\} \subseteq S$ such that $s_s = s_0$ and $\forall i \in [1, m], \delta(s_{i-1}, \ell_i.symbol) = s_i$; (iii) $\ell_m.ts - \ell_1.ts \leq \tau$.*

It should be observed that Definition 4.3 does not require an occurrence to reach a terminal stage. This feature gives security experts complete freedom in deciding whether or not a certain subsequence must be considered “critical” (i.e., with a high severity). Thus, any prefix of a complete path in the pattern can correspond to a critical subsequence the framework must take into account. Terminal stages are used to semantically represent the “final goal” of the attacker. Moreover, they help the retrieval algorithm as they signal that a subsequence can no longer be extended.

The following definition formalizes the way we characterize the severity of a subsequence and the attribute-based correlation among log tuples.

Definition 4.4 (Severity and Correlation Functions). *Given a pattern P and a log L , the severity w.r.t. P is a function $sev_P : 2^L \rightarrow \mathbb{N}$. Moreover, the correlation w.r.t. P is a function $\gamma_P : 2^L \rightarrow \{\text{true}, \text{false}\}$ such that $\gamma_P(X) = \text{true}$ for all subsets $X \subseteq L$ that, based on their attribute values, can be part of a same occurrence.*

We assume transitivity of function γ_P , that is, if $\gamma_P(X_1 \cup X_2) = \text{true}$ and $\gamma_P(X_2 \cup X_3) = \text{true}$, then $\gamma_P(X_1 \cup X_3) = \text{true}$. It should also be observed that it is natural to assume $sev_P(X) = 0$ when X is not an occurrence of P in L .

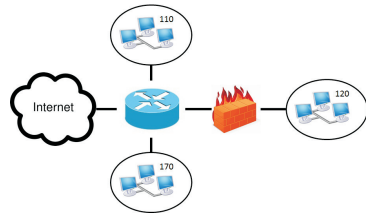
We are now ready to define the alert correlation problem we address.

Definition 4.5 (Alert Correlation Problem). *Given a set \mathcal{P} of patterns, a log L , and a number $k \in \mathbb{N}$, the alert correlation problem consists in finding a set $\mathcal{O} = \{O_1, \dots, O_k\}$ such that: (i) each O_i is an occurrence in L of a pattern $P_i \in \mathcal{P}$; (ii) $\forall i \in [1, k], \gamma_{P_i}(O_i) = \text{true}$; (iii) $\forall i \in [1, k-1], sev_{P_i}(O_i) \geq sev_{P_{i+1}}(O_{i+1})$; (iv) there do not exist a pattern $P \in \mathcal{P}$ and an occurrence $O \notin \mathcal{O}$ of P in L such that $sev_P(O) > sev_{P_k}(O_k)$.*

In Definition 4.5, the second condition states that all tuples in each occurrence $O_i \in \mathcal{O}$ must be correlated to one another; the third condition states that \mathcal{O} contains occurrences in decreasing order of severity value; the fourth condition ensures that the occurrences in \mathcal{O} are the ones with the top- k severity values. We do not assume that $\forall i, j$ with $i \neq j$, $P_i \neq P_j$ – in other words, set \mathcal{O} can contain two different occurrences of the same pattern. It should be noted that if the security expert is only interested in contiguous occurrences (as the majority of existing approaches do), our proposed framework can be straightforwardly extended to post-process the retrieved occurrences and filter out non-contiguous ones.

Example 4.6. Returning to our running example, suppose we want to find the occurrences of the patterns in the log of Figure 4.2 (top left). In this case, log tuples are of the form $\langle id, symbol, ts, sourceIP, targetIP \rangle$. We assume that γ_{P_1} and γ_{P_2} consider log tuples as correlated if their *sourceIP*s are equal and their *targetIP*s are in the same subnetwork w.r.t. the example network in Figure 4.2 (top right). Moreover, sev_{P_1} and sev_{P_2} return the values in Figure 4.1 if the *targetIP*s of the tuples are outside the firewall – values are doubled if the *targetIP*s are inside the firewall. The resulting sub-sequences are listed in Figure 4.2 (bottom), ordered by severity value.

id	symbol	ts	sourceIP	targetIP
100	scan	12	160.57.91.110	110.80.70.120
101	reverse key	13	160.57.91.110	110.80.70.120
102	scan	14	130.10.71.151	120.15.62.140
103	buffer overflow	15	190.23.41.170	170.21.88.124
104	web exploit	16	130.10.71.151	120.15.62.141
105	SQL injection	24	190.23.41.170	170.21.88.124
106	information loss	26	190.23.41.170	170.21.88.124
107	ICMP redirect	28	160.57.91.110	110.80.70.122
108	ARP spoofing	29	160.57.91.110	110.80.70.129
109	DoS	32	190.23.41.170	170.21.88.124
110	information loss	35	130.10.71.151	120.15.62.146



Sub-sequence	Pattern	Severity	Duration
$O_1 = \{102, 104, 110\}$	P_1	240	21
$O_2 = \{102, 104\}$	P_1	80	2
$O_3 = \{100, 101, 107\}$	P_2	75	16
$O_4 = \{100, 101\}$	P_2	45	1
$O_5 = \{102\}$	P_1	50	0
$O_6 = \{100\}$	P_2	25	0
$O_7 = \{100\}$	P_1	20	0
$O_8 = \{100, 101, 107, 108\}$	P_2	0	17

Fig. 4.2. Example log (top left), network (top right), and sub-sequences (bottom).

Note that O_8 is not an occurrence of P_2 according to Definition 4.3, since its duration is 17 time units which is longer than the maximum duration of any occurrence of P_2 (that is, 16 time units). The set $\mathcal{O} = \{O_1, \dots, O_4\}$ is

a solution for the alert correlation problem with $k = 4^2$. In fact, it satisfies all of the conditions of Definition 4.5: (i) each O_i has an associated pattern P_i for which it is an occurrence in L ; (ii) $\gamma_{P_1}(O_1) = \gamma_{P_2}(O_2) = \gamma_{P_3}(O_3) = \gamma_{P_4}(O_4) = true$; (iii) $sev_{P_1}(O_1) \geq sev_{P_1}(O_2) \geq sev_{P_2}(O_3) \geq sev_{P_2}(O_4)$; (iv) $sev_{P_1}(O_5) \leq sev_{P_2}(O_4)$ and $sev_{P_3}(O_6) \leq sev_{P_2}(O_4)$.

In the characterization of the complexity of the alert correlation problem we target, we make the realistic assumption that the computation of functions γ and sev can be performed in polynomial time. We therefore denote the complexity of computing such functions as $O(poly_{\gamma,sev}(x))$, that is a polynomial in the cardinality x of the set to which the functions are applied. The following result establishes the overall complexity of the problem.

Proposition 4.7. *The worst-case asymptotical time complexity of solving the alert correlation problem is $\Omega\left(\log k \cdot \sum_{P=\langle S, s_s, S_t, \delta, \tau \rangle \in \mathcal{P}} (\tau^{|S|} \cdot poly_{\gamma,sev}(\tau))\right)$.*

To see why the above result is true, it suffices to observe the following. τ is the maximum cardinality of an occurrence of P , so $\tau^{|S|}$ is the maximum possible number of occurrences of P in L . It should be observed that the existence of a “local time window” where alerts can be “connected” is common to all the models that allow to constrain the length of the sub-sequences (see, e.g., [83]) – obviously, without such constraints, this term would become $|L|^{|S|}$. Moreover, $poly_{\gamma,sev}(\tau)$ represents the time needed to check the correlation among the tuples of an occurrence of P and to compute their severity. Finally, to extract the top- k occurrences, it suffices to maintain a priority queue of maximum size k while scanning the whole set of occurrences – this takes time $\log k$ for each occurrence.

4.3 The AC-Index

In this section we describe our proposed AC-Index, whose objective is that of efficiently “tracking” the occurrences of a given set of patterns in a log. The index is updated as soon as a new log tuple enters the system, and it contains a priority queue whose content represents the top- k occurrences found so far in the log.

We denote the set of patterns as \mathcal{P} . Without loss of generality, we assume $\bigcap_{\langle S, s_s, S_t, \delta, \tau \rangle \in \mathcal{P}} S = \emptyset$. Moreover, we use \mathcal{S} to denote the set $\bigcup_{\langle S, s_s, S_t, \delta, \tau \rangle \in \mathcal{P}} S$. Finally, given an alert symbol $sym \in \mathcal{A}$, we define $stages(sym) \subseteq \mathcal{S}$ as the set of non-terminal stages having an incoming edge labeled with sym — formally, $\forall s \in stages(sym), \exists \langle S, s_s, S_t, \delta, \tau \rangle \in \mathcal{P}$ such that $s \in S, s \neq s_s, s \notin S_t$, and $\exists s' \in S$ such that $\delta(s', sym) = s$.

² Note that a security expert may want to discard O_2 and O_4 because they are prefixes of O_1 and O_3 respectively.

Definition 4.8 (AC-Index). Given a set \mathcal{P} of patterns on a log L , an AC-Index $I_{\mathcal{P}}$ is a tuple $\langle Tables, MainTable, PQ \rangle$ where:

- *Tables* is a set containing a table $table(s)$ for each $s \in stages(sym)$ with $sym \in \mathcal{A}$. $table(s)$ contains rows of the form (PL, sev) where PL is a list of pointers to tuples in L , and $sev \in \mathbb{N}$ is the severity value corresponding to the set of tuples pointed by PL ;
- *MainTable* is a table where each row is of the form (sym, Z) , where $sym \in \mathcal{A}$ and Z is a set of pointers to tables $table(s)$;
- *PQ* is a priority queue containing pairs of the form (PL, sev) that are copies of table rows in $tables(s)$. The size of *PQ* is bounded by k and the priority is the value of sev .

In the AC-Index, a row $(PL = \{\ell_0^\uparrow, \dots, \ell_m^\uparrow\}, sev) \in table(s)$ corresponds to an occurrence $O = \{\ell_0, \dots, \ell_m\}$ in L of a pattern $P = \langle S, s_s, S_t, \delta, \tau \rangle \in \mathcal{P}$ with $sev = sev_P(O)$ and $\delta(s', \ell_m.symbol) = s$ for some $s' \in S$. Following the definition of set *stages*, no table is built for neither initial stages (because such stages cannot correspond to occurrences) nor terminal stages (because we do not need to store non-extendable occurrences). In *MainTable*, a row (sym, Z) encodes the fact that, for each table $tables(s)$ pointed by Z there exists a stage $s \in S$ with at least one ingoing edge labeled with sym . Finally, *PQ* always contains the k occurrences found so far with higher severity values. Moreover, if requested by the security expert, *PQ* can be configured in such a way that it will discard the occurrences that are prefixes of some occurrence of the same pattern. In our running example, O_2 and O_4 would be discarded since they are prefixes of O_1 and O_3 , respectively.

Example 4.9. Figure 4.3 (left) shows the initial status of the AC-Index built over pattern $P_1 = \langle S, s_s, S_t, \delta, \tau \rangle$. At this stage, *PQ* and all $table(s)$ are empty. *MainTable* contains a number of rows equal to the number of distinct alert symbols labeling edges that end in non-terminal stages.

Figure 4.3 (right) shows the pseudo-code of the **Insert** algorithm that indexes a new log tuple ℓ_{new} with associated alert symbol $\ell_{new}.symbol$.

In the algorithm, Lines 6-9 deal with the case where s is a start stage, by creating a new occurrence. Specifically, it creates a new row table r and adds it to *PQ* and to $table(s')$, where s' is the stage reached from s by following the edge labeled with sym . Lines 11-20 check whether the new log tuple ℓ_{new} can be correlated with those in the existing occurrences. If it does (Lines 13-20), it is appended to such occurrences and the latter are added to *PQ*. Otherwise, i.e., if it does not fit in the time window τ , then the last log tuple of each occurrence that can not be extended is removed from its related table (Line 22). Observe that this implicitly corresponds to a pruning process that is applied during the construction of the index.

Example 4.10. Figure 4.4 shows the status of the AC-Index after indexing log tuples from 102 to 110 of our running example when considering pattern

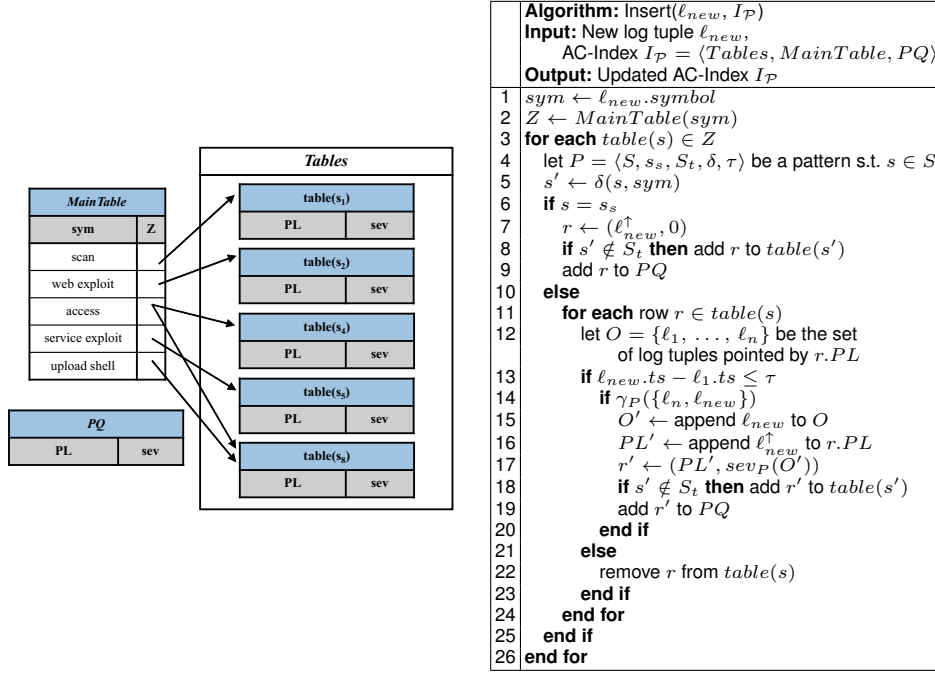


Fig. 4.3. Example initial index status (left) and Insert algorithm (right).

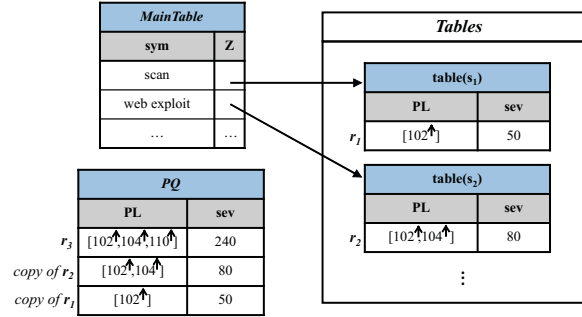


Fig. 4.4. Example index status after indexing log tuples from 102 to 110 of the log of Figure 4.2 (top left).

$P_1 = \langle S, s_s, S_t, \delta, \tau \rangle$ only. The indexing process can be divided into 3 distinct macro-steps:

1. The first processed log tuple is $\langle 102, scan, \dots \rangle$. Since there exists a row ($scan, \{table(s_1)^\uparrow\}$) in *MainTable*, row $r_1 = (PL = [102^\uparrow], sev = 50)$ is added to $table(s_1)$ (50 is the severity value returned by sev_{P_1}). Then, a copy of r_1 is added to *PQ*. Log tuple $\langle 103, buffer\ overflow, \dots \rangle$ is skipped because there are no rows in *MainTable* with $sym = buffer\ overflow$.

2. Log tuple $\langle 104, \text{web exploit}, \dots \rangle$ can be correlated with $\langle 102, \text{scan}, \dots \rangle$, because $\delta_{P_1}(s_1, \text{scan}) = s_2$ and $\gamma_{P_1}(102, 104) = \text{true}$. Thus, row $r_2 = ([102^\uparrow, 104^\uparrow], 80)$ is added to $\text{table}(s_2)$ and PQ . Log tuples from 105 to 109 are skipped because none of them can be correlated with log tuples 102 or 104. As an example, tuple $\langle 106, \text{information loss}, \dots \rangle$ cannot be linked to the occurrence $O = \{102, 104\}$ although $\delta_{P_1}(s_2, \text{information loss}) = s_3$, because $\gamma_{P_1}(\{102, 104, 106\}) = \text{false}$ due the *IPAttacker* attribute value, which is distinct from that of tuples 102 and 104.
3. Log tuple $\langle 110, \text{information loss}, \dots \rangle$ can be correlated with $\{102, 104\}$, because $\delta_{P_1}(s_2, \text{information loss}) = s_3$ and $\gamma_{P_1}(\{104, 110\}) = \text{true}$. However, in this case, a new row $r_3 = ([102^\uparrow, 104^\uparrow, 110^\uparrow], 240)$ is directly added to PQ because there does not exist $\text{table}(s_3)$ since s_3 is a terminal stage.

As the example shows, we only need to store occurrences in *Tables* if they can be extended. In fact, when an occurrence ends in a terminal stage it is no longer extendable, so it can be directly stored in PQ – this is why the AC-Index does not contain any $\text{table}(s)$ with s being a terminal stage. The following result ensures that Algorithm **Insert** solves the alert correlation problem both correctly and optimally.

Proposition 4.11. *Given a log L , the execution of Algorithm **Insert** on all tuples in L terminates, and after the execution, the content of PQ represents the correct solution to the alert correlation problem. The worst-case asymptotical time complexity of **Insert** is $O\left(\log k \cdot \sum_{P=\langle S, s_s, S_t, \delta, \tau \rangle \in \mathcal{P}} (\tau^{|S|} \cdot \text{poly}_{\gamma, \text{sev}}(\tau))\right)$.*

4.4 Experimental Results

In this section we report on the experimental assessment we performed on our proposed AC-Index when applied to both real-world and synthetic patterns and logs. We implemented the whole framework in Java and run the experiments on an Intel Core i7-3770K CPU clocked at 3.50GHz, with 12GB RAM, running Windows 8.

We ran three different rounds of experiments. In the first round, we used real-world patterns P_1 and P_2 of Figure 4.1 and P_3 and P_4 of Figure 4.5. In the second round, we used synthetic patterns P_5 and P_6 of Figure 4.6, in order to outline the behavior of our framework when varying the “density” of the patterns (number of edges w.r.t. the number of vertices). In fact, much denser patterns usually yield a much bigger AC-Index as each log tuple can be attached to many more occurrences.

For the first and second round, we built synthetic logs consisting of 300K tuples. Each log was built by combining a set of sub-logs, each of which is a sequence of alert symbols that can represent an occurrence of a given pattern. Specifically, a log combines several sub-logs $\{L_1, \dots, L_n\}$ where each L_i is built by considering a path from an initial to a terminal stage in a pattern.

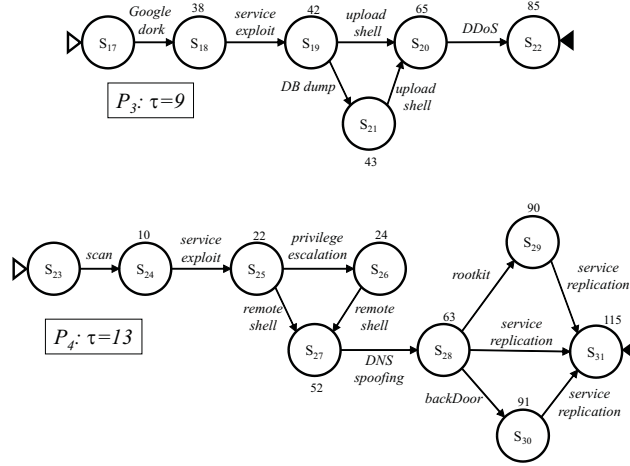


Fig. 4.5. Real-world patterns P_3 and P_4 .

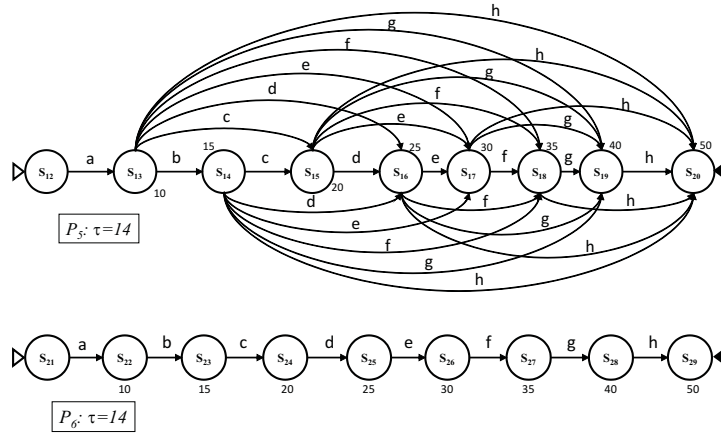


Fig. 4.6. Synthetic patterns P_5 and P_6 .

These sub-logs were built and combined under six different *log generation modes*, each corresponding to a possible real-world scenario:

1. each sub-log only contains alert symbols in its corresponding pattern, and the sub-logs are concatenated;
2. same as mode 1, except that some alert symbols are *replaced* with “noise”, i.e. with symbols not present in the corresponding pattern, with a certain frequency;
3. same as mode 1, except that noise is *inserted* in the sequence, i.e., it is added between alert symbols which are present in the pattern;

4. same as mode 1, except that a certain percentage of each L_i partially overlaps with L_{i+1} ;
5. same as mode 2, but with partial overlap as in mode 4;
6. same as mode 3, but with partial overlap as in mode 4.

We performed 14 runs for each of the first and second round. The log generation mode, noise frequency, and overlap percentage used are reported in Figure 4.7. For each run indicated in the figure, the values of the other parameters were set to the defaults (in bold) – for instance, run 3 was performed with noise frequency 3/10 and overlap percentage 40%.³ We assumed worst-case behavior of function γ , i.e., it always returns *true*. We also performed experiments with much larger logs (1M tuples) – interestingly, the performance we obtained in terms of tuples processed per second was 5.1% worse at most.

Log generation mode	Noise frequency	Overlap percentage
1 (run 1)	1/10 (run 7)	20% (run 12)
2 (run 2)	2/10 (run 8)	30% (run 13)
3 (run 3)	3/10 (run 9)	40% (run 14)
4 (run 4)	4/10 (run 10)	50% (run 15)
5 (run 5)	5/10 (run 11)	60% (run 16)
6 (run 6)		

Fig. 4.7. Parameter values used for each experimental run.

Finally, in the third round, we used a 112K-tuple log produced by running SNORT [92] on the second, fourth, and fifth week of inside traffic from the *1999 DARPA intrusion detection evaluation dataset* [93] and manually extracted 14 patterns from it. In this round, function γ was set to return *true* when the alerts shared the same destination IP address, and we fixed $\tau = 10$.

Figure 4.8 reports the results of the first round. In particular, Figure 4.8 (top left) shows the number of log tuples processed per second when varying the log generation mode (runs 1–6), Figure 4.8 (top right) shows the variation with respect to noise frequency (runs 7–11), and Figure 4.8 (bottom) the variation with respect to overlap percentage (runs 12–16).

The results confirm our expectations and show extremely good overall performances. As expected, the presence of noise in the log or overlap between consecutive instances reduces the overall number of occurrences, thus improving performances. Moreover, when noise appears *instead of* alert symbols of actual interest (which we believe is an even more realistic case), we obtain better results than when noise appears *between* such symbols. Generally, the number of tuples processed per second is extremely high – it is consistently higher than 765K, and around 1.4M on average. In both Figure 4.8 (top right) and Figure 4.8 (bottom) the trend is basically linear in the frequency of noise

³ For simplicity of presentation, the run with all parameters set to default values is reported as three separate runs (6, 9, and 14) in Figure 4.7.

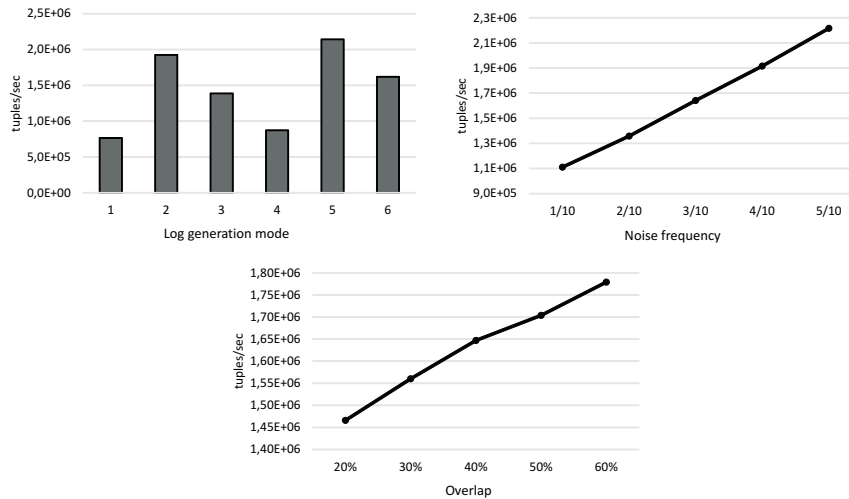


Fig. 4.8. Tuple rates in the first round of experiments.

and percentage of overlap – in these experiments, the average tuple rate is around 1.6M tuples/sec.

Figure 4.9 shows the results obtained in the second round, which again appear very satisfactory. We can notice in Figure 4.9 (top left) that the performance loss is always around 40% when moving from a sparse pattern (P_6) to a much denser one (P_5). The tuple rate never dropped below 260K tuples/sec, and it was around 700K tuples/sec on average. In the experiments where we fixed the log generation mode to 6 and varied noise frequency and overlap percentage (top right and center left of the figure) the performance loss was always around 60%. It should be observed that the number of paths in P_5 is 64 times that of P_6 . Thus, the relationship between the number of paths and the tuple rates is much less than linear.

For the second round, we also measured the number of occurrences and the indexing time per tuple normalized by the number of occurrences. As expected (Figure 4.9 (center right)), the number of occurrences is lower when using P_6 . Interestingly, the normalized indexing time (bottom left) shows very small variations with respect to the specific configuration used (8% on average). Finally, the maximum size of the AC-Index (bottom right) using P_5 is much larger – the difference was around 60% on average (again, showing a sub-linear relationship with the number of paths in the patterns). Moreover, in this case the size of the AC-Index shows very small variations with respect to the configuration used.

Finally, Figure 4.10 shows the results of the third round of experiments, when varying the number of patterns considered. Here, the tuple rate (top left) is consistently higher than 140K tuples/sec, and around 410K tuples/sec

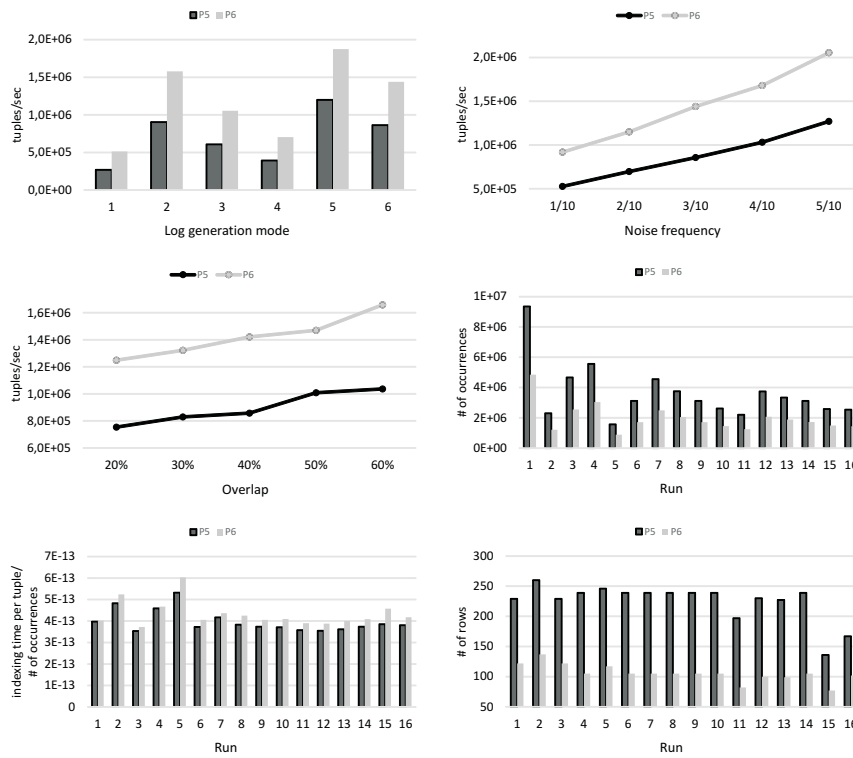


Fig. 4.9. Tuple rates (top and center left), number of occurrences (center right), normalized indexing time per tuple (bottom left), and maximum size of the AC-Index (bottom right) in the second round of experiments.

on average – again, it appears closely dependent on the actual number of occurrences in the log (top right). Interestingly, the normalized indexing time (bottom left) shows relatively small variations even with respect to the number of patterns used. As expected, the maximum size of the AC-Index (bottom right) is larger when indexing for more patterns – however, its size is always kept under 140 rows.

4.5 Summary

In this chapter we proposed an indexing technique for alert correlation that supports DFA-like patterns and user-provided correlation functions and provides very fast retrieval of occurrences of the patterns. The experimental results have proven that, although the supported model is very expressive, the framework is able to guarantee a very high efficiency of the retrieval process. It is capable of processing logs that enter the system at extremely large

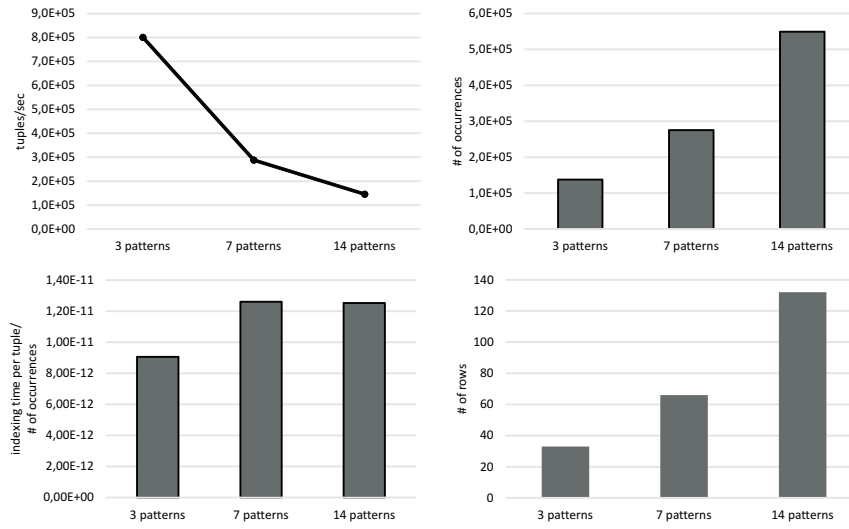


Fig. 4.10. Tuple rates (top left), number of occurrences (top right), normalized indexing time per tuple (bottom left), and maximum size of the AC-Index (bottom right) in the third round of experiments.

rates – orders of magnitude of 100K–1M tuples/sec are definitely sufficient for fully covering a wide range of real-world applications. Moreover, the framework scales well w.r.t. the amount of noise in the log, the overlap between consecutive occurrences, and the number of occurrences retrieved.

Malevolent Activity Detection with Hypergraph-Based Models

5.1 Introduction

In numerous security scenarios, given a sequence of logged *actions*, it is necessary to look for all subsequences that represent a *malevolent activity*. Typical scenarios where the *malevolent activity detection* problem is of prominent importance are those where actions correspond to interdependent security vulnerabilities, exploits, or alerts in a computer network. However, the problem is also crucial in other scenarios – for instance, when looking at data recorded by a surveillance system or when analyzing binary code to identify malware.

Many interesting graph-based approaches have been proposed in the past for modeling malevolent activities. Graphs are a very natural formalism for representing multi-step activities. However, the most common form of graph (*simple directed graph*, that is a set of vertices and a set of edges that are ordered pairs of vertices) is usually employed – this form does not capture well those scenarios where an activity can appear in multiple “forms”, depending on the *order* in which the actions it is composed of are performed. For instance, attacks to enterprise networks can correspond to non-predetermined sequences of actions; alert correlation processes are typically applied to alerts that are stored in different, possibly non-synchronized logs; polymorphic malwares change the order of binary instructions to overcome signature-based detection.

After a set of malevolent activity models are defined, the main problem is that of analyzing a log of actions in order to check whether it contains sequences that fit the models – in their simplest form, logs are sequences of tuples, each having (at least) an associated action and a timestamp. The ability to detect such *instances* of the activities in a log, and the efficiency in doing so, is clearly crucial in many security scenarios.

We propose a *hypergraph-based* framework for modeling and detecting malevolent activities. We summarize the main contributions below.

- First, we propose a hypergraph-based model for malevolent activities. The model is capable of representing many different kinds of activities in a compact way, by allowing the security expert to include (i) *order-independent sets of action symbols* along with (ii) *temporal constraints* on the execution of sets of actions and (iii) *cardinality constraints* on the number of occurrences of the actions (these capabilities, in turn, enable a host of useful features, some of which are discussed in Section 5.3).
- Second, we provide a *formal in-depth treatment* of the problems regarding (i) consistency, equivalence, and minimality of hypergraph-based models, and (ii) detection of activity instances.
- Third, we develop an index structure specifically designed for hypergraph-based activity models, along with its associated maintenance and retrieval algorithms, that allow the *efficient identification of instances of multiple models in a given log* (besides other interesting “minor” features, that are quickly discussed in Section 5.7).

5.1.1 Running Example

Consider the example log shown in Fig. 5.1 (bottom) and assume that the security expert identifies the log as an instance of an attack. In particular,

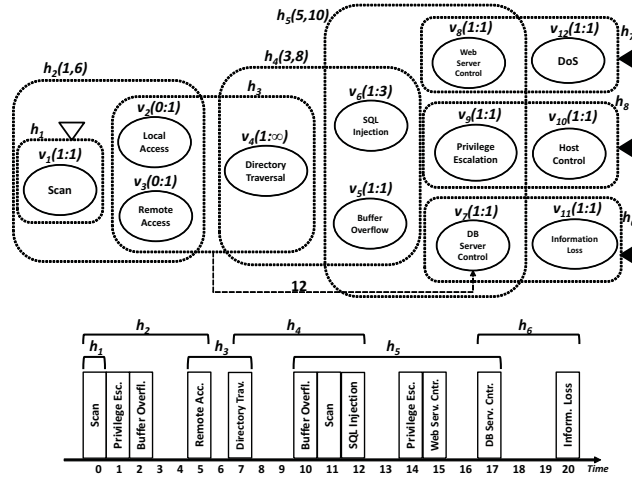


Fig. 5.1. Example activity model (top) and log (bottom).

the security expert tells us the log contains an attack because (i) it starts with exactly one **Scan** action followed by a **Remote Access** action, and these actions appear in a “window” which lasts between 1 and 6 time units; (ii) after the above, the log contains one **Directory Traversal** action; (iii) then, the log contains exactly one **Buffer Overflow** action and between 1 and 3 **SQL**

Injection actions (in any order), and the last three actions appear in a window which lasts between 3 and 8 time units; (iv) the last two actions are followed by a **Web Server Control** action, a **Privilege Escalation** action, and a **DB Server Control** action (in any order), the last five actions appear in a window which lasts between 5 and 10 time units, and the **DB Server Control** action appears at most 12 time units after the **Remote Access** action; (v) then, the log contains an **Information Loss** action.

In addition, the security expert could provide us with further information about *other attacks having a similar structure*. For instance, he could tell us that (i) a **Local Access** action could be present instead of (or together with) the **Remote Access** action; (ii) any number of **Directory Traversal** actions can be present; (iii) **DoS** and **Host Control** actions can conclude the attack instead of the **DB Server Control** action.

This kind of information about the structure of attacks is usually derived from historical statistical data and enriched with the specific technical knowledge provided by the domain expert. It should be observed that too restrictive choices about the possible number of occurrences of the actions and about the duration of temporal windows could expose the security expert to the possibility of missing attack instances in the presence of “random” actions in the log. An effective model must therefore support forms of constraints that, while giving full freedom to the security expert, provide the possibility of relaxing them when needed (for instance, by enlarging the duration of temporal windows at will).

An activity model that captures the above scenario is reported in Fig. 5.1 (top). Vertices (depicted with plain circles) correspond to the actions of interest while hyperedges (represented with dotted rounded boxes) represent correlated groups of vertices.

Hyperedge h_1 is a *start* hyperedge, which is indicated by the white arrow. This means that an activity instance must start with a **Scan** action (associated with vertex v_1). Vertices can have a cardinality constraint that specifies lower and upper bounds on the number of occurrences of their associated action in the instance. In this case, the constraint “(1:1)” states that the instance will contain precisely one **Scan** action.

Hyperedge h_2 specifies that vertices v_1 , v_2 , and v_3 are correlated: the model requires the instance to contain optional actions **Local Access** and **Remote Access** besides the **Scan** action from hyperedge h_1 – optionality is expressed through the cardinality constraints “(0:1)”. In general, the sub-sequence of an activity instance that contains the actions included in a hyperedge is called a *segment* of the instance. The model prescribes no specific order of appearance among the actions in the segment. In addition, hyperedges can have a temporal constraint that specifies lower and upper bounds on the number of time units the corresponding segment lasts. In this case, the constraint “(1,6)” states that all the actions in h_2 must be performed, no matter of the ordering, between 1 and 6 time units.

The other hyperedges impose similar constraints – however, hyperedges h_6 , h_7 , and h_8 are *terminal* (indicated with the black arrows), meaning that an instance must end with segments corresponding to these hyperedges.

The model supports further temporal constraints *between* hyperedges (represented by dashed edges in the figure). Such constraints specify upper bounds on the time that can pass between the start of the segment corresponding to the first hyperedge and the start of the segment corresponding to the second hyperedge. In this case, the constraint between h_3 and h_6 requires that the segment corresponding to h_6 starts at most 12 time units after the one corresponding to h_3 .

Fig. 5.1 (bottom) shows how the segments of the instance correspond to the hyperedges in the example model. It is easy to verify that the segments indicated in the figure satisfy all the constraints dictated by the model. For instance, the segment corresponding to h_2 contains one **Scan** action and one **Remote Access** action, so it satisfies the cardinality constraints over the vertices in h_2 , and it lasts $5 - 0 = 5$ time units, so it satisfies the temporal constraint over h_2 . The same applies to the segments corresponding to all of the other hyperedges. In addition, the time between the start of the segment corresponding to h_3 and that of h_6 is $17 - 5 = 12$, so the instance satisfies the temporal constraint between h_3 and h_6 . Moreover, the segments in the figure allow to trace a path from the start hyperedge h_1 to the terminal hyperedge h_6 .

5.2 Modeling Malevolent Activities

We start by defining activity models. Besides the topological structure (vertices and hyperedges), the definition identifies sets (S and T) of start and terminal hyperedges, a function (λ) that assigns an action symbol to each vertex along with lower and upper bounds on the number of occurrences of the symbol, a function (τ) that specifies lower and upper bounds on the temporal extension of each hyperedge, and a function (ϵ) that specifies upper bounds on the difference between the start times of different hyperedges. We assume that an alphabet \mathcal{A} of symbols is given, univocally identifying the actions of interest.

Definition 5.1 (Activity Model). *An activity model defined over the set \mathcal{A} of actions is a tuple $M = \langle \mathcal{H}, \lambda, \tau, \epsilon, S, T \rangle$ where:*

- $\mathcal{H} = (V, H)$ is a hypergraph, where V is a finite set of vertices and H is a set of hyperedges (i.e., for each $h \in H$, $h \subseteq V$).
- $\lambda : V \rightarrow \mathcal{A} \times \mathbb{N}^0 \times (\mathbb{N}^+ \cup \{\infty\})$ is a vertex labeling function that associates with each vertex $v \in V$ a triple of the form (a, l, u) , with $l \leq u$, which specifies the action of v along with its cardinality constraints.¹

¹ The intended meaning of the symbol ‘ ∞ ’ is that there is no upper bound.

- $\tau : H \rightarrow \mathbb{N}^0 \times (\mathbb{N}^+ \cup \{\infty\})$ is a (partial) function that expresses temporal constraints on hyperedges – $\text{domain}(\tau)$ will denote the set of hyperedges $h \in H$ such that $\tau(h)$ is defined;
- $\epsilon : H \times H \rightarrow \mathbb{N}^+$ is a (partial) function that expresses temporal constraints on ordered pairs of distinct hyperedges – $\text{domain}(\epsilon)$ will denote the set of pairs of hyperedges $h_i, h_j \in H$ such that $\epsilon(h_i, h_j)$ is defined;
- $S, T \subseteq H$ are nonempty sets of start and terminal hyperedges, respectively.

The following example shows how the model of our running example is formalized according to Definition 5.1.

Example 5.2. In the activity model $M = \langle \mathcal{H}, \lambda, \tau, \epsilon, S, T \rangle$ of Fig. 5.1 (top) we have:

- $V = \{v_1, \dots, v_{12}\}$;
- $H = \{h_1 = \{v_1\}, h_2 = \{v_1, v_2, v_3\}, h_3 = \{v_2, v_3, v_4\}, h_4 = \{v_4, v_5, v_6\}, \dots\}$;
- $\lambda(v_1) = (\text{Scan}, 1, 1)$, $\lambda(v_2) = (\text{Local Access}, 0, 1)$, $\lambda(v_3) = (\text{Remote Access}, 0, 1)$, $\lambda(v_4) = (\text{Directory Traversal}, 1, \infty)$, $\lambda(v_5) = (\text{Buffer Overflow}, 1, 1)$, etc.;
- $\text{domain}(\tau) = \{h_2, h_4, h_5\}$, $\tau(h_2) = (1, 6)$, $\tau(h_4) = (3, 8)$, $\tau(h_5) = (5, 10)$;
- $\text{domain}(\epsilon) = \{(h_3, h_6)\}$, $\epsilon(h_3, h_6) = 12$;
- $S = \{h_1\}$, $T = \{h_6, h_7, h_8\}$.

A path in an activity model is a sequence of overlapping hyperedges beginning with a start hyperedge – a complete path is one that ends with a terminal hyperedge.

Definition 5.3. A path π in an activity model $M = \langle \mathcal{H}, \lambda, \tau, \epsilon, S, T \rangle$ is a sequence h_1, \dots, h_m of distinct hyperedges from \mathcal{H} such that $h_1 \in S$ and $\forall i \in \{2, \dots, m\}$, $h_{i-1} \cap h_i \neq \emptyset$. Moreover, π is said to be complete if $h_m \in T$.

A log is a sequence ℓ_1, \dots, ℓ_n , with $n > 0$ and where each ℓ_i is a tuple $\langle att_1, \dots, att_k \rangle$ of attributes (e.g., user-id, IP, etc.). A sub-log of a log L is a subsequence of L (and therefore a log itself). In the following, we assume that a ‘timestamp’ attribute, here just formalized as a natural number, encodes the time point (w.r.t. an arbitrary but fixed time granularity) at which the action represented by a log tuple occurs. Moreover, for each $i, j \in \{1, \dots, n\}$ with $i < j$, it holds that $\ell_i.\text{timestamp} < \ell_j.\text{timestamp}$, i.e., the sequence reflects the temporal ordering of the tuples.² Moreover, we assume that an ‘action’ attribute encodes the action represented by a tuple.

Before defining the instance of an activity model, we introduce the notion of *m-segmentation*. As previously discussed, in order to be considered an instance, a log must necessarily contain a set of segments, each corresponding to a hyperedge in the model. An *m-segmentation* identifies such segments and, since the hyperedges in a path are overlapping, it requires that the first tuple of a segment must be part of the previous segment.

² Note that we are assuming here, w.l.o.g., that there are no tuples with the same timestamp. Indeed, this can always be guaranteed by assuming a sufficiently fine time granularity.

Definition 5.4 (m -segmentation). Let $L = \ell_1, \dots, \ell_n$ be a log and let $m > 0$ be a natural number. A segment of L is a pair (s, t) of natural numbers such that $1 \leq s \leq t \leq n$. An m -segmentation of L is a sequence $(1 = s_1, t_1), \dots, (s_m, t_m = n)$ of pairs of natural numbers such that (i) $\forall i \in \{1, \dots, m\}$, (s_i, t_i) is a segment of L , and (ii) $\forall i \in \{1, \dots, m - 1\}$, $s_{i+1} \leq t_i$.

Example 5.5. Consider the log L of our running example, reported in Fig. 5.2 – for the moment, ignore the timestamps.

Log tuple	action	timestamp
ℓ_1	Scan	0
ℓ_2	Privilege Escalation	1
ℓ_3	Buffer Overflow	2
ℓ_4	Remote Access	5
ℓ_5	Directory Traversal	7
ℓ_6	Buffer Overflow	10
ℓ_7	Scan	11
ℓ_8	SQL Injection	12
ℓ_9	Privilege Escalation	14
ℓ_{10}	Web Server Control	15
ℓ_{11}	DB Server Control	17
ℓ_{12}	Information Loss	20

Fig. 5.2. Log used in the example of Fig. 5.1.

The sequence $(1, 1), (1, 4), (4, 5), (5, 8), (6, 11), (11, 12)$ is a 6-segmentation of L that segments it into the sub-logs $L_1 = \ell_1$, $L_2 = \ell_1, \dots, \ell_4$, $L_3 = \ell_4, \ell_5$, $L_4 = \ell_5, \dots, \ell_8$, $L_5 = \ell_6, \dots, \ell_{11}$, and $L_6 = \ell_{11}, \dots, \ell_{12}$.

Finally, given a log $L = \ell_1, \dots, \ell_n$, we define the temporal distance between two tuples ℓ_i and ℓ_j in L as $d(\ell_i, \ell_j) = |\ell_j.timestamp - \ell_i.timestamp|$.

We are now ready to give a formal definition of the instance of an activity model. An instance of an activity model M over a complete path in M is a log that can be segmented in such a way that, for each segment: (1) the actions represented by the tuples in the segment and the cardinalities of their occurrences comply with the constraints in M over the corresponding hyperedge; (2) the segment does not include unnecessary tuples as its start or end; (3) the temporal extension of the segment complies with the constraints specified by function τ (if present); (4) the start tuple of the segment and those of all subsequent segments comply with the constraints specified by function ϵ (if present).

Definition 5.6 (Instance of an Activity Model). Assume that $M = \langle \mathcal{H}, \lambda, \tau, \epsilon, S, T \rangle$ is an activity model over \mathcal{A} . Let $\pi = h_1, \dots, h_m$ be a complete path in M , and let $L = \ell_1, \dots, \ell_n$ be a log. Then, we say that L is an

instance of M over π , denoted by $L \models_{\pi} M$, if there exists an m -segmentation $(s_1, t_1), \dots, (s_m, t_m)$ of L such that $\forall i \in \{1, \dots, m\}$,

1. $\forall v \in h_i$, if $\lambda(v) = (a, l, u)$, then $l \leq |\{\ell \in \ell_{s_i}, \dots, \ell_{t_i} \mid \ell.action = a\}| \leq u$;
2. $\exists v_s, v_t \in h_i$ such that $\lambda(v_s) = (\ell_{s_i}.action, l_s, u_s)$ and $\lambda(v_t) = (\ell_{t_i}.action, l_t, u_t)$;
3. if $h_i \in \text{domain}(\tau)$, then $l_i \leq d(\ell_{s_i}, \ell_{t_i}) \leq u_i$, with $\tau(h_i) = (l_i, u_i)$;
4. $\forall j \in [1, m]$ s.t. $i < j$ and $(h_i, h_j) \in \text{domain}(\epsilon)$, it holds that $d(\ell_{s_i}, \ell_{s_j}) \leq \epsilon(h_i, h_j)$.

Example 5.7. Consider the activity model M of our running example and the path $\pi = h_1, h_2, h_3, h_4, h_5, h_6$ in M . The log L in Fig. 5.2 is an instance of M over π because the 6-segmentation $(1, 1), (1, 4), (4, 5), (5, 8), (6, 11), (11, 12)$ of L is such that (see Fig. 5.1) the sets of actions associated with sub-logs $L_1 = \ell_1$, $L_2 = \ell_1, \dots, \ell_4$, $L_3 = \ell_4, \ell_5$, $L_4 = \ell_5, \dots, \ell_8$, $L_5 = \ell_6, \dots, \ell_{11}$, and $L_6 = \ell_{11}, \ell_{12}$, are “minimal” supersets of the sets of actions associated with hyperedges h_1, h_2, h_3, h_4, h_5 and h_6 , respectively (Conditions 1 and 2 in Definition 5.6) and temporal constraints hold (Conditions 3 and 4 in Definition 5.6): $\tau(h_2) = (1, 6)$ and $1 \leq d(\ell_1, \ell_4) = 5 \leq 6$; $\tau(h_4) = (3, 8)$ and $3 \leq d(\ell_5, \ell_8) = 5 \leq 8$; $\tau(h_5) = (5, 10)$ and $5 \leq d(\ell_6, \ell_{11}) = 7 \leq 8$; $\epsilon(h_3, h_6) = 12$ and $d(\ell_1, \ell_4) = 9 \leq 12$.

5.3 Discussion: Features of the Hypergraph-Based Model

The features of the proposed hypergraph-based activity model are manifold – in this section, we discuss some of those we find the most interesting and potentially useful and convenient for the security expert.

5.3.1 Synchronization Independency

Several kinds of activities may not be evident after analysing a single log but may be exposed after correlating information provided by multiple logs. For instance, intrusion detection systems may collect data from system logs, services, and node messages; operating systems keep track of various kinds of events; servers keep extensive records of their operations; applications log errors, warnings, and failures; firewalls track packets at different points in a network; cloud services often generate different logs for network, applications, database, and programming interfaces [94, 95, 96]. Bringing together all this information enables more effective activity detection. The most common problem in such scenarios is the lack of synchronization among different logs, that occurs when the acquisition points are only synchronized within small groups, or not synchronized at all. Our proposed hypergraph-based model allows the security expert to manage the lack of synchronization by grouping action symbols from different logs that may appear in different orders. Consider for instance the scenario depicted in Fig. 5.3 (top), which describes a

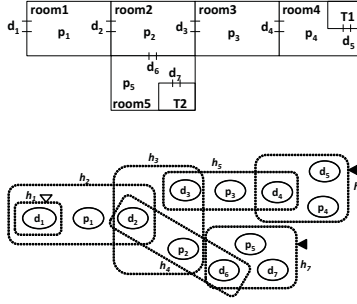


Fig. 5.3. Example scenario (top) and activity model (bottom). Every vertex of the hypergraph has cardinality constraint $(1 : 1)$.

floor controlled with both door sensors and Closed-Circuit TV (CCTV). The floor consists of five rooms, in two of which there is a target asset ($T1$ and $T2$) a thief may want to steal. When somebody opens a door, an alert d_i is produced by the door sensor, while when somebody crosses a room, an alert p_i is produced by the CCTV system. Now assume the latter is slightly slower in producing alerts due to the delay introduced by video processing. Thus, it may happen that when the thief crosses a room, the CCTV system sends the corresponding alert *after* the thief opens the next door. This means that a pair of alerts p_i, d_j may not necessarily appear in the “expected” order. This situation can be described in a very simple way with the activity model depicted in Fig. 5.3 (bottom). Each hyperedge $h_i, i = 2, \dots, 7$, expresses the fact that there is a pair of alerts p_i, d_j that may occur in any order (but always after a d_k with $j \neq k$). For instance, in h_2 , p_1 and d_2 may occur in any order (after d_1) and so on. It should also be noted that, if the security expert prefers to make the detection process less tied to the output of the CCTV system (which can sometimes be not fully reliable), then it suffices to just replace the cardinality constraints $(1 : 1)$ with $(0 : 1)$ for each vertex p_i .

5.3.2 Compactness

One of the most immediate advantages of our proposal is the compactness of the models. Fig. 5.4 shows a graph-based model for the scenario of Fig. 5.3 – the hypergraph-based model is clearly much more compact (and readable) than its graph-based counterpart. Moreover, additional edges (dotted in the figure) must be added to impose independency from the CCTV system.

Compactness is even more evident in more “extreme” scenarios – consider for instance a case where three sets of symbols must occur in a specific order, $\{a, b, c\} \rightarrow \{d, e, f\} \rightarrow \{g, h, i\}$, independently of the order in which symbols in the same set occur (but all of the symbols must occur). Fig. 5.5 (top) shows the hypergraph model for this scenario, while Fig. 5.5 (bottom) depicts the corresponding graph – the graph has 36 vertices and 54 edges, whereas the

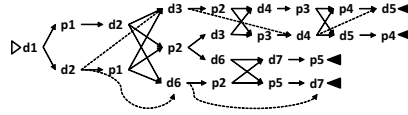


Fig. 5.4. Graph-based model for the scenario of Fig. 5.3 (bottom).

hypergraph has just 9 vertices and 3 hyperedges. All the graph-based models that do not include specific extensions for order-independency (such as, e.g., those present in SP-graphs [24]), suffer heavily from this issue.

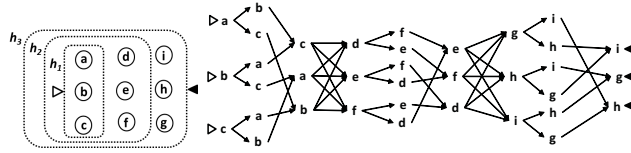


Fig. 5.5. Hypergraph- (top) and graph- (bottom) based models for the sequence of actions $\{a, b, c\} \rightarrow \{d, e, f\} \rightarrow \{g, h, i\}$. Every vertex of the hypergraph has cardinality constraint $(1 : 1)$.

5.3.3 Modeling XORs and Contiguous Sequences

Our proposed model can easily express the usual AND and OR relationships between different sub-paths. Modeling ANDs just requires the involved action symbols to be constrained with a minimum cardinality equal to 1, whereas ORs require minimum cardinality equal to 0 plus appropriate temporal constraints on hyperedges. XORs can be easily expressed as well. Consider the example graph-based model of Fig. 5.6 (left), where symbols a and x are mutually exclusive, and assume that so are symbols c and y . Moreover, assume that all symbols must appear contiguously in the log. In the corresponding hypergraph-based model, shown in Fig. 5.6 (right), all of the symbols can be inserted in a single path h_1, h_2, h_3 . In order to impose mutual exclusiveness, we combine the cardinality constraints over v_1 and v_2 , which require both a and x to appear at most once, with the temporal constraint over h_1 , which requires either a or x to appear exactly once (to express an OR on a and x , it would instead suffice to use the temporal constraint “(0,1)” over h_1 , that allows both symbols to appear). In addition, the fact that symbols must be contiguous is expressed through the temporal constraints over h_2 and h_3 , after assuming that timestamps are strictly consecutive (this can obviously be obtained after a trivial preprocessing of the log).

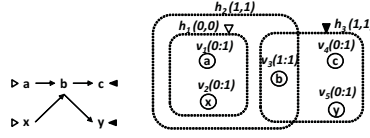


Fig. 5.6. Graph-based model with XORs (left) and the corresponding hypergraph-based model (right).

5.3.4 “Weak” Dependency

The proposed model allows the security expert to easily capture situations where there are groups of actions the detection task is not strongly dependent on. This might be the case of a malware detection task: polymorphic malwares may contain pieces of code they generate in order to circumvent signature-based malware detection. For instance, [97] presents a malware detection system that looks at typical actions occurring during the infection lifecycle: target scanning, infection exploit, binary download and execution, etc. They assume that not all these actions are required nor that every action will be detected. Using weak dependency, we can design a hypergraph-based model with just one hyperedge for each potentially generated piece of code, in order to take into account every possible form. An example of this kind of usage is provided in Section 5.7.

5.4 Consistency of Activity Models

In this section we study the consistency of our proposed activity models and the complexity of checking whether a given activity model is consistent. We start by defining the consistency of a path.

Definition 5.8 (Consistency of a path). *Let π be a complete path in an activity model $M = \langle \mathcal{H}, \lambda, \tau, \epsilon, S, T \rangle$. We say that π is consistent w.r.t. M if there is an instance L of M over π , i.e., if there is a log L such that $L \models_{\pi} M$.*

To detect the consistency of a complete path $\pi = h_1, \dots, h_m$ in M , we associate a *support graph* with π , denoted by $SG(M, \pi) = \langle N, E, \omega \rangle$, that is a node- and edge-weighted directed graph where:

- the set N of nodes are the hyperedges of π ;
- there is precisely an edge from h_{α} to h_{β} in E for each $(h_{\alpha}, h_{\beta}) \in \text{domain}(\epsilon)$ with $\alpha < \beta$;
- $\omega(h_i) = \begin{cases} l_i & \text{if } h_i \in \text{domain}(\tau) \text{ and } \tau(h_i) = (l_i, u_i); \\ 0 & \text{if } h_i \notin \text{domain}(\tau); \end{cases}$
- $\omega(h_{\alpha}, h_{\beta}) = \epsilon(h_{\alpha}, h_{\beta})$.

Example 5.9. Consider the activity model M' obtained from the model M of our running example (Fig. 5.1 (top)) by adding the temporal constraint $\epsilon(h_2, h_5) = 3$. The graph $SG(M, \pi = h_1, h_2, h_3, h_4, h_5, h_6)$ is shown in Fig. 5.7. By definition, the graph has 6 nodes/hyperedges, and two edges corresponding to the two elements in the domain of ϵ .

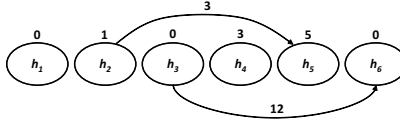


Fig. 5.7. Support graph associated with the path $h_1, h_2, h_3, h_4, h_5, h_6$.

The following result gives us necessary and sufficient conditions for a path to be consistent.

Proposition 5.10. *Let M be an activity model and let $\pi = h_1, \dots, h_m$ be a complete path in M . Then, π is consistent w.r.t. M if and only if for each edge (h_α, h_β) in $SG(M, \pi)$, it holds that $\sum_{i=\alpha}^{\beta-1} \omega(h_i) \leq \omega(h_\alpha, h_\beta)$.*

For instance, in Fig. 5.7, we have that $\omega(h_2) + \omega(h_3) + \omega(h_4) = 4 > 3 = \epsilon(h_2, h_5)$. Thus, by Proposition 5.10, π is not consistent w.r.t. M' .

We now define our notion of consistency for activity models.

Definition 5.11 (Consistency of an Activity Model). *Assume that $M = \langle \mathcal{H}, \lambda, \tau, \epsilon, S, T \rangle$ is an activity model. We say that M is consistent if $\forall h_s \in S, \forall h_m \in T$ there is a path π starting with h_s and ending with h_m , respectively, that is consistent.*

We refer the reader to [98] for less restrictive variants of the above notion. To complete our discussion, we characterize the complexity of consistency checking.

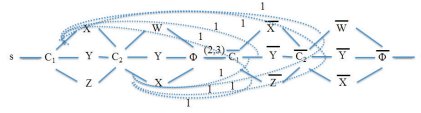
Theorem 5.12. *Deciding whether a given activity model is consistent is NP-complete, even if $|S| = |T| = 1$.*

Membership in **NP** is trivial. For the hardness, we give a reduction from the MONOTONE ONE-IN-THREE 3SAT problem, which is known to be **NP**-complete [99]. The problem is a variant of the classical satisfiability problem, where the input instance is a conjunction of clauses, with each clause consisting of exactly three variables (i.e., negation is not allowed). The goal is to determine whether there is a truth assignment to the variables so that each clause has exactly one true variable (and thus exactly two false variables).

Let $\phi = C_1 \wedge \dots \wedge C_m$ be a Boolean formula taken as input and such that each clause $C_i = x_{i,1} \vee x_{i,2} \vee x_{i,3}, \forall i \in \{1, \dots, m\}$, consists of exactly 3 variables. Based on ϕ , we define an activity model $M(\phi) = \langle \mathcal{H}, \lambda, \tau, \epsilon, S, T \rangle$ over a set \mathcal{A} of actions, where $\mathcal{H} = (V, H)$ and such that:

- $V = \{s, \phi, t, \bar{\phi}\} \cup \{C_i, x_{i,j}, \bar{C}_i, \bar{x}_{i,j} \mid i \in \{1, \dots, m\} \wedge j \in \{1, 2, 3\}\}$;
- the set H exactly contains the following hyperedges:
 - $h_s = \{s, C_1\}$;
 - $h_{i,j} = \{C_i, x_{i,j}\}$ and $h_{i,j}^\wedge = \{x_{i,j}, C_{i+1}\}$, $\forall i \in \{1, \dots, m-1\} \wedge j \in \{1, 2, 3\}$;
 - $h_{m,j} = \{C_m, x_{m,j}\}$ and $h_{m,j}^\wedge = \{x_{m,j}, \phi\}$, $\forall j \in \{1, 2, 3\}$;
 - $\bar{h}_s = \{\phi, \bar{C}_1\}$;
 - $\bar{h}_{i,j} = \{\bar{C}_i, \bar{x}_{i,j}\}$ and $\bar{h}_{i,j}^\wedge = \{\bar{x}_{i,j}, \bar{C}_{i+1}\}$, $\forall i \in \{1, \dots, m-1\} \wedge j \in \{1, 2, 3\}$;
 - $\bar{h}_{m,j} = \{\bar{C}_m, \bar{x}_{m,j}\}$ and $\bar{h}_{m,j}^\wedge = \{\bar{x}_{m,j}, \bar{\phi}\}$, $\forall j \in \{1, 2, 3\}$; and,
 - $h_t = \{\bar{\phi}, t\}$;
- $\lambda(v) = (v, 1, 1)$, for each $v \in V$; in fact, $\mathcal{A} = V$;
- $\text{domain}(\tau) = \{\bar{h}_s\}$, and $\tau(\bar{h}_s) = (2, 3)$;
- $S = \{h_s\}$, $T = \{h_t\}$;
- for each variable of the form $x_{i,j}$ with $i \in \{1, \dots, m\}$ and $j \in \{1, 2, 3\}$, for each clause C_z with $z \in \{1, \dots, m\}$ where $x_{i,j}$ occurs (possibly with $i = z$), and for each variable $x_{z,k} \neq x_{i,j}$ with $k \in \{1, 2, 3\}$ occurring in C_z and different from $x_{i,j}$, we have that: $(h_{i,j}, \bar{h}_{z,k}) \in \text{domain}(\epsilon)$ and $\epsilon(h_{i,j}, \bar{h}_{z,k}) = 1$.

As an example, the formula $\phi = (x \vee y \vee z) \wedge (x \vee y \vee w)$ is a YES instance to the MONOTONE ONE-IN-TREE 3SAT problem, as it witnessed by the truth assignment where x is the only variable evaluating true. The hypergraph associated with the activity model $M(\phi)$ is shown in Fig. 5.8, where for the sake of readability only arrows associated with constraints involving the variable x in ϕ are depicted.



(\Leftarrow) Let π be a consistent path in $M(\phi)$ starting with h_s and terminating with h_t . Note that by definition of a path, π must be of the following form: $\pi = h_s, h_{1,a_1}, h_{1,a_1}^\wedge, \dots, h_{m,a_m}, h_{m,a_m}^\wedge, \bar{h}_s, \bar{h}_{1\bar{a}_1}, \bar{h}_{1\bar{a}_1}^\wedge, \dots, \bar{h}_{m,\bar{a}_m}, \bar{h}_{m,\bar{a}_m}^\wedge, h_t$, where $a_1, \dots, a_m, \bar{a}_1, \dots, \bar{a}_m \in \{1, 2, 3\}$. Now, recall that $\tau(\bar{h}_s) = (2, 3)$, i.e., in particular, the temporat extension of \bar{h}_s is at least 2 time units. Given the construction of ϵ , it follows that π cannot contain any pair of hyperedges in the domain of ϵ . Therefore, for each hyperedge h_{i,a_i} , with $i \in \{1, \dots, m\}$, if the variable x_{i,a_i} occurs in the clause C_z , with $z \in \{1, \dots, m\}$, then the hyperedge \bar{h}_{z,\bar{a}_z} is actually such that $x_{z,\bar{a}_z} = x_{i,a_i}$. Let now θ be the truth assignment such that the variable x_{i,a_i} evaluates true, for each $i \in \{1, \dots, m\}$ (and the remaining variables evaluate false). Note that θ is satisfying. Then, assume by contradiction that there is a clause C_z and two distinct variables $x_{z,\alpha}$ and $x_{z,\beta}$ evaluating true in θ . By construction of θ , it must be the case that $x_{z,\alpha} = x_{i',a_{i'}}$ and $x_{z,\beta} = x_{i'',a_{i''}}$, where i' and i'' are two indices in $\{1, \dots, m\}$. However, by the above observations, \bar{h}_{z,\bar{a}_z} is actually such that $x_{z,\bar{a}_z} = x_{i',a_{i'}} = x_{i'',a_{i''}}$. That is, $i' = i''$, which is impossible if the variables $x_{z,\alpha}$ and $x_{z,\beta}$ are distinct.

5.5 Equivalence and Minimality of Activity Models

In this section, we define precedence relations over activity models with the aim of identifying redundancies in their definitions and semantically-equivalent minimal models.

We start by defining a purely syntactic ordering relation “ \preceq ” over activity models, with the objective of ensuring that if $M_1 \preceq M_2$, then M_2 is “more general” than M_1 , in the sense that the set of instances of M_1 is a subset of those of M_2 .

More formally, let $M_1 = \langle (V_1, H_1), \lambda_1, \tau_1, \epsilon_1, S_1, T_1 \rangle$ and $M_2 = \langle (V_2, H_2), \lambda_2, \tau_2, \epsilon_2, S_2, T_2 \rangle$ be two activity models. We write $M_1 \preceq M_2$ if the following conditions are satisfied:

- $V_1 \subseteq V_2$;
- there exists a mapping $f : H_1 \rightarrow H_2$ such that $\forall h_1 \in H_1, h_1 \subseteq f(h_1)$, and $\forall v \in f(h_1) \setminus h_1, \lambda(v) = (a, l, u)$ implies $l = 0$; we call f the *witness mapping* of $M_1 \preceq M_2$;
- the restriction of λ_2 over the vertices in V_1 coincides with λ_1 (i.e. $\forall v \in V_1, \lambda_2(v) = \lambda_1(v)$);
- if $\tau_1(h_1)$ is not defined, then $\tau_2(f(h_1))$ is not defined; if $\tau_1(h_1) = (l_1, u_1)$ then either $\tau_2(f(h_1))$ is not defined or $\tau_2(f(h_1)) = (l_2, u_2)$ with $l_2 \leq l_1$ and $u_2 \geq u_1$;
- if $\epsilon(h_1, h'_1)$ is not defined, then $\epsilon(f(h_1), f(h'_1))$ is not defined; otherwise, either $\epsilon(f(h_1), f(h'_1))$ is not defined or $\epsilon(f(h_1), f(h'_1)) \geq \epsilon(h_1, h'_1)$;
- $\forall h_1 \in S_1, f(h_1) \in S_2$ (resp. $\forall h_1 \in T_1, f(h_1) \in T_2$).

Example 5.13. Consider the activity model $M_1 = \langle (V_1, H_1), \lambda_1, \tau_1, \epsilon_1, S_1, T_1 \rangle$ of our running example and the activity model $M_2 = \langle (V_2, H_2), \lambda_2, \tau_2, \epsilon_2, S_2, T_2 \rangle$ of Fig. 5.9. We can easily check that $M_1 \preceq M_2$ since:

- $V_2 = \{v^*, v_1, \dots, v_{12}\} \supseteq V_1$;
- $H_2 = \{h_1^*, \dots, h_8^*\}$, where $h_1^* = h_1, h_2^* = h_2, h_3^* = h_3, h_4^* = \{v_2, v_3, v_4, v_5, v_6\}, h_5^* = h_5, h_6^* = \{v^*, v_7, v_{11}\}, h_7^* = h_7$, and $h_8^* = h_8$; intuitively, the witness mapping $f : H_1 \rightarrow H_2$ in condition (2) is such that $f(h_i) = h_i^*$, for each $i \in \{1, \dots, 8\}$;
- $\lambda_2(v^*) = (\text{Scan}, 0, u)$, while the restriction of λ_2 over the vertices in V_1 coincides with λ_1 ;
- $\text{domain}(\tau^*) = \{h_2^*, h_4^*, h_5^*\}$; $\tau(h_2^*) = (1, 6)$, $\tau(h_4^*) = (3, 8)$, $\tau(h_5^*) = (2, 18)$; for instance, note that condition (4) holds on $\tau(h_5^*) = (2, 18)$ given that $\tau(h_5) = (5, 10)$;
- $\text{domain}(\epsilon^*) = \{(h_3^*, h_6^*)\}$ and $\epsilon(h_3^*, h_6^*) = 13$; in particular, note that condition (5) holds because $\epsilon(h_3, h_6) = 12$;
- $S_2 = \{h_1^*\}$ and $T_2 = \{h_6^*, h_7^*, h_8^*\}$.

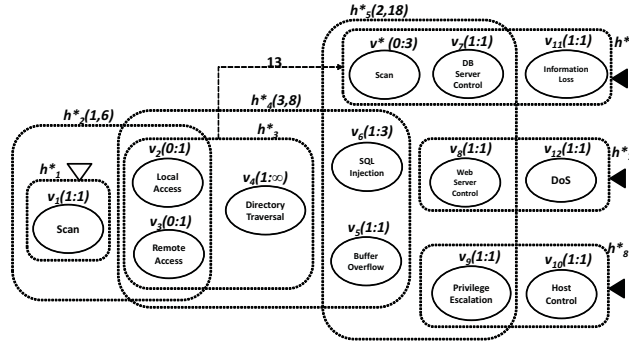


Fig. 5.9. Example activity model M_2 .

For an activity model M , we define $\mathcal{L}(M) = \{L \mid \text{there is a path } \pi \text{ s.t. } L \models_{\pi} M\}$ as the set of all the logs that are instances of M . The above precedence relation over activity models leads to the following semantic characterization.

Theorem 5.14. *Given two activity models M_1 and M_2 such that $M_1 \preceq M_2$, it holds that $\mathcal{L}(M_1) \subseteq \mathcal{L}(M_2)$.*

A proof sketch for Theorem 5.14 is reported in the Appendix.

As an example, the log reported in Fig. 5.2, which belongs to $\mathcal{L}(M_1)$ with M_1 being the model of our running example, is also in $\mathcal{L}(M_2)$ with M_2 being the model discussed in Example 5.13.

It is easy to verify that the above notion of minimality can be checked in polynomial time.

Proposition 5.15. *Given two activity models M_1 and M_2 , deciding whether $M_1 \preceq M_2$ is feasible in polynomial time.*

We are now ready to define a concept of semantic minimality, which we call “ \preceq -minimality”. The idea is to define a partial order \preceq over a set of activity models in such a way that a minimum element is a model whose definition does not contain redundant specifications. We say that two models M_1 and M_2 are *log-equivalent* if $\mathcal{L}(M_1) = \mathcal{L}(M_2)$. It should be observed that, by Theorem 5.14, if two models are such that $M_1 \preceq M_2$ and $M_2 \preceq M_1$, then they are actually log-equivalent. We write $M_1 \trianglelefteq M_2$ if $M_1 \preceq M_2$ and the two models are log-equivalent. This notion of semantic minimality is intractable, as dictated by the following result.

Theorem 5.16. *Given two activity models M_1 and M_2 , deciding whether $M_1 \trianglelefteq M_2$ is **co-NP-hard**.*

5.6 The Malevolent Activity Detection Problem

In this section we formally characterize the malevolent activity detection problem we are interested in and its complexity. The problem is basically that of checking whether a log is an instance of an activity model. The following definition formalizes this.

Definition 5.17 (Malevolent Activity Detection Problem). *Given an activity model M and a log L , determine whether there exists a complete path π in M such that $L \models_{\pi} M$.*

We now characterize the complexity of the above problem.

Theorem 5.18. *The malevolent activity detection problem is **NP-complete**.*

Theorem 5.18 establishes that there are cases where detecting a malevolent activity is not feasible in polynomial time (unless **P=NP**). Intuitively, the exponential blowup is mainly due to the fact that, while analyzing the log: (i) it is necessary to maintain all possible “partial” instances (prefixes of the log that are “instances” over incomplete paths) of the activity model M ; (ii) many different incomplete paths in M can be associated with each partial instance; (iii) many different segmentations can be associated with each (*partial instance, incomplete path*) pair. Recent works on the detection instances of temporal graph-based models in sequences of logged actions [82, 16] have shown that acceptable detection times in real-world cases can be obtained by using index structures that limit the number of partial instances to be maintained. These structures exploit temporal constraints to only look at the set of (partial) instances that lie within a fixed temporal window.

5.7 Indexing and Detecting Activity Instances

In this section we present our proposed index structure, called AM-Index, and its associated maintenance and retrieval algorithms, that exploit temporal (and cardinality) constraints to prune partial instances of *hypergraph-based models* as soon as possible. Moreover, they are designed to be capable of *concurrently tracking instances of multiple models*. Finally, besides retrieving the instances of any of the given models in the log, they return, for each instance, the path followed in the corresponding model.

Given an activity model $M = \langle (V, H), \lambda, \tau, \epsilon, S, T \rangle$ and a hyperedge $h \in H$, we define $in_M(h) = \{h' \in H \text{ s.t. there exists a complete path } \pi \text{ in } M \text{ containing the subsequence } h', h\}$. Moreover, for the sake of conciseness, if $\lambda(v) = (a, l, u)$, then we also write $label(v) = a$.

Definition 5.19 (AM-Index). *Given a set \mathcal{M} of activity models and a log L , an AM-Index $I_{\mathcal{M}}$ is a pair $\langle \mathcal{T}, \mathcal{I} \rangle$ where \mathcal{T} is a set containing a table $table(h)$ for each hyperedge in \mathcal{M} – each row $r \in table(h)$ represents a segment of the log and has several components: a list $PL[a]$ of pointers to log tuples for each $v \in h$ with $label(v) = a$; a (possibly null) pointer previous to a table row; a boolean flag completed; the identifier model of a model in \mathcal{M} ; a timestamp $minTS$. \mathcal{I} is a set containing, for each model $M \in \mathcal{M}$, a list $instances(M)$ of pointers to table rows.*

The components of every row r of a table $table(h)$ represent the following information: (i) each pointer list $PL[a]$ contains pointers to log tuples l with $l.action = a$ that are a part of the log segment associated with h and represented by r ; (ii) *previous* points to a row of $table(h')$ with $h' \in in_M(h)$ – this row represents a segment to which the segment represented by r can be linked in order to extend a partial instance; (iii) *completed* is *true* iff r represents a segment that completely satisfies h ; (iv) *model* is the identifier of the model containing h ; (v) *minTS* is the minimum $l.timestamp$ where l is a log tuple pointed by a list $PL[x]$ for each x such that $\exists v \in h$ with $label(v) = x$. The pointers in $instances(M)$ point to table rows that represent log segments corresponding to terminal hyperedges in M which have been completed.

Example 5.20. Fig. 5.10 shows the status of the AM-Index after indexing the log of our running example up to tuple ℓ_{10} .

At this stage, the index contains the following information (note that ℓ_2 , ℓ_3 , and ℓ_7 do not appear in the index since they cannot be part of the instance of any vertex):

- The row in $table(h_1)$ represents the fact that hyperedge h_1 of model M is completed by a segment that includes the log tuple ℓ_1 (that is instance of vertex v_1 of h_1). In addition, $minTS = 0$ means that the segment starts at time 0.

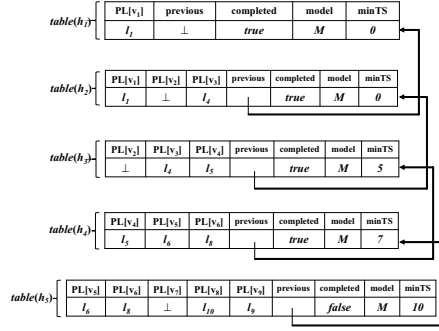


Fig. 5.10. Status of the AM-Index after indexing the log of the running example up to tuple ℓ_{10} .

- The row in $table(h_2)$ represents the fact that hyperedge h_2 of model M is completed by a segment that includes log tuples ℓ_1 and ℓ_4 (note that, since a tuple for v_2 is not needed, $PL[v_2]$ can remain null in this row). Moreover, the *previous* pointer points to the row in $table(h_1)$ as the segment represented by the row in $table(h_2)$ follows the former in the (partial) activity instance being represented.
- The row in $table(h_3)$ represents the fact that hyperedge h_3 of model M is completed by a segment that includes log tuples ℓ_4 and ℓ_5 . The *previous* pointer points to the row in $table(h_2)$ as the segment represented by the row in $table(h_3)$ follows the former in the (partial) activity instance being represented.
- The row in $table(h_4)$ represents the fact that hyperedge h_4 is completed by a segment that includes log tuples ℓ_5 , ℓ_6 and ℓ_8 . The *previous* pointer points to the row in $table(h_3)$ as the segment represented by the row in $table(h_4)$ follows the former in the (partial) activity instance being represented.
- The row in $table(h_5)$ represents the fact that hyperedge h_5 is partially completed by a segment that includes log tuples ℓ_6 , ℓ_8 , ℓ_9 and ℓ_{10} , but we still have to encounter a log tuple corresponding to vertex v_7 .

5.7.1 Insertion of Tuples and Retrieval of Instances

Fig. 5.11 (top) shows the pseudo-code of the **AM-Insert** algorithm that indexes a new log tuple l_{new} with associated action $l_{new}.action$. In the algorithm, Lines 4-10 check whether l_{new} , with associated action $a = l_{new}.action$, can be added to existing segments (i.e., to any row $r \in table(h)$ with $r.completed = false$, where h is an hyperedge containing a), by verifying cardinality and temporal constraints. If so, a new pointer to l_{new} is added to the pointer list $r.PL[a]$, and the new segment is passed to the **CheckCompleted** procedure to check whether r represents a segment that completely satisfies h . Otherwise, r is immediately pruned as it represents a non-extendable segment. Lines

	Algorithm AM.Insert (l_{new}, I_M) Input: New log tuple l_{new} , AM-Index I_M Output: Updated AM-Index I_M
1	$a \leftarrow l_{new}.action$
2	for each $M = \langle \mathcal{H}, \lambda, \tau, \epsilon, S, T \rangle \in \mathcal{M}, \mathcal{H} = (H, V)$ do
3	for each (h, v) s.t. $h \in H, v \in h$, and $label(v) = a$ do
	// Can l_{new} be added to an existing segment?
4	for each table row $r \in table(h)$ s.t. $r.completed = false$ do
5	if $l_{new}.timestamp - r.minTS + 1 \leq u'$ with $\tau(h) = (\cdot, u')$
	and $size(r.PL[a]) + 1 \leq u$ with $\lambda(v) = (\cdot, \cdot, u)$ then
6	add l_{new}^i to $r.PL[a]$; checkCompleted (h, r)
7	else // The segment violates the constraints
8	remove r from $table(h)$
9	end if
10	end for
	// If h is a start hyperedge, create a row for a new segment
11	if $h \in S$ then
12	$PL \leftarrow \emptyset$
13	for each $z \in h$ do $PL[label(z)] \leftarrow \perp$
14	$r \leftarrow \langle PL, \perp, false, M, l_{new}.timestamp \rangle$; $r.PL[a] \leftarrow \{l_{new}^i\}$
15	add r to $table(h)$; checkCompleted (h, r)
16	end if
17	end for
18	end for
	Procedure checkCompleted (h, r) Input: Hyperedge h , index table row r
19	if $l' \leq l_{new}.timestamp - r.minTS + 1$ with $\tau(h) = (l', \cdot)$
	and $\forall w \in h, l_w \leq size(r.PL[label(w)])$ with $\lambda(w) = (\cdot, l_w, \cdot)$ then
20	$r.completed \leftarrow true$
21	if $h \in T$ then add r^\dagger to $instances(M)$ // M is the model where h is
22	for each h' s.t. $h \in in_M(h')$ do
23	$r' \leftarrow \langle PL = \emptyset, r^\dagger, false, M, 0 \rangle$
24	for each $z \in h \cap h'$ do $r'.PL[label(z)] \leftarrow r.PL[label(z)]$
25	$r'.minTS \leftarrow$ minimum timestamp of the log tuples pointed by r'
26	if $(h, h') \notin domain(\epsilon)$ or $r'.minTS - r.minTS \leq \epsilon(h, h')$ then
27	add r' to $table(h')$
28	end if
29	end for
30	end if
	Algorithm AM.Retrieve ($I_M = \langle \mathcal{T}, \mathcal{I} \rangle, \mathcal{M}'$) Input: AM-Index I_M , set $\mathcal{M}' \subseteq \mathcal{M}$ of activity models Output: Set I of tuples of the form (M, l_s, l_t, π) such that $M \in \mathcal{M}'$ and $l_s, \dots, l_t \models_\pi M$
1	$I \leftarrow \emptyset$
2	for each $M \in \mathcal{M}'$ do
3	for each $r^\dagger \in instances(M)$ do
4	$l_t \leftarrow$ most recent log tuple pointed by r
5	$\pi \leftarrow \{hyperedge \text{ represented by } r\}$
6	$r_{curr} \leftarrow r$
7	while $r_{curr}.previous \neq \perp$ do
8	$r_{curr} \leftarrow r_{curr}.previous$
9	add the hyperedge represented by r_{curr} to π
10	end while
11	$l_s \leftarrow$ less recent log tuple pointed by r_{curr}
12	add (M, l_s, l_t, π) to I
13	end for
14	end for
15	return I

Fig. 5.11. AM.Insert (top) and AM.Retrieve (bottom) algorithms.

11-16 deal with the case where a is in a start hyperedge – the algorithm simply creates a new row r pointing to l_{new} and adds r to $table(h)$. Procedure

CheckCompleted takes as input a table row r and an hyperedge h , and checks whether the segment of log tuples pointed by r is an instance and/or if it can be further extended. More specifically, Line 20 sets $r.completed = true$ because, according to the constraints expressed by h , the segment of log tuples pointed by r completely satisfies h . Line 21 stores a new instance by adding a pointer to r in $instances(M)$ if h is a terminal hyperedge. In lines 22-29, for each outgoing hyperedge h' of h , a new row r' is created with $r'.previous = r$ and $r'.PL[label(z)] = r.PL[label(z)]$ for each $z \in h \cap h'$. In other words, if h has at least one outgoing hyperedge h' , then h and h' have at least one vertex z in common, thus the index must contain rows r in $table(h)$ and r' in $table(h')$, both pointing the same log tuples l with $l.action = label(z)$.

Example 5.21. Fig. 5.12 shows the evolution of the AM-Index when indexing tuples ℓ_{11} and ℓ_{12} of the log of our running example. When indexing tuple

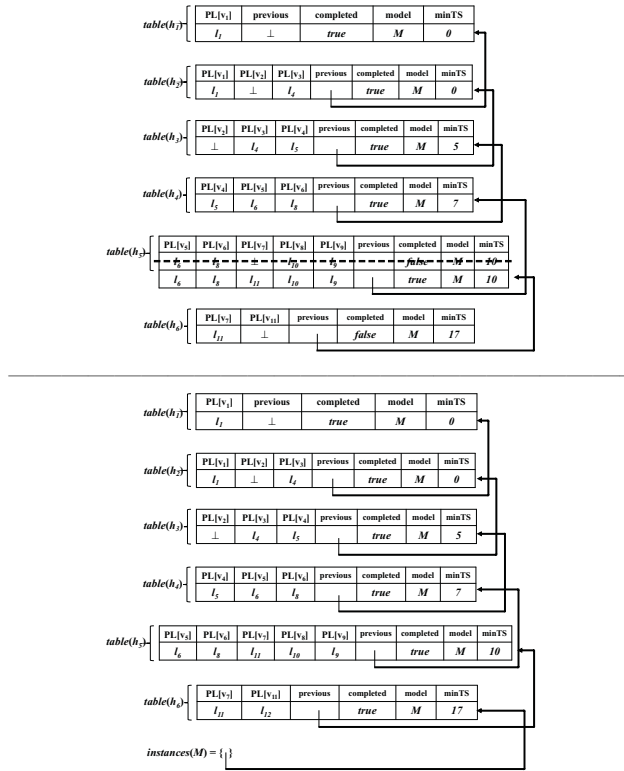


Fig. 5.12. Evolution of the AM-Index when indexing tuples ℓ_{11} (top) and ℓ_{12} (bottom) of the log of the running example.

ℓ_{11} (Fig. 5.12 (top)) the **AM.Insert** algorithm identifies the row in $table(h_5)$

as one that represents a segment that can be extended by ℓ_{11} (ℓ_{11} is an instance of vertex v_7). Thus, the algorithm replaces the \perp pointer in $PL[v_7]$ with a pointer to ℓ_{11} (note that, for clarity of presentation, the figure shows a deletion of the row). Then, procedure **CheckCompleted** verifies that the segment is completed, so it marks the row as completed and creates a new row in $table(h_6)$ that contains, for each vertex in common between h_5 and h_6 , a copy of the pointer list of the vertex. This new row gets completed after indexing ℓ_{12} , that is an instance of vertex v_{11} (Fig. 5.12 (bottom)). Moreover, since h_6 is a terminal hyperedge, a pointer to the row is added to $instances(M)$.

In order to precisely characterize the time complexity of the **AM_Insert** algorithm, we need to introduce some concepts. First, we define the *maximum size* of the models in \mathcal{M} as $S_{max} = \max_{M \in \mathcal{M}} (\sum_{h \text{ in } M} |h|)$. In addition, we compute the *temporal window* within which all instances must lie as $W = \min (\max_{M \in \mathcal{M}, D \in \Delta(\pi)} \omega_m(D), |L|)$, where:

- π is a complete path in $M = \langle \mathcal{H}, \lambda, \tau, \epsilon, S, T \rangle$;
- $\Delta(\pi)$ is the set of all maximal subsets of $domain(\epsilon)$ that do not contain overlapping pairs of hyperedges w.r.t. π (we say that two pairs (h_i, h_j) and (h'_i, h'_j) overlap w.r.t. π if h_i appears before h'_i and h'_i appears before h_j in π);
- we define $\omega_m(h_i) = u_i$ if $h_i \in domain(\tau)$ and $\tau(h_i) = (l_i, u_i)$, and $\omega_m(h_i) = \infty$ otherwise;
- given a set $D \in \Delta(\pi)$, we define $\omega_m(D) = \sum_{(h_i, h_j) \in D} \epsilon(h_i, h_j) + \sum_{h_k \text{ not covered by } D} \omega_m(h_k)$, where we say that an edge h_k is covered by D if the latter contains a pair (h_i, h_j) and h_k appears between h_i and h_j in π .

The presence of temporal constraints, as already pointed out in Section 5.6, has a direct impact on the actual portion of the index that is affected by the insertion of a new log tuple. This is confirmed by the following proposition.

Proposition 5.22. *The **AM_Insert** algorithm terminates and correctly updates $I_{\mathcal{M}}$ in time $O(|\mathcal{M}| \cdot W^{S_{max}})$.*³

Corollary 5.23. *Given a set of models \mathcal{M} and a log L , the corresponding AM-Index can be built in time $O(|L| \cdot |\mathcal{M}| \cdot W^{S_{max}})$.*

Fig. 5.11 (bottom) shows the pseudo-code of the **AM_Retrieve** algorithm that, whenever executed, returns the instances of a (sub)set of the activity models w.r.t. the set of log tuples that have been indexed. The instances are represented as tuples of the form (M, l_s, l_t, π) , where l_s and l_t are the first and last tuple of an instance of M over a complete path π . The algorithm takes as input an AM-Index $I_{\mathcal{M}} = \langle \mathcal{T}, \mathcal{I} \rangle$ and a set of models $\mathcal{M}' \subseteq \mathcal{M}$ (note that it does not make any difference if the user is willing to extract instances

³ We assume that $size(PL[\cdot])$ is computable in time $O(1)$ – this can obviously be achieved by continuously keeping track of list sizes.

of just a subset of the models indexed by the AM-Index). For each pointer $r^\uparrow \in \text{instances}(M)$, the algorithm assigns to l_t the most recent log tuple pointed by r , i.e., the last tuple of a log segment that completely satisfies a terminal hyperedge (Line 4). The algorithm follows the chain of $r.\text{previous}$ pointers backwards, from r to the row representing a start hyperedge (Lines 7-10). The last row of the chain will be recognized due to its previous attribute set to \perp . This way, hyperedges represented by all rows of the chain are added to π . Finally, Line 11 assigns to l_s the less recent log tuple pointed by the last row, i.e., the first log tuple from which the current instance begins.

Proposition 5.24. *Given a log L and the AM-Index $I_{\mathcal{M}}$ built over L by the **AM.Insert** algorithm, the **AM.Retrieve** algorithm terminates and correctly returns the set of all instances of the models in \mathcal{M} contained in L in time $O(W \cdot N_I)$, where $N_I = |\cup_{M \in \mathcal{M}} \text{instances}(M)|$.*

Additional Pruning. A simple yet effective concurrent pruning process is also performed during the maintenance of the AM-Index. When the **AM.Insert** algorithm eliminates a non-completed row from $\text{table}(h)$ because it violates the constraints, we follow the chain of previous pointers starting from that row and eliminate any row $r \in \text{table}(h')$ if the following conditions hold: (i) r is not pointed by any other row (this can be checked efficiently by maintaining the number of pointers to each row); (ii) r cannot be extended by a row corresponding to a segment for a hyperedge $h'' \neq h$ such that $h' \in \text{in}_M(h'')$ for some M . It should also be noted that the **AM.Retrieve** algorithm can easily be modified to accommodate the cases where the security expert wants to retrieve “current” completed instances multiple times while new log tuples get indexed, so it is useless to return the same instance twice. To this end, it suffices to remove r^\uparrow from $\text{instances}(M)$ after Line 12.

Design of Efficient Activity Models. Proposition 5.22 suggests that in some basic cases the security expert can make activity models more efficient in a very simple way. Since the efficiency of the detection process increases when the sum of the cardinalities of the hyperedges (which affects the value of S_{max} in the proposition) decreases, it is better to minimize (whenever possible) the number of shared vertices among different hyperedges. Consider for instance the model of Fig. 5.5 (top). It can easily be observed that, since the model does not contain temporal constraints, it does not actually matter which vertex is shared between h_1 and h_2 (or h_2 and h_3) – the extracted instances will stay the same since the only semantic modification lies in the log tuples where segments start or end. Thus, if we re-design the model as shown in Fig. 5.13, the detection process will be faster.

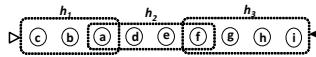


Fig. 5.13. Alternative structure for the model of Fig. 5.5 (top).

Inclusion of Unlikely Action Symbols. It should also be observed that action symbols for which very few tuples (or none) appear in the log do not impact the performance of the detection process at all. In fact, the **AM_Insert** algorithm only considers the symbols that actually appear in the log, and the only consequence of additional symbols is the creation of further index tables (with very few or no rows). This is convenient for the security expert, who can completely disregard the likelihood of seeing a symbol a in the log before deciding to include an a -labeled vertex in the model.

5.7.2 Performance of the AM-Index

The AM-Index and its associated insertion and retrieval algorithms have been implemented and integrated in a system prototype. In the following, we discuss the results of the experimental assessment we conducted in order to assess the performance they obtain in terms of log tuples processed per second and maximum size of the index.

We ran three rounds of experiments. In Round 1, we used a set of real-world activity models comprising the one used as our running example (Fig. 5.1 (top)) together with three additional ones. For Round 2, we added cardinality constraints to the models in order to analyze how the **AM_Insert** algorithm behaves when working with more constrained models. Details about the additional activity models and cardinality constraints are reported in the Appendix. For these two rounds, synthetic logs were generated according to the following procedure:

- (1) For each model $M \in \mathcal{M}$, we built the full set $P(M)$ of complete paths in the model.
- (2) For each model M , we randomly generated a set $L(M)$ (with $|L(M)| = 2K$) of sequences of action symbols that can represent an instance of M , hence covering the full spectrum of potential behaviors encoded by M . To generate each sequence in $L(M)$, we picked (uniformly at random) a complete path $\pi \in P(M)$. Then, for each hyperedge h in π , we first added l instances of each vertex $v \in h$ with $\lambda(v) = (a, l, u)$, then we added a random number of instances of vertices in h by taking into account their upper bounds and the range specified by $\tau(h)$.
- (3) For each model M , a fixed percentage (TP in the following) of the sequences in $L(M)$ was truncated to half of its size, in order to include partial instances. This way, we were able (i) to better control the number of complete instances in the generated logs, and (ii) to more deeply assess the efficiency of the framework in pruning partial instances. In each experiment, we fixed $TP \in \{20\%, 50\%, 80\%\}$, with a default of 20%.
- (4) We built a list of sequences $L_1, \dots, L_{|\mathcal{M}|*2K}$ by extracting each L_i from one of the sets $L(M)$ (chosen uniformly at random), until all such sets are empty.
- (5) We built the final sequence by combining those in the above list according to four different *combination modes*:

- CM 1: The final sequence is obtained by just concatenating the sequences in the list.
- CM 2: Same as CM 1, but, for each L_i a fixed percentage (NF in the following) of symbols not present in the corresponding model is inserted (uniformly at random) – hence simulating a certain degree of “noise”. In each experiment, we fixed $NF \in \{10\%, 30\%, 50\%\}$, with a default of 30%.
- CM 3: The final sequence is obtained by making each L_i partially overlap, by a fixed percentage (OP in the following) with L_{i+1} . In each experiment, we fixed $OP \in \{20\%, 40\%, 60\%\}$, with a default of 40%.
- CM 4: Same as CM 2, but with partial overlap as in CM 3.

(6) We assigned consecutive timestamps to the final sequence.

Table 5.1 shows the size of the resulting logs.

Table 5.1. Size of the log in Rounds 1 and 2 (default case in bold).

Varying parameter	Log size	Varying parameter	Log size
CM 1	78,622	TP 20%	102,376
CM 2	102,190	TP 50%	91,014
CM 3	78,622	TP 80%	74,610
CM 4	102,376		
NF 10%	86,651	OP 20%	102,141
NF 30%	102,376	OP 40%	102,376
NF 50%	117,939	OP 60%	102,253

In Round 3, we manually built 4 models that describe possible structures of a basic malware consisting in a shellcode that creates a user and writes a file. The models capture sequences of alphanumeric instruction codes – the sequences are of the form (**Function1** OR **Function2**) followed by (**Command1** OR **Command2**), or the other way around, where functions are fully expanded and commands are represented as single actions (thus assuming a simple pre-processing of the binary). The activity model for **Function1** can be found in the Appendix – in order to take into account every possible form the malevolent code can take, the model uses single hyperedges for different potentially generated sequences of instructions, thus applying the weak dependency feature discussed in Section 5.3. We also built 3 logs by injecting instances of the models (with additional action symbols not present in the models) in the binary of the PuTTY application [100]. The final sizes of the logs were around 524K tuples (with 1 instance of the malware), 531K tuples (100 instances), and 601K tuples (1000 instances).

In all the activity models used in the experiments, we avoided the use of function ϵ , in order to put a limit on the chances for pruning. All the experiments were run on an Intel Core i7-3770K CPU clocked at 3.5GHz, running Windows 8.1, with 12GB RAM.

Fig. 5.14 (top) shows the number of log tuples processed per second in Rounds 1 and 2.

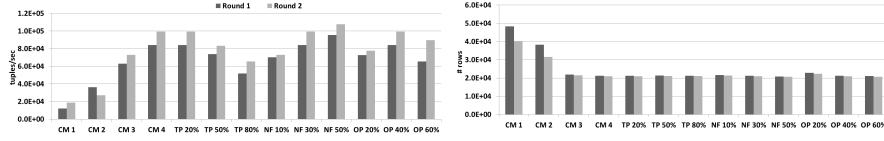


Fig. 5.14. Tuple rates (top) and maximum size of the AM-Index (bottom) in Rounds 1 and 2.

The results confirm our expectations and show very satisfying overall performances. Generally, the number of tuples processed per second is very high – in 8 out of 10 cases it is higher than 50K tuples/sec, and it is over 67K on average. We can notice that, in the majority of cases, the presence of “noise” in the log or overlap between consecutive instances have an immediate benefit on performance, basically because they reduce the overall number of instances to be detected. On the other hand, the effect of truncating more sequences worsens performances – this can be explained by observing that more partial instances have to be managed (before pruning occurs) in these cases. If we examine the differences between Rounds 1 and 2, i.e. when moving to more constrained activity models that allow for fewer instances, the results again confirm our expectations – we obtain a performance gain in all but one case, with an average gain around 17% (and upto 55.8%).

Fig. 5.14 (bottom) reports the maximum size of the AM-Index for Rounds 1 and 2 in terms of number of table rows. Interestingly, the size of the AM-Index appears extremely stable with respect to the configurations used: if we exclude the CM 1 and CM 2 cases (which correspond to a much larger number of instances in the log) the differences among the other cases are always under 6.8% (and 1.78% on average). On the other hand, the index appears to use slightly more resources per instance when using more constrained models: on average, the index is just 4.2% smaller in Round 2.

Finally, Fig. 5.15 reports the results for Round 3. Again, the framework appears to provide extremely good overall performance (160K tuples/sec. on average). In addition, we can notice that both the tuple rates and the maximum size of the index scale in a largely sub-linear way with respect to the number of instances in the log.

5.7.3 Additional Information About the Experimental Setting

The three real-world activity models used in Rounds 1 and 2 in addition to the model in Fig. 5.1 (top) are shown in Fig. 5.16. The temporal constraints added in Round 2 are:

- $\tau(h_3) = (1, 2)$ in the model of Fig. 5.1 (top);

5.7. Indexing and Detecting Activity Instances

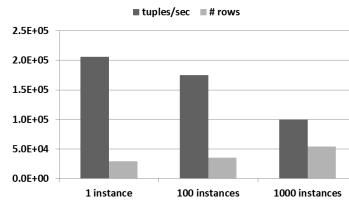


Fig. 5.15. Tuple rates and maximum size of the AM-Index in Round 3.

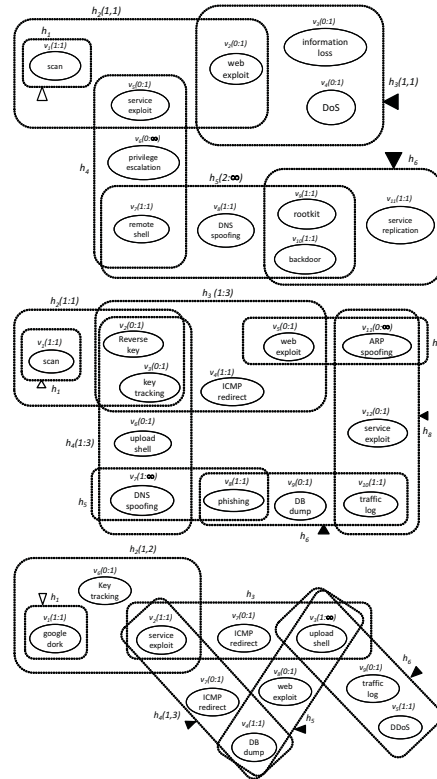


Fig. 5.16. Activity models used in Rounds 1 and 2.

- $\tau(h_4) = (1, 4)$ and $\tau(h_6) = (1, 1)$ in the model of Fig. 5.16 (top);
- $\tau(h_5) = (1, 3)$ and $\tau(h_7) = (1, 3)$ in the model of Fig. 5.16 (center);
- $\tau(h_3) = (1, 4)$ in the model of Fig. 5.16 (bottom).

The activity model for **Function1** is reported in Fig. 5.17.

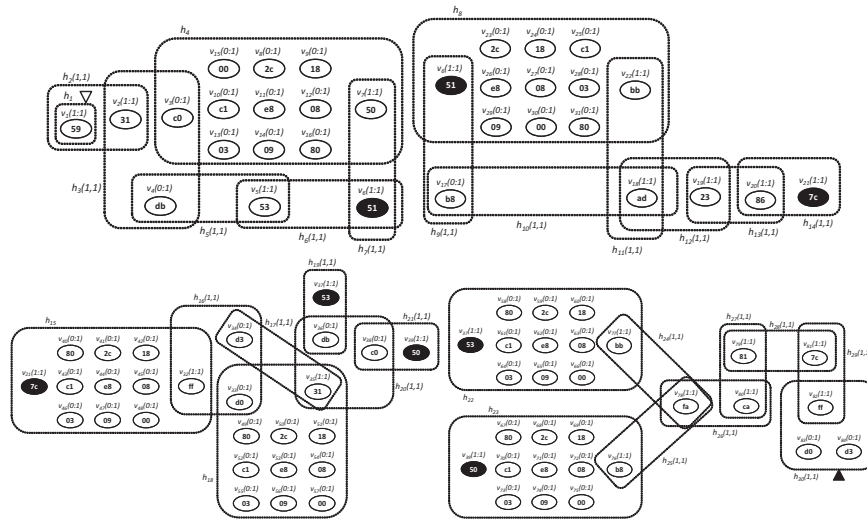


Fig. 5.17. Activity model for Function1 (some vertices, depicted in black, are represented twice for ease of presentation).

5.8 Summary

This chapter has presented a hypergraph-based framework for modeling and detecting malevolent activities. We have studied and rigorously characterized consistency checking, equivalence, and minimality of models, along with the general problem of hypergraph-based activity detection. In addition, we have developed an index data structure that proved very efficient in extracting occurrences of multiple activity models.

A Virtual Environment for realistic Cyber-Security Scenarios

This Chapter describes SmallWorld, a scalable software platform designed to reproduce realistic scenarios achieved by the immersion of real systems into a software defined virtual environment. SmallWorld enables the assessment, teaching and learning of Cyber-Security related aspects in different areas and for various purposes. One of the main features of SmallWorld is the support for designing and building complex scenarios which are dynamic and reactive and where a number of autonomous software agents can be deployed. Agents are able to reproduce the behaviors of human users and/or malicious applications into a SmallWorld scenario making it a more realistic testing environment.

6.1 Introduction

Cyber security issues have an ever increasing social-economical impact both for citizens and enterprises, then the availability of tools allowing to improve the awareness of cyber-space threats, to learn how handle them and to assess the effectiveness of prevention and defence solutions is critical for the safeness of IT services. Traditionally, security assessment and penetration testing activities are performed on real networks while the training of security specialists is made on insulated and static virtualized systems. The typical cyberspace user learns security issues only by direct experience exposing himself to high risks.

This Chapter proposes SMALLWORLD, a software platform enabling the assessment, teaching and learning of security-related aspects in different areas and for various purposes. SMALLWORLD exploits innovative and state-of-the-art virtualization and cloud technologies for reproducing in a realistic setting a hybrid environment where large distributed computer systems can be deployed and from where they can interact with real life entities (users, software and hardware). One of the main features of SMALLWORLD is the support for designing and building complex scenarios which are dynamic and reactive and where a number of autonomous software agents can be deployed.

Agents are able to reproduce the behaviors of human users and/or malicious applications into a SMALLWORLD scenario making it a more realistic testing environment. By using agents, according to their intelligence, the scenario may evolve over the time and can produce, for example, unexpected and unpredictable events that are interesting to study and analyze through simulation logs.

Keeping skills up to date is an extremely important requirement for security professionals. As example, penetration testing skills take time to develop while the information security landscape changes on a daily basis. A platform like SMALLWORLD allows penetration testers to build out complex virtual networks that can be used as practice labs.

Business companies and enterprises can resort to SMALLWORLD to provide training courses to their employees and make them aware of security risks, reducing incidents and hence saving money. SMALLWORLD can also be useful for cyber security certification entities and incidents response teams (SOC, CERT and CSIRT), which may build real-like challenges for assessing skills of candidates/employees. SMALLWORLD can be used by researchers to generate real system logs in order to analyze the propagation of malwares and test the effectiveness of new intrusion detection/prevention algorithms.

Another application of SMALLWORLD is the support for cyber-space users in advancing their learning path from familiarity to assessment by making available realistic training scenarios. In this way, users acquire practical skills performing exercises and get experience in the adverse nature of the field. Furthermore, they can apply their newly acquired skills to novel and mutable situations (thanks to agents) where success depends on their ability to make the right decisions quickly and eventually work in a team [101]. For these reasons, SMALLWORLD is particularly useful in building and manage Cyber Ranges. To better understand this assertion we have to briefly explain what a Cyber Range is. A Cyber Range [102] is a virtual environment that is used for cyber-warfare training and cyber-technology development. It provides tools that help strengthen the stability, security and performance of cyber-infrastructures and IT systems used by government and military agencies. Cyber Ranges function like Shooting or Kinetic Ranges, facilitating training in weapons, operations or tactics. Thus, cyber-warriors and IT professionals employed by various agencies train, develop and test Cyber Range technologies to ensure consistent operations and readiness for real world deployment. Goals of a Cyber Range include:

- Replicate large scale, complex and diverse networks and users activities
- Enable the development and deployment of state-of-the-art cyber-testing capabilities
- Facilitate the scientific use of cyber-testing methods
- Provide a virtual environment for the quantitative, qualitative and realistic assessment of potentially ground-breaking cyber-technologies for research and development

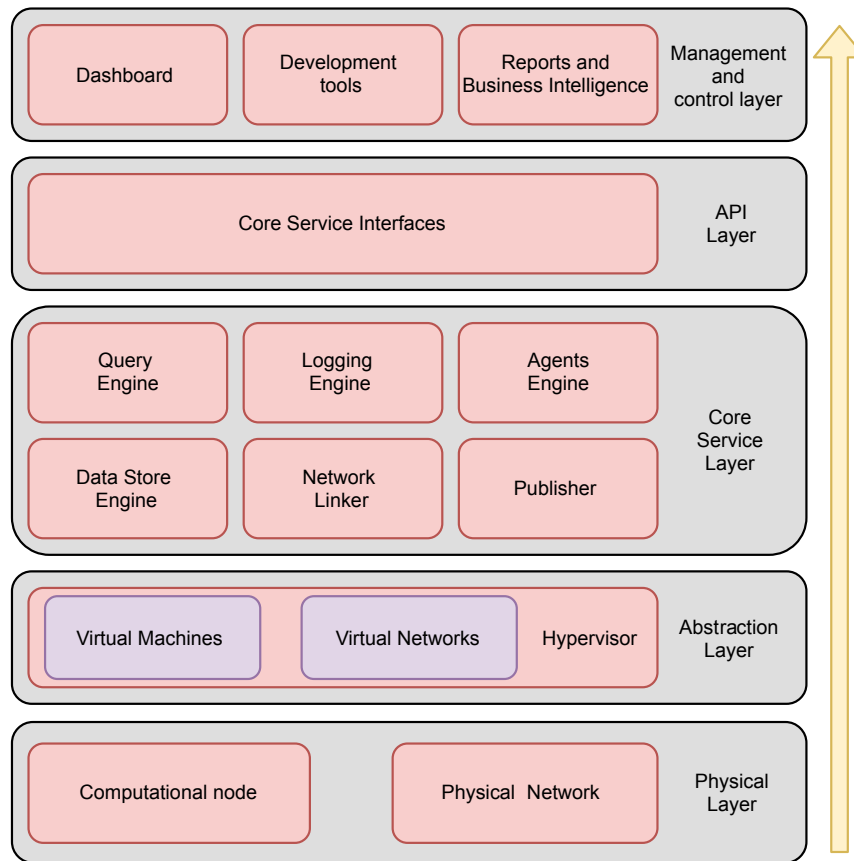


Fig. 6.1. SmallWorld Architecture

6.2 SmallWorld

SmallWorld has been developed with the main objective to be extensible and hypervisor-independent. To achieve these goals, it has been designed as multiple layers system, where the components of each layer cooperate among them to implement higher abstraction level services by exploiting the underlying tiers. A schema of the resulting architecture is reported in Fig. 6.1.

The five layers of the SmallWorld architecture are described in details by the following subsections.

6.2.1 Physical layer

This layer hosts computational, storage and networking hardware configured in a suitable way in order to offer fault tolerance, business continuity and

data replication mechanisms services for proper and scalable operation of the hypervisors. The above hypervisor layer abstracts and hides bare metal details which can then be easily changed/improved for scalability purposes without impacting on the overall system operations. In particular, at this level we find:

- Routers, switches, firewalls and other networking devices, providing links among computational and storage nodes. This layer is also in charge of opening SMALLWORLD entities to the world outside. It is important to properly choose these components in order to ensure suitable performances for proper handling of complex scenarios.
- Storage area networks (SAN) and network attached storages (NAS), which offer, at low level, data storage, backup and replication. These are critical components because they are in charge of storing and handling all the platform data and meta-data. So an high level of reliability is required and delivered, e.g from ad-hoc RAID configurations.
- Servers and elaboration nodes which offer the computational resources needed to execute the SMALLWORLD scenarios.
- Other hardware, e.g. uninterruptible power supplies (UPS), employed to protect the whole systems from an unexpected power disruption that could cause data loss and service downtime.

6.2.2 Abstraction Layer

This layer hosts the virtual machine monitor and the network hypervisor, which respectively enable to define via software the virtual computational nodes, along with the above operating systems and software layers (software defined systems) and the virtual network infrastructure (software-defined networking). There exists many hypervisors solutions that offers these features and that be employed in this layer. The current prototype of SmallWorld relies on Openstack [103, 104], however other implementations, i.e. VirtualBox [105], OpenNebula [106], are planned.

The required features for a hypervisor to be used as SmallWorld components are the following:

- Multi-User, i.e. it must be possible to create users and groups with different privileges;
- Multi-Tenancy, i.e. a single instance of the environment has to be able to serve multiple clients. Each client is called a tenant and each of them has the ability to customize some parts of the application. While SmallWorld gives access to multiple users, each tenant's data must be kept isolated and invisible to other tenants;
- Snapshot support, i.e. it should be possible to save, reset and reload the state of a virtual-machine.

The network hypervisor allows to create virtual networks that are completely decoupled and independent from the underlying physical network. This hypervisor enables segments of a virtual network and network devices, such as

switches, routers and firewalls, to be managed independently and to be provisioned dynamically to accommodate the traffic demands.

6.2.3 Core Service Layer

This layer hosts the main software components that implements the core SmallWorld features which are in turn exposed by the above API layer. These components exploits the hardware abstractions offered by the hypervisors and can be implemented on top of different virtualization products.

Each block shown in Fig. 6.1 for this layer acts like a bridge between the above layers and the abstraction layer or physical layer. The *Network Linker* communicates with the underlying network hypervisor and introduces facilities to manage the networking services (routing, switching, bandwidth shaping, firewalling, policies). The *Publisher* is responsible to install applications (e.g. vulnerable software, malware, etc.) and agents in a scenario. The *Datastore Engine* handles information that must be stored into suitable databases on the basis of the data type. This component does not use the abstraction layer.

Data kept by the datastore engine are retrieved by the *Query Engine*, and used by the *Management and Control Layer* to gather and compute statistics about the platform usage, e.g. users' and agents' activities, network bandwidth usage, traffic logs and other informations.

The *Agent Engine* is basically an agent based [107] [108] real-time simulation engine. It performs four main functions: (i) translates Agent Behavior from agent description language (ADL) format (see next section) to executable code; (ii) provides an API for deploying and planning all simulation steps; (iii) executes agents' behaviors in cooperation with each other providing an efficient messages delivery system; (iv) exposes an interface to extract efficiently simulation logs. The Engine is able to perform a scalable simulation by deploying a distributed simulation environment [109] over a pool of available machines. A *Controller* entity permits to add *worker nodes* to the simulation each of which handles the execution of a little cluster of agents. The Controller orchestrates the behaviors of workers nodes.

6.2.4 API Layer

This layer introduces a platform independent API which defines the SmallWorld interface. This API is used for the implementation of the applications of the Management and Control Layer and is the key to implement the SmallWorld scenarios design and development independently from the software technologies used in the underlying layers.

The SmallWorld API is made available both as a Java framework and as a set of REST services.

6.2.5 Management and Control Layer

Thanks to the efforts made in the underlying levels, it has been possible to realize a set of GUIs completely independent from the technologies and software used in the rest of the environment and capable to provide simple yet effective management and development tools.

In particular, the following facilities are provided:

- A Dashboard, from where is possible to manage the scenarios, agents and virtual-machines. It also allows to display system usage and statics, set scenario parameters, handle user access and account management.
- A Report tool, which provides statistical data about the running scenarios.
- A set of Development Tools which include: an agent development tool, a scenario development tool and a virtual-system development tool.

SMALLWORLD provides different kinds of access and two type of installation, i.e. in site or in cloud, given that is suitable for many different usages. An enterprise user that needs to transfer a great amount of data (virtual machines images, logs from a system or a data warehouse) into SMALLWORLD or does not want to export personal data and can afford a variable hardware investment may opt for a in site deployment solution. The features to deliver to the client, and the respective cost, are fully customizable thanks to the modular design of the environment. On the other hand, SMALLWORLD can also be deployed on a cloud environment and made available as a service. This last solution allows the user to have immediate access to SMALLWORLD avoiding hardware investment and configuration efforts. The next subsections summarize the main SMALLWORLD tools which support the user in the design and development of cybersecurity scenarios.

6.2.6 Scenario Development Tool

A scenario is a composition of virtual machines, agents, policies and network devices. SMALLWORLD provides a *Scenario Development Tool* (SDT) realized to help the user in deploying a preloaded environment or to build a new one, starting from a repository of virtual machines, agents, configurations and devices. It is a graphical tool that allows the user to create scenarios, such as network topologies and system configuration, using graphic simplifications, such as Drag-and-Drop elements, similar to those offered by orchestration frameworks like Juju. Projects can be available at any time and anywhere, as they can be saved in the cloud. All scenario information are stored in description files, written in a *Scenario Definition Language* (SDL) which is then used by the SMALLWORLD engine in order to deploy the scenario on chosen computational nodes, through the use of the publisher and the network service. The SDL is a meta-language used to express:

- Network properties, such as bandwidth limits, IP classes, NAT addresses, Firewalling policies, etc.;

- Nodes details, such as Operative System, installed Software and loaded Agents;
- Access points, such as remote access policies are defined here, like VPN or VNC tunnel.

6.2.7 Virtual-System Development Tool

SMALLWORLD comes with a set of pre-built VMs hosting software components, vulnerable applications and malwares. SMALLWORLD allows to include in its repository both new VMs, built from scratch, or VMs obtained as customizations of those already available. A new VM is built starting from a clean image of the target Operating System. The IDE assists the user in writing an initial configuration script that the system will execute at the first boot in such a manner to configure the network settings of the system and install the required software and plug-ins. The tool allows users to add new programs to existing VMs exploiting the container technology [110]. A container hosting a bugged version of a software, can be prepared outside the SMALLWORLD environment and then uploaded when ready. The developed container can become part of the software and vulnerabilities repository and may be easily deployed on any container-aware VM. All this elements (VM images, software and malware) are stored inside the SMALLWORLD repository and made accessible from other tools.

6.2.8 Agent Development Tool

The Agent Development Tool (ADT) is a graphical tool realized to design the behavior of agents with a visual paradigm. Behaviors are visually specified as finite state machines. The tool helps the user to make a consistent agent behavior by automatically carrying a series of checks on the automaton in order to validate its execution flow. It provides some pre-configured complex actions in addition to the basic functions defined in the standard package. Through a graphical approach users can add new types of messages, states and transitions and create custom functions. In order to define agent behavior [111] [112], preserving an appropriate abstraction level, a meta-language, called *Agent Definition Language (ADL)*, has been defined. The ADL is based on XML and allows to describe a finite state automaton in terms of its states, transitions and actions. States can be hierarchical [113] and each transition allows to inject default or custom actions such as issuing HTTP Requests or making a system log. A type of transition, named *Temporal Transition*, has been introduced for simulating the waiting for completion of an activity. The waiting time may be established by sampling a random number from a suitable distribution. Through the definition of particular messages the language makes possible the cooperation of two different agent behavior and the exchange of local parameters and configuration. The ADL source code is then translated into an agent executable code which can be instantiated and run by the Agent Engine.

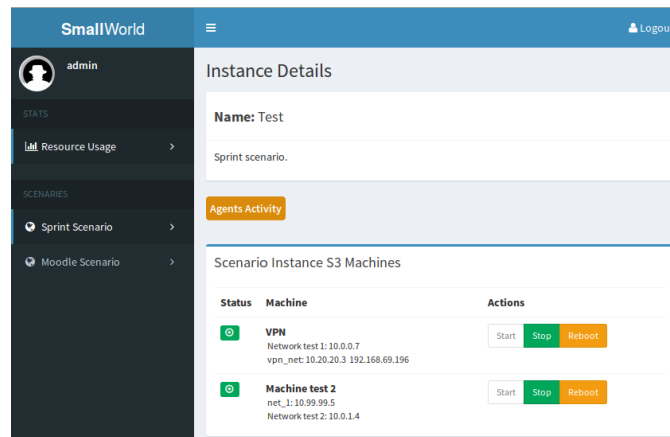


Fig. 6.2. Management Dashboard

6.2.9 Repositories

One of the SMALLWORLD main strengths is its rich catalog of vulnerable software, operating systems, network templates and agent behaviors delivered with the platform. SMALLWORLD is equipped with repositories which contain:

- Operating Systems: a set of particular version or distribution of an OS affected by security issues;
- Software: a set of particular versions of vulnerable software;
- Agents: a set of pre-modeled agents that can reproduce the behavior of an attacker, of an employee or of a general user.

Each item is enriched with meta-data that describe the item itself. This is particular useful because it allows the user to search the needed component for its scenario by querying, through the query engine, the repository on meta-data information. In order to allow the search engine to suggest the user a list of components related to the search query they are arranged in categories: vulnerability typology, difficult level, skills required, etc.

6.2.10 Management Dashboard

The management Dashboard (see Fig. 6.2 for a screenshot) is the web-based interface designed to communicate with SMALLWORLD's API for handling scenario instances. It also allows to monitor the available resources of the server machine which hosts the scenarios. Administrators can customize the configuration parameters of both scenarios and agents. The Dashboard is able to start, pause and kill agents' instances at any time after they are injected into a scenario. Once a new scenario is instantiated, an administrator can obtain information about the status of the active machines and can monitor

the agents' logs gathered by the Dashboard. Logs can be filtered with several log levels to facilitate the monitoring phase. In addition, the management dashboard allows to define a VPN entry point for accessing the scenario from the external of SMALLWORLD. Once this entry point has been defined, the administrator can request the creation of a certain number of VPN certificates. A possible extension provides the integration within the dashboard of the Scenario Development Tool discussed in the above section.

6.3 Case Studies and experimental results

In this section we present two applications of SMALLWORLD. The first focuses on the assessment of security properties. It is a laboratory used during a master course to introduce students to the main information security concepts through a series of practical examples that includes: assessment, network and port scanning, auditing, remote and local exploitation. The second instead concerns the cybersecurity learning process and has been realised throughout the integration of SMALLWORLD with the Moodle e-learning platform. A new specific scenario, more complex than that of the first case study, has been designed and implemented. The scenario was devised to be used as test and training laboratory during the second Cybersecurity Master course organized by Poste Italiane, in collaboration with the PosteCERT and the University of Calabria. The course has been held in September 2016, and many details cannot be disclosed because of intellectual property constraints.

To conclude, we quantitatively evaluated the performances of SMALLWORLD in particular we investigated about its scalability. We took advantage of the scenario proposed in the second case study to set up a number of experiments in order to assess CPU consumption, memory consumption and start-up time vs. a variable number of instances of the proposed scenario.

6.3.1 Case study I : security assessment

The aim of the first scenario, see Fig. 6.3, is to demonstrate the basic functionalities and usage of the proposed platform and has been developed during a master thesis at the University of Calabria.

It is composed by three servers, one load-balancer, one firewall, one router, one red agent and a series of white agents, the details of each component are listed below:

- Kali Linux host, connected to the VPN access point external network 192.168.73.X;
- Windows Server that hosts a vulnerable web-log (blog) platform;
- Linux Server hosting a vulnerable hotel booking system;
- Network router, which forwards packets to the target host and acts as VPN access point;

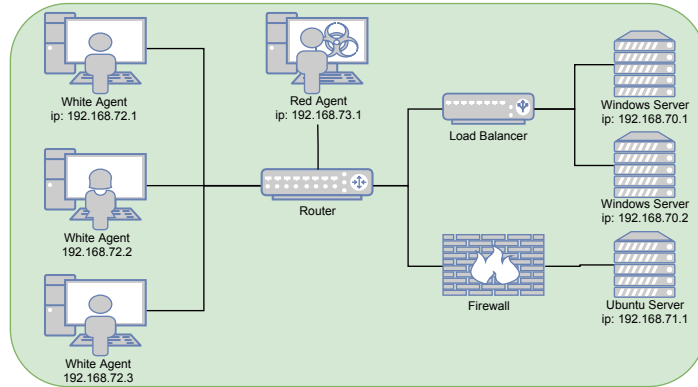


Fig. 6.3. An example of scenario in SMALLWORLD

- PFSense VM, connected to internal network 192.168.70.X and to the external network 192.168.73.X, which acts as load balancer;
- Many White Agents and one Red Agent, connected to the external network 192.168.73.X, that are respectively in charge of generating normal network traffic and executing attacks against the machines of the network.

Table 6.1 summarizes the software installed and a portion of the systems vulnerabilities. At the current development stage, it is possible to execute a pre-configured scenario using the dashboard, described in the previous section. Once a scenario is up and running we can start the agents execution and observe their actions.

Table 6.1. Software and vulnerabilities

OS	Software	Vulnerabilities	IP address
Windows Server	Apache 2 Httpd, PHP, MySQL, WordPress	Scf Exploit	192.168.70.1
Windows Server	Apache 2 Httpd, PHP, MySQL, WordPress	Scf Exploit	192.168.70.2
Ubuntu Server	Apache 2 Httpd, PHP, MySQL, Joomla!	Shellshock, SQL Injection	192.168.71.1

An instance of a White Agent reproduces the behavior of a typical user which browses a website. Using a Java browser engine, it is able to generate the same kind of traffic which would result from a human user interaction. In particular it can navigate between the pages of a portal, add comments or book reservations.

An instance of a Red Agent emulates the behavior of an attacker with the intent to violate the systems deployed in the running scenario. The activity

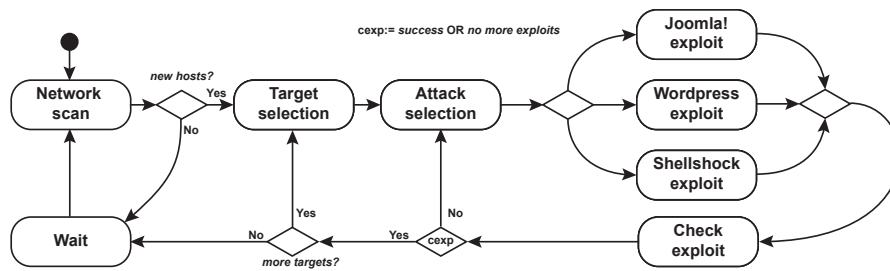


Fig. 6.4. Red Agent Activity Diagram

diagram depicted in Fig. 6.4 describes the execution flow of a Red Agent. For the proposed example, three different kind of attacks have been implemented:

1. **Shellshock**. It exploits a family of security bugs that affected the Unix `bash` shell. Many Internet services use `bash` to process certain requests. If a vulnerable version of `bash` is installed, an attacker can gain unauthorized access to a computer system.
2. **Scf Exploit**. Creative Contact Form (v. 0.9.7), a Wordpress extension, is liable to a vulnerability that lets attackers upload arbitrary files to the affected computer; this can result in arbitrary code execution within the context of the vulnerable application. An attacker can exploit this issue using a web browser.
3. **SQL Injection**. The K2 component (v. 1.0.1) for Joomla! (v. 1.X, 2.x and 3.X) is liable to an SQL-injection vulnerability because it fails to properly sanitize user-supplied data before using it in an SQL query. Exploiting this issue could allow an attacker to compromise the application, to gain access, to modify data or to exploit latent vulnerabilities in the underlying database.

Each time an exploit succeeds the agent loads a backdoor into the server and sends a notification to the dashboard, then a message is shown to the user.

6.3.2 Case study II: e-learning

Since one of the main SMALLWORLD intents is to support the learning process of new knowledge in the cybersecurity domain, it was considered appropriate to integrate it with an e-learning platform, such as Moodle.

Moodle (Modular Object-Oriented Dynamic Learning Environment) is an open-source e-learning platform based on the idea that learning would be facilitated by the production of tangible objects. It is realized in PHP and JavaScript languages with a modular approach in order to facilitate and encourage the development of plugins that implement additional features. The goal of this integration is to further enhance the learning process, adding a

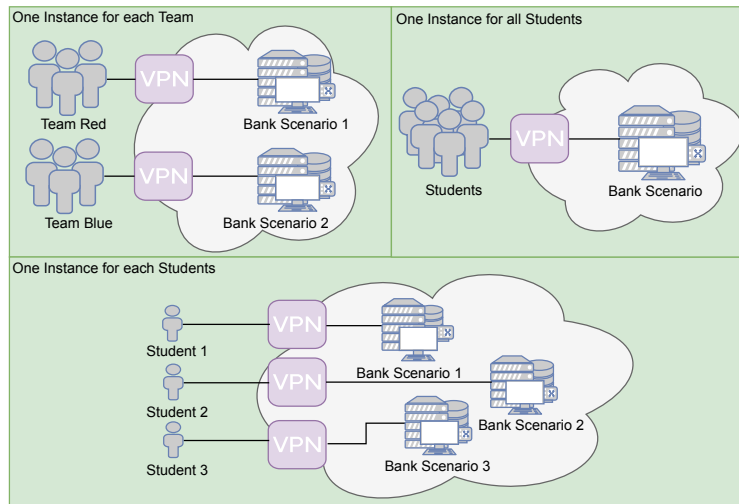


Fig. 6.5. Plugin Configuration types

practical element to one or more quizzes, defined by Moodle. In this way, the user is no more limited to answer questions using only his theoretical knowledge, but she/he also faces practical challenges, which allow her/him to better consolidate the concepts she/he studied.

In the *Computer Science Curricula 2013* document [101], the ACM defined three types of desirable learning outcomes:

- **Familiarity:** indicates the student theoretical comprehension of the proposed concepts. This is usually achieved via books and lectures.
- **Usage:** indicates the student conceptual comprehension, he can apply it correctly when it's required. This is usually achieved by a mix of lectures and practical exercises.
- **Assessment:** indicates that the student can correctly recognize a given concept in practice and apply it as solution to some related problem. This is usually achieved via lectures and practical experimentations.

The integration of SMALLWORLD within Moodle eases the achievements of the above goals. Such integration has been obtained by developing two plugins.

One of the plugins allows the administrators to authenticate on SMALLWORLD through Moodle and retain the access on both platforms without having to type several times their credentials.

The other plugin allows to create an association between quizzes and scenario(s), as well as making available VPN credentials to the students.

After defining one or more quizzes within the e-learning platform through the normal process, the teacher/administrator can assign one or more scenar-

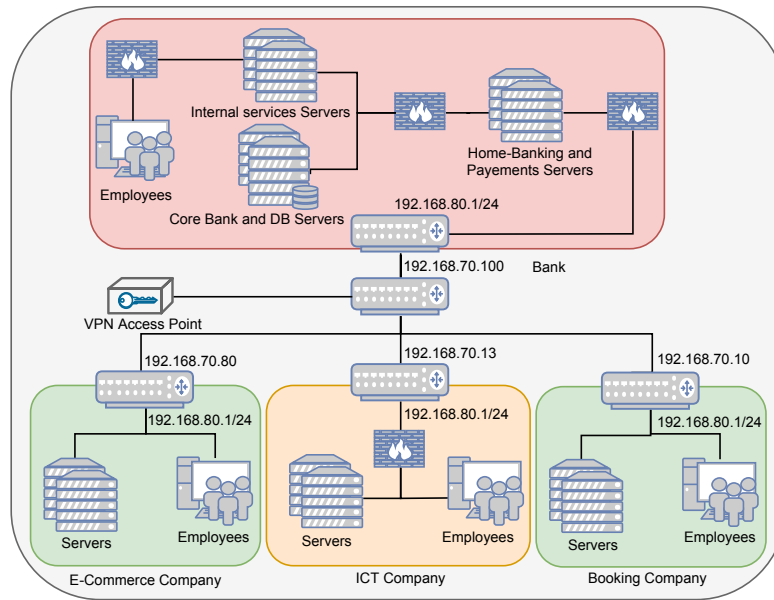


Fig. 6.6. An example of a scenario composed by more sub-scenarios

ios to each of them, in which the students will have access for solving the proposed questions.

Through the plugin it is possible to decide whether the scenario should be shared among all students, or a new instance should be built for each of them. For matches among teams it is also possible to create instances of scenario in which only players of the same team can collaborate.

After the configuration and deployment scenario phase, the screen shows the current status of the VPN, which require a bit of time to be generated. When SMALLWORLD makes available the certificates, each student participating in the quizzes will have his personal VPN through which he can access to the scenario.

Through the integration of LDAP provided by both Moodle and SMALLWORLD, it was possible to create a common database of users.

Using the proposed integration, the scenario depicted in Fig. 6.6 was realized. The scenario has been designed to be complex enough to be proposed as test and training laboratory for the second edition of the Cybersecurity Master course (9 servers, 5 routers, 2 firewall and 4 white agents) organized by Poste Italiane in collaboration with the PosteCERT and the University of Calabria. It reproduces the behavior of four cooperating companies: a Bank (SmallBank), an e-commerce (SmallSell), a booking company (SmallBook) and a ICT development firm (SmallDev).

Considering today's Cybesecurity scene, the ability to reproduce scenarios involving e-commerce and booking companies is mandatory for a platform like SMALLWORLD. In the past few years, several attacks has been perpetrated against on-line shops, like Target [114] and Ebay [115], just to recall two cases where data of thousands of users have been exposed to criminals.

The entities involved in the scenario are:

- The main router 192.168.70.X;
- The VPN access point, which allows the connection of external users (students, developers, administrators) inside the network of the running scenario;
- Four border routers, 192.168.70.10, 192.168.70.13, 192.168.70.89 and 192.168.70.100, that isolate the intranet of each company from the main network;
- A certain number of Firewalls, used to filter malicious traffic and protect the private networks;
- A certain number of Servers (LAMP Servers, SQL Servers, Tomcat Servers etc.), used to properly run the companies' software (websites, web applications, web services etc.)
- A group of White Agents inside the companies that simulates the behavior of normal employees (accepts payments, transfers money, updates accounts, etc.);

In this case, the installed software is very heterogeneous, so both for brevity and confidentiality reasons a complete disclosure is avoided, but we can argue the reasons that are behind the choice of the four companies. Having said that, below a bit more details on such companies are provided, explaining what are their goals and the network topology adopted.

SmallSell employs the e-commerce whose front-end is implemented with the popular Content Management System (CMS) Prestashop. Its subnet is composed of some servers which host the service and the database, in addition to a number of computers used by the company employees. Periodically, the employees connect to the CMS administration portal through which they perform their functions such as verifying payments, by interacting with the bank, shipping orders or emitting refunds.

The ICT Company offers a wide variety of services that include web services for file hosting, email accounts and web hosting to the users of the scenario. The network topology consists of a web server and a software testing server. Given the presence of sensitive data, including personal files and private software, SmallDev employs a firewall between the Gateway and the server. In this context, the employees perform uploads and downloads of files, to emulate software updating or development.

SmallBook is a company entirely similar to the e-commerce one, with the only difference consisting in the vulnerability of the CMS exposed to the users.

SmallBank offers financial services and payment gateways, like credit card payments, wire transfer and home banking to the other companies. The bank's

network is structured in three levels. The first level contains two servers, which set out the payments and home banking services to the users. A firewall protects the entire network from unauthorized traffic, coming from the outside. The two servers, in order to carry out the services, must access the db servers. The latter are located in the second level of the network. This level is separated from the first through a second firewall. The objective of this firewall is to prevent unauthorized access, if at least one of the two first-level server gets compromised. At the second level of the network there are also the servers used by the employees to manage the entire set of services offered by the bank. Their typical daily activities are e.g. transaction approvals, account management, money transfer and so on. The same employees reside in the third level, which is isolated from the rest of the network by means of a third firewall, to avoid malicious insiders.

The different network topologies, jointly to the used services, determine different levels of difficulty, that SMALLWORLD users can face. Such difficulties are made explicit in Fig. 6.6, with the colors associated with the networks.

Inside this “small world” it is possible to reproduce large scale and highly risk attacks in a controlled environment populated with synthetic, yet realistic, data with the aim of showing how even small bugs can open a door to hackers.

Leaving the scenario running for several days, the agents generate a series of events that lead to the evolution of the scenario. The result of this execution is a set of log files that can be accessed in different levels of the system (e.g. pcap or Apache logs files), taking advantage of the Logging Service. These data may be useful to study the behavior of the employees to understand how to detect anomalous activities.

To speedup the development time, Wordpress and a series of its extensions have been used to reproduce a multi structure booking system. Every component has been analyzed and tested.

The main aim of this scenario is to spur students, or general SMALLWORLD users, to find their own way to complete the challenge which consists in conquering at least one machine per sub-network, or, in other words, to hack at least one machine for each company, and respond to the proposed quizzes, using the information gained from the scenario.

6.4 Summary

Cyber security is currently a rapidly growing field due to the increasing number of cyber incidents to which companies are subject. In most cases the best security strategy is to prevent and investigate possible scenarios that an attack could cause. Companies and incident response teams need a system to provide training courses built on top of real-like challenges and to perform security assessment and penetration tests without compromising the production systems. IT users need to smoothly learn how to safely live in a cyber space by increasing their awareness of threats without exposing themselves to real

risks. Researchers working in cybersecurity field, and not only, need a great amount of real-like system logs, security environments to test new algorithms or to study malware propagation.

We argue that the flexibility of virtual environments will play a critical role in many cybersecurity related aspects. Problems like the assessment of newly devised intrusion detection techniques, the evaluation of skills of cyber defense team members, the evaluation of the disruptive effects caused by the diffusion of new malware, are just few examples of issues that cannot be directly addressed in production systems even though they require realistic operating environments in order to be suitably performed.

SMALLWORLD has been designed as a modular, scalable and evolvable platform able to address the above mentioned needs. It has a great improvement potential over time because its features may increase and evolve as the result of real case-study applications. For example, because of their pervasiveness in the everyday life of people, very important issues are those related to the security of mobile devices and applications. In the near future, SMALLWORLD could be extended to accommodate suitable features, e.g. like those reported in [116], for coping with mobile devices.

SMALLWORLD is able to exploit all the advantages induced by the available hypervisor technology, OpenStack, in terms of: scalability, availability, elasticity, performance and so on.

Conclusions and Future Work

This Chapter presents our final thoughts about Cyber-Security and the contributions provided by this thesis, a brief overview about ongoing and future work is also reported.

7.1 Ongoing and Future Work

All the proposed solutions in this thesis have been implemented as prototype or proof-of-concept, so the main purpose was always to get a working solution in the shortest possible time. Due to its intrinsic complexity, a lot of effort has been made during the development of SMALLWORLD and for prototyping purposes we decided to build the platform on top of OpenStack. This decision streamlined the above process, for example, OpenStack provides a complete support to manage the entire network level (subnets, routing, VPN, load balancer, IDS and so on), called Network Function Virtualization (NFV), and infrastructure provision (but not virtual-machine provision), it also exposes a complete set of API useful to manage the life cycle of a virtual-machine and to configure the environment. However, these were only few examples, in fact we know that OpenStack is a complete solution for cloud computing and for this reason its support in building a prototype was huge. Beyond the prototyping face, OpenStack is not the better choice. In fact, as we need to customize our platform and features or if we need to move to another Hypervisor that offers less support than OpenStack a lot of effort is required in order to implement all the missing pieces (Compute, Networking, Storage, Orchestration, Database, etc.). For these reasons, after have established the scientific value and the industrial interest in SMALLWORLD, I decided to take a step back and develop all the main layers of SMALLWORLD without relying on any particular Hypervisor or ready-for-use-solution.

The idea consists in using OpenVSwitch, a Software Defined Network tool, to realize a network abstraction layer shared between all computation nodes. A virtual-machine acting as scenario router (also known as VPNRouter), with

DHCP server enabled, and VPN access point was also needed. In this way is possible to create as many scenario networks as we want without relying on any particular Hypervisor feature. We just need to attach a VPNRouter to each set of VMs that make up a particular scenario and we are ready to go. Another key technology used to add a second level of abstraction is Vagrant. Vagrant fits in here by automating some of those boring repetitive tasks involved in building and configuring, groups of, virtual-machines and containers. These tasks become a much quicker and less involved process. You do not have to actually interact with a Hypervisor or the booting OS at all, just with Vagrant. It introduces the concept of boxes, these are the packages for Vagrant environments and in their most basic form they are an Operating Systems tailored to run with Vagrant and available from a public or private repository. So rather than downloading an Ubuntu image then booting a VirtualBox VM using that image, we can download the relevant Vagrant box and just use the 'vagrant up' command from our command-line to bring up a new Ubuntu VM, without touching the Hypervisor nor going through all the boot prompt. The development of these two layers was not a simple task but the results are very important. With Hacking Square, we can switch, for example, from OpenStack to KVM no additional efforts and without changing anything but only the Hypervisor name.

The project, resulting from the improvements described above, is known as HACKING SQUARE [117] and is currently available on-line and freely accessible. It is used during different grad school class work or to host the exercises and exams of the course named *"Metodi e Strumenti per la Sicurezza Informatica"* for the master degree in Cyber Security at DIMES, University of Calabria.

The platform can be managed, in a very simple way, from the web application. The administrator, or the teacher, can add, edit or delete new content, configurations, users and meta-data, while students can see the available scenarios, send a request access for a scenario to an administrator, download the VPN certificates to access to the corresponding scenarios, read teaching material and write to the forum. In order to make the building of new scenarios easier, to reduce the total amount of space in the repository and to improve the platform efficiency we are promoting the idea to use as few base boxes (vanilla Operating Systems) as possible and complete the customization of a virtual-machine through the use of a provisioning file. We need hard-code software or configurations in a box only when the vulnerability or the service we want to exploit can't be provisioned, for example a bug in a particular kernel version or services available in a particular Operative System version. Many other features characterize HACKING SQUARE, but a full description of the platform is beyond the scope of this section. We are constantly working to improve the platform and features offered, like tools, content and virtualized network functions to advance in the build of always more complex scenarios, functionalities to assisting students or also extend the virtual-machines deployment not only on local servers, but also on cloud services. Finally, we

hope these efforts will serve to establish HACKING SQUARE as a reference platform to design, build and deploy Cyber Ranges.

7.2 Conclusions

The history show us that criminals will dependably be anxious to break into our computers to steal and do damage. This because there will always some vulnerability in our systems, which give criminals the chance to do their malicious deeds. Computer engineers can design systems that are less prone to be violated. Security has been enhancing steadily for over 10 years. In 2000, password hashing algorithms were weaker, recommendations for solid passwords were uncommon, hacking strategies that are currently normally stopped were waiting to be put to use. Home computer firewalls started to show up and have enhanced relentlessly. Antivirus tools have enhanced and are almost universally installed. Security updates are automatically scheduled and happen, generally, noiselessly in background. Engineers are now mindful of coding habits that are probably going to leave flaws that hackers can use to break in or cause damage. Quality testing now effectively and strongly follows security gaps. These are all real advances that have made computers substantially more impervious to hacking and are probably going to keep on improving security. However more criminals are striking and plotting to strike every day. A portion of these criminals are very skilled and dedicated as the engineers are dedicated to build computer systems. In fact, for each new technique for defeating assaults that is produced, roused hackers are looking for approaches to break or circumvent it. What's more, in the end they do succeed, despite the fact that achievement is getting increasingly hard for them. In any case, the ideal toolset that will prevent every breach is not upcoming and is not probably ever to be there. The computing environment itself, offering more services every year and increasing the number of users day by day, continually adds new challenges. The growth in the power and minimization of the computers sizes is constantly anticipated to be near the end, however every year, they get littler, less expensive, and more powerful. The global attack surface grows as these IoT devices are utilized for more and more purposes, inviting hackers to exploit them and reuse them for more complex attacks. Nevertheless, the security of our computers has increased, the challenge is to grow our safety faster than the criminals assault us. This thesis proposed many tools and frameworks that we can learn to use in order to build a more effective security and more conscious users.

A

Appendix

A.1 Proof Sketch for Theorem 5.14

Statement: Given two activity models M_1 and M_2 such that $M_1 \preceq M_2$, it holds that $\mathcal{L}(M_1) \subseteq \mathcal{L}(M_2)$.

Assume that f is the witness mapping of $M_1 \preceq M_2$. Let $L = \ell_1, \dots, \ell_n$ be a log and let $\pi = h_1, \dots, h_m$ be a path such that $L \models_\pi M_1$. Consider the path $\pi' = f(h_1), \dots, f(h_m)$. We claim that $L \models_{\pi'} M_2$, so $L \in \mathcal{L}(M_2)$. Indeed, assume that $(1 = s_1, t_1), \dots, (s_m, t_m = n)$ is the m -segmentation of L required to exist according to Definition 5.6 $L \models_\pi M_1$. Precisely the same segmentation can be used to show that $L \models_{\pi'} M_2$. In fact, this follows by inspecting Definition 5.6 and by noticing that the conditions that have to be satisfied by the segmentation are satisfied by M_2 : The functions τ_2 and ϵ_2 are just less restrictive than τ_1 and ϵ_1 , and each fresh vertex $v \in V_2 \setminus V_1$, is such that its associated lower bound is 0.

A.2 Proof Sketch for Theorem 5.16

Statement: Given two activity models M_1 and M_2 , deciding whether $M_1 \leq M_2$ is **co-NP**-hard.

By inspecting the proof of Theorem 5.12 it emerges that the problem of deciding whether $\mathcal{L}(M_2) \neq \emptyset$ is **NP**-hard. Consider then a trivial model M_1 defined over the emptyset of vertices. The problem of checking whether $\mathcal{L}(M_2) \neq \emptyset$ reduces to checking whether $M_1 \leq M_2$ does *not* hold.

A.3 Proof Sketch for Theorem 5.18

Statement: The malevolent activity detection problem is **NP**-complete.

Membership in **NP** is trivial: it suffices to use π and the corresponding $|\pi|$ -segmentation as a polynomially-verifiable witness. We prove **NP**-hardness by polynomial-time reduction from HAMILTONIAN PATH [99]. Let $G_{in} = (V_{in}, E_{in})$ be an undirected graph. In order to decide whether G_{in} contains a Hamiltonian path, we build an activity model $M = \langle \mathcal{H}, \lambda, \tau, \epsilon, S, T \rangle$ with $\mathcal{H} = (V, H)$, $V = V_{in}$, $H = \{\{v_1, v_2\} \mid (v_1, v_2) \in E_{in}\}$, $\forall v \in V$, $\lambda(v) = (x, 1, 1)$, $domain(\tau) = domain(\epsilon) = \emptyset$, $S = T = H$. Then, we build a log $L = \ell_1, \dots, \ell_n$ where $n = |V|$ and $\forall \ell_i \in L$, $\ell_i.action = x$ and $\ell_i.timestamp = i$. Now, in order to include all log tuples in an activity instance, all of the vertices in V must be traversed exactly once. Thus, there exists a complete path π in M such that $L \models_{\pi} M$ if and only if G_{in} contains a Hamiltonian path.

References

- [1] A Astorino, A Chiarello, M Gaudio, and A Piccolo. Malicious url detection via spherical classification. *Neural Computing and Applications*, pages 1–7, 2016.
- [2] Andrea Pugliese, Antonino Rullo, and Antonio Piccolo. The ac-index: Fast online detection of correlated alerts. In *International Workshop on Security and Trust Management*, pages 107–122. Springer, 2015.
- [3] Antonella Guzzo, Andrea Pugliese, Antonino Rullo, Domenico Sacca, and Antonio Piccolo. Malevolent activity detection with hypergraph-based models. *IEEE Transactions on Knowledge and Data Engineering*, 29(5):1115–1128, 2017.
- [4] Angelo Furfaro, Antonio Piccolo, and Domenico Saccà. Smallworld: A test and training system for the cyber-security. *European Scientific Journal, ESJ*, 12(10), 2016.
- [5] Angelo Furfaro, Antonio Piccolo, Domenico Saccà, and Andrea Parise. A virtual environment for the enactment of realistic cyber security scenarios. In *Cloud Computing Technologies and Applications (CloudTech), 2016 2nd International Conference on*, pages 351–358. IEEE, 2016.
- [6] Angelo Furfaro, Luciano Argento, Andrea Parise, and Antonio Piccolo. Using virtual environments for the assessment of cybersecurity issues in iot scenarios. *Simulation Modelling Practice and Theory*, 73:43–54, 2017.
- [7] John Pescatore and Greg Young. Defining the next-generation firewall. *Gartner RAS Core Research Note*, 2009.
- [8] Christoph Michael and Anup Ghosh. Using finite automata to mine execution data for intrusion detection: A preliminary report. In *Recent Advances in Intrusion Detection*, pages 66–79. Springer, 2000.
- [9] R. Sekar, Mugdha Bendre, Dinakar Dhurjati, and Pradeep Bollineni. A fast automaton-based method for detecting anomalous program behaviors. In *Security and Privacy*, pages 144–155, 2001.
- [10] Seyit Ahmet Çamtepe and Bülent Yener. Modeling and detection of complex attacks. In *SecureComm*, pages 234–243, 2007.
- [11] Yu Liu and Hong Man. Network vulnerability assessment using bayesian networks. In *SPIE*, pages 61–71, 2005.
- [12] Marcel Frigault, Lingyu Wang, Anoop Singhal, and Sushil Jajodia. Measuring network security using dynamic bayesian network. In *QoP*, pages 23–30, 2008.

References

- [13] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeanette M Wing. Automated generation and analysis of attack graphs. In *Security and Privacy*, pages 273–284, 2002.
- [14] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *CCS*, pages 217–224, 2002.
- [15] Massimiliano Albanese, Sushil Jajodia, Andrea Pugliese, and V. S. Subrahmanian. Scalable analysis of attack scenarios. In *ESORICS*, 2011.
- [16] Massimiliano Albanese, Andrea Pugliese, and V. S. Subrahmanian. Fast activity detection: Indexing for temporal stochastic automaton-based activity models. *IEEE Trans. Knowl. Data Eng.*, 25(2):360–373, 2013.
- [17] Steven Noel, Eric Robertson, and Sushil Jajodia. Correlating intrusion events and building attack scenarios through attack graph distances. In *ACSAC*, pages 350–359, 2004.
- [18] Lingyu Wang, Steven Noel, and Sushil Jajodia. Minimum-cost network hardening using attack graphs. *Computer Communications*, 29(18):3812–3824, 2006.
- [19] Sebastian Roschke, Feng Cheng, and Christoph Meinel. A new alert correlation algorithm based on attack graph. In *CISIS*, pages 58–67, 2011.
- [20] Lingyu Wang, Anyi Liu, and Sushil Jajodia. Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts. *Computer Communications*, 29(15), 2006.
- [21] Ching-Hao Mao, Hsing-Kuo Pao, Christos Faloutsos, and Hahn-Ming Lee. Sbad: Sequence based attack detection via sequence comparison. In *PSDML*, pages 78–91, 2010.
- [22] Steven Noel and Sushil Jajodia. Managing attack graph complexity through visual hierarchical aggregation. In *Workshop on Visualization and Data Mining for Computer Security*, pages 109–118, 2004.
- [23] Shuzhen Wang, Zonghua Zhang, and Youki Kadobayashi. Exploring attack graph for cost-benefit security hardening: A probabilistic approach. *Computers & Security*, 32:158–169, 2013.
- [24] Ravi Jhavar, Barbara Kordy, Sjouke Mauw, Sasa Radomirovic, and Rolando Trujillo-Rasua. Attack trees with sequential conjunction. In *SEC*, pages 339–353, 2015.
- [25] Andrea Pugliese, Antonino Rullo, and Antonio Piccolo. The AC-index: Fast online detection of correlated alerts. In *Security and Trust Management*, pages 107–122, 2015.
- [26] Barbara Kordy, Ludovic Piètre-Cambacédès, and Patrick Schweitzer. DAG-based attack and defense modeling: Don’t miss the forest for the attack trees. *Computer Science Review*, 13-14:1–38, 2014.
- [27] Vivek Shandilya, Chris B. Simmons, and Sajjan G. Shiva. Use of attack graphs in security systems. *Journal Comp. Netw. and Communic.*, 2014:818957:1–818957:13, 2014.
- [28] Giovanni Vigna. A topological characterization of tcp/ip security. In *FME*, pages 914–939, 2003.
- [29] Virginia N. L. Franqueira, Raul H. C. Lopes, and Pascal van Eck. Multi-step attack modelling and simulation (msams) framework based on mobile ambients. In *ACM SAC*, pages 66–73, 2009.
- [30] Wolter Pieters. Ankh: Information threat analysis with actor-network hypergraphs. CTIT technical report series, 2010.

-
- [31] Benjamin Morin, Ludovic Mé, Hervé Debar, and Mireille Ducassé. M2d2: A formal data model for ids alert correlation. In *RAID*, pages 115–127, 2002.
 - [32] Fabrizio Baiardi, Claudio Telmon, and Daniele Sgandurra. Hierarchical, model-based risk management of critical infrastructures. *Rel. Eng. & Sys. Safety*, 94(9):1403–1415, 2009.
 - [33] Jorge G. Silva and Rebecca Willett. Hypergraph-based anomaly detection of high-dimensional co-occurrences. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(3):563–569, 2009.
 - [34] Christopher R. Johnson, Mirko Montanari, and Roy H. Campbell. Automatic management of logging infrastructure. In *NCAE Workshop on Insider Threat*, St Louis, MO, USA, 2010.
 - [35] Massimiliano Albanese, Sushil Jajodia, Andrea Pugliese, and V. S. Subrahmanian. Scalable analysis of attack scenarios. In *ESORICS*, pages 416–433, 2011.
 - [36] Andrew P. Kosoresow and Steven A. Hofmeyr. Intrusion detection via system call traces. *IEEE Software*, 14(5):35–42, 1997.
 - [37] Anton Babenko, Leonardo Mariani, and Fabrizio Pastore. Ava: automated interpretation of dynamically detected anomalies. In *ISSTA*, 2009.
 - [38] Joel Branch, Alan Bivens, and Taek Kyeun Lee. Denial of service intrusion detection using time dependent deterministic finite automata. In *Graduate Research Conference*, 2002.
 - [39] Peng Ning, Yun Cui, Douglas S. Reeves, and Dingbang Xu. Techniques and tools for analyzing intrusion alerts. *ACM Trans. Inf. Syst. Secur.*, 7(2):274–318, 2004.
 - [40] Hanli Ren, Natalia Stakhanova, and Ali A. Ghorbani. An online adaptive approach to alert correlation. In *DIMVA*, 2010.
 - [41] Gideon Creech and Jiankun Hu. A semantic approach to host-based intrusion detection systems using contiguous and discontinuous system call patterns. *IEEE Trans. Computers*, 63(4):807–819, 2014.
 - [42] Tim Bass. Intrusion detection systems and multisensor data fusion. *Communications of the ACM*, 43(4):99–105, 2000.
 - [43] Frédéric Cuppens and Alexandre Mieke. Alert correlation in a cooperative intrusion detection framework. In *S&P*, 2002.
 - [44] Alfonso Valdes and Keith Skinner. Probabilistic alert correlation. In *RAID*, 2001.
 - [45] Jagrati Agrawal, Yanlei Diao, Daniel Gyllstrom, and Neil Immerman. Efficient pattern matching over event streams. In *SIGMOD*, 2008.
 - [46] Daniel Gyllstrom, Jagrati Agrawal, Yanlei Diao, and Neil Immerman. On supporting kleene closure over event streams. In *ICDE*, 2008.
 - [47] Alan Demers, Johannes Gehrke, Mingsheng Hong, Mirek Riedewald, and Walker White. Towards expressive publish/subscribe systems. In *EDBT*, 2006.
 - [48] Alan J. Demers, Johannes Gehrke, Biswanath Panda, Mirek Riedewald, Varun Sharma, and Walker M. White. Cayuga: A general purpose event monitoring system. In *CIDR*, 2007.
 - [49] T. Benzal, R. Braden, D. Kim, C. Neuman, A. Joseph, K. Sklower, R. Ostrenga, and S. Schwab. Experience with deter: a testbed for security research. In *Testbeds and Research Infrastructures for the Development of Networks and Communities, 2006. TRIDENTCOM 2006. 2nd International Conference on*, 2006.

- [50] C. Siaterlis, A.P. Garcia, and B. Genge. On the use of emulab testbeds for scientifically rigorous experiments. *IEEE Communications Surveys & Tutorials*, 15(2):929–942, 2013.
- [51] Stephen Schwab, Brett Wilson, Calvin Ko, and Alefiya Hussain. Seer: A security experimentation environment for deter. In *Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test on DETER Community Workshop on Cyber Security Experimentation and Test 2007*, pages 2–2. USENIX Association, 2007.
- [52] Fabrice Bellard. QEMU, a fast and portable dynamic translator. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference, ATEC '05*, pages 41–41, Berkeley, CA, USA, 2005. USENIX Association.
- [53] Michael Goldweber Renzo Davoli. View-OS: Change your View on Virtualization. In *Proc. of Linux Kongress*, 2009.
- [54] OpenVZ. <http://openvz.org>.
- [55] elearningsecurity. <https://www.elearnsecurity.com>.
- [56] Pentestit. <https://lab.pentestit.ru>.
- [57] The hacker academy. <http://hackeracademy.com>.
- [58] Pentest laboratory. <http://pentestlab.org>.
- [59] Offensive security. <http://www.offensive-security.com>.
- [60] Jian Zhang, Phillip A Porras, and Johannes Ullrich. Highly predictive blacklisting. In *USENIX Security Symposium*, pages 107–122, 2008.
- [61] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. Beyond blacklists: learning to detect malicious web sites from suspicious urls. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1245–1254. ACM, 2009.
- [62] Laura Palagi and Marco Sciandrone. On the convergence of a modified version of svm light algorithm. *Optimization methods and Software*, 20(2-3):317–334, 2005.
- [63] Annabella Astorino, Antonio Fuduli, and Manlio Gaudioso. Dc models for spherical separation. *Journal of Global Optimization*, 48(4):657–669, 2010.
- [64] Annabella Astorino, Antonio Fuduli, and Manlio Gaudioso. Margin maximization in spherical separation. *Computational Optimization and Applications*, 53(2):301–322, 2012.
- [65] Annabella Astorino and Manlio Gaudioso. Ellipsoidal separation for classification problems. *Optimization Methods and Software*, 20(2-3):267–276, 2005.
- [66] Annabella Astorino and Manlio Gaudioso. A fixed-center spherical separation algorithm with kernel transformations for classification problems. *Computational Management Science*, 6(3):357–372, 2009.
- [67] Judah Ben Rosen. Pattern separation by convex programming. *Journal of Mathematical Analysis and Applications*, 10(1):123–134, 1965.
- [68] Olvi L Mangasarian. Linear and nonlinear separation of patterns by linear programming. *Operations research*, 13(3):444–452, 1965.
- [69] Kristin P Bennett and Olvi L Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optimization methods and software*, 1(1):23–34, 1992.
- [70] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.
- [71] Hoai An Le Thi, Hoai Minh Le, Tao Pham Dinh, and Ngai Van Huynh. Binary classification via spherical separator by dc programming and dca. *Journal of Global Optimization*, 56(4):1393–1407, 2013.

-
- [72] Pham Dinh Tao and Le Thi Hoai An. A dc optimization algorithm for solving the trust-region subproblem. *SIAM Journal on Optimization*, 8(2):476–505, 1998.
- [73] Pham Dinh Tao et al. The dc (difference of convex functions) programming and dca revisited with dc models of real world nonconvex optimization problems. *Annals of operations research*, 133(1-4):23–46, 2005.
- [74] <http://www.phishtank.com/>. Phishtank: .
- [75] <http://opendirectory.org/>. Open directory:.
- [76] Pedro Garcia-Teodoro, Jesús E. Díaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 28(1-2):18–28, 2009.
- [77] Sandeep Kumar and Eugene H. Spafford. A pattern matching model for misuse intrusion detection. In *National Computer Security Conference*, 1994.
- [78] Cristian Molinaro, Vincenzo Moscato, Antonio Picariello, Andrea Pugliese, Antonino Rullo, and V. S. Subrahmanian. Padua: Parallel architecture to detect unexplained activities. *ACM Trans. Internet Techn.*, 14(1):3, 2014.
- [79] Animesh Pacha and Jung-Min Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Comp. Networks*, 51(12):3448–3470, 2007.
- [80] Claudio Piciarelli, Christian Micheloni, and Gian Luca Foresti. Trajectory-based anomalous event detection. *IEEE Trans. Circuits Syst. Video Techn.*, 18(11):1544–1554, 2008.
- [81] Taeshik Shon and Jongsub Moon. A hybrid machine learning approach to network anomaly detection. *Inf. Sci.*, 177(18):3799–3821, 2007.
- [82] Massimiliano Albanese, Sushil Jajodia, Andrea Pugliese, and V. S. Subrahmanian. Scalable analysis of attack scenarios. In *ESORICS*, 2011.
- [83] Massimiliano Albanese, Andrea Pugliese, and V. S. Subrahmanian. Fast activity detection: Indexing for temporal stochastic automaton-based activity models. *IEEE Trans. Knowl. Data Eng.*, 25(2):360–373, 2013.
- [84] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Comp. Networks*, 31(23-24):2435–2463, 1999.
- [85] Mansour Sheikhan and Zahra Jadidi. Misuse detection using hybrid of association rule mining and connectionist modeling. *World Applied Sciences Journal*, 7:31–37, 2009.
- [86] Giovanni Vigna and Richard A. Kemmerer. Netstat: A network-based intrusion detection system. *Journal of Computer Security*, 7(1):37–71, 1999.
- [87] Reza Sadoddin and Ali Ghorbani. Alert correlation survey: framework and techniques. In *PST*, 2006.
- [88] Christopher Kruegel, Fredrik Valeur, and Giovanni Vigna. *Intrusion detection and correlation: challenges and solutions*, volume 14. Springer Science & Business Media, 2004.
- [89] Xinming Ou, Sudhakar Govindavajhala, and Andrew W Appel. Mulval: A logic-based network security analyzer. In *USENIX*, 2005.
- [90] Fredrik Valeur, Giovanni Vigna, Christopher Krügel, and Richard A. Kemmerer. A comprehensive approach to intrusion detection alert correlation. *IEEE Trans. Dependable Sec. Comput.*, 1(3):146–169, 2004.
- [91] Jian-shu Liu, Ren-hou Li, Yun-long Liu, and Zhen-yao Zhang. Multi-sensor data fusion based on correlation function and fuzzy integration function. *Systems Engineering and Electronics*, 28(7):1006–1009, 2006.

References

- [92] Martin Roesch et al. Snort: Lightweight intrusion detection for networks. In *LISA*, volume 99, pages 229–238, 1999.
- [93] Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba, and Kumar Das. The 1999 DARPA off-line intrusion detection evaluation. *Comp. Networks*, 34(4):579–595, 2000.
- [94] Kleber Vieira, Alexandre Schulter, Carlos Westphall, and Carla Westphall. Intrusion detection for grid and cloud computing. *It Professional*, (4):38–43, 2009.
- [95] Cristina Abad, Jed Taylor, Cigdem Sengul, William Yurcik, Yuanyuan Zhou, and Ken Rowe. Log correlation for intrusion detection: A proof of concept. In *CSAC*, pages 255–264, 2003.
- [96] Ashutosh Singh and Horacio Gonzalez Velez. Hierarchical multi-log cloud-based search engine. In *CISIS*, pages 211–219, 2014.
- [97] Guofei Gu, Phillip A Porras, Vinod Yegneswaran, Martin W Fong, and Wenke Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *Usenix Security*, volume 7, pages 1–16, 2007.
- [98] Antonella Guzzo, Andrea Pugliese, Antonino Rullo, and Domenico Saccà. Intrusion detection with hypergraph-based attack models. In *Graph Structures for Knowledge Representation and Reasoning*, 2013.
- [99] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.
- [100] Simon Tatham. PuTTY SSH and telnet client, 2016. <http://www.putty.org>. Version 0.65 for Windows.
- [101] ACM/IEEE-CS Joint Task Force on Computing Curricula. Computer science curricula 2013. Technical report, ACM Press and IEEE Computer Society Press, December 2013.
- [102] Lori Pridmore, Patrick Lardieri, and Robert Hollister. National cyber range (ncr) automated test tools: Implications and application to network-centric support tools. In *AUTOTESTCON, 2010 IEEE*, pages 1–4. IEEE, 2010.
- [103] Guillaume Aubuchon, Kathy Cacciatore, Morgan Fainberg, and Chris Hoge. Expediting digital workflow with openstack. <https://www.openstack.org/assets/pdf-downloads/OpenStack-Workflow-White-Paper-Letter-Final.pdf>, 2015.
- [104] Omar Sefraoui, Mohammed Aissaoui, and Mohsine Eleuldj. Openstack: Toward an open-source solution for cloud computing. *International Journal of Computer Applications*, 55(3):38–42, October 2012.
- [105] Gregory King. Oracle VM 3: Building a demo environment using Oracle VM VirtualBox. <http://www.oracle.com/technetwork/server-storage/vm/ovm3-demo-vbox-1680215.pdf>, 2012.
- [106] Dejan Milojicic, Ignacio M. Llorente, and Ruben S. Montero. OpenNebula: A cloud management tool. *IEEE Internet Computing*, 15(2):11–14, 2011.
- [107] Robert M Axelrod. *The complexity of cooperation: Agent-based models of competition and collaboration*. Princeton University Press, 1997.
- [108] Jacques Ferber. *Multi-agent systems: an introduction to distributed artificial intelligence*, volume 1. Addison-Wesley Reading, 1999.
- [109] Richard M Fujimoto. *Parallel and distributed simulation systems*, volume 300. Wiley New York, 2000.
- [110] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.

-
- [111] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Intelligent agents III agent theories, architectures, and languages*, pages 21–35. Springer, 1997.
 - [112] Michael Wooldridge and Nicholas R Jennings. Intelligent agents: Theory and practice. *The knowledge engineering review*, 10(02):115–152, 1995.
 - [113] Franco Cicirelli, Angelo Furfaro, and Libero Nigro. Modelling and simulation of complex manufacturing systems using statechart-based actors. *Simulation Modelling Practice and Theory*, 19(2):685 – 703, 2011.
 - [114] Chris Poulin. What retailers need to learn from the target breach to protect against similar attacks. <https://securityintelligence.com/target-breach-protect-against-similar-attacks-retailers>, January 2014.
 - [115] Shane Schick. Security researcher: ebay vulnerabilities could have led to drive-by attacks. <https://securityintelligence.com/news/security-researcher-ebay-vulnerabilities-led-drive-attacks/>, April 2015.
 - [116] Saman Zonouz, Amir Houmansadr, Robin Berthier, Nikita Borisov, and William Sanders. Secloud: A cloud-based comprehensive and lightweight security solution for smartphones. *Computers & Security*, 37:215–227, 2013.
 - [117] Hacking square. <http://hackingsquare.net>.