Manchester Metropolitan University

Doctoral Thesis

A MACHINE-LEARNING-BASED SYSTEM FOR REAL-TIME ADVANCED PERSISTENT THREAT DETECTION AND PREDICTION

IBRAHIM GHAFIR

A thesis submitted in partial fulfilment of the requirements of the Manchester Metropolitan University for the degree of Doctor of Philosophy

School of Computing, Mathematics and Digital Technology
Faculty of Science & Engineering

Manchester, 2017

Declaration of Authorship

This work or any part thereof has not previously been presented in any form to the University or to any other body whether for the purposes of assessment, publication or for any other purpose (unless otherwise indicated). Save for any express acknowledgements, references and/or bibliographies cited in the work, I confirm that the intellectual content of the work is the result of my own efforts and of no other person.

The right of Ibrahim Ghafir to be identified as author of this work is asserted in accordance with ss.77 and 78 of the Copyright, Designs and Patents Act 1988.

Ibrahim Ghafir

Acknowledgements

I wish to express my greatest thanks and sincere gratitude to my director of studies, Dr. Mohammad Hammoudeh, for his insightful directions, thoughtful discussions and heartily support. Without his guidance and constant encouragement, this PhD would not have been achievable. In addition, my life has been enriched personally and professionally by working with him.

I also want to thank the other members in my doctoral supervision team: Dr. Liangxiu Han, Dr. Robert Hegarty and Dr. Moi Hoon Yap, there supervision has been invaluable and their sustained support throughout this research has proven to be both endless and priceless. Furthermore, their guidance and feedback have helped me in all the time of research and writing of this thesis.

My thanks are due to my parents for their unlimited love and support. They taught me the value of knowledge and have stood by me at all times, providing constant support, encouragement and love. I admire them for all of their accomplishments in life, for their patience and independence, and for all of the knowledge and wisdom they have passed to me over the years.

Abstract

It is widely cited that cyber attacks have become more prevalent on a global scale. In light of this, the cybercrime industry has been established for various purposes such as political, economic and socio-cultural aims. Such attacks can be used as a harmful weapon and cyberspace is often cited as a battlefield. One of the most serious types of cyber attacks is the Advanced Persistent Threat (APT), which is a new and more complex version of multi-step attack. The main aim of the APT attack is espionage and data exfiltration, which has the potential to cause significant damage and substantial financial loss.

This research aims to develop a novel system to detect and predict APT attacks. A Machine-Learning-based APT detection system, called MLAPT, is proposed. MLAPT runs through three main phases: (1) Threat detection, in which eight methods are developed to detect different techniques used during the various APT steps. The implementation and validation of these methods with real traffic is a significant contribution to the current body of research; (2) Alert correlation, in which a correlation framework is designed to link the outputs of the detection methods, aiming to find alerts that could be related and belong to one APT scenario; and (3) Attack prediction, in which a machine-learning-based prediction module is proposed based on the correlation framework output, to be used by the network security team to determine the probability of the early alerts to develop a complete APT attack. The correlation framework and prediction module are two other major contributions in this work. MLAPT is experimentally evaluated and the presented system is able to predict APT in its early steps with a prediction accuracy of 84.8%.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The World Wide Web is everywhere with a presently estimated size of roughly 46.4 billion web pages, as indexed by Google engine [2]. The Internet has developed from a system providing a local connection service dealing with simple documents to what is currently a robust and adaptable platform used to spread information and deliver applications. Furthermore, organisations and corporations have increasingly depended on the Internet sharing significant information online. However, with this evolution of the Internet, the number of cyber attacks have rapidly increased. Cybercrime has become more prevalent because it can be launched from outside the target's network and, in many cases, it is difficult to identify the attacker [3]. As a result, attackers have become organised and a cybercrime industry has been established for various purposes such as political, economic and socio-cultural aims [4]. Moreover, cyber attacks can be used to spy on governments; thus, they can be used as a strong weapon and the cyberspace is often cited as a battlefield [5].

1.1 Motivation

The annual cost of cyber attacks was $3 trillion in 2015. Moreover, it is expected to increase to more than $6 trillion per annum by 2021 [6]. This expensive cost has brought much interest in the research and investment towards developing new methods and techniques for cyber attacks defence. For this reason, intrusion detection and prevention systems (IDPSs) [7], explored later in Section 2.2 on page 16, have been suggested to be used to protect computer networks and mitigate this threat.

Although virus scanners, firewalls and IDPSs have been able to detect and prevent many of cyber attacks, cyber-criminals in turn have developed more advanced methods and techniques to intrude into the target's network and exploit its resources. In addition, many of the defence approaches assume that if the organisations' defences are too hard to breach, the attacker will try to find an easier victim. Nonetheless, according to a technical report by Trend Micro [8], this assumption is no longer valid with the rise of targeted attacks, also known as Advanced Persistent Threats (APTs), in which both cyber-criminals and hackers are targeting selected organizations and persisting until they achieve their goals.

The APT attack, explained further in Section 2.1 on page 13, is targeting a specific organisation and it is performed through several steps. The main aim of APT is espionage and then data exfiltration. Therefore, APT is considered as a new and more complex version of multi-step attack [9]. These APTs form a problem for current detection methods as they use advanced techniques like social engineering [10] and make use of unknown vulnerabilities. Moreover, the economic damages due to a successful APT attack

can be very expensive. The expected cost of attacks is the major motivation for the investments in intrusion detection and prevention systems [11]. APTs are currently one of the most serious threats to the companies and governments [12].

Most of the research in the area of APT detection, as explored later in Chapter 3 on page 35, has focused on analysing already identified APTs [13] [14] [15] [16] [17] [18] [19], or detecting a particular APT that uses a specific piece of malware [20]. Some works have attempted to detect potential APT attacks. However, they face serious shortcomings in achieving real time detection [21], detecting all APT attack steps [21], balance between false positive and false negative rates [20], and correlating of events spanning over a long period of time [22] [23]. Therefore, it is assumed that there is a need to research new approaches and techniques regarding the APT attacks detection.

1.2 Research Questions

This research aims to develop a novel system to detect and predict APT attacks which can make a significant contribution to intrusion detection systems. The effectiveness of the proposed approach, which is its ability to detect APT attacks, should be high. This should be combined with high accuracy resulting into a low number of false warnings. This motivates the first research question:

Research question 1: How to develop an efficient system to detect the APT attack in a systematic way?

The detection system should support the real time detection because if an attack (or an attempted attack) is detected quickly, it can be much easier to trace back to the attacker, minimise the damage and prevent further break-ins. This motivates the second research question:

Research question 2: How effective is the developed system in terms of the processing speed?

Detecting the APT attack before completing its life cycle can prevent the attackers from achieving their goals and stealing the data. This motivates the third research question:

Research question 3: How good is the developed system in terms of the prediction of the APT attack in its early steps?

The main objective of this thesis is to answer the research questions.

1.3 Contributions

To answer the *first research question*, a machine-learning-based APT detection system, called MLAPT, has been developed. MLAPT runs through three main phases: threat detection, alert correlation and attack prediction. In the first phase, the sniffed data traffic is scanned to detect possible techniques used in the APT attack life cycle. To this end, eight detection modules have been developed and tested; each module implements a method to detect one technique used in one of the APT attack steps. These detection modules are as follows: disguised exe file detection (DeFD), malicious file hash detection (MFHD), malicious domain name detection (MDND), malicious IP address detection (MIPD), malicious SSL certificate detection (MSSLD),

domain flux detection (DFD), scan detection (SD), and Tor connection detection (TorCD). The output of this phase is alerts, also known as events, triggered by the individual modules. In the second phase, the alerts raised by the individual detection modules are fed to the newly designed (FCI) correlation framework. The aim of the correlation framework is to find alerts which could be related and belong to one APT attack scenario. The process in this phase undergoes three main steps: alerts filter (AF), to identify redundant or repeated alerts; clustering of alerts (AC), which most likely belong to the same APT attack scenario; and correlation indexing (CI), to evaluate the degree of correlation between alerts of each cluster. The main objective of using the correlation framework is to reduce the false positive rate of the MLAPT detection system. In the final phase, a machine-learning-based prediction module (PM) is designed and implemented based on a historical record of the monitored network. This module can be used by the network security team to determine the probability of the early alerts to develop a complete APT attack.

To answer the *second research question*, MLAPT has been developed in such a way that all the modules and algorithms in the first and second phases do not depend on storing the data and then analysing it, instead, those modules and algorithms are able to process the network traffic in real time and generate their outputs (events) accordingly.

To answer the *third research question*, a prediction module, which uses a historical record of the monitored network and applies machine learning techniques, has been developed. This module allows the network security team to predict the APT attack in its early steps and apply the required procedure to stop it before the attack completes its life cycle.

Although the detection modules methodologies exist in the literature, their implementation and validation in real traffic are significant contributions to the field. The correlation framework and prediction module are two other major contributions in this thesis.

The thesis results and contributions have been published in the following journals and conference proceedings:

- Ibrahim Ghafir, Mohammad Hammoudeh, Vaclav Prenosil, Liangxiu Han and Robert Hegarty. *MLAPT: A Correlation-Based System for Real-Time Advanced Persistent Threat Detection and Prediction.* Journal paper, under review.

- Ibrahim Ghafir, Vaclav Prenosil, Mohammad Hammoudeh and Umar Raza. *Malicious SSL Certificate Detection: A Step Towards Advanced Persistent Threat Defence.* In Proceedings of International Conference on Future Networks and Distributed Systems. Cambridge, United Kingdom: ACM Digital Library, 2017. ISBN 978-1-4503-4844-7. doi:10.475/123_4.

- Ibrahim Ghafir, Vaclav Prenosil, and Mohammad Hammoudeh. *Botnet Command and Control Traffic Detection Challenges: A Correlation-based Solution.* International Journal of Advances in Computer Networks and Its Security (IJCNS), New York, USA: theIRED, 2017, vol. 7, Issue 2, p. 27-31. ISSN 2250-3757.

- Ibrahim Ghafir, Vaclav Prenosil, Ahmad Alhejailan and Mohammad Hammoudeh. *Social Engineering Attack Strategies and Defence Approaches*. In Proceedings of International Conference on Future Internet of Things and Cloud. Vienna, Austria: IEEE Xplore Digital Library, 2016. p. 145-149, 5 pp. ISBN 978-1-5090-4052-0. doi:10.1109/FiCloud.2016.28.

- Ibrahim Ghafir, Vaclav Prenosil, Jakub Svoboda and Mohammad Hammoudeh. *A Survey on Network Security Monitoring Systems*. In Proceedings of International Conference on Future Internet of Things and Cloud. Vienna, Austria: IEEE Xplore Digital Library, 2016. p. 77-82, 6 pp. ISBN 978-1-5090-3946-3. doi:10.1109/W-FiCloud.2016.30.

- Ibrahim Ghafir and Vaclav Prenosil. *Malicious File Hash Detection and Drive-by Download Attacks*. In Suresh Chandra Satapathy, K. Srujan Raju, Jyotsna Kumar Mandal, Vikrant Bhateja. Proceedings of the Second International Conference on Computer and Communication Technologies, series Advances in Intelligent Systems and Computing. Hyderabad: Springer, 2016. p. 661-669, 9 pp. Vol. 379. ISBN 978-81-322-2516-4. doi:10.1007/978-81-322-2517-1_63.

- Ibrahim Ghafir and Vaclav Prenosil. *Proposed Approach for Targeted Attacks Detection*. In Sulaiman, H.A., Othman, M.A., Othman, M.F.I., Rahim, Y.A., Pee, N.C.. Advanced Computer and Communication Engineering Technology, Lecture Notes in Electrical Engineering. Phuket: Springer International Publishing, 2016. p. 73-80, 9 pp. Vol. 362. ISBN 978-3-319-24582-9. doi:10.1007/978-3-319-24584-3.

- Ibrahim Ghafir, Jakub Svoboda, Vaclav Prenosil. *A Survey on Botnet Command and Control Traffic Detection*. International Journal of

Advances in Computer Networks and Its Security (IJCNS), New York, USA: theIRED, 2015, vol. 5, Issue 2, p. 75-80. ISSN 2250-3757.

- Ibrahim Ghafir and Vaclav Prenosil. *Advanced Persistent Threat and Spear Phishing Emails.* In Proceedings of International Conference Distance Learning, Simulation and Communication. Brno, Czech Republic: University of Defence, 2015. p. 34-41, 8 pp. ISBN: 978-80-7231-992-3.

- Ibrahim Ghafir and Vaclav Prenosil. *Blacklist-based Malicious IP Traffic Detection.* In Proceedings of Global Conference on Communication Technologies (GCCT). Thuckalay, India: IEEE Xplore Digital Library, 2015. p. 229-233, 5 pp. ISBN 978-1-4799-8552-4. doi:10.1109/GCCT.2015.7342657.

- Jakub Svoboda, Ibrahim Ghafir, Vaclav Prenosil. *Network Monitoring Approaches: An Overview.* International Journal of Advances in Computer Networks and Its Security (IJCNS), New York, USA: theIRED, 2015, vol. 5, Issue 2, p. 88-93. ISSN 2250-3757.

- Ibrahim Ghafir and Vaclav Prenosil. *DNS traffic analysis for malicious domains detection.* In Proceedings of International Conference on Signal Processing and Integrated networks. Noida, India: IEEE Xplore Digital Library, 2015. p. 613 - 618, 6 pp. ISBN 978-1-4799-5990-7. doi:10.1109/SPIN.2015.7095337.

- Ibrahim Ghafir and Vaclav Prenosil. *Advanced Persistent Threat Attack Detection: An Overview.* International Journal of Advances in Computer Networks and Its Security (IJCNS), New York, USA: theIRED, 2014, Volume 4, Issue 4, p. 50-54. ISSN: 2250-3757.

- Ibrahim Ghafir and Vaclav Prenosil. *DNS query failure and algorithmically generated domain-flux detection.* In Proceedings of International Conference on Frontiers of Communications, Networks and Applications. Kuala Lumpur, Malaysia: IEEE Xplore Digital Library, 2014. p. 1-5, 5 pp. ISBN 978-1-78561-072-1. doi:10.1049/cp.2014.1410.

- Ibrahim Ghafir, Jakub Svoboda and Vaclav Prenosil. *Tor-based malware and Tor connection detection.* In Proceedings of International Conference on Frontiers of Communications, Networks and Applications. Kuala Lumpur, Malaysia: IEEE Xplore Digital Library, 2014. p. 1-6, 6 pp. ISBN 978-1-78561-072-1. doi:10.1049/cp.2014.1411.

- Ibrahim Ghafir, Martin Husak and Vaclav Prenosil. *A Survey on Intrusion Detection and Prevention Systems.* In Proceedings of student conference Zvůle 2014, IEEE/UREL. Zvůle, Czech Republic: Faculty of Electrical Engineering and Communication, Brno University of Technology, 2014. p. 10-14, 5 pp. ISBN 978-80-214-5005-9.

- Ibrahim Ghafir and Vaclav Prenosil. *POSTER: Network-based Advanced Persistent Threat Attack Detection.* In 8th International Conference on Autonomous Infrastructure, Management and Security (AIMS). 2014.

1.4 Restrictions

The detection modules, in the first phase of MLAPT, are able to detect the most common techniques possibly used in the APT attack life cycle. If a new technique is used in the future in one of the APT steps, MLAPT will

not be able to detect the corresponded step. However, MLAPT will still able to detect part of the APT attack scenario based on the other steps.

1.5 Thesis Structure

The remainder of the thesis is organized as follows:

Chapter 2 provides the reader with an overview of the background of this research. The advanced persistent threat (APT) attack is defined and the steps of this attack are explained. Besides, the intrusion detection and prevention systems (IDPSs) are explored. In addition, machine learning (ML) categories and some approaches are presented.

Chapter 3 explores the state of the art for the APT attack detection. This chapter introduces previous research findings on APTs and describes current attempts for the APT attacks detection.

Chapter 4 presents the proposed Machine-Learning-based APT detection system (MLAPT). The architecture of MLAPT is introduced first, along with a brief definition of the three main phases of MLAPT: threat detection, alert correlation and attack prediction. Following this, more description and details regarding each MLAPT phase are given.

Chapter 5 explains the implementation of MLAPT and mentions to all frameworks, tools and programming languages used for this implementation. As MLAPT consists of three main phases: Detection, Correlation, and Prediction; the implementation algorithms of each phase are presented separately.

Chapter 6 introduces the evaluation of MLAPT and presents the results achieved. Additionally, a comparison between the developed system MLAPT and other existing systems is provided.

Chapter 7 concludes the thesis and summarises the achievements. Furthermore, a future work is proposed.

Chapter 2

Background

This chapter provides the reader with an overview of the background of this research. The advanced persistent threat (APT) attack is defined and the steps of this attack are explained. Besides, the intrusion detection and prevention systems (IDPSs) are explored. In addition, machine learning (ML) categories and some approaches are presented.

2.1 An Overview of the APT Life Cycle

APT refers to Advanced Persistent Threat. APTs are a cybercrime category directed at business and political targets. APTs require a high degree of stealth over a long period of operation in order to be successful. The attacker usually aims for more than immediate financial gain, and infected systems continue to be compromised even after the target's network has been breached and initial goals reached [24]. Figure 2.1 depicts the steps of the APT attack [1].

FIGURE 2.1: Typical steps of the APT attack [1].

1. Intelligence gathering: This initial step aims to get information regarding the target, like its organizational structure, IT environment and even about people who are working for that target. For this purpose, the attacker can use public sources (LinkedIn, Facebook, Twitter, etc) and prepare a customized attack. Through this step the attacker tries to find and organize accomplices, build or acquire tools, and research target/infrastructure/employees.

2. Initial compromise (Point of entry): Performed by use of social engineering and spear phishing, over email, using software vulnerabilities [25]. Another popular infection method is planting malware on a website which the victim employees will be likely to visit.

3. Command and control (C&C) communication: After an organization's perimeter has been breached, continuous communication between the infected host and the C&C server should be preserved to instruct and guide the compromised machine. These communications are usually protected by Secure Sockets Layer (SSL) encryption, making it difficult to identify if the traffic directed to sites is malicious. Another technique can be used in this step is domain flux technique [26]; an exploited host may try to connect to a large number of domain names which are expected to be C&C servers. The goal of this technique is to make it difficult or even impossible to shut down all of these domain names.

4. Lateral movement: Once getting access to the target's network, the attacker laterally moves throughout the target's network searching for new hosts to infect. The attacker can use brute force attack [27], to obtain information such as a user password or personal identification number (PIN); an automated software is used to generate a large number of consecutive guesses as to the value of the desired data. Another technique is pass the hash attack [28], in which the attacker steals a hashed user credential and, without cracking it, reuses it to trick an authentication system into creating a new authenticated session on the same network.

5. Asset/Data discovery: This step aims to identify and isolate the noteworthy assets within the target's network for future data exfiltration. Port scanning can be used for this step [29].

6. Data exfiltration: Data of interest is transmitted into external servers which are controlled by the attacker. There are some techniques used for data exfiltration like built-in file transfer, via FTP or HTTP and via the Tor anonymity network [30].

2.2 Intrusion Detection and Prevention Systems (IDPS)

IDPSs are used to monitor different systems and use various approaches to detect intrusions to the monitored systems. These intrusions can result from external/internal attacks or misuse by the system users. Figure 2.2 shows the general architecture of the intrusion detection and prevention system.

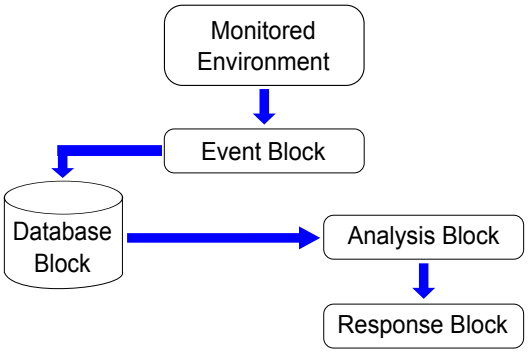

FIGURE 2.2: The general architecture of the IDPS.

This general architecture consists of four main blocks: event block, database block, analysis block and response block. First, the events of the monitored system are collected by the *event block*, then the events are sent to the *database block* to be stored. Following this, the stored events are processed by the the *analysis block* and an alert can be sent to the *response block*, which aims to respond to the malicious activity and stop it [31].

Different approaches are followed for IDPS, among them, four methodologies are widely used. These are the anomaly based, signature based, stateful protocol analysis based and hybrid based. The last methodology, the hybrid base, is a combination of the other ones and provides better performance capabilities; thus, it is mostly used by the present intrusion detection

systems. Despite having the same general model, the four methodologies are different in the way of processing the events collected from the monitored system. The main purpose of all methodologies is to detect if a breach into the monitored system has occurred [32].

2.2.1 Anomaly Based Methodology

This methodology is based on a comparison between the monitored activity and a baseline profile. The baseline profile is built within the training phase, in this phase the IDPS monitors the normal behaviour of the system and learns the environment to develop the baseline profile. The baseline profile can be built for systems, networks, users and so on. This profile can be fixed or dynamic. When the fixed profile is built, it does not change; While the dynamic profile changes when the monitored system changes. Building a dynamic profile requires the IDPS to keep updating the profile which adds more load to the system, moreover, this makes the system available to the evasion. To evade an IDPS uses a dynamic profile, the attacker can perform the attack over a long period of time; when building the profile, the IDPS combines the malicious activities with the profile and considers them as normal system changes [33].

In the detection phase, a threshold is set and any deviations for the monitored activities from the baseline profile are considered as malicious. Anomaly based methodologies are effective at detecting previously unknown attacks, known as zero-day attacks, without any updates to the system. Three main techniques are used by anomaly IDPSs: knowledge/data-mining, machine learning based and statistical anomaly detection [34].

Figure 2.3 shows the general architecture of an anomaly based IDPS. A detector is used to observe the monitored environment, the IDPS then compares the monitored activity against a baseline profile. If the monitored activity matches the baseline profile, that means, it is a normal activity and no action is required to be taken. If a deviation from the baseline is found, the IDPS checks the predefined threshold: if the deviation is within the threshold range, the IDPS updates the baseline profile; if the deviation is out of the threshold range, the monitored activity is reported as a malicious one and an alert is raised [31].

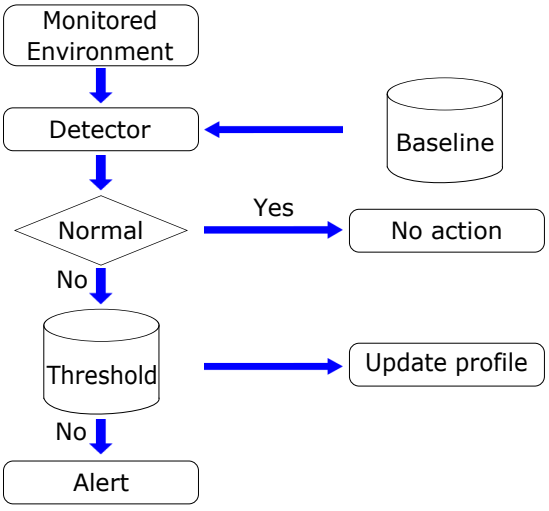

FIGURE 2.3: The general architecture of an anomaly based IDPS.

2.2.2 Signature Based Methodology

This methodology is based on a comparison between the monitored system signatures and a signatures database. This database contains a list of known attack signatures. If a match is found, the monitored system signature is reported as a malicious one and an alert is raised. The signature based methodology does not require a deep inspection for each single environment

activity or every packet in the network traffic, and it only looks for known signatures in the database. Therefore, signature based IDPS does not add extra load to the system [35]. As the signature based IDPS does not require a baseline profile, there is no need for the training phase used the anomaly based IDPS. Thus, the signature based IDPS is easy to be deployed and put directly in the detection phase. This methodology is very effective at detecting previously known attacks as their signatures are known and included in the signatures database. However, zero-day attacks cannot be detected as their signatures are not known yet [36].

Figure 2.4 shows the general architecture of a signature based IDPS. A detector is used by the IDPS to monitor the system signatures, the IDPS then compares the captured signatures with the signatures database: if a match is found, the monitored signature is reported as an attack and an alert is raised; if there is no match, no action is required to be taken [31].

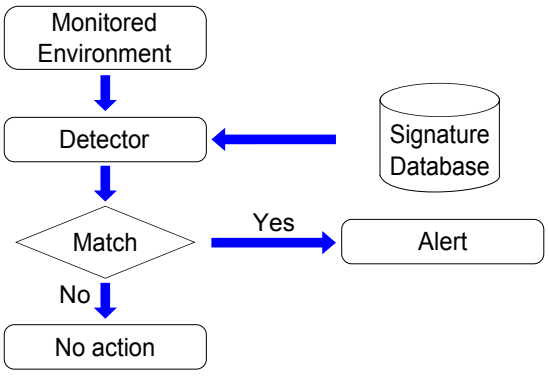

FIGURE 2.4: The general architecture of a signature based IDPS.

Since the signature based IDPS depends on the signatures database which contains known attacks and it is not possible to detect the new attacks before updating the database and adding the new signatures, this makes the IDPS open to the evasion. The attacker can amend the known attack and target a system which has an outdated database, in doing so, the attacker

can avoid the IDPS as the modified attack is considered as a new one. For this reason, the signature based IDPS requires to be updated continuously based on significant intelligence feeds which provide the IDPS with the new attacks [37].

2.2.3 Stateful Protocol Analysis Based Methodology

The stateful protocol analysis methodology is based on a comparison between the monitored behaviour and a protocol database. The protocol database contains profiles of how protocols should behave. The protocol profiles are created and built by the vendors. While the signature based IDPS only matches the monitored behaviour against a list of signatures, the stateful protocol analysis based IDPS has a comprehensive understanding of the protocol behaviour and performs a deep analysis of the interactions between the protocols and applications. This makes the stateful protocol analysis methodology add extra overhead to the system [38].

Figure 2.5 shows the general architecture of the stateful protocol analysis based IDPS. This architecture is similar to that of the signature based IDPS. A detector is used by the IDPS to monitor the system behaviour, the IDPS then compares the observed behaviour with the protocols database: if the observed behaviour does not meet the expected protocol behaviour, the monitored behaviour is reported as an attack and an alert is raised; if the observed behaviour meets the expected protocol behaviour, no action is required to be taken [31].

In spite of the fact that this methodology has a comprehensive understanding of how protocols should behave, the stateful protocol analysis based IDPS is still open to the evasion. The attacker can perform the attack and

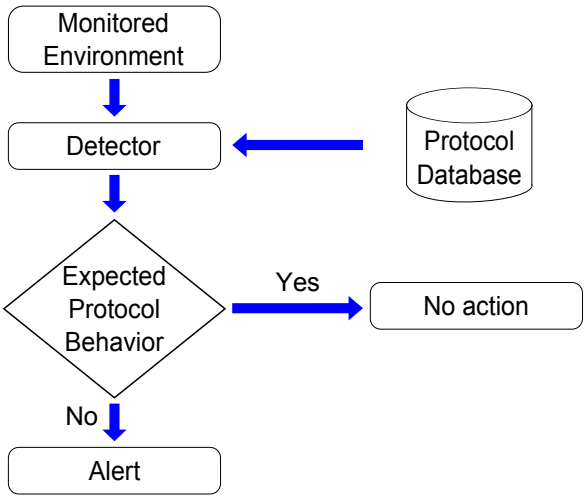

FIGURE 2.5: The general architecture of a stateful protocol analysis based IDPS.

stay within the expected behaviour of the protocols, in doing so, the attacker can bypass the IDPS. Over the last decade, stateful protocol analysis methodology has not been used alone for the IDPS. This methodology has been integrated and combined with the other IDPS methodologies. Most of the IDPSs depend on the anomaly, signature, and hybrid techniques. Thus, the stateful protocol analysis is not used any more as a standalone IDPS methodology [39].

2.2.4 Hybrid Based Methodology

To get the advantages of more than one methodology, a combination between two or more methodologies is used, this leads to a better technique, known as hybrid based methodology. The hybrid based IDPS is able to detect more malicious activities than each individual methodology alone [40].

Figure 2.6 shows the general architecture of a hybrid based IDPS. The monitored environment is processed and analysed by the three methodologies respectively, and the result is a better IDPS [31].

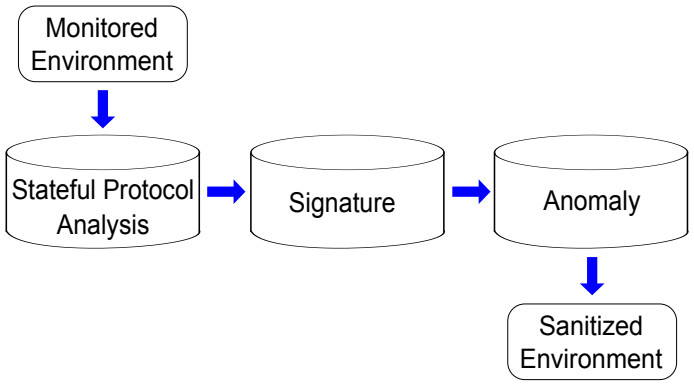

FIGURE 2.6: The general architecture of a hybrid based IDPS.

2.3 Machine Learning

Usually computers are programmed to perform their functions, while in machine learning systems, computers can learn without being obviously programmed. Machine learning gives computer programs the ability to develop themselves when the environment changes or new data comes. For this reason, machine learning is considered as a type of artificial intelligence (AI) [41]. Machine learning analyses the data with the aim of finding patterns and extracting program functions accordingly. The main task of machine learning is to find models. The model should represent the data and describe the main function of the system [42].

Based on the way how the model is built, machine learning is classified into four main categories: (1) supervised learning [43], when the dataset

includes the predictor features which are all labelled; (2) unsupervised learning [44], when the dataset includes the predictor features but does not include the labels; (3) semi-supervised learning [45], when the dataset includes the predictor features in which some of them have labels, while some are not labelled; and (4) reinforcement learning [46], which provides machines and software agents with the ability to automatically make the best decision within a given environment.

2.3.1 Machine Learning Approaches

Many approaches are used for machine learning [47]. This section presents only the ones used in this research, mainly for the purpose of data classification.

2.3.1.1 Decision Tree Learning

Decision tree is one of the supervised learning models, in which the final decision is made through a sequence of internal functions. A decision tree includes inner nodes and final leaves. The inner node performs a specific function to decide which branch to follow to the next inner node, and so on until a leaf node is reached and the final decision is made [48].

Table 2.1 shows an example of a training dataset and its decision tree is shown in Figure 2.7. The nodes in the decision tree represents the dataset attributes $at1$, $at2$, $at3$ and $at4$; the decision tree branches are formed based on the attributes values $a1$, $b1$, $c1$, $a2$, $b2$, $c2$, $a3$, $b3$, $a4$ and $b4$; and the terminal leaves in the decision tree indicates the dataset classes Yes and No [49].

TABLE 2.1: An example of a training dataset.

at_1	at_2	at_3	at_4	$Class$
a_1	a_2	a_3	a_4	Yes
a_1	a_2	a_3	b_4	Yes
a_1	a_2	a_3	a_4	Yes
a_1	a_2	b_3	b_4	No
a_1	c_2	a_3	a_4	Yes
a_1	c_2	a_3	b_4	No
b_1	b_2	b_3	b_4	No
c_1	b_2	b_3	b_4	No

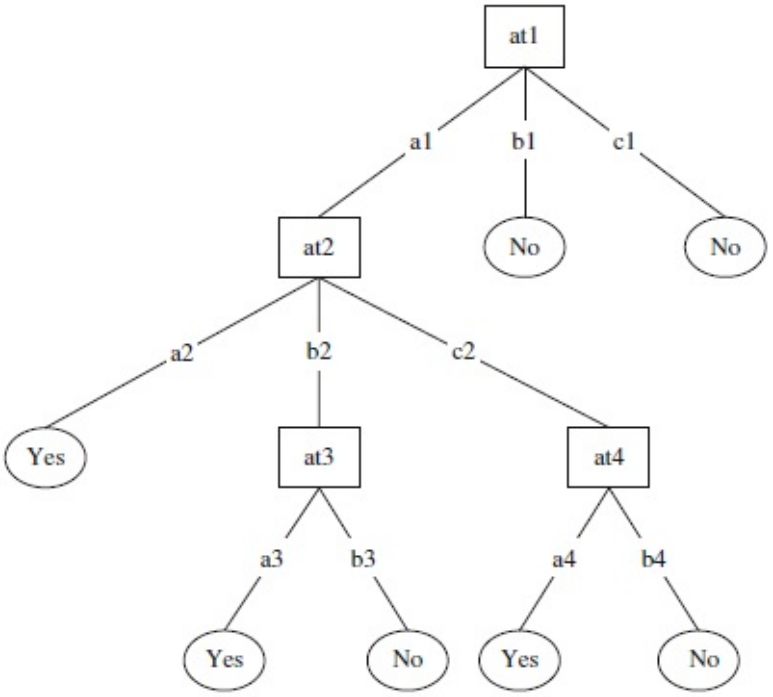

FIGURE 2.7: An example of a decision tree.

The decision tree process consists of two main phases: the first one is the growth phase. This phase includes a recursive dividing of the dataset until creating a decision tree in which each node is related to one class or more dividing to the node results in other nodes, which do not go over a specific threshold. The second phase is the pruning phase. The purpose of this phase is to prevent the over-fitting of the training dataset by creating

a sub-tree, in other word, this phase works on generalizing the decision tree which was generated in the growth phase. The process in the second phase is known as post-pruning, while it is called pre-pruning in the first phase [50].

The decision tree algorithm applies a discrete function on the input attributes to decide regarding partitioning the dataset. That function is chosen based on some partitioning measures such as Gini impurity, Information gain and Variance reduction.

Gini impurity is a measure of misclassification, which applies in a multi-class classifier context. Gini coefficient applies to binary classification and requires a classifier that can in some way rank examples according to the likelihood of being in a positive class [51]. For a given set of items with J classes, f_i is the fraction of items labelled with class i in the set, where $i \in \{1, 2, ..., J\}$, the *Gini impurity* is calculated as follows:

$$I_G(f) = \sum_{i=1}^{J} f_i(1 - f_i) = \sum_{i=1}^{J}(f_i - {f_i}^2) \tag{2.1}$$

$$= \sum_{i=1}^{J} f_i - \sum_{i=1}^{J} {f_i}^2 = 1 - \sum_{i=1}^{J} {f_i}^2 \tag{2.2}$$

Information gain is equal to the total entropy for an attribute if for each of the attribute values a unique classification can be made for the result attribute [52]. The *Information gain* $IG(T, a)$ for a given attribute a in a training set T, is based on the concept of *entropy* $H(T)$, given as:

$$H(T) = -\sum_{i=1}^{J} p_i \log_2 p_i \tag{2.3}$$

Where $p_1, p_2, ..., p_n$ are fractions represent the percentage of each class presented in the child node which results from a split in the tree.

$$IG(T, a) = H(T) - H(T|a) \tag{2.4}$$

$$= H(T) - \sum_{v \in vals(a)} \frac{|\{\mathbf{x} \in T | x_a = v\}|}{|T|} \cdot H(\{\mathbf{x} \in T | x_a = v\}) \tag{2.5}$$

Where each of the form $(x, y) = (x_1, x_2, x_3, ..., x_k, y)$, $x_a \in vals(a)$ is the value of the a attribute of example x, y is the corresponding class label.

Variance reduction is the search for alternative and more accurate estimators of a given quantity. Simple variance reduction methods often are remarkably effective and easy to implement [53]. The *variance reduction* of a node N is defined as the total reduction of the variance of the target variable x due to the split at this node:

$$\begin{aligned} I_V(N) = & \frac{1}{|S|^2} \sum_{i \in S} \sum_{j \in S} \frac{1}{2}(x_i - x_j)^2 - \\ & \left(\frac{1}{|S_t|^2} \sum_{i \in S_t} \sum_{j \in S_t} \frac{1}{2}(x_i - x_j)^2 + \frac{1}{|S_f|^2} \sum_{i \in S_f} \sum_{j \in S_f} \frac{1}{2}(x_i - x_j)^2 \right) \end{aligned} \tag{2.6}$$

Where S is the set of pre-split sample indices, S_t is the set of sample indices for which the split test is true, and S_f is the set of sample indices for which the split test is false.

The algorithm continues splitting the set into subsets until no split meets the splitting measure or a stopping feature is reached. The main task of the decision tree algorithm is to determine the best split, this split divides the

dataset into two or more subsets which includes one, two or more classes. The best subset is the one which contains only one class and it is known a pure subset, otherwise it is impure [54].

2.3.1.2 Support Vector Machines

In Support Vector Machine (SVM) algorithm, a hyperplane is created to represent the feature points in the space. This hyperplane splits the feature points into different categories. The last hyperplane state is specified by the support vectors which are the most representative points in the boundaries. The final hyperplane should represent the best distance between support vectors and the space categories. Thus the final hyperplane classify the points and divide the space into different regions [55].

For classification tasks, the kernel trick method [56] is used by SVM algorithms to allow the use of multidimensional hyperplanes. By using these hyperplanes, SVM learners can process multivariate data for classification purposes [57].

For a given data $(\mathbf{x_i}, y_i)$, for $i = 1, ...m$, where m is the total number of samples, $\mathbf{x_i} \in \chi \subseteq \mathbb{R}^d$ is the input of sensor readings and $y_i \in \{+1,\text{-}1\}$ represent each of the labels. This model learns from data to find a hyperplane function such as:

$$f(x) = \mathbf{w^T x} + b \tag{2.7}$$

where $f(x)$ represents the plane as a function of the training samples $\mathbf{x}$, b is the bias term and $\mathbf{w^T}$ is the *weight vector*. In order to learn the model

parameters b and $\mathbf{w}$, the following function should be optimised:

$$\frac{1}{2}||w||^2, \quad s.t. \;\; y_i(w^T\mathbf{x_i} + b) \geq 1 \tag{2.8}$$

which is solved by means of a Lagrangian multiplier, yielding the optimisation expression:

$$\min_{\alpha} \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \mathbf{x_i^T x_j} - \sum_{i=1}^{m} \alpha_i \tag{2.9}$$

When the last equation is solved using quadratic programming, returns a matrix of α coefficients for each $\mathbf{x_i}$. Substituting the values in α into one of the Lagrangian constraints the value for $\mathbf{w}$ and for the bias term b can be solved. Besides, α matrix includes zero and non-zero coefficients. The non-zero values indicates the support vectors, mentioned above, which are the feature points in the space which create the hyperplane [58].

SVM can split the non-linear points using the kernel method resulting in an output similar to that one of a linear model. This method gives SVM algorithm the feature of flexibility and the ability to handle more complex data. However, the kernel method adds more computational cost due to the increased complexity of the quadratic programming optimisation [59].

2.3.1.3 k-Nearest Neighbours

k-Nearest Neighbours (kNN) is a simple machine learning approach used for pattern recognition either for classification or regression. The k closest samples to the test object in the feature space are selected as an input, where k is a positive integer. The result is based on the use purpose: For classification purpose, the algorithm process returns a class. The decision is

based on a voting among the k closest samples to the object, when $k = 1$, the object is then classified into the same category of that single closest sample. For regression purpose, the algorithm process returns a property value. The result is based on the average value of the k closest samples values [60].

In kNN algorithms, a weight can be added to the participation of each neighbour. In doing so, the closer samples participates more to the final decision than the farther ones. Usually a weight of $1/d$ is assigned to each neighbour, where d is the distance between the object and the neighbour [61]. In spite of the fact that no training dataset is clearly needed for kNN algorithms, the k nearest neighbours can be considered as a training set. In kNN classification, the classes of the selected neighbours are known, and so on for the property values in kNN regression.

Figure 2.8 shows an example of kNN classification. This examples includes two classes: the first one is represented by the blue squares and the second class is represented by the red triangles. The kNN task is to classify the object represented by the green circle either to the first or second class. The value of k plays a significant role in the final decision. When $k = 3$, based on the voting among the three nearest neighbours (inside the solid line circle), the object is classified as a red triangle. However, when $k = 5$, based on the voting among the five nearest neighbours (inside the dashed line circle), the object is classified as a blue square [62].

Many distance measures are used in kNN algorithms, the most common one is the *Euclidean distance* [63]. For given points x and y, where $x = (x_1, x_2, ..., x_n)$ and $y = (y_1, y_2, ..., y_n)$, the Euclidean distance is calculated as follows:

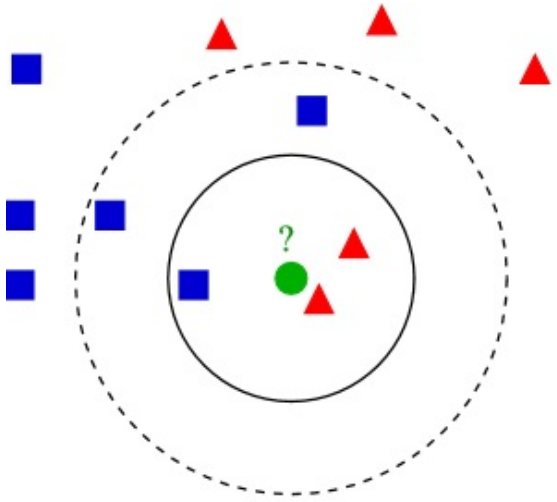

FIGURE 2.8: An example of kNN classification.

$$\mathrm{d}(\mathbf{x},\mathbf{y}) = \mathrm{d}(\mathbf{y},\mathbf{x}) = \sqrt{(y_1 - x_1)^2 + (y_2 - x_2)^2 + \cdots + (y_n - x_n)^2} \tag{2.10}$$

$$= \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2} \tag{2.11}$$

Some other distance measures used in kNN algorithms are: Hamming distance [64], Manhattan distance [65] and Minkowski distance [66].

2.3.1.4 Ensemble Learning

Ensemble learning is to combine more than one machine learning (ML) approach to learn the model. The final decision of the model is made based on the outputs of all ML approaches involved in the process. The need for ensemble learning comes from the fact that, in real-word practice, there is no optimal approach and each one has its own limitations. Thus, ensemble

learning makes use of the strengths and manage the weaknesses of each approach resulting in the best model which is able to make the best decision overall. The model built by ensemble learning has better accuracy than any other individual contributed approach [67].

Many methods are used to combine the outputs of ML approaches, the most commonly used ones are: the linear combiner, the product combiner and the voting combiner. The three methods are not complex and have presented good results in handling with different problems. However, a specific combiner can be used for a particular problem [68].

The linear combiner can be used to get the class probability estimates for both regression and classification models. However the models output should be real-valued numbers. For a given model $f_t(y|x)$, for a set of $t = \{1...T\}$, the estimate of the probability of class y given input x is:

$$\bar{f}_t(y|x) = \sum_{t=1}^{T} w_t f_t(y|x) \tag{2.12}$$

Where w_t is the weight, the simple probability estimate is when $w_t = 1/T$.

The product combiner is given as follows:

$$\bar{f}_t(y|x) = \frac{1}{Z} \prod_{t=1}^{T} f_t(y|x)^{w_t} \tag{2.13}$$

Where Z is a normalization factor to ensure f is a valid distribution.

As previously mentioned, the linear and product combiners are used only when the models outputs are real-valued numbers. Nonetheless, when the models outputs are class labels, the voting combiner can be used. Each

individual model votes for a specific class and the ensemble model select the class of the largest vote, which expressed as:

$$\sum_{t=1}^{T} w_t d_{t,j}(x) = max_{j=1,...,C} \sum_{t=1}^{T} w_t d_{t,j} \tag{2.14}$$

Where $t = \{1...T\}$, $j = \{1...C\}$; T is the number of classifiers and C is the number of classes; $d_{t,j} \in \{1, 0\}$ based on the classifier t if it selects j or not.

2.4 Threat Projection and Impact Assessment

Threat projection and impact assessment have similar, if not the same, interpretation in the fusion community. Two different fusion tasks can be considered, the first one is threat projection, which focuses on predicting future attack actions, i.e. where in the network will be attacked and what exploit(s) will be executed? The second task is impact assessment, which may be interpreted as estimating damages caused by current and future attack actions. Estimating damages or effects of attack actions on a network clearly depends on network contexts, some of which may not be automatically gathered, such as the criticality of a machine (due to data stored or its operational role) in the network. Damages to be caused by future actions also depend on threat projection [69].

2.5 Summary

This chapter presents an overview of advanced persistent threat (APT), intrusion detection and prevention systems (IDPSs) and machine learning

(ML). APT is a new and more complex version of known multi-step attack, usually directed at business and political targets.

IDPSs are used to detect intrusions to the monitored systems using different methodologies such as: (1) anomaly based methodology, which is based on a comparison between the monitored activity and a baseline profile; (2) signature based methodology, which is based on a comparison between the monitored system signatures and a signatures database; (3) stateful protocol analysis based methodology, which is based on a comparison between the monitored behaviour and a protocol database; and (4) hybrid based methodology, which is a combination between two or more methodologies.

Machine learning gives the computers the ability to learn without being explicitly programmed. Some of ML approaches, used in this research, are described such as: (1) decision tree learning, in which the final decision is made through a sequence of internal functions; (2) support vector machine, in which a hyperplane is created to represent the feature points in the space; (3) k-nearest neighbours, in which the k closest samples are voting for the final decision; and (4) ensemble learning, in which more than one machine learning (ML) approach are combined to learn the model.

Chapter 3

Advanced Persistent Threats: Detection and Defence

This chapter explores the state of the art for the APT detection. The related work is categorised into three main areas: previous research findings on APTs, analysing already identified APTs and detecting possible APT attacks.

3.1 Previous Research Findings on APT Attacks

In February 2013, Mandiant, an information security company in the USA, released their APT1 report [13] revealing one of China's cyber spying units. In this report, Mandiant presents significant information regarding the APT1 activities. More than 141 companies were compromised by APT1 and the size of the exfiltrated data was hundreds of terabytes. APT1 showed and

proved that it had the ability to attack tens of corporations simultaneously. The target corporations were working on 20 different industries. Once APT1 breached the victim's network and got the point of entry, the attackers maintained periodic visits to the target's network over several months or years. The categories of the stolen data were various including technology designs, company bylaws for corporations, manufacturing costs, meeting minutes, performance analysis results, operating agreements, business progress, non-disclosure agreements, memorandums of understanding and employment agreements. Moreover, APT1 were able to get access to all companies' emails and lists of contacts. For example, the stolen data from only one company "x" was 6.5 terabytes, this data had been stolen over 10 months. APT1 targeted companies which used English as their main language. 87% of the compromised organizations were located in English-speaking countries.

Regarding the APT1 infrastructure, it is found that the APT1 computer systems are deployed in 13 countries. APT1 controls more than 937 Command and Control (C&C) servers hosted on 849 different IP addresses. More than 97% of the detected connections between the attackers and the APT1 infrastructure were established from IP addresses registered in Shanghai using machines configured to use Chinese language.

In October 2012, an investigation, known as (Red October) [14], was performed by Kaspersky Lab's Global Research and Analysis Team. This research had been conducted after the computer networks of many international diplomatic service agencies had been compromised. As a result of this investigation, a network of computer systems used for spying was detected and analysed. Collecting information from the target companies was the major aim of the intruders. The attackers were targeting various diplomatic and governmental corporations around the world. In each case the attacks were

constructed of multiple stages, with the information gathered in each stage informing the subsequent stages of the arrack. For instance, the gathered credentials had been used frequently over a periodic visits to the victim's network.

Regarding the attackers' infrastructure, at least 60 domain names were established and many C&C servers, used as proxies to hide the main C&C server, were hosted in several countries (most of them in Germany and Russia). This infrastructure had been used to control the compromised networks. A multi-functional kit was developed by the attackers to collect the data from the infected machines. the attackers used more than one channel for C&C server communications, therefore, in case one channel was closed the attacker could still have access to the compromised computer. In addition to the personal computers, the attackers were able to breach and gather information from mobile devices, removable disk drives and enterprise network equipment. The research team mentioned that some vulnerabilities in Microsoft Word and Microsoft Excel had been exploited to breach the target's network and get the point of entry.

In January 2010, Google released a report on APTs called Operation Aurora [15]. The report stated that Operation Aurora had started in July 2009 and was successful to breach 34 companies' networks. In addition to Google, Dow Chemical, Northrop Grumman, Morgan Stanley, Symantec and Yahoo had been targeted by this attack. The first step of those attacks depended on social engineering techniques and used spear phishing, the attacks had started by sending emails to the employees of the target company, those emails contained links to malicious websites and were formatted in such a way to persuade the victim to click on the links. Malicious JavaScript codes

was inserted into those websites to exploit a vulnerability in the Internet Explorer browser. According to security vendor McAfee, the techniques used in the attacks were more advanced in comparison to those used in previous attacks.

The Google report revealed that Operation Aurora used a variety of malware to compromise the targets' networks. After installing the malware, C&C communications were maintained to control and instruct the infected machines. All C&C communications were encrypted to avoid the traditional intrusion detection systems, therefore, those connections were established via TCP port 443, the default port used by secure http. After getting the point of entry, the attackers started to used the current infected machines to compromise other machines in the same network; this technique, known as pivoting, enabled the attackers to avoid the detection rules set by firewalls. In doing so, the attackers were able to move in the target network, search for vulnerabilities, steal date and transmit it into external servers which were controlled by the attackers.

In March 2009, an investigation study on *GhostNet* [16] was released by the *SecDev* group in Canada. This investigation exposed a cyber spying network, called *GhostNet*, operated by attackers located in China. The attackers targeted the Tibetan organizations in many countries around the world. The *SecDev* research group was granted full access to the targeted Tibetan organizations' networks including the Private Office of the Dalai Lama. After analysing the targeted networks and available data, the research team revealed insecure web-based interfaces to four control servers. The attackers used those interfaces to get the point of entry and breach into the targeted networks. By tracking those four servers traffic, the *SecDev* group detected a large-scale network of exploited machines. More than 1295

machines in 103 countries were compromised, in which almost 30% of them are considered as high-value assets located in governmental ministries and embassies.

The compromised machine was instructed by the attacker to download a Trojan known as *gh0st RAT*. The pieces of *gh0st RA* were hosted on servers located on the island of Hainan in China. This Trojan gives the attacker a full and real-time control on the infected machines. Meaning, the attacker was able to use the compromised machine to upload and download files, moreover, the attacker had access to the microphones, speakers and cameras connected to the exploited computers. Social engineering techniques were used to compromise large number of machines in the target network. Spear phishing emails were sent to selected individuals, those emails contained malicious attachments, in which Trojan horse programmes and exploit codes were packed with the attached documents. The attacker then, exploiting the infected computer, started to move in the target network and search for the noteworthy servers which contain the data of interest. The research team believed that the majority of the high-value targets were infected by receiving malicious emails appeared as they were contained legitimate documents and sent from individuals they contacted them before.

3.2 Analysing Already Identified APT Attacks

An investigation framework to find the possible targets of the APT attack is proposed in [17]. The framework is based on analysing a known APT victim, extracting the victim attributes and then the developed engine searches for the possible APT targets. The authors state that the proposed system can reduce the false positive rate with regards to N-gram based approaches.

The work in [18] is based on a big data of APT attacks provided by Symantec over the year 2011. The authors use TRIAGE data analytic to classify the data into groups of APTs in which each group of attacks most likely performed by the same attackers. The next step is to process and analyse the characteristics and dynamics of each group separately, aiming to determine the modus operandi attributes of the attackers. To investigate the spread and complexity degree of each attacks group, the authors analyse the malicious attachments used in the attacks campaigns. The authors claim that most of the attacks used social engineering techniques to get the point of entry to the targets' networks. The attackers used simple malware, however, they exploited unknown vulnerabilities at that time.

An in-depth analysis of Duqu is presented in [20]. A European organisation was targeted by attackers using the Duqu malware to steal data. The authors propose the Duqu detector toolkit, which consists of six investigation tools developed to detect the Duqu malware involved in the APT attacks. The proposed tools can be classified into three main areas: *FindPNFnoINF*, *FindDuquSys* and *FindDuquTmp* tools, to find file existence anomalies; *FindDuquReg* and *CalcPNFEntropy* tools, to find registry entries and properties of files; and *GetProcMem* tool, to analyse code injection into running processes. The outputs of all those tools are then written into a specific log for a possible correlation of the findings. The authors admit that the empirical results shows a high rate of false negatives. Moreover, the detection output needs to be carefully investigated by the network security team. Besides, the detection tools are developed to detect the APT attacks particularly performed using the Duqu malware, this means, the attack cannot be detected when using a different piece of malware.

In [19], the authors use an undirected graph to build a map of the

APT activities. They mention that they can find related APTs based on the targets shared between different attacks. They can then determine the clusters which could led to the attackers or malware developers.

3.3 Approaches to Detecting APT Attacks

In this section, some APT detection systems are analysed in more details than others as they are used for a comparison with the developed system in Section 6.3 on page 140.

TerminAPTor, an APT detector, is described in [70]. This detector uses information flow tracking to find the links between the elementary attacks, which are triggered within the APT life cycle. TerminAPTor depends on an agent, which can be a standard intrusion detection system, to detect those elementary attacks. This agent also records the information flows and generates alerts in a chronological order. Each alert can be triggered by several events. Each event is considered as a flow of information and has four attributes: the event type, the timestamp, a list of references of input objects, and a list of references of output objects. All events are fed to TerminAPTor which assumes that the agent can detect all the elementary attacks. The main objective of TerminAPTor is to decide if two events are related to each other and find the APT scenarios. To consider two events are correlated, there must be an information flow from the first event output to the input of the second event. Therefore, each event should be determined to which elementary attack belongs and then a tag is created for each new detected attack. Each tag is attached to an object which is part of an information flow and all tags are propagated through the monitored system by information flows. The APT detector processes the information flows in a

chronological order and checks the flow input, if it is tagged, and the event, if it is part of an attack. TerminAPTor, based on specific rules, can then ignore the triggered event, propagate the tags, create a new APT scenario or add the event into a created APT scenario. The authors evaluated TerminAPTor by simulating only two APT scenarios and mentioned that the APT detector needs to be improved by filtering the false positives.

An APT detection system based on C&C domains detection is introduced in [71]. This work analyses the C&C communication and states a new feature that the access to C&C domains is independent while the access to legal domains is correlated. Based on this feature, a new concept, which is concurrent domains in the domain name service records (CODD), is proposed. CODDs are the domains queried by the same host during a specific time window. The suggested method depends on evaluating the correlation between the hosts and domains and applies classification algorithms. The detection of C&C domains relies on three features of C&C communications which are: (1) C&C communication is HTTP-based, (2) victims connect to C&C domains lowly and slowly, and (3) access to C&C domains is independent. The detection method is based on the fact that C&C domains have significantly less CODDs than benign ones. This is due to the loading of inlined components, such as ads and multimedia from other domains which are auto accessed when a web page is loaded, and the continuous access of legitimate domains. However, C&C domains do not have inlined components and independently accessed. Despite the fact that the detection system achieved significant results when validated on a public dataset, the authors mentioned that the detection can be easily evaded when the infected hosts connect to the C&C domains while users are surfing the Internet. This case increases the number of CODDs for C&C domains causing to be classified as benign ones. Moreover, missing the detection of C&C domains leads to failure in

APT detection since this system depends on detecting only one step of the APT life cycle.

An approach for APT detection based on spear phishing detection is explored in [72]. This approach depends on mathematical and computational analysis to filter spam emails. Tokens, which are considered as a group of words and characters such as (click here, free, Viagra, replica), should be defined for the detection algorithm to separate legitimate and spam emails. For each new email, the whole message is processed as a string, then Bayes' theorem is applied and the conditional probability and multiplicative rule are calculated. A threshold is given and the email is classified. This approach can be efficient when spear phishing attack is targeting hosts randomly. However, this is not the case in APT where the attacker targets selected host and crafts the malicious email carefully to appear as it is sent to that specific host. Meaning, the spear phishing email might not include any of the tokens which are necessary for the algorithm process. Additionally, depending on one step for APT detection leads the system to fail when missing that step.

A statistical APT detector, similar to TerminAPTor detector [70] previously mentioned on page 41, is developed in [73]. This system considers that APT undergoes five states which are delivery, exploit, installation, C&C and actions; and several activities are taken in each state. The generated events in each state are correlated in a statistical manner. The correlation, across multiple data sources, undergoes three levels: (1) Host-level combination, to identify malicious host-level activities; (2) Host-neighbourhood scoring, to detect potential movements of the attacker to infect other hosts; (3) Across-host combination, to find machines targeted by the attack. The events are extracted from four sources: mail, windows event, web, and domain name server (DNS) logs. Apart from the mail logs, all the data sources provide

the IP addresses of the event's source and destination which are used for the purpose of correlation. This system requires significant expert knowledge to set up and maintain.

An active-learning-based framework for malicious PDFs detection is suggested in [74]. These malicious PDFs can be used in the early steps of APT to get the point of entry. The system collects all PDFs transferred over the network, then all known benign and malicious files are filtered by the "known files module" which depends on white lists, reputation systems and antivirus signature repository. Following this, the remaining files "unknown files" are checked for their compatibility as viable PDF files. Based on the fact that most of the malicious files are incompatible, the unknown incompatible files are blocked giving the system a significant reduction of the computational cost. The unknown compatible PDFs are then processed by a detection model using SVM classifier with the radial basis function (RBF) kernel. This model is trained on a training dataset contains malicious and benign PDF files. As an output of the detection model, the unknown compatible PDF files can be classified as benign or malicious, or suspected as informative. The PDF files are considered as informative when the classifier is not confident regarding its classification (the files lie inside the SVM margin) or the files have the maximal distance from the SVM separating hyperplane. These informative files are sent to a security expert to be analysed manually, then they are labelled and added to the training dataset to retrain the detection model. In addition, the malicious labelled PDFs are added to the antivirus signature repository to maintain its updatability. Although this framework has not been implemented and validated yet, it highlights and addresses the updatability issue of the current detection models. However, this suggested approach detects only one step of the APT life cycle.

A classification model for APT detection is presented in [75]. This model is built based on machine learning algorithms. First, the legitimate traffic for normal users is analysed aiming to extract the CPU usage, memory usage, open ports and number of files in the system32 folder. A piece of malware, which is previously used for the APT attack, is injected into the network and the four features are extracted. The dataset of benign and malicious features are used to train the detection model using different machine learning algorithms. The trained models are validated on a test dataset and the best one, which has the highest accuracy, is chosen. The model can then be used for detection. It is mentioned that the random forest classifier has a detection accuracy of 99.8%. However, this model is limited to detect APT only if the attacker used the same piece of malware used when training the model.

An approach based on Data Leakage Prevention (DLP) is proposed in [76]. This approach focuses on detecting the last step of APT which is the data exfiltration. A DLP algorithm is used to process the data traffic to detect data leaks and generate "fingerprints" according to the features of the leak. These fingerprints might have various types of information such as data destination or file hash. However, they should have a standardized format such as Extensible Markup Language (XML). The fingerprints are then sent to a fingerprint database to acquire information regarding the data leak. The acquired information is provided to external cyber counterintelligence (CCI) sensors in order to track the location or path of the leaked data. The CCI sensors can be operated by the government or trusted partners such as operating system manufacturers, antivirus software companies and Internet service providers. These sensors have passive access to several nodes on the Internet and can search for a match with the leaked data fingerprints. If a match is found, this means either the leaked data location has been

found or the leaked data has passed through the corresponded sensor node. This information is passed back to the CCI analysis unit to be used along with other information provided by other CCI sensors to detect the actual attacker. This approach is limited to detect only one step of APT which is the data exfiltration. In addition, it cannot achieve the real time detection as the CCI analysis unit should wait for the information from the sensors. Moreover, it is not guaranteed that the CCI sensors can provide the required information regarding the leaked data fingerprints.

A working prototype, SPuNge, is presented in [21]. The proposed approach depends on the gathered data on the hosts' side and aims to detect possible APT attacks. The authors state that their system can successfully filter the large number of the threat events on the users' side into a manageable amount can be processed effectively. SPuNge uses a set of clustering methods to classify the infected machines into groups in which each group has the same malicious activity such as connecting to C&C servers, drive-by download attack or exploit kits connection. Then the machines' location and organisations' field of work (e.g., oil & gas or government) are correlated to detect malicious attack activities. SPuNge undergoes two main phases, in the first one, the detected malicious URLs are analysed. Those URLs can be connected by the hosts' computers over HTTP(S) with an Internet browser or by malware installed on the infected machines. The computers which show a similar activity are then determined. For example, all computers which request the same URLs during the same attack campaign are grouped together. And so on for the computers which connect to the same URLs because they are infected with the same malware. To achieve the first phase aims, SPuNge relies on a set of clustering methods to group the machines according to the requested malicious URLs. In the second phase, the connections of the machines clustered in the first phase are correlated to find

servers, companies or networks which are possibly involved in APT activities. For instance, machines can be correlated if they are located in the same location or working for the same organisation. To achieve the second phase aims, SPuNge uses an analysis framework to detect the malicious activities involved in APTs. Although the authors claim that SPuNge can work effectively on APTs detection, their system basically depends on detecting one activity of the APT attack, which is malicious URL connection, and does not consider the other activities of APT. Meaning, if the detection system misses the malicious URL connection, the whole APT scenario will not be detected. In addition, the proposed approach is based on a gathered data provided by an anti-virus vendor, in other words, the system needs to be fed with the data and cannot achieve real time detection.

A context-based framework for APT detection is explained in [77]. This framework is based on modelling APT as an attack pyramid in which the top of the pyramid represents the attack goal, and the lateral planes indicates the environments involved in the APT life cycle. Theses lateral planes are different from one organisation to another according to the environments where the events are recorded. This approach considers four main planes: (1) physical plane, records events that relate potential targets to physical machines or working locations; (2) user plane, records events that are related to privileged users who have access to sensitive data; (3) network plane, this plane includes all the events recorded by firewalls, intrusion detection and prevention systems, network flow sensors, routers and VPN access; and (4) application plane, records server and end host application logs, and application gateways such as http, SIP, RTP, DNS, SSH, ftp, telnet. The recorded events are collected and the profile selection is applied. The framework then links the events into contexts which are fed to the alert system. For each context, the alert system applies the detection rules which can be classified

into three major categories: (1) signature based rules, match the observed events and behaviour against known signatures database; (2) profiling based rules, compare the observed behaviour or profile with a behaviour baseline; and (3) policy based rules, which are the static rules set according to the organization policies. If an APT is detected by the alert system, an alert is sent to the security analyst to start investigating the alert. This detection framework is similar to the statistical detector [73], previously mentioned on page 43, and requires significant expert knowledge to set up and maintain.

With regards to the processing of multiple streams of events, IBM suggests a conceptual model for event processing in [22] and describes the basic requirements to design an efficient correlation system. The work explained in [23] is based on finite state machines and uses a query language for event processing. Both systems can process the events in real time, a key limitation shared by both approaches is that they cannot detect so called low & slow attacks, which take place over an extended time period.

3.4 Summary

Taking into consideration the great damage and substantial financial loss caused by the APT attacks, and based on the fact that the current intrusion detection methods still have weaknesses in the detection of APT, there is a need to research new approaches and techniques regarding the APT detection. Most of the research in the area of APT detection has focused on analysing already identified APTs or detecting a particular APT attack which uses a specific piece of malware. Some previous studies have attempted to detect potential APT attacks. However, they face serious shortcomings in achieving real time detection [21], detecting all APT attack steps [21],

balance between false positive and false negative rates [20], and correlating of events spanning over a long period of time [22] [23].

This thesis presents a Machine-Learning-based APT detection system (MLAPT) which runs through three main phases: threat detection, alert correlation and attack prediction. The suggested detection modules take into consideration most of the APT attack steps and the developed algorithms are able to process the network traffic in real time. Besides, the proposed correlation framework can reduce the false positive rate resulting from the detection system. To the best of our knowledge, the prediction module is a novel approach in terms of using machine learning to predict APT in its early steps.

Chapter 4

A Machine-Learning-Based System for Real-Time APT Detection and Prediction

In this chapter, the proposed Machine-Learning-based APT detection system (MLAPT), developed by the author, is presented. The architecture of MLAPT is introduced first, along with a brief definition of the three main phases of MLAPT: Detection; Correlation; and Prediction. Following this, more description and details regarding each MLAPT phase are given. This chapter defines and explains the MLAPT phases, while the implementation of MLAPT is presented in the following chapter.

4.1 Design Rationale

Since the APT attack is a multi-step attack, the detection of this attack should go through the detection of the techniques used within the APT life

cycle. Therefore, detection modules should be developed to detect the most common techniques used in the APT attack steps. However, detecting an APT technique itself does not mean detecting an APT attack.

Even though an individual module alert indicates a technique which can possibly be used in an APT attack, this technique can be used for other types of attacks or it can be even a benign one. For example, domain flux, port scanning and malicious C&C communications, used in the APT attack, can be also used for botnet attacks [78]. Moreover, Tor network connection, used for data exfiltration in the APT attack, can also be used legally to protect the confidentiality of a user traffic [79]. Thus, individually these detection modules are ineffective and their information should be fused to build a complete picture regarding an APT attack. For this reason, a correlation framework should be developed to link the outputs of the detection modules and reduce the false positive rate of the detection system.

Predicting the APT attack in its early steps would minimise the damage and prevent the attacker from achieving the goal of data exfiltration. With a historical record of the correlation framework output, machine learning algorithms can be used to train a prediction model. As the purpose of the prediction model is to classify the early alerts of the correlation framework, classification algorithms should be selected to train the model.

4.2 MLAPT Architecture

MLAPT runs through three main phases: threat detection, alert correlation and attack prediction. Figure 4.1 shows the architecture of MLAPT.

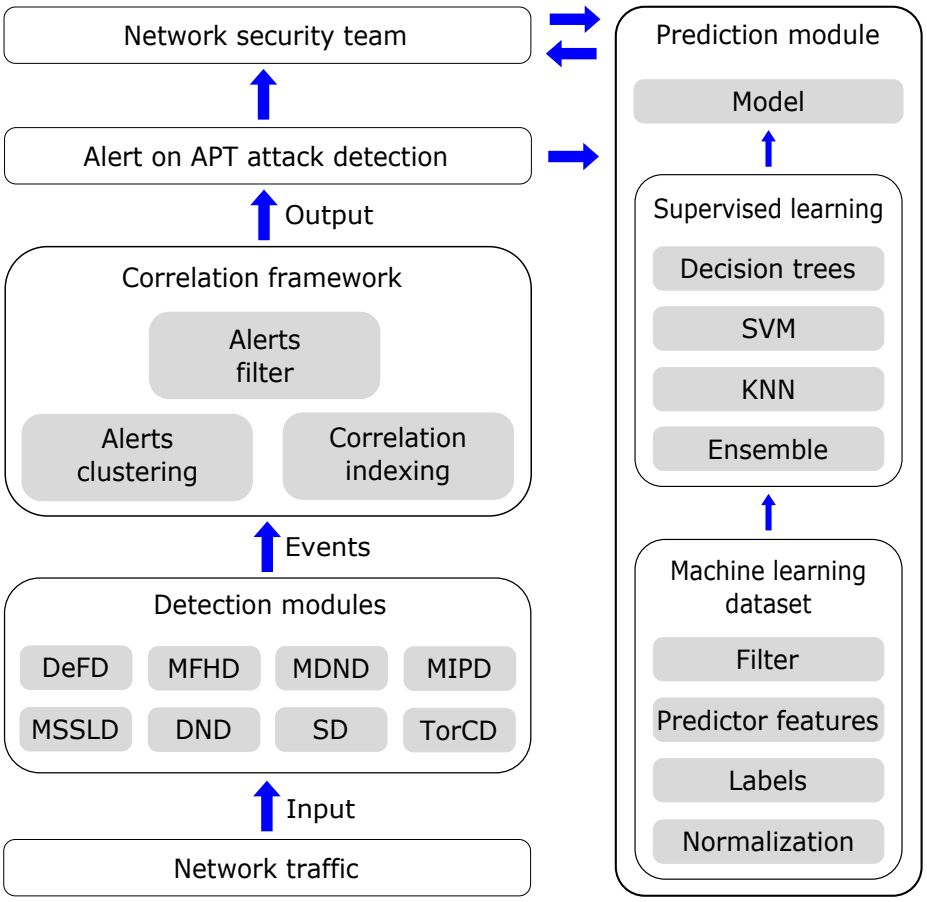

FIGURE 4.1: The Architecture of MLAPT.

Initially, the network traffic is scanned and processed to detect possible techniques used in the APT life cycle. To this end, eight detection modules have been developed; each module implements a method to detect one technique used in one of APT attack steps, and it is independent from the other modules. MLAPT implemented eight modules, presented later in Section 4.2.1 on page 54, to detect the most commonly used techniques in the APT life cycle. The output of this phase are alerts, also known as events, triggered by individual modules.

The alerts raised by individual detection modules are then fed to the correlation framework. The aim of the correlation framework is to find alerts

could be related and belong to one APT attack scenario. The process in this phase undergoes three main steps: alerts filter to identify redundant or repeated alerts; clustering of alerts which most likely belong to the same APT attack scenario; and correlation indexing to evaluate the degree of correlation between alerts of each cluster.

In the final phase, a machine-learning-based prediction module is used by the network security team to determine the probability of the early alerts to develop a complete APT attack. This allows the network security team to predict the APT attack in its early steps and apply the required procedure to stop it before completion and minimize the damage. The detection of APT is different from the prediction. The detection can be when two or more steps of APT are correlated. However, the prediction can be achieved after the first two steps of APT are linked.

Combining all the proposed modules in one system makes it relevant to APTs, as the detection modules are developed to detect most commonly used techniques in the APT life cycle. The outputs of those detection modules are then fed to the correlation framework to link them aiming to find the APT scenario. The prediction module is based on the correlation framework output to achieve the system functionality of the APT prediction.

4.2.1 MLAPT Detection Modules

The detection modules play a significant role in MLAPT to achieve its functionality. MLAPT implemented eight modules, these modules are chosen to detect the most commonly used techniques in APT, according to several reports like Mandiant APT1 report [80] and Trend Micro report [1], and they are able to form a clear picture on the APT attack scenario. The detection of

all modules is in real time, as MLAPT can process the sniffed network traffic immediately and does not need to store it. Some of the detection modules are blacklist-based, some of these blacklists are publicly published and some are related to private projects. All used blacklists are automatically updated within MLAPT, based on different intelligence feeds at once. Taking into consideration the APT steps, mentioned in Section 2.1 on page 13, Table 4.1 shows the MLAPT detection modules for each APT step.

TABLE 4.1: The MLAPT detection modules for each APT step.

APT step	**Detection modules**
Step 1 Intelligence gathering	This initial step includes a passive process which cannot be detected through network traffic monitoring, so there are no detection modules.
Step 2 Point of entry	Disguised exe file detection Malicious file hash detection Malicious domain name detection
Step 3 C&C communication	Malicious IP address detection Malicious SSL certificate detection Domain flux detection
Step 4 Lateral movement	This is internal traffic within the target's network. MLAPT monitors the inbound and outbound traffic, so there are no detection modules.
Step 5 Asset/Data discovery	Scanning detection
Step 6 Data exfiltration	Tor connection detection

4.2.1.1 Disguised exe File Detection (DeFD)

According to Mandiant APT1 report [80], spear phishing is the most commonly used technique to get the point of entry in APT. The spear phishing

emails contain either a malicious attachment or a hyperlink to a malicious file. The subject line and the text in the email body are usually relevant to the recipient. Executable files supposed to end in .exe are made to appear as simple document files (pdf, doc, ppt, excel) to convince the victim to click on it.

The DeFD module detects disguised exe files over the connections. In other words, it detects if the content of the file is exe while the extension is not exe. The network traffic is processed, all connections are analysed and all exe files identified when transferring over the connections are filtered. This filtering is based on the file content. Following this, the file name extension should be checked to decide about raising an alert on disguised exe file detection.

4.2.1.2 Malicious File Hash Detection (MFHD)

The MFHD module detects any malicious file downloaded by one of the network hosts. It is based on a blacklist of malicious file hashes [13]. The network traffic is processed, all connections are analysed and MD5, SHA1 and SHA256 hashes are calculated for each new file identified when transferring over a connection. The calculated hashes are then matched with the blacklist.

4.2.1.3 Malicious Domain Name Detection (MDND)

The MDND module is used to detect any connection to a malicious domain name. It is based on a blacklist of malicious domain names [81] [82] [83] [84] [85] [86]. DNS traffic is filtered, all DNS requests are analysed and the queries are matched with the blacklist.

4.2.1.4 Malicious IP Address Detection (MIPD)

The MIPD module detects any connection between an infected host and a C&C server. The detection is based on a blacklist of malicious IPs of C&C servers [87] [88] [89] [90]. MIPD processes the network traffic to search for a match in the source and destination IP addresses for each connection with the IP blacklist.

4.2.1.5 Malicious SSL Certificate Detection (MSSLD)

C&C communications are usually protected by Secure Sockets Layer (SSL) encryption, which makes it difficult to identify malicious traffic. MSSLD aims at detecting C&C communications based on a blacklist of malicious SSL certificates [91] [92]. This blacklist consists of two forms of SSL certificates, the *SHA1 fingerprints* and the *serial & subject*, which are associated with malware and malicious activities. The network traffic is processed and all secure connections are filtered. The SSL certificate of each secure connection is then matched with the SSL certificate blacklist.

4.2.1.6 Domain Flux Detection (DFD)

One common technique used for C&C communications is the domain flux technique, in which each infected machine separately uses a Domain Generation Algorithm (DGA) to generate a list of domain names [93]. By using the domain flux technique, the infected host attempts to query and connect to a large number of generated domain names, which are expected to link the host to the C&C servers. This technique makes it difficult for law enforcement to successfully shut down a large number of domains. To prevent

infected hosts from connecting to the C&C servers, law enforcement needs to pre-register all the domains which an infected host queries every day before the attacker registers them [94].

The DFD module detects algorithmically generated domain flux, where the infected host queries for the existence of a large number of domains, whilst the owner has to register only one. This leads to the failure of many of DNS queries. DFD utilizes DNS query failures to detect domain flux attacks. The network traffic is processed, particularly DNS traffic. All DNS query failures are analysed and a threshold for DNS query failures from the same IP address is imposed to detect domain flux attacks and identify infected hosts.

4.2.1.7 Scan Detection (SD)

Network scanning provides list of open ports, closed ports and filtered ports. Network related details such as IP address, MAC address, router, gateway filtering, firewall rules, etc. can be obtained through such scan [95].

The SD module detects port scanning attacks which aims to identify the noteworthy servers and services for future data exploitation. SD is based on tracking all failed connection attempts, and a threshold for those failed attempts is imposed over a specific time interval to detect scanning attacks and identify infected hosts.

4.2.1.8 Tor Connection Detection (TorCD)

Tor [96] [97] is an anonymous communication network used to secure the privacy of user traffic by encrypting all connections through the overlay network. Tor uses onion routing to direct client's traffic over a circuit of different relays to its destination, denying any single relay to know the complete path of the traffic [98]. Tor is often misused by criminals and hackers to remotely direct and instruct infected machines [99].

The TorCD module detects any connection to a Tor network. It is based on a list of Tor servers which is publicly published [100]. The network traffic is processed and the source and destination IP addresses for each connection are matched with Tor servers list.

4.2.2 FCI Correlation Framework

This phase of MLAPT takes the output of all detection modules (the generated alerts) as an input, and aims to find alerts could be correlated and belong to one APT attack scenario. FCI (Filter, Cluster, and Index) runs through three main steps: (1) Alerts filter, which filters redundant or repeated alerts; (2) Alerts clustering, which clusters alerts which potentially belong to the same APT attack scenario; and (3) Correlation indexing, which evaluates the correlations between alerts of each cluster.

In Section 4.2.1, eight attack detection modules are presented, each module detects one possible technique used in one of the APT steps. The output of each module is an alert which is generated when an attack is detected. Each alert has seven attributes *(alert_type, timestamp, src_ip, src_port, dest_ip, dest_port, infected_host)*. Table 4.2 summarizes the

steps of the APT attack that can be detected by MLAPT and the alerts which can be generated for each step.

TABLE 4.2: The APT attack detectable steps and alerts.

APT step	**Alerts**
(A) Step 2 Point of entry	(a1) disguised_exe_alert (a2) hash_alert (a3) domain_alert
(B) Step 3 C&C communication	(b1) ip_alert (b2) ssl_alert (b3) domain_flux_alert
(C) Step 5 Asset/Data discovery	(c1) scan_alert
(D) Step 6 Data exfiltration	(d1) tor_alert

All alerts generated by the detection modules are fed to the correlation framework. However, those alerts are not the only ones detected by the the modules. When an APT technique is detected, and before an alert is generated, the module checks whether the same alert has been generated during the previous day, if so, the alert is ignored. This alerts suppression reduces the computational cost of the FCI correlation framework. The FCI process steps will be explained in this section. As an output of the FCI correlation framework, two main alerts can be generated:

- *apt_full_scenario_alert:* This alert is generated when FCI detects a *full* APT attack scenario during a specific time window, called the correlation time. This is the period in which APT is expected to complete its life cycle. A full attack scenario is one in which all possible detectable steps of an APT are detected by FCI. In other words, FCI detects four correlated steps of an APT, i.e. four different alerts each

one is from a different step. Based on Table 4.2, and taking into consideration the APT life cycle, FCI is able to detect nine possible full scenarios of APT (APT-full). These possible full APT scenarios can be expressed as:

$$A = [a_1 \vee a_2 \vee a_3] \tag{4.1a}$$

$$B = [b_1 \vee b_2 \vee b_3] \tag{4.1b}$$

$$C = [c_1] \tag{4.1c}$$

$$D = [d_1] \tag{4.1d}$$

$$APT_{full} = A \wedge B \wedge C \wedge D \tag{4.1e}$$

- *apt_sub_scenario_alert:* This alert is generated when FCI detects two or three, rather than all, correlated steps of an APT attack during a specific time window. In this partial attack detection scenario, alerts from one or two steps were not generated. Thus, FCI can generate two types of this alert: *apt_sub_scenario_two_steps_alert*; and *apt_sub_scenario_three_steps_alert*. FCI is able to detect forty six possible APT sub-scenarios which can be expressed as:

$$\begin{aligned} APT_{sub} = {} & [A \wedge (B \vee C \vee D)] \vee [B \wedge (C \vee D)] \vee [C \wedge D] \vee \\ & [(A \vee B) \wedge (C \vee D)] \vee [A \wedge B \wedge C] \vee [A \wedge C \wedge D] \vee [B \wedge C \wedge D] \end{aligned} \tag{4.2}$$

The APT can still compromise the target network for months or years if it is not detected. However, one week might be enough for APT to complete one life cycle.

4.2.2.1 Alerts Filter (AF)

The first module of the FCI correlation framework filters redundant or repeated alerts. The AF module takes all alerts generated by the various detection modules as an input. For each new generated alert, the alerts filter checks if the alert has been generated during the correlation time window. If the new alert is the same type and has the same attributes of a recorded one, then the new alert is ignored. This filtering module reduces computational cost of the FCI correlation framework.

4.2.2.2 Alerts Clustering (AC)

This module clusters alerts which most likely belong to the same APT attack scenario. One cluster can represent a possible APT full or sub-scenario, i.e. it can contain one, two, three or four different alerts; each alert for a different APT step. The AC module takes the AF output, all alerts generated by the detection modules after repeated ones are filtered, as an input. All incoming alerts are stored by AC for a correlation time. For each new alert, the AC module checks all stored alerts for the clustering possibility. The clustering algorithm in this module is scenario-based, which utilizes three main rules:

- Alert step: Alerts for the same APT attack step cannot be in one cluster.
- Alert type: Alerts of the same type cannot be in one cluster.
- Alert time: Cluster's alerts should be all triggered within the correlation time, and alerts order should be corresponded with the APT life cycle. Meaning, if *t(d), t(c), t(b)* and *t(a)* are the times when the alerts from the APT steps *six, five, three* and *two*, respectively, have

been triggered, the clustering algorithm can classify those alerts into one cluster only if they meet the following two conditions:

```
t(d) > t(c) > t(b) > t(a)
t(d) - t(a) <= Correlation_time
```

The AC module has four processing engines, explained later in Section 5.2.2 on page 94, each engine processes all alerts which belong to one of the APT detectable steps. Based on the incoming alert step, a corresponded engine runs. As a result of AC process, the new incoming alert can be classified into an existing APT cluster, a new APT cluster can be created, or the new alert is ignored as it does not meet the rules and cannot be clustered at all. The output of AC is APT clusters. Each cluster contains a maximum of four alerts, which potentially belong to one APT full or sub-scenario. The produced cluster alerts are evaluated using the correlation index algorithm, presented in the following Section 4.2.2.3 on page 63, to decide whether they are correlated. The prediction module uses a historical record of the monitored network and takes the correlation dataset, built by FCI over six months or more, as an input. The correlation dataset contains the correlated clusters, both full and sub APT scenarios, and the correlation index $Corr_{id}$ for each cluster.

4.2.2.3 Correlation Indexing (CI)

The third processing module evaluates the correlations between alerts in each cluster to determine if they belong to a full or sub APT attack scenario. This module has two major functions. The first function is to evaluate the correlations between alerts when building the cluster. The goal of this correlation process is to filters clusters having uncorrelated alerts. The second function

calculates the correlation index of each cluster by the end of the correlation window. The latter function is essential to build a historical record of the monitored network to be used in the next module of the FCI correlation framework, namely the prediction module.

The correlation indexing (CI) algorithm makes use of the attributes of each alert in the cluster to calculate the cluster's correlation index $Corr_{id}$. To find the $Corr_{id}$ for each cluster, the CI algorithm calculates the correlation between each two alerts (steps) in the cluster. Therefore, three values are calculated within each cluster: $Corr_{ab}$, the correlation between the second step ($alert_1$) and the third step ($alert_2$) of APT; $Corr_{bc}$, the correlation between the third step ($alert_2$) and the fifth step ($alert_3$) of APT; and $Corr_{cd}$, the correlation between the fifth step ($alert_3$) and the sixth step ($alert_4$) of APT.

The clustering algorithm is based on *alert_type* and *timestamp* attributes of each alert. However, the correlation indexing algorithm is based on *infected_host* and *scanned_host* attributes. To calculate $Corr_{ab}$, $Corr_{bc}$ and $Corr_{cd}$, taking into consideration the APT attack life cycle and the attributes of each alert in the cluster, the CI algorithm utilizes the following rules:

$$Corr_{ab} = \begin{cases} 1, & \text{if } [alert_2, infected\_host_2] = [alert_1, infected\_host_1] \\ 0, & otherwise \end{cases}$$

$$Corr_{bc} = \begin{cases} 1, & \text{if } [alert_3, infected\_host_3] = [alert_2, infected\_host_2] \\ & \text{or } [alert_3, infected\_host_3] = [alert_1, infected\_host_1] \\ 0, & otherwise \end{cases}$$

$$Corr_{cd} = \begin{cases} 1, & \text{if } [alert_4, infected\_host_4] = [alert_3, scanned\_host] \\ & \text{or } [alert_4, infected\_host_4] = [alert_3, infected\_host_3] \\ & \text{or } [alert_4, infected\_host_4] = [alert_2, infected\_host_2] \\ & \text{or } [alert_4, infected\_host_4] = [alert_1, infected\_host_1] \\ 0, & otherwise \end{cases}$$

When $Corr_{ab}$ equals to 1, this means there is a correlation between the second step and the third step of APT and the corresponding alerts can be in one cluster. When $Corr_{ab}$ equals to 0, there is no correlation and the two alerts cannot be in one cluster. And so on for $Corr_{bc}$ and $Corr_{cd}$.

The CI algorithm calculates the cluster's correlation index $Corr_{id}$ using the following equation:

$$Corr_{id} = Corr_{ab} + Corr_{bc} + Corr_{cd} \tag{4.3}$$

Since $Corr_{ab}$, $Corr_{bc}$ and $Corr_{cd}$ values can be only 1 or 0, the cluster's correlation index $Corr_{id}$ is always positive and can take one of the following values:

- 0; there is no correlation between any of the cluster's alerts, and the cluster's alerts cannot belong to one APT attack scenario.
- 1; there is a correlation between two different steps of an APT attack, and the cluster's alerts belong to one APT sub-scenario "*apt_sub_scenario_two_steps*".
- 2; there is a correlation between three different steps of an APT attack, and the cluster's alerts belong to one APT sub-scenario "*apt_sub_scenario_three_steps*".
- 3; there is a correlation between four different steps (all detectable steps) of an APT attack, and the cluster's alerts belong to one APT full scenario "*apt_full_scenario*".

All the clusters and their correlation index values are recorded into a specific dataset, the *correlation_dataset*, to be used in the Prediction module.

4.2.3 Prediction Module (PM)

This module is used by the network security team to estimate the probability of an *apt_sub_scenario_two_steps_alert*, generated by the FCI correlation framework, to develop a complete APT attack. In practical terms, it predicts if FCI will generate an *apt_full_scenario_alert* in the future based on the attributes of the current *apt_sub_scenario_alert*. This prediction gives the network security team a sign to perform more forensics on the corresponding two suspicious connections and deny the attacker to complete the APT life cycle. The prediction module uses a historical record of the monitored network and applies machine learning techniques to achieve its functionality.

PM takes the correlation dataset, built by FCI over six months or more, as an input. The required period of time to build the correlation dataset depends on the number of correlated clusters generated by FCI. This number affects the number of samples used to train the prediction model, as it is explained in Section 5.3.1 on page 108. The correlation dataset contains the correlated clusters, both full and sub APT scenarios, and the correlation index $Corr_{id}$ for each cluster. The process in this module undergoes three main steps: (1) Preparing the dataset, to be available to be consumed by machine learning algorithms; (2) Training the prediction model, different machine learning algorithms are applied and the best model which has the highest accuracy is chosen; and (3) Using the model for prediction, the security team apply the model on FCI real time alerts. The output of this module is a prediction model used by the network security team for live traffic monitor and APT prediction. Section 5.3 provides more details about the components of PM, the way that probabilistic calculations are made, and how the network security team can use the output of this module for prediction.

4.3 Summary

This chapter presents the MLAPT architecture. MLAPT runs through three main phases: Detection; Correlation; and Prediction. In the first phase, eight detection modules have been developed, each detection module is to detect one technique possibly used in one of the APT steps. These detection modules are as follows: disguised exe file detection (DeFD), malicious file hash detection (MFHD), malicious domain name detection (MDND), malicious IP address detection (MIPD), malicious SSL certificate detection (MSSLD),

domain flux detection (DFD), scan detection (SD), and Tor connection detection (TorCD). In the second phase, the FCI correlation framework aims to find alerts could be related and belonged to one APT attack scenario. The process in this phase undergoes three main steps: alerts filter (AF), alerts clustering (AC), and correlation indexing (CI). In the last phase, the prediction module (PM) aims to build a prediction model used by the network security team for live traffic monitor and APT prediction.

Chapter 5

MLAPT Implementation

In this chapter, the implementation of MLAPT will be introduced and the used frameworks, tools and programming languages will be mentioned. As MLAPT consists of three main phases: threat detection, alert correlation and attack prediction; the implementation algorithms of each phase will be presented separately.

5.1 Implementation of the Detection Modules

There are several approaches to network security monitoring. However, there is no best approach and each approach performs best in a certain environment and fits different purposes. Wireshark [101] is an effective tool for manual analysis, predominantly of small capture files. Tcpdump is packet-oriented approach that works well in scenarios where filtering individual packets by L3/L4 attributes, like IP address, TCP flags and payload bytes, is sufficient. It does not work well for stream reassembly or L7 protocol analysis [102]. Snort [103] and Suricata [104] work well when the objective is to

match patterns in network data. Bro [105] [106] allows development of advanced detection methods and offers the best software/environment for the development of novel detection or processing techniques. It can be used for continuous monitoring of high-throughput networks. Moreover, the scripting environment is extensible in a memory-safe language specialized in network data processing.

All detection modules are implemented on top of *Bro*. Bro is a passive, open-source network traffic analyser. It is primarily a security monitor which inspects all traffic on a link in depth for signs of suspicious activity. The most immediate benefit gained from deploying Bro is an extensive set of log files which record a network's activity in high-level terms. These logs include not only a comprehensive record of every connection seen on the wire, but also application-layer transcripts such as, e.g., all HTTP sessions with their requested URIs, key headers, MIME types, and server responses; DNS requests with replies; and much more. Bro event engine reduces the incoming packet stream into a series of higher-level *events*, more than 300 events. These events reflect network activity in policy-neutral terms, i.e. they describe *what* has been seen, but not *why*, or whether it is significant. The MLAPT detection modules consume and handle some of Bro events; all connection information (such as *timestamp, src_ip, src_port, dest_ip, dest_port*) can be extracted from those events, in addition to more specific information related to each individual event.

The MFHD, MDND and MSSLD modules make use of *Bro Intelligence Framework* [107]; this framework enables the modules to consume data from different data sources and make it available for matching, as explained later in the implementation section of each module.

All blacklists of blacklist-based detection modules are automatically updated based on different intelligence feeds at once. The automatic update runs parallel with the detection modules process and there is no need to stop or restart MLAPT. This parallel-running feature allows a continuous live monitoring of the network traffic and supports real time detection. Furthermore, to keep the integrity of the blacklist files during updates, MLAPT detection modules keep using the current copy of the blacklists while the source copy is updated. After blacklist update is complete, the modules start using the updated copy. Those blacklists files are stored at the server that monitors the network traffic.

Based on the detection module, two automatic update mechanisms have been applied. Figure 5.1 shows the automatic update of the blacklists used by the MFHD, MDND and MSSLD modules. The user *crontab file* is configured to run *blacklist_ update.sh* each day at 3:00 am, this shell script connects through Internet to the data source servers and downloads updated blacklists of malicious file hashes, malicious domain names and malicious SSL certificate hashes into a new *blacklist.intel* text file. This text file is connected to the *Intelligence Framework*, which consumes it as explained later in the implementation section of each module.

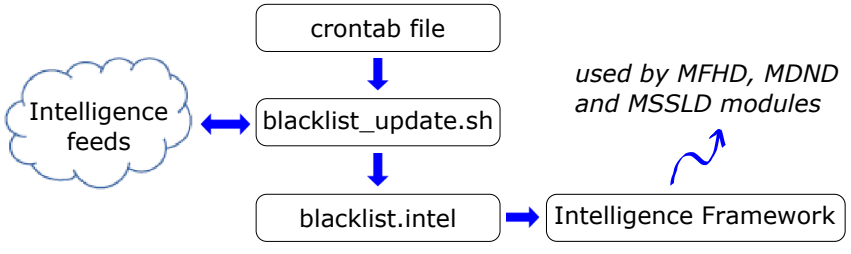

FIGURE 5.1: Automatic update of the blacklists used by the MFHD, MDND and MSSLD modules.

Figure 5.2 shows the automatic update of the blacklists used by the MIPD, MSSLD and TorCD modules. The user *crontab file* is configured to run *blacklist_update.sh* each day at 3:00 am, this shell script connects through Internet to the intelligence feeds and downloads updated blacklist of malicious IPs, malicious SSL certificates and Tor servers into *ip_blacklist.txt*, *ssl_blacklist.txt* and *Tor_servers_list.txt* files, respectively. The *Input Framework* [108], built in Bro, enables the modules to use those text files as an input to MLAPT. The Input Framework reads *ip_blacklist.txt*, *ssl_blacklist.txt* and *Tor_servers_list.txt* files into *t_ip_blacklist* table, *bad_ssl* group and *t_tor_server* table, respectively, which are used by the corresponding module as explained later in the implementation section of each module.

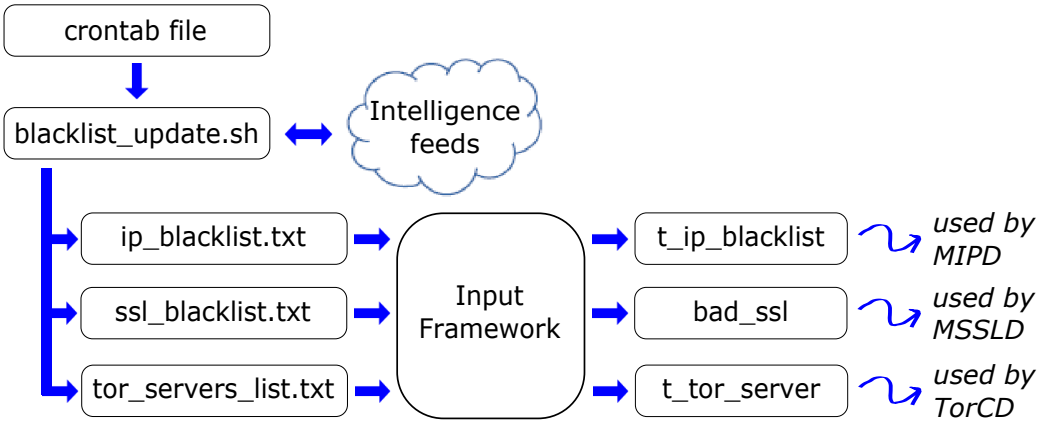

FIGURE 5.2: Automatic update of the blacklists used by the MIPD, MSSLD and TorCD modules.

As an output of each detection module, in case of an APT technique is detected, a corresponding event (alert) is generated. This event is to be used in the FCI correlation framework as explained later in Section 5.2 on page 93. Additionally, an alert email is sent to RT (Request tracker) [109] where the network security team can perform additional forensics and respond to the triggered alert. It is assumed that the network security team

responds to the generated alert within 24 hours, therefore, the detection modules suppress all the same alerts (the same alert is the one which has the same infected host and the same malicious item) into one alert per day, so no repeated alert emails bother the network security team. Moreover, this alerts suppression reduces the computational cost of the FCI correlation framework. To this end, after an alert is generated, the module adds the triggered alert into a specific corresponding table where it stays for one day to ensure that the module does not generate the same alert within the next 24 hours. When an APT technique is detected, and before an alert is generated, the module checks the corresponding table in order to conclude if the same alert has been generated during the previous day, if so, the alert is ignored. Along with generating a new alert, information regarding the alert and the malicious connection (*alert_type, timestamp, src_ip, src_port, dest_ip, dest_port, infected_host, malicious_item*) is written into a specific log (individual log for each APT technique detection) to keep a historical record of the monitored network.

In case of cryptographically embedded payloads for APTs paradigms, even the connections are encrypted, the detection modules (except DeFD and MFHD) are still effective as they depend on investigating the packets' headers and not the payload.

5.1.1 Implementation of the Disguised exe File Detection Module (DeFD)

Algorithm 1 shows the implementation pseudo-code of the DeFD module. The network traffic is processed; this module waits for

file_over_new_connection event to be generated by Bro. This event indicates that a file has been seen in the process of being transferred over a connection [110] [111]. Then *describe()* function is applied on that file; this function provides a text description regarding metadata of the file, so the file name can be extracted.

Algorithm 1 Implementation pseudo-code of DeFD

```
Get t_exe_file table
Get file_over_new_connection event
fname ← file name
if the connection is established by a host from the monitored
    network then
    if the file MIME type is in t_exe_file table then
        if file MIME type = fname extension then
            if the same disguised_exe_alert has been generated over
                the last day then
                goto End
            else
                Generate an event (disguised_exe_alert)
                Write disguised_exe_alert into disguised_exe_detection.log
                Send an alert email to RT
                Suppress the same disguised_exe_alert over the next day
            end if
        else
            goto End
        end if
    else
        goto End
    end if
else
    goto End
end if
End
```

This method is able to detect disguised exe files in both cases, uploaded and downloaded, but DeFD aims to detect only downloaded disguised exe files, as they are the ones used in the second step of APT. Thus, DeFD checks the current connection, in which a new file has seen being transferred,

if it is established by a host from the monitored network. This is done by checking the connection source IP address through *is_local_addr* function; this function returns true if an address corresponds to one of the defined local networks, false if not. For this reason, the *subnet* of the monitored network should be defined.

Following this, the *mime_type*, which can be extracted from *file_over_new_connection* event, is checked for its presence in *t_exe_file* table. MIME stands for (Multipurpose Internet Mail Extensions). It is a way of identifying Internet files according to their nature and format. For example, using the "Content-type" header value defined in a HTTP response, the browser can open the file with the proper extension/plugin [112]. *t_exe_file* table contains the MIME types of the files which DeFD aims to find and filter, i.e. MIME types of exe files. Therefore, if the transferred file is an exe file (based on its *mime_type*), DeFD checks whether the extension in the file name is exe; this file name extension is extracted from the output of the *describe()* function previously mentioned. If the file name extension is not exe, this means it is a disguised exe file. Before an alert is raised, DeFD checks if an alert regarding the same host and for the same disguised exe file has been generated during the previous day. This check is to ensure that DeFD does not generate the same alert about the same set (host, file) during one day, therefore, DeFD checks if the current set exists in the *t_suppress_disguised_exe_alert* table, this table contains all detected sets during the last day.

If the current set (host, file) had not been detected during the previous 24 hours, DeFD generates *disguised_exe_alert* event to be used in the FCI correlation framework. The malicious connection information is written into a specific log *disguised_exe_detection.log*, to keep a historical record of the

monitored network. An alert email regarding disguised exe file detection is sent to RT, where the network security team can perform additional forensics and respond to it. The current detected set (host, file) is added into the *t_suppress_disguised_exe_alert* table where it stays for one day to ensure that DeFD does not generate another alert about the same set during the same day. The written information into *disguised_exe_detection.log* is:

```
timestamp = c$start_time,
alert_type = "disguised_exe_alert"
connection  = c$id
infected_host = c$id$orig_h
malicious_file = fname
```

This module does not account for malware not being attached to executables, called fileless malware. This type of malware evades detection by reducing or eliminating the storage of any binaries on disk and instead hides its code in the registry of a compromised host. Malware authors have made detection challenging through techniques such as polymorphism, implanting watchdogs, revoking permissions, and more [113].

5.1.2 Implementation of the Malicious File Hash Detection Module (MFHD)

Algorithm 2 shows the implementation pseudo-code of the MFHD module. MFHD makes use of *Bro Intelligence Framework* mentioned in Section 5.1 on page 69. In this module, the intelligence framework is configured to monitor all file hashes which are identified when transferring over the network traffic. This framework is connected to *blacklist.intel* text file, which contains the files hashes blacklist.

Algorithm 2 Implementation pseudo-code of MFHD

```
1: Get malicious fills hashes blacklist (blacklist.intel)
2: Get file_new event
3: Calculate MD5, SHA1 and SHA256 hashes
4: Send MD5, SHA1 and SHA256 hashes to Bro Intelligence Framework
5: if MD5, SHA1 or SHA256 hashes are in blacklist.intel then
6: |   if the connection is oriented to a host from the monitored
7:         network then
8: |   |   if the same hash_alert has been generated over the last
9:             day then
10:            goto End
11: |   |   else
12:            Generate an event (hash_alert)
13:            Write hash_alert into blacklist_detection_hash.log
14:            Send an alert email to RT
15:            Suppress the same hash_alert over the next day
16: |   |   end if
17: |   else
18:        goto End
19: |   end if
20: else
21:    goto End
22: end if
23: End
```

The network traffic is processed; MFHD waits for *file_new* event to be generated by Bro. This event indicates that the analysis of a new file has started [114]. MFHD then calculates MD5, SHA1 and SHA256 hashes for the current file; these calculations are performed by three functions, which are *add_analyzer_md5*, *add_analyzer_sha1*, and *add_analyzer_sha256*. To ensure the real time process, the calculation of file hashes is limited to the files of size up to 500 KB, based on the fact that the average size of a malware sample is 338 KB [115]. All calculated hashes are sent to the intelligence framework where its presence should be checked within the intelligence dataset (*blacklist.intel* text file). When a piece of intelligence data, which in this case is the calculated file hash, is detected, the intelligence

framework generates *Intel::match* event. This event is generated when any *indicator_type* of intelligence data is detected [116], the intelligence dataset may contain many *indicator_types* for intelligence data (such as *ADDR*, *DOMAIN*, *CERT_HASH*) and not only *FILE_HASH indicator_type*. Thus, the *indicator_type* in this module is *FILE_HASH*.

This method is able to detect the malicious file in both cases, uploaded and downloaded, but MFHD aims to detect only downloaded malicious files. Therefore, MFHD checks if the connection is oriented to a host from the monitored network. This is done by checking the connection destination IP address through *is_local_addr* function mentioned in Section 5.1.1 on page 73. Before raising an alert, MFHD checks if an alert regarding the same host and for the same file hash has been generated during the previous 24 hours to avoid sending several alerts about the same set (host, hash) during one day. For this reason, MFHD checks if the current set exists in *t_suppress_hash_alert* table, this table contains all detected sets during the previous day.

MFHD then generates *hash_alert* event, writes the malicious connection information into a specific log *blacklist_detection_hash.log*, sends an alert email regarding the malicious file hash detection to RT and adds the current detected set (host, hash) into *t_suppress_hash_alert* table. The written information into *blacklist_detection_hash.log* is:

```
timestamp = s$conn$start_time
alert_type = "hash_alert"
connection  = s$conn$id
infected_host = s$conn$id$resp_h
malicious_hash = s$indicator
```

5.1.3 Implementation of the Malicious Domain Name Detection Module (MDND)

Algorithm 3 shows the implementation pseudo-code of the MDND module. MFHD makes use of *Bro Intelligence Framework* mentioned in Section 5.1 on page 69. In this module, the intelligence framework is configured to monitor all the domain names which are seen in DNS query requests traffic. This framework is connected to *blacklist.intel* text file, which contains the domain names blacklist.

Algorithm 3 Implementation pseudo-code of MDND

```
1: Get malicious domain names blacklist (blacklist.intel)
2: Filter DNS traffic
3: Extract DNS query requests
4: Extract the query (the requested domain name)
5: Send domain name to Bro Intelligence Framework
6: if domain name is in blacklist.intel then
7:  |  if the connection is established by a host from the monitored
8:  |      network then
9:  |  |  if the same domain_alert has been generated over the last
10: |  |      day then
11: |  |      goto End
12: |  |  else
13: |  |      Generate an event (domain_alert)
14: |  |      Write domain_alert into blacklist_detection_domain.log
15: |  |      Send an alert email to RT
16: |  |      Suppress the same domain_alert over the next day
17: |  |  end if
18: |  else
19: |      goto End
20: |  end if
21: else
22:     goto End
23: end if
24: End
```

The network traffic is processed and filtered into DNS traffic; MDND then extracts all DNS query requests. All domain names which are discovered in DNS traffic are sent to the intelligence framework where its presence should be checked within the intelligence dataset (*blacklist.intel* text file). When a piece of intelligence data, which in this instance is the malicious domain name, is detected, the intelligence framework generates *Intel::match* event mentioned in Section 5.1.2 on page 76. The *indicator_type* in this module is *DOMAIN*. Following this, MDND checks if the connection is established by a host from the monitored network, therefore, the connection source IP address is checked through the *is_local_addr* function. Before raising an alert, MDND checks if an alert regarding the same host and for the same malicious domain name has been generated during the last day, for this purpose, *t_suppress_domain_alert* table is checked.

MDND then generates *domain_alert* event, writes the malicious connection information into a specific log *blacklist_detection_domain.log*, sends an alert email regarding the malicious domain name detection to RT and adds the current detected set (host, domain) into *t_suppress_domain_alert* table. The written information into *blacklist_detection_domain.log* is:

```
timestamp = s$conn$start_time
alert_type = "domain_alert"
connection  = s$conn$id
infected_host = s$conn$id$orig_h
malicious_domain = s$indicator
```

5.1.4 Implementation of the Malicious IP Address Detection Module (MIPD)

Algorithm 4 shows the implementation pseudo-code of the MIPD module. The network traffic is processed; this module waits for *new_ connection* event to be generated by Bro. This event is generated for every new connection and it is raised with the first packet of a previously unknown connection [117].

Algorithm 4 Implementation pseudo-code of MIPD

```
1: Get malicious IP addresses blacklist (t_ip_blacklist table)
2: Get new_connection event
3: Check if the connection is to a malicious IP:
4: if the connection destination IP is in t_ip_blacklist then
5:     if the connection source IP belongs to the monitored network
6:         then
7:         if the same ip_alert has been generated over the last day
8:             then
9:             goto Check if the connection is from a malicious IP:
10:        else
11:            Generate an event (ip_alert)
12:            Write ip_alert into blacklist_detection_ip.log
13:            Send an alert email to RT
14:            Suppress the same ip_alert over the next day
15:        end if
16:    else
17:        goto Check if the connection is from a malicious IP:
18:    end if
19: else
20:    goto Check if the connection is from a malicious IP:
```

```
21: end if
22: Check if the connection is from a malicious IP:
23: if the connection source IP is in t_ip_blacklist then
24: |   if the connection destination IP belongs to the monitored
25:         network then
26: |   |   if the same ip_alert has been generated over the last day
27:             then
28:             goto End
29: |   |   else
30:             Generate an event (ip_alert)
31:             Write ip_alert into blacklist_detection_ip.log
32:             Send an alert email to RT
33:             Suppress the same ip_alert over the next day
34: |   |   end if
35: |   else
36:         goto End
37: |   end if
38: else
39:     goto End
40: end if
41: End
```

Through *new_connection* event, MIPD checks both connection sides' IP addresses to detect if the connection is to or from a malicious IP. If the connection destination IP exists in *t_ip_blacklist* table, this means, the connection is to a malicious IP. MIPD then checks the connection source IP through the *is_local_addr* function to determine if the connection is established by a host from the monitored network. When a malicious connection is detected and before raising an alert, MIPD checks the *t_suppress_ip_alert*

table to determine if the same *ip_ alert* has been generated within the last day. MIPD then generates *ip_ alert* event, writes the malicious connection information into a specific log *blacklist_ detection_ ip.log*, sends an alert email regarding the malicious IP detection to RT and adds the current detected set (host, ip) into *t_ suppress_ ip_ alert* table. The written information into *blacklist_ detection_ ip.log* is:

```
timestamp = c$start_time
alert_type = "ip_alert"
connection = c$id
infected_host = c$id$orig_h
malicious_ip = c$id$resp_h
```

When the connection is from a malicious IP, the same procedure, when the connection is to a malicious IP, is followed paying attention to the source and destination IP addresses as shown in Algorithm 4.

5.1.5 Implementation of the Malicious SSL Certificate Detection Module (MSSLD)

As the blacklist consists of two forms of malicious SSL certificates (*SHA1 fingerprints* and *serial & subject*), two methods are followed for malicious SSL certificate detection. The first one is intelligence-based MSSLD, shown in Algorithm 5, and the second method is event-based MSSLD, shown in Algorithm 6.

In the intelligence-based MSSLD, the *Bro Intelligence Framework* is used and configured to monitor all secure connections SSL certificates' hashes. This framework is connected to *blacklist.intel* file, which contains the SSL certificate blacklist. After extracting all secure connections traffic,

Algorithm 5 Implementation pseudo-code of intelligence-based MSSLD

```
1: Get malicious SSL certificates hashes blacklist (blacklist.intel)
2: Filter secure connections traffic
3: Extract SSL certificate hash
4: Send SSL certificate hash to Bro Intelligence Framework
5: if SSL certificate hash is in blacklist.intel then
6:  |  if the connection source IP belongs to the monitored network
7:  |       then
8:  |  |  if the same ssl_alert had not been generated over the last
9:  |  |       day then
10: |  |       Generate an event (ssl_alert)
11: |  |       Write ssl_alert into blacklist_detection_ssl.log
12: |  |       Send an alert email to RT
13: |  |       Suppress the same ssl_alert over the next day
14: |  |  end if
15: |  else if the connection destination IP belongs to the monitored
16: |       network then
17: |  |  if the same ssl_alert had not been generated over the last
18: |  |       day then
19: |  |       Generate an event (ssl_alert)
20: |  |       Write ssl_alert into blacklist_detection_ssl.log
21: |  |       Send an alert email to RT
22: |  |       Suppress the same ssl_alert over the next day
23: |  |  end if
24: |  else
25: |       goto End
26: |  end if
27: else
28:     goto End
29: end if
30: End
```

SSL certificates hashes are passed to the intelligence framework to be checked against the intelligence data set *blacklist.intel*. When a match with any *indicator_type* of the intelligence data is found, the intelligence framework generates an *Intel::match* event. Through this event, if the *indicator_type* is *CERT_HASH*, it means this connection has a malicious SSL certificate. Next, both connection sides, source and destination IP addresses, are checked

Algorithm 6 Implementation pseudo-code of event-based MSSLD

```
1: Get malicious SSL certificates [serials and subjects] (bad_ssl group)
2: Filter secure connections traffic
3: Get x509_certificate event
4: Extract SSL certificate [serial and subject]
5: if SSL certificate [serial and subject] is in bad_ssl then
6:  |  if the connection source IP belongs to the monitored network
7:  |      then
8:  |  |  if the same ssl_alert had not been generated over the last
9:  |  |      day then
10: |  |      Generate an event (ssl_alert)
11: |  |      Write ssl_alert into blacklist_detection_ssl.log
12: |  |      Send an alert email to RT
13: |  |      Suppress the same ssl_alert over the next day
14: |  |  end if
15: |  else if the connection destination IP belongs to the monitored
16: |      network then
17: |  |  if the same ssl_alert had not been generated over the last
18: |  |      day then
19: |  |      Generate an event (ssl_alert)
20: |  |      Write ssl_alert into blacklist_detection_ssl.log
21: |  |      Send an alert email to RT
22: |  |      Suppress the same ssl_alert over the next day
23: |  |  end if
24: |  else
25: |      goto End
26: |  end if
27: else
28:     goto End
29: end if
30: End
```

through the *is_local_addr* function to check if the connection is established to or from the monitored network. To avoid raising the same alert within the same day, the *t1_suppress_ssl_alert* table is checked to ensure that it does not contain the same detected [host IP address, SSL certificate hash] set.

MSSLD then generates *ssl_alert* event, writes the malicious connection information into a specific log *blacklist_detection_ssl.log*, sends an alert email regarding the malicious SSL certificate detection to RT and adds the current detected set [host IP address, SSL certificate hash] into *t1_suppress_ssl_alert* table. The written information into *blacklist_detection_ssl.log* is:

```
timestamp = s$conn$start_time
alert_type = "ssl_alert"
connection  = s$conn$id
infected_host = s$conn$id$orig_h
malicious_ssl = s\$indicator
```

In the event-based MSSLD, the network traffic is processed and filtered into secure connections traffic, and then *x509_certificate* event can be generated for encountered X509 certificates [118]. Through this event, the serial and subject of the X509 certificate are checked for the certificate presence in the *bad_ssl* group. This group contains many of serials and subjects of malicious X509 certificates. If a match is found, the module should determine if the connection is established to or from one of the monitored network hosts; accordingly, both the source and destination IP addresses are checked through *is_local_addr* function. Before an *ssl_alert* is raised, the *t2_suppress_ssl_alert* table is to be checked to ensure that the same alert was not raised previously during the same day.

In conforming CA certificates, the value of the subject key identifier must be the value placed in the key identifier field of the authority key identifier extension of certificates issued by the subject of this certificate. Applications are not required to verify that key identifiers match when performing certification path validation. Therefore, matching the serial and subject of

the X509 certificate, regardless the certificate version, should be effective to detect the malicious activities.

As in the previous intelligence-based method, MSSLD generates *ssl_alert* event, writes the malicious connection information into a specific log *blacklist_detection_ssl.log*, sends an alert email regarding the malicious SSL certificate detection to RT and adds the current detected set [host IP address, SSL certificate hash] into *t2_suppress_ssl_alert* table.

5.1.6 Implementation of the Domain Flux Detection Module (DFD)

Algorithm 7 shows the implementation pseudo-code of the DFD module. DNS traffic is extracted and processed; this module waits for *dns_message* event to be generated by Bro. This event is generated for any DNS message and provides information regarding the connection to DNS server [119].

Through *dns_massage* event, DFD checks for two conditions: (1) If this connection is established by a host from the monitored network using the *is_local_addr* function; (2) If the *dns_message* is due to DNS error of NXDOMAIN. A NXDOMAIN code means that the domain name does not exist, either not registered or invalid. This information can be extracted from *dns_message* event (*cdnsrcode_name=="NXDOMAIN"*). If the two conditions are met, the source IP address, which queries for unregistered domain names, is saved in the *t_dns_failure* table. This table counts DNS query failures of the same IP address. If the current IP address exists in the *t_dns_failure* table, the counter is increased by one (*++ t_dns_failure[cidorig_h]*).

Algorithm 7 Implementation pseudo-code of DFD

```
Get dns_failure_threshold
Extract DNS traffic
Get dns_message event
if the connection is established by a host from the monitored
    network then
    if dns_message event is due to DNS error of NXDOMAIN then
        if the host IP is not in t_dns_failure table then
            write host IP into t_dns_failure
            host IP counter ← 1
        else
            Increase host IP counter by 1
            if host IP counter > dns_failure_threshold then
                Delete host IP from t_dns_failure
                Reset host IP counter to zero
                if the same domain_flux_alert has been generated
                    over the last day then
                    goto End
                else
                    Generate an event (domain_flux_alert)
                    Write domain_flux_alert into domain_flux.log
                    Send an alert email to RT
                    Suppress the same domain_flux_alert over the
                    next day
                end if
            else
                goto End
            end if
        end if
    else
        goto End
    end if
else
    goto End
end if
End
```

When the number of DNS query failures exceeds the specified threshold, *dns_failure_threshold*, the current IP address is deleted from the *t_dns_failure* table to be removed from the counting, i.e. to reset the counter of this IP address to zero. The threshold is set to 50 DNS query failures per 5 minutes based on the fact that recent malware can generate 50,000 domain names every day [120]. Then, if the IP address of the potentially infected host does not exist in the *t_suppress_domain_flux_alert* table, to prevent more than one alert regarding the same IP address per day, DFD writes the following information into *domain_flux.log*:

```
timestamp = c$start_time
alert_type = "domain_flux_alert"
connection  = c$id
infected_host = c$id$orig_h
domain_name = c$dns$query
```

DFD also generates *domain_flux_alert* event, sends an alert email regarding the domain flux detection to RT and adds the current detected host IP address into *t_suppress_domain_flux_alert* table.

5.1.7 Implementation of the Scan Detection Module (SD)

The SD module makes use of *bro.scan* function, which is shipped by default with Bro [121]. bro.scan detects if an attacking host appears to be scanning a single victim host on several ports (port scanning). It also detects if a host appears to be scanning a number of destinations on a single port (address scanning). The detection is based on tracking all failed connection attempts over a specific time interval [122], and depends on four events: (1) *connection_attempt*, generated for an unsuccessful connection

attempt [123]; (2) *connection_ rejected*, generated for a rejected TCP connection [124]; (3) *connection_ reset*, generated when an endpoint aborted a TCP connection [125]; and (4) *connection_ pending*, generated for each still-open TCP connection when Bro terminates [126].

SD uses the same script as *bro.scan* and only minor modifications are made on it to generate *scan_ alert* event, write the malicious connection information into a specific log *scan_ detection.log* and send an alert email to RT.

5.1.8 Implementation of the Tor Connection Detection Module (TorCD)

Algorithm 8 shows the implementation pseudo-code of the TorCD module. The network traffic is processed; this module waits for *connection_ established* event to be generated by Bro. This event is generated when a SYN-ACK packet is seen in response to a SYN packet during a TCP handshake [127].

Algorithm 8 Implementation pseudo-code of TorCD

```
1: Get Tor servers list (t_tor_server table)
2: Get connection_established event
3: Check if the connection is to a Tor network:
4: if the connection destination IP is in t_tor_server then
5: |  if the connection source IP belongs to the monitored network
6:       then
7: |  |  if the same tor_alert has been generated over the last day
8:          then
9:             goto Check if the connection is from a Tor network:
```

```
        else
            Generate an event (tor_alert)
            Write tor_alert into tor_detection.log
            Send an alert email to RT
            Suppress the same tor_alert over the next day
        end if
    else
        goto Check if the connection is from a Tor network:
    end if
else
    goto Check if the connection is from a Tor network:
end if
Check if the connection is from a Tor network:
if the connection source IP is in t_tor_server then
    if the connection destination IP belongs to the monitored
        network then
        if the same tor_alert has been generated over the last day
            then
            goto End
        else
            Generate an event (tor_alert)
            Write tor_alert into tor_detection.log
            Send an alert email to RT
            Suppress the same tor_alert over the next day
        end if
    else
        goto End
    end if
```

38: **else**
39: **goto** *End*
40: **end if**
41: **End**

TorCD is able to detect the Tor connections in both cases, to or from a Tor network. However, this module aims to detect only connections from the monitored network to a Tor network. Therefore, through *connection_established* event, TorCD checks if the connection destination IP exists in *t_tor_server* table, this means, the connection is to a Tor network. The module then checks the connection source IP through the *is_local_addr* function to determine if the connection is established by a host from the monitored network. Following this, as in previous modules, *tor_alert* event is generated; an alert email is sent to RT; the following information is written into *tor_detection.log*:

```
timestamp = c$start_time
alert_type = "tor_alert"
connection  = c$id
infected_host = c$id$orig_h
tor_server = c$id$resp_h
```

This module considers any connection to a Tor entrance node is a suspicious one. Nonetheless, this alert is not considered as an attack till it is correlated with another step of APT. Furthermore, there are around 21 anonymous networks [128]. However, Tor network is the most commonly used one by malware. Detecting the connections to other anonymous networks can be achieved using this module by adding their nodes' IPs to *t_tor_server* table.

5.2 Implementation of the FCI Correlation Framework

The FCI framework is implemented in two versions. The first one is implemented on top of Bro to be used on live traffic for real time detection; it can be also used offline on PCAP (Packet Capture) files. The second version is implemented in Python [129] to be used offline on saved alerts' logs. Using FCI offline-version is useful when having a PCAP file for a network which is not monitored by Bro.

5.2.1 Implementation of the Alerts Filter (AF) Module

Algorithm 9 shows the implementation pseudo-code of the AF module. When

Algorithm 9 Implementation pseudo-code of AF

```
1: Get the correlation_time
2: Get t_detection_modules_alerts table
3: Get a new alert from one of the detection modules
4: if the new alert is in t_detection_modules_alerts then
5:     Ignore the new alert
6: else
7:     write the new alert into t_detection_modules_alerts
8:     Send the new alert to AC
9: end if
10: End
```

generating a new alert by one of the detection modules, the AF module checks *t_detection_modules_alerts* table to determine if the same alert has been generated within the last *correlation_time*. *t_detection_modules_alerts* table contains all alerts which have been generated by the detection modules and sent to AC within the last *correlation_time*. Thus, AF either (1) ignores the new alert, if it is a repeated one; or (1) sends the new alert

to AC, to be processed and clustered, and (2) writes the new alert into *t_detection_modules_alerts* table where it is saved for the next *correlation_time*.

5.2.2 Implementation of the Alerts Clustering (AC) Module

All produced APT clusters are recorded into a specific dataset, the *clustered_dataset*, to be consumed by the next module, namely the correlation indexing module. The clustering algorithm dataset "*clustered_dataset*" consists of clusters. Each cluster contains a maximum of four alerts and each alert represents one of the APT detectable steps:

1. $alert\_1 \in \{disguised\_exe\_alert, hash\_alert, domain\_alert\}$.
2. $alert\_2 \in \{ip\_alert, ssl\_alert, domain\_flux\_alert\}$.
3. $alert\_3 \in \{scan\_alert\}$.
4. $alert\_4 \in \{tor\_alert\}$.

Algorithm 10 shows the implementation pseudo-code of the AC module. Alert clustering can affect the performance of the correlation indexing and the prediction module as well. For this reason, the first function of CI, evaluating the correlations between the cluster's alerts, mentioned in Section 4.2.2.3 on page 63, is also implemented within the AC algorithm. Implementing the first function of CI within AC reduces the computational cost of the FCI correlation framework, since AC does not classify any new alert into a cluster unless it is correlated with the cluster alerts, as explained later in this section.

Algorithm 10 Implementation pseudo-code of AC and first function of CI

```
Get the time window TW (the correlation time)
Get the new alert from AF
if new alert == disguised_exe_alert Or new alert == hash_alert
    Or new alert == domain_alert then
    goto alert_1 processing engine:
else if new alert == ip_alert Or new alert == ssl_alert
    Or new alert == domain_flux_alert then
    goto alert_2 processing engine:
else if new alert == scan_alert then
    goto alert_3 processing engine:
else if new alert == tor_alert then
    goto alert_4 processing engine:
end if

alert_1 processing engine:
Start a new cluster
Write the new alert into alert_1

alert_2 processing engine:
for each cluster in the clustered_dataset do
    if alert_1 != None And alert_2 == None
        And alert_3 == None And alert_4 == None then
        if time > time_1 And time - Time_1 <= TW then
            if infected == infected_1 then
                Add the new alert into the current cluster
                Write the new alert into alert_2
```

```
27:                 Generate an event apt_sub_scenario_two_steps_alert
28:                 Write apt_sub_scenario_two_steps_alert into
29:                 apt_sub_scenario_two_steps_detection.log
30:                 Send apt_sub_scenario_two_steps_alert to RT
31:                 End for loop
32:         else
33:             goto Next cluster:
34:         end if
35:     else
36:         goto Next cluster:
37:     end if
38: else
39:     goto Next cluster:
40: end if
41: Next cluster:
42: if the current cluster is the last one in the clustered_dataset then
43:     Start a new cluster
44:     Write the new alert into alert_2
45: else
46:     do for statements for the next cluster in the clustered_dataset
47: end if
48: end for
49:
50: alert_3 processing engine:
51: for each cluster in the clustered_dataset do
52: if alert_1 != None And alert_2 != None
53:     And alert_3 == None And alert_4 == None then
54:     if time > time_2 And time - Time_1 <= TW then
```

```
55:         if infected == infected_2 then
56:             Add the new alert into the current cluster
57:             Write the new alert into alert_3
58:             Generate an event apt_sub_scenario_three_steps_alert
59:             Write apt_sub_scenario_three_steps_alert into
60:             apt_sub_scenario_three_steps_detection.log
61:             Send apt_sub_scenario_three_steps_alert to RT
62:             End for loop
63:         else
64:             goto Next cluster:
65:         end if
66:     else
67:         goto Next cluster:
68:     end if
69: else if alert_1 != None And alert_2 == None
70:     And alert_3 == None And alert_4 == None then
71:     if time > time_1 And time - Time_1 <= TW then
72:         if infected == infected_1 then
73:             Add the new alert into the current cluster
74:             Write the new alert into alert_3
75:             Generate an event apt_sub_scenario_two_steps_alert
76:             Write apt_sub_scenario_two_steps_alert into
77:             apt_sub_scenario_two_steps_detection.log
78:             Send apt_sub_scenario_two_steps_alert to RT
79:             End for loop
80:         else
81:             goto Next cluster:
82:         end if
```

```
83:         else
84:             goto Next cluster:
85:         end if
86:     else if alert_1 == None And alert_2 != None
87:         And alert_3 == None And alert_4 == None then
88:         if time > time_2 And time - Time_2 <= TW then
89:             if infected == infected_2 then
90:                 Add the new alert into the current cluster
91:                 Write the new alert into alert_3
92:                 Generate an event apt_sub_scenario_two_steps_alert
93:                 Write apt_sub_scenario_two_steps_alert into
94:                 apt_sub_scenario_two_steps_detection.log
95:                 Send apt_sub_scenario_two_steps_alert to RT
96:                 End for loop
97:             else
98:                 goto Next cluster:
99:             end if
100:        else
101:            goto Next cluster:
102:        end if
103:    else
104:        goto Next cluster:
105:    end if
106:    Next cluster:
107:    if the current cluster is the last one in the clustered_dataset then
108:        Start a new cluster
109:        Write the new alert into alert_3
110:    else
```

```
111:          do for statements for the next cluster in the clustered_dataset
112:    end if
113: end for
114:
115: alert_4 processing engine:
116: for each cluster in the clustered_dataset do
117:    if alert_1 != None And alert_2 != None
118:          And alert_3 != None And alert_4 == None then
119:       if time > time_3 And time - Time_1 <= TW then
120:          if infected == infected_3 Or infected == scanned then
121:                Add the new alert into the current cluster
122:                Write the new alert into alert_4
123:                Generate an event apt_full_scenario_alert
124:                Write apt_full_scenario_alert into
125:                apt_full_scenario_detection.log
126:                Send apt_full_scenario_alert to RT
127:                End for loop
128:          else
129:                goto Next cluster:
130:          end if
131:       else
132:             goto Next cluster:
133:       end if
134:    else if alert_1 != None And alert_2 != None
135:          And alert_3 == None And alert_4 == None then
136:       if time > time_2 And time - Time_1 <= TW then
137:          if infected == infected_2 then
138:                Add the new alert into the current cluster
```

```
139:                Write the new alert into alert_4
140:                Generate an event apt_sub_scenario_three_steps_alert
141:                Write apt_sub_scenario_three_steps_alert into
142:                apt_sub_scenario_three_steps_detection.log
143:                Send apt_sub_scenario_three_steps_alert to RT
144:                End for loop
145:            else
146:                goto Next cluster:
147:            end if
148:        else
149:            goto Next cluster:
150:        end if
151:    else if alert_1 != None And alert_2 == None
152:        And alert_3 != None And alert_4 == None then
153:        if time > time_3 And time - Time_1 <= TW then
154:            if infected == infected_3 Or infected == scanned then
155:                Add the new alert into the current cluster
156:                Write the new alert into alert_4
157:                Generate an event apt_sub_scenario_three_steps_alert
158:                Write apt_sub_scenario_three_steps_alert into
159:                apt_sub_scenario_three_steps_detection.log
160:                Send apt_sub_scenario_three_steps_alert to RT
161:                End for loop
162:            else
163:                goto Next cluster:
164:            end if
165:        else
166:            goto Next cluster:
```

```
167:        end if
168:    else if alert_1 == None And alert_2 != None
169:        And alert_3 != None And alert_4 == None then
170:        if time > time_3 And time - Time_2 <= TW then
171:            if infected == infected_3 Or infected == scanned then
172:                Add the new alert into the current cluster
173:                Write the new alert into alert_4
174:                Generate an event apt_sub_scenario_three_steps_alert
175:                Write apt_sub_scenario_three_steps_alert into
176:                apt_sub_scenario_three_steps_detection.log
177:                Send apt_sub_scenario_three_steps_alert to RT
178:                End for loop
179:            else
180:                goto Next cluster:
181:            end if
182:        else
183:            goto Next cluster:
184:        end if
185:    else if alert_1 != None And alert_2 == None
186:        And alert_3 == None And alert_4 == None then
187:        if time > time_1 And time - Time_1 <= TW then
188:            if infected == infected_1 then
189:                Add the new alert into the current cluster
190:                Write the new alert into alert_4
191:                Generate an event apt_sub_scenario_two_steps_alert
192:                Write apt_sub_scenario_two_steps_alert into
193:                apt_sub_scenario_two_steps_detection.log
194:                Send apt_sub_scenario_two_steps_alert to RT
```

```
195:                End for loop
196:            else
197:                goto Next cluster:
198:            end if
199:        else
200:            goto Next cluster:
201:        end if
202:    else if alert_1 == None And alert_2 != None
203:        And alert_3 == None And alert_4 == None then
204:        if time > time_2 And time - Time_2 <= TW then
205:            if infected == infected_2 then
206:                Add the new alert into the current cluster
207:                Write the new alert into alert_4
208:                Generate an event apt_sub_scenario_two_steps_alert
209:                Write apt_sub_scenario_two_steps_alert into
210:                apt_sub_scenario_two_steps_detection.log
211:                Send apt_sub_scenario_two_steps_alert to RT
212:                End for loop
213:            else
214:                goto Next cluster:
215:            end if
216:        else
217:            goto Next cluster:
218:        end if
219:    else if alert_1 == None And alert_2 == None
220:        And alert_3 != None And alert_4 == None then
221:        if time > time_3 And time - Time_3 <= TW then
222:            if infected == infected_3 then
```

```
223:                Add the new alert into the current cluster
224:                Write the new alert into alert_4
225:                Generate an event apt_sub_scenario_two_steps_alert
226:                Write apt_sub_scenario_two_steps_alert into
227:                apt_sub_scenario_two_steps_detection.log
228:                Send apt_sub_scenario_two_steps_alert to RT
229:                End for loop
230:            else
231:                goto Next cluster:
232:            end if
233:        else
234:            goto Next cluster:
235:        end if
236:    else
237:        goto Next cluster:
238:    end if
239:    Next cluster:
240:    if the current cluster is the last one in the clustered_dataset then
241:        Ignore the new alert
242:    else
243:        do for statements for the next cluster in the clustered_dataset
244:    end if
245: end for
246: End
```

First, the AC module determines to which one of the APT steps the *new alert*, coming from the AF module, belongs. MLAPT can detect four steps of the APT life cycle, mentioned in Section 4.2.2, Table 4.2 on page 60.

Based on the *new alert* step, AC has four processing engines, each engine processes all alerts which belong to one APT step.

For *alert_1 processing engine*, the second step of APT is the first detectable step, therefore, as soon as an alert of the second APT step is triggered, AC starts a new cluster and writes the new alert into *alert_1*.

For *alert_2 processing engine*, when a new alert for the third step of APT is triggered, the AC module checks all the clusters in the *clustered_dataset*. The cluster of interest is the one that has *alert_1* and the other alerts (*alert_2, alert_3, alert_4*) are still missed. For that cluster of interest, the algorithm checks time attributes: *time*, the time when the current processed alert is triggered; and *time_1*, the time when the *alert_1* is triggered. For the new alert to be considered, those time attributes should meet two conditions: $time > time\_1$ and $time - time\_1 <= TW$; whereas TW stands for the time window "correlation time". Following this, the first function of the CI module checks the *infected_host* attributes: *infected*, the infected host of the current processed alert; and *infected_1*, the infected host of alert_1. If both infected host attributes are matched, the current processed alert is added into the current cluster of interest as *alert_2*. In addition, an event *apt_sub_scenario_two_steps_alert* is generated and an alert email is sent to RT informing the network security team regarding this APT sub scenario detection. When one of the previous checks fails, AC checks if the current cluster is the last one in the *clustered_dataset*: if true, a new cluster is started and the current processed alerts is added as *alert_2*; if false, the process is to be repeated again for the next cluster.

For *alert_3 processing engine*, when a new alert for the fifth step of APT is triggered, AC checks all the clusters in the *clustered_dataset*. There are three cases for the cluster of interest: (1) when the cluster has *alert_1* and

alert_2 and the other alerts "*alert_3* and *alert_4*" are missed; (2) when the cluster has *alert_1* and the other alerts "*alert_2*, *alert_3*, *alert_4*" are missed; (3) and when the cluster has *alert_2* and the other alerts "*alert_1*, *alert_3*, *alert_4*" are missed.

For the first case of cluster of interest, AC checks all time attributes which should meet two conditions: *time* > *time_2* and *time* - *time_1* <= *TW*. Following this, CI checks all infected host attributes that should meet the condition *infected* == *infected_2*, as *alert_1* and *alert_2* are already in the cluster so it is guaranteed that *infected_1* == *infected_2* and there is no need for the first function of CI to check it. The current processed alert is then added into the current cluster of interest as *alert_3*, an event *apt_sub_scenario_three_steps_alert* is generated, and an alert email is sent to RT informing the network security team regarding this APT sub-scenario detection. If one of the previous checks is failed, it is checked if the current cluster is the last one in *clustered_dataset*: if true, a new cluster is started and the current processed alerts is added as *alert_3*; if false, the process is to be repeated again for the next cluster.

For the second and third case of cluster of interest, the process is similar to the first case, as shown in Algorithm 10, taking into consideration the corresponded time and infected host attributes.

For *alert_4 processing engine*, the first step is to find the cluster of interest in the *clustered_dataset*. When a new alert for the sixth step of APT is triggered, AC checks all the clusters in the *clustered_dataset*. There are seven cases for the cluster of interest: (1) when the cluster has *alert_1*, *alert_2*, and *alert_3*, and the last alert "*alert_4*" is missed; (2) when the cluster has *alert_1* and *alert_2* and the other alerts "*alert_3* and *alert_4*" are missed; (3) when the cluster has *alert_1* and *alert_3* and the other

alerts "*alert_2* and *alert_4*" are missed; (4) when the cluster has *alert_2* and *alert_3* and the other alerts "*alert_1* and *alert_4*" are missed; (5) when the cluster has *alert_1* and the other alerts "*alert_2*, *alert_3*, and *alert_4*" are missed; (6) when the cluster has *alert_2* and the other alerts "*alert_1*, *alert_3*, and *alert_4*" are missed; (7) and when the cluster has *alert_3* and the other alerts "*alert_1*, *alert_2*, and *alert_4*" are missed.

The process of all cases of cluster of interest in *alert_4* processing engine is similar to the process in *alert_3* processing engine explained above. The AC algorithm checks all time attributes of the cluster; after that, the CI algorithm checks all infected host and scanned host attributes; if all conditions "presented in Algorithm 10" are met, the current processed alert is added into the current cluster of interest as *alert_4*. Based on the cluster of interest, three events can be generated as an output of *alert_4* processing engine: *apt_full_scenario_alert* for case 1; *apt_sub_scenario_three_steps_alert* for cases 2, 3, and 4; and *apt_sub_scenario_two_steps_alert* for cases 5, 6, and 7. In addition, an alert email is sent to RT informing the network security team regarding this APT full or sub-scenario detection. If one of the algorithms' conditions fails, the process moves to the next cluster in *clustered_dataset* or it is ended if the current cluster is the last one.

5.2.3 Implementation of the Correlation Indexing (CI) Module

The first function of CI, evaluating the correlations between the cluster's alerts, is implemented within AC algorithm, as explained in the previous Section 5.2.2 on page 94. The implementation of the second function of CI,

calculating the correlation index $Corr_{id}$ for each cluster by the end of the correlation window, is shown in Algorithm 11.

Algorithm 11 Implementation pseudo-code of the second function of CI

```
1: Get the clustered_dataset
2: for each cluster in the clustered_dataset do
3:   if infected_2 == infected_1 then
4:     Corr_ab ← 1
5:   else
6:     Corr_ab ← 0
7:   end if
8:   if infected_3 == infected_2 Or infected_3 == infected_1 then
9:     Corr_bc ← 1
10:  else
11:    Corr_bc ← 0
12:  end if
13:  if infected_4 == scanned Or infected_4 == infected_3
14:    Or infected_4 == infected_2 Or infected_4 == infected_1 then
15:    Corr_cd ← 1
16:  else
17:    Corr_cd ← 0
18:  end if
19:  Corr_id == Corr_ab + Corr_bc + Corr_cd
20:  Write the current cluster with its Corr_id into the
21:  correlation_dataset
22: end for
23: End
```

To calculate $Corr_{id}$ for each cluster, the CI module makes use of the attributes of each alert in the cluster, applies the correlation rules mentioned in Section 4.2.2.3 on page 63, and calculates the correlation index $Corr_{id}$ based on the equation 4.3 mentioned also in Section 4.2.2.3 on page 65 .

5.3 Implementation of the Prediction Module (PM)

The PM module uses machine learning techniques to achieve its functionality. The process in this module undergoes three main steps: (1) Preparing the dataset, implemented in Python; (2) Training the prediction model, implemented in MATLAB [130]; and (3) Using the model for prediction, in Python and MATLAB.

5.3.1 Preparing the Machine Learning Dataset

Building the *machine_ learning_ dataset* is based on the *correlation_ dataset*, which is the output of the FCI correlation framework over a period of six months or more. The *correlation_ dataset* contains the correlated clusters, both full and sub APT scenarios, and the correlation index $Corr_{id}$ for each cluster. To prepare the *machine_ learning_ dataset*, PM makes the following modifications on the *correlation_ dataset*:

- The prediction of *apt_ sub_ scenario_ two_ steps_ alert* to complete the APT life cycle is based on the first two detectable steps of APT, therefore, only the clusters containing at least alerts for the first two detectable steps, i.e. *alert_ 1* and *alert_ 2*, are kept; the other clusters are filtered out of the *correlation_ dataset*.
- Based on the $Corr_{id}$ value, the *correlation_ dataset* clusters can be classified into four classes: Class 3, for APT full scenario and the cluster has four correlated alerts; Class 2, for APT sub-scenario and the cluster has three correlated alerts; Class 1, for APT sub-scenario and

the cluster has two correlated alerts; and Class 0, the cluster has only one alert. The *machine_learning_dataset* contains only two classes: Class 1 for APT full scenario; and Class 0, for uncompleted APT scenario. Thus, the PM module considers: (1) Class 3, in the *correlation_dataset*, as Class 1, for the *machine_learning_dataset*; and (2) Classes 2, 1, and 0, in the *correlation_dataset*, as Class 0, for the *machine_learning_dataset*.

- The class prediction is based on the first two detectable steps of APT, therefore, all the columns related to the third and fourth detectable alerts, i.e. *alert_3* and *alert_4* attributes, are filtered out of the *correlation_dataset*.

- Since the chosen machine learning classifiers work with numeric values, columns which are not numeric in the *correlation_dataset* are represented in a numerical format for the *machine_learning_dataset*. The *alert_type* values are mapped to numbers from 1 to 6, and the columns which contain IPs values (*src_ip_1, dest_ip_1, infected_host_1, src_ip_2, dest_ip_2, infected_host_2*) are mapped to numeric values using *socket* [131] and *struct.unpack* [132] functions built in Python.

5.3.2 Training the Prediction Model

As the task is to predict classes, classification methods are chosen and different machine learning algorithms are applied on *machine_learning_dataset* to train the model. The model is trained using four machine learning approaches, commonly used for classification problems, which are: decision tree learning, support vector machine, k-nearest neighbours and ensemble learning. Those ML approaches are previously explained in Section 2.3.1 on

page 23. The prediction accuracy of each trained model is calculated and the best model, which has the highest prediction accuracy, is chosen. The best model is saved to be used by the network security team.

5.3.3 Using the Model for Prediction

When a new *apt_sub_scenario_two_steps_alert* is generated by the correlation framework, the new data, i.e. the cluster attributes, is prepared as explained above in Section 5.3.1 on page 108, then the prediction model, which has been trained and chosen in the previous step, is applied.

As a result, the network security team can determine the probability of the current alert to complete the APT life cycle, and apply the required procedure to stop the attack before completion and achieving the final aim of data exfiltraition.

5.4 Summary

This chapter presents the MLAPT implementation. The Bro's scripting language is used in the implementation of all detection modules. The FCI correlation framework is implemented in two versions. The first one is implemented on top of Bro to be used on live traffic for real time detection; it can also be used offline on PCAP files. The second version is built in Python to be used offline on saved alerts' logs. The prediction module PM makes use of Python and Matlab to achieve its functionality.

All blacklists of blacklist-based detection modules are automatically updated based on different intelligence feeds at once. The MLAPT detection modules consume and handle some of Bro events, can generate eight events (alerts), send alert emails to RT and write alerts information into specific logs. Those detection modules' alerts are fed to the FCI correlation framework to be filtered, clustered, and the $Corr_{id}$ is calculated for each cluster. FCI can generate three types of alerts: *apt_full_scenario_alert*, *apt_sub_scenario_two_steps_alert* and *apt_sub_scenario_three_steps_alert*. FCI writes all the correlated clusters along with the $Corr_{id}$ of each cluster into the *correlation_dataset*. Based on the *correlation_dataset*, the PM module prepares *machine_learning_dataset*, applies different machine learning algorithms to find the best model and saves the prediction model to be used by the network security team.

Chapter 6

Evaluation Results

In this chapter, the evaluation of MLAPT is introduced and the achieved results are presented. As MLAPT consists of three main phases: threat detection, alert correlation and attack prediction; the evaluation of MLAPT undergoes the evaluation of the three phases respectively. Additionally, a comparison between the developed system MLAPT and other existing systems is provided.

6.1 Evaluation Metrics

Two main measures should be taken into consideration when evaluating MLAPT, the first one is the detection/prediction accuracy and the second measure is the processing speed.

6.1.1 Detection/Prediction Accuracy

To determine the detection accuracy of MLAPT, *True Positive Rate (TPR)* and *False Positive Rate (FPR)* are calculated for the detection modules and the correlation framework. Effective IDS can differentiate between real attacks and normal traffic. In general, four different results can be returned by an IDS: *True Positive (TP)*, when an attack is predicted as an attack; *False Positive (FP)*, when normal traffic is predicted as an attack; *False Negative (FN)*, when an attack is predicted as normal traffic; and *True Negative (TN)*, when normal traffic is predicted as normal traffic. In the literature, the efficiency of IDS is commonly measured by the false positive and false negative alarm rates [133]. The *Recall (R)* or *True Positive Rate (TPR)* is the proportion of correctly predicted attacks to the actual size of the attack, as calculated using equation 6.1:

$$TPR = \frac{TP}{TP + FN} \tag{6.1}$$

The *Specificity* is the proportion of correctly predicted normal traffic to the actual size of the normal traffic, as calculated using the equation 6.2:

$$Specificity = \frac{TN}{TN + FP} \tag{6.2}$$

The *False Positive Rate (FPR)* or false alarm rate, is calculated using the equation 6.3:

$$FPR = 1 - Specificity = \frac{FP}{TN + FP} \tag{6.3}$$

To determine the prediction accuracy of MLAPT, two measures of the prediction module are calculated: (1) the prediction accuracy which is the proportion of correct predictions, defined in equation 6.4; and (2) the confusion matrix [134] which gives more detailed analysis than the prediction accuracy, described in Table 6.1.

$$Prediction\ Accuracy = \frac{Number\ of\ samples\ predicted\ correctly}{Total\ number\ of\ samples} \quad (6.4)$$

TABLE 6.1: Confusion matrix description.

	Predicted 0	**Predicted 1**
Actual 0	TN	FP
Actual 1	FN	TP

6.1.2 Processing Speed

The processing speed is a significant feature of MLAPT. Even a high-quality IDS is not effective if its processing cost is too high, since the resulting loss of packets increases the probability that an attack is not detected. The processing load exerted by this algorithm depends on the characteristics of the rules as well as on the network traffic. Rules generally fall into one of two categories, depending on whether they apply to the packet header or the payload. *Header rules* inspect the packet header in an attempt to detect specific combinations of features, such as the source and destination address, port numbers, checksums or sequence numbers. *Payload rules* attempt to match a specific byte sequence in a packet's payload. IDS rules may also

combine header and payload specific match conditions. Since header size is generally fixed, the processing cost of applying header rules is nearly constant for each packet regardless of actual packet size, while the cost of payload rules scales with the packet size [135]. The processing speed of MLAPT can be calculated by a comparison between the real time of the attack and the detection time of the attack reported by one of the detection modules and the correlation framework.

6.2 Experimental Evaluation of MLAPT

6.2.1 Evaluation of the Detection Modules

Two main methods were used to evaluate the detection modules. In the first one, the detection modules were applied on pcap files which contain malicious traffic. Each pcap file was provided by a different third party, pcap file size and data source are mentioned in the evaluation section of each detection module. In the second evaluation method, Bro was installed on an experimental server (2x 4-core Intel Xeon CPU E5530 @ 2.40 GHz, 12 GB RAM) with passive access to part of the university campus live traffic (200 Mbps, 200 users, 550 nodes) via an optical TAP (Test Access Port). The detection modules were run on the experimental server and the network was monitored for one month.

6.2.1.1 Evaluation of the Disguised exe File Detection Module (DeFD)

To evaluate the effectiveness of the DeFD module, a download of disguised exe file was simulated via the campus network. An experimental server was set up, using Bro, to passively monitor the campus live traffic. DeFD was run on that experimental server for the purpose of disguised exe file detection. Two exe files were randomly selected, *SkypeSetup.exe* and *ViberSetup.exe*, and their names' extensions (*exe*) were changed into *pdf* and *doc* extensions, i.e. *SkypeSetup.pdf* and *ViberSetup.doc* respectively. The two disguised exe files were uploaded to the *speedyshare.com* public server. Then a host working on a computer connected to the Internet through the monitored network was used to connect to that public server and download the modified files. As shown in Figure 6.1, DeFD was able to detect both malicious downloads and write the information regarding each connection into a specific log.

#fields	timestamp	alert_type	orig_h	orig_p	resp_h	resp_p	infected_host	malicious_file
#types	time	string	addr	port	addr	port	addr	string
1407424021.202210		disguised_exe_alert	[redacted]	56973	207.244.73.42	80	[redacted]	SkypeSetup.pdf
1407424040.255414		disguised_exe_alert	[redacted]	53105	207.244.73.42	80	[redacted]	ViberSetup.doc
#close	2014-08-07-19-15-07							

FIGURE 6.1: The log produced by the DeFD module.

This experiment was repeated a hundred of times using different disguised exe files with different extensions. DeFD was able to detect all the malicious files, the average detection delay was 270 ms with a standard deviation of 54 ms.

6.2.1.2 Evaluation of the Malicious File Hash Detection Module (MFHD)

To evaluate the effectiveness of the MFHD module, two experiments were performed. In the first one, MFHD was applied on a pcap file which contains traffic infected by the *Nuclear EK* malware, which has an MD5 file hash value of *dc5c71aef24a5899f63c3f9c15993697* [136]. This pcap file was analysed by the provider and the ground truth was known. The infection was delivered by drive-by download attack and five malicious IPs were involved. MFHD successfully detected the malicious file *Nuclear EK* malware and identified the connection over which the malware was downloaded. Note that the ground truth blacklist was not provided to MFHD. Figure 6.2 shows part of *blacklist_detection_hash.log* produced by MFHD. The log file contains more information regarding the malicious connection than given in the figure, such as the source IP address, source and destination ports, but the figure shows only the connection timestamp, alert type, infected host and malicious file hash.

```
#fields   timestamp   alert_type  infected_host     malicious_hash
#types    time        string      addr              string
1411999086.158059  hash_alert  192.168.204.148  dc5c71aef24a5899f63c3f9c15993697
#close 2015-01-25-15-56-24
```

FIGURE 6.2: Part of the log produced by the MFHD module.

In the second evaluation experiment, the campus live traffic was monitored to detect hosts involved in downloading malicious files. MFHD was set up to create a log file of detected malicious file hashes. MFHD was run on the experimental server mentioned before in Section 6.2.1 on page 116. The monitoring was done for one month. The list of hosts involved in downloading malicious files was matched with the results of a malicious IP address

detection module. As shown in Figure 6.3, 19 hosts were detected by MFHD, involved in downloading malicious files; and 37 hosts were detected by MIPD, involved in malicious IP address connections. Within detected hosts, 12 were detected involved in both malicious IP connection and downloading malicious file, indicating that there was malware infection.

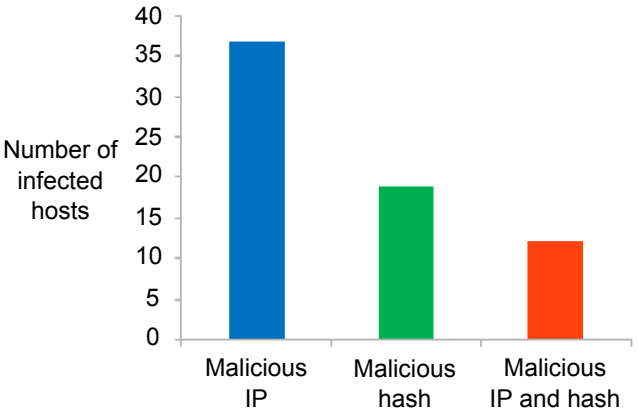

FIGURE 6.3: Detected hosts by the MFHD and MIPD modules.

MFHD also sent an alert email, in real time, regarding each malicious file hash detection to RT; where the network security team can perform additional forensics and respond to it. Figure 6.4 shows an example of a *Malicious_Hash* ticket, which was emailed to RT.

Greetings,

the security team CSIRT-MU detected involvement of the IP address [redacted] into the following incident:

Incident type: Malicious_Hash
Time of detection: 2014-12-10 14:43:18 +0100
IP address [redacted]
Domain name: ---

Details of this incident can be found at this address:
https://reports.csirt.muni.cz/A4FE72DC-65A8-1C74-8627-5664BE78D651

Best regards,
CSIRT-MU, the security team

Date: Wed, 10 Dec 2014 14:43:36 +0100

FIGURE 6.4: An example of a Malicious_Hash ticket.

6.2.1.3 Evaluation of the Malicious Domain Name Detection Module (MDND)

Three evaluation experiments were performed to test the MDND module in terms of effectiveness and real time detection. In the first one, MDND was applied on a pcap file. This pcap file contains traffic infected by malware with MD5 hash *f73c538c1558b1e02f52743534ca967e* [137]. The infection was delivered by a *Neutrino exploit kit (EK)* and six domains were involved. This fact was exploited to set the ground truth. The module consumes the pcap file and produces a log file. MDND was able to detect five out of six malicious domains which were involved in that infection while one of those malicious domains was not included in the module *blacklist.intel* file. Note that the ground truth blacklist was not provided to the MDND module. Figure 6.5 shows part of *blacklist_detection_domain.log* produced by the detection module.

```
#fields   timestamp      alert_type     infected_host  malicious_domain
#types    time           string         addr           string
1387853424.602463  domain_alert  192.168.1.107  brixton-beds.co.uk
1387853432.372462  domain_alert  192.168.1.107  flirtlivejasmin.com
1387853432.830202  domain_alert  192.168.1.107  www.dana123.com
1387853433.186351  domain_alert  192.168.1.107  restofthebesta.com
1387853433.685025  domain_alert  192.168.1.107  www.rightmedia.com
#close 2014-11-06-16-14-30
```

FIGURE 6.5: Part of the log produced by the MDND module.

In the second evaluation experiment, the campus live traffic was monitored for malicious domains detection. MDND was run on the experimental server mentioned before in Section 6.2.1 on page 116. The monitoring was done for one month. The list of hosts involved in malicious domain connections was matched with the results of a malicious IP address detection module. As shown in Figure 6.6, 22 hosts involved in malicious domain connections were detected, by MDND; and 37 hosts involved in malicious IP address connections were detected, by MIPD. Within detected hosts, 14 were involved in both malicious IP and domain connections, which indicated that they were infected with malware.

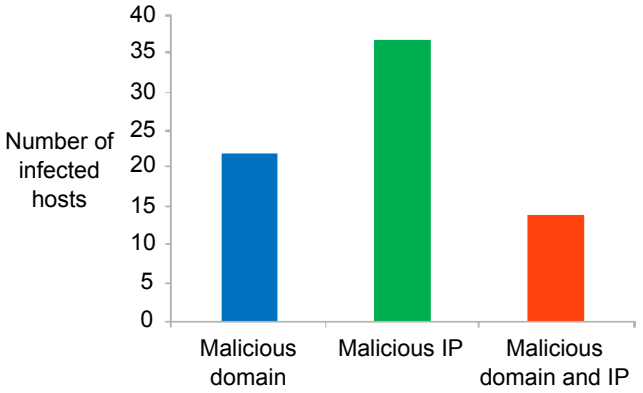

FIGURE 6.6: Detected hosts by the MDND and MIPD modules.

MDND also sent a real time alert email to report the malicious domain activities to RT. Figure 6.7 shows an example of a *Malicious_ Domain* ticket

which was sent by email to RT.

Greetings,

the security team CSIRT-MU detected involvement of the IP address
[redacted] into the following incident:

Incident type: Malicious_Domain
Time of detection: 2014-11-06 16:21:39 +0100
IP address [redacted]
Domain name: cqufwm.com

Details of this incident can be found at this address:
https://reports.csirt.muni.cz/A4FE72DC-65A8-1C74-8627-5664BE69D651

Best regards,
CSIRT-MU, the security team

Date: Wed, 10 Dec 2014 14:43:36 +0100

FIGURE 6.7: An example of a Malicious_Domain ticket.

The third evaluation experiment is to evaluate the real time detection capability of MDND. In this experiment, a script connecting to random malicious domains from the blacklist was installed on a computer in the monitored network. MDND was set up to send a report to RT as soon as a malicious connection was detected. The experiment consisted of the following steps. First, a script initiated connection to a randomly picked address from the blacklist of malicious domains. It noted the connection time with millisecond precision. Second, MDND detected a malicious connection after the first step and automatically created an RT ticket. Third, the RT ticket was received and the time of arrival, with millisecond precision, is noted . Finally, the connection start-up time and the time of RT ticket arrival were compared to calculate the detection delay. The average detection delay was found to be 310 ms with a standard deviation of 63 ms. Figure 6.8 shows the detection delay results of MDND.

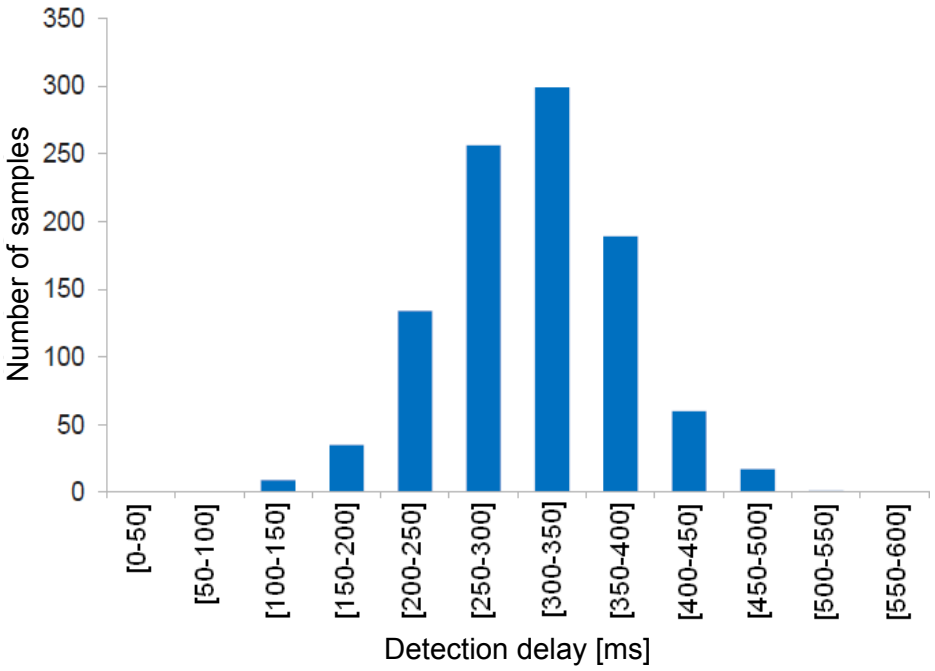

FIGURE 6.8: The real time detection of the MDND module.

6.2.1.4 Evaluation of the Malicious IP Address Detection Module (MIPD)

Two experiments were performed to evaluate the MIPD module. In the first evaluation experiment, three datasets each containing traffic carrying different malware were used. MIPD was applied on those pcap files which were analysed by the provider, therefore, the ground truth was known. The first pcap file contained traffic infected by the malware (*PizzaHut_ Coupon.exe*) with MD5 hash *191a02952905cc0037753700636c3339* [138]. The infection was delivered by an email attachment sent by Asprox botnet, which uses phishing emails and sends fake *Pizza Hut* emails with the subject line: *Free Pizza*. The second pcap file traffic was infected by the malware (*Label-CA-Toronto.exe*) with MD5 file hash *a6ba2cadc7c6891a5f437b212a18ac52* [139].

The infection was also delivered by Asprox botnet phishing email, but different malicious IP was used. The third pcap file traffic was infected by a piece of malware with MD5 file hash *dc5c71aef24a5899f63c3f9c15993697* [140]. The infection was delivered by drive-by-download attack and five malicious IPs were involved.

The MIPD module was set up to record the malicious IP connections into a log file. MIPD was able to detect all malicious IP addresses, which were involved in the three infections described in the previous paragraph without the knowledge of the ground truth blacklist. Figures 6.9, 6.10 and 6.11 shows part of *blacklist_detection_ip.log* produced by MIPD for each used pcap file. The log file contains other useful information such as the communication source and destination ports.

```
#fields   timestamp      alert_type  infected_host     malicious_ip
#types    time           string      addr              addr
1414537682.066774  ip_alert    192.168.56.255  192.168.56.101
1414537713.067728  ip_alert    85.12.29.172    192.168.56.101
#close 2014-11-29-20-40-04
```

FIGURE 6.9: Part of the log produced by the MIPD module for the first pcap file.

```
#fields   timestamp      alert_type  infected_host     malicious_ip
#types    time           string      addr              addr
1410463527.855812  ip_alert    184.107.222.130 172.16.165.133
1410463534.997906  ip_alert    172.16.165.2    172.16.165.133
#close 2014-11-29-20-47-41
```

FIGURE 6.10: Part of the log produced by the MIPD module for the second pcap file.

#fields	timestamp	alert_type	infected_host	malicious_ip
#types	time	string	addr	addr
1411999086.158059		ip_alert	192.168.204.148	148.251.154.3
1411999091.202658		ip_alert	192.168.204.148	148.251.154.29
1411999091.713753		ip_alert	192.168.204.148	89.40.71.156
1411999102.462283		ip_alert	192.168.204.148	79.133.219.113
1411999178.285571		ip_alert	192.168.204.148	77.87.78.127
#close 2014-11-29-20-56-24				

FIGURE 6.11: Part of the log produced by the MIPD module for the third pcap file.

In the second evaluation experiment, as in the previous two detection modules, the campus live traffic was monitored for malicious IP connections detection. The detected hosts by MIPD were matched with the detected ones of the MFHD and MDND modules. The result of this experiment is presented before in the previous Section 6.2.1.2, Figure 6.3 on page 119 and Section 6.2.1.3, Figure 6.6 on page 121. MIPD also sent an alert email, in real time, regarding each malicious IP detection similar to those RT tickets sent by the previous modules.

6.2.1.5 Evaluation of the Malicious SSL Certificate Detection Module (MSSLD)

To evaluate the MSSLD module, a virtual Internet-connected network was built, malware samples were injected into the virtual network, the network traffic was recorded into pcap files, and then the MSSLD module was applied on those pcap files.

As illustrated in Figure 6.12, two Windows XP SP3 virtual machines were connected to a physical consumer-grade router, which provided connection to the Internet. The virtual machines behaved as physical computers in a home network and were able to communicate with each other. The Virtual

machines traffic was recorded into two separate pcap files using the nictrace VirtualBox functionality [141]. Because no software besides the operating system and the malware was installed on the virtual machines and the operating system updates were disabled, the majority of the captured traffic was initiated by the installed malware. Moreover, it becomes easy to establish the ground truth.

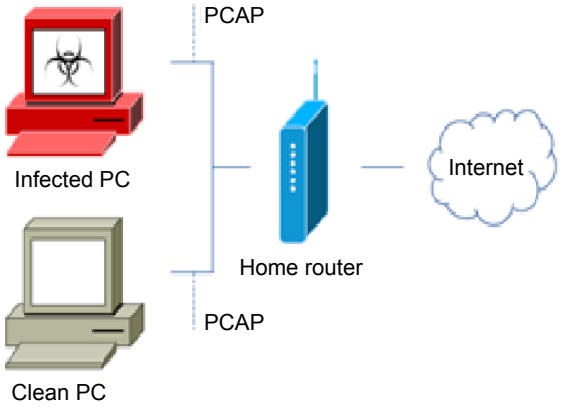

FIGURE 6.12: Topology of the implemented virtual network.

Two malware samples were injected independently into the virtual network for 5 minutes each. The first one is the *Trojan.Win32.Inject.sbqz*, also known as TorrentLocker, which has MD5 hash value of *aabe2844ee61e1f2969d7a96e1355a99*. The second injected malware sample is the *Trojan.Win32.Staser.bazr* malware, which has MD5 hash value of *e161a4d2716eb83552d3bd22ce5d603c*. The C&C servers for these two malware uses SSL certificates for communication over *https*. When the MSSLD module was applied on the captured pcap files, it successfully detected the malicious traffic as shown in Figure 6.13.

```
#fields  timestamp  alert_type  infected_host  malicious_ssl
#types  time        string      addr           string
138.857709  ssl_alert  192.168.1.101  45c0c2f1fa15b0ac5ce5fc018992a6ecf7e1e6bc
143.725647  ssl_alert  192.168.1.101  48a79b6bc3b9616f1e62fa4014997087673b358f
207.147152  ssl_alert  192.168.1.102  d8af2f6a1a2ba2b1b6e1a260e791fcab88cc2c8d
#close  2015-03-26-18-50-32
```

FIGURE 6.13: Part of a log produced by the MSSLD module.

6.2.1.6 Evaluation of the Domain Flux Detection Module (DFD)

This module was evaluated using two experiments. In the first evaluation experiment, the DFD module was applied on a pcap file provided by the Malware Capture Facility Project (MCFP) [142]. MCFP is an effort from the Czech Technical University ATG Group for capturing, analysing and publishing real and long-lived malware traffic. The captured network traffic took place between November 2013 and January 2014 in their capture facility. After analysing the captured data, MCFP found that a piece of malware used the domain flux technique to connect to its C&C server. There were a large group of packets going to the IP address 192.35.51.30, destination port 53/TCP. The content of these packets are DNS requests asking for domains being generated with a Domain Generation Algorithm (DGA). DFD was set up to consume the MCFP pcap file. It was able to detect the domain flux communications used by that malware and identify the infected host. All information related to the domain flux technique detection was written into a specific log as shown in Figure 6.14.

orig_h addr	orig_p port	resp_h addr	resp_p port	infected_host addr	domain_name string
10.0.2.107	29219	192.35.51.30	53	10.0.2.107	ndyotrc.com
10.0.2.107	29222	192.35.51.30	53	10.0.2.107	kbzmyrj.net
10.0.2.107	29225	192.35.51.30	53	10.0.2.107	asptecbd.ru
10.0.2.107	29228	192.35.51.30	53	10.0.2.107	yrtuqrbuk.cc

FIGURE 6.14: Part of a log produced by the DFD module.

In the second evaluation experiment, the virtual network mentioned before in Section 6.2.1.5 Figure 6.12 on page 126, was used to gather data. The *HEUR:Trojan.Win32.Generic* malware of MD5 value *fbb354f6773fb81927a59008cd9fd3a6* was executed on one of the virtual machines for 48 hours and the traffic was recorded into a pcap file. Through manual traffic analysis, it was found that the infected machine proceeded to use the domain flux technique to connect to the C&C servers over ports 1778, 3363, 3478 and 3479. The DFD module was applied on the captured pcap file, and the domain flux was detected and written into a log similar to that one shown in Figure 6.14.

6.2.1.7 Evaluation of the Tor Connection Detection Module (TorCD)

Three evaluation experiments were used to evaluate the TorCD module in terms of effectiveness and real time detection. In the first evaluation experiment, TorCD was applied on six pcap files with total size of 31.6 MB. Those pcap files contain recorded traffic of *Trojan.Tbot* (Skynet Tor Botnet) malware, which uses Tor network to communicate with its C&C centre [143]. All pcap files had been analysed by the third party, so the ground truth was already known. All Tor connections were detected and Figure 6.15 shows part of the log produced by the TorCD module.

In the second evaluation experiment, the *Trojan-Spy.Win32.Zbot.qvcn* malware of MD5 hash value *52d3b26a03495d02414e621ee4d0c04e*) was run on the virtual network, described in Section 6.2.1.5 Figure 6.12 on page 126, for ten hours and the traffic was recorded into a pcap file. It was found that the malware communicated solely through the Tor network and did not exhibit other activities. In the first two minutes, the malware initiated 67

```
#fields   timestamp        alert_type  infected_host   tor_server
#types    time             string      addr            addr
1349621090.423721  tor_alert  172.16.253.130  208.83.223.34
1349621090.945925  tor_alert  172.16.253.130  86.59.21.38
1349621102.634850  tor_alert  172.16.253.130  74.120.13.132
1349621102.628802  tor_alert  172.16.253.130  96.47.226.20
1349621102.670488  tor_alert  172.16.253.130  96.44.189.102
#close  2015-03-24-18-06-02
```

FIGURE 6.15: Part of a log produced by the TorCD module.

connections to 67 addresses belonging to the Tor network and transferred 3698 kB of data. The flow of data dropped for the remaining time, but new connections were still made. These findings were used to establish the ground truth. TorCD was applied on the captured pcap file and all Tor connections were detected and written into a log similar to that one shown in Figure 6.15.

In the third evaluation experiment, the campus live traffic was monitored for Tor connections detection. TorCD was run on the experimental server, mentioned before in Section 6.2.1 on page 116, and the monitoring was done for one month. The list of hosts involved in Tor connections was matched with the results of domain flux detection module. 14 hosts were detected by TorCD, involved in Tor connections, and 7 hosts were detected by DFD, involved in domain flux connections. Within the detected hosts, 5 were involved in both Tor and domain flux connections, which indicated that they were infected with malware. The TorCD module also sent an alert email, in real time, regarding each Tor connection detection similar to those RT tickets sent by the previous modules.

In the end of the detection modules evaluation section, it was found that: (1) When the detection modules is applied on the pcap files, either

the files provided by the third parties or the ones captured through the implemented virtual network mentioned in Section 6.2.1.5 Figure 6.12 on page 126, all the MLAPT detection modules, apart from MDND, were able to detect the malicious connections with a true positive rate (TPR) of 100% and a false positive rate (FPR) of 0%. In the first evaluation experiment of MDND, the module was able to detect five out of six malicious domains while missed one malicious domain which was not included in the MDND module blacklist, i.e. TPR is 83% and FPR is 0%. For this reason, all blacklists of the blacklist-based detection modules are automatically updated every day based on different intelligence feeds at once. (2) When the detection modules were run on the experimental server, mentioned in Section 6.2.1 on page 116, for the live traffic monitor, all the MLAPT detection modules were able to detect and report the malicious connections in real time.

6.2.2 Evaluation of the FCI Correlation Framework

In the absence of any publicly available data which contains APT attack traffic, which can be used in the evaluation of the FCI framework. We had to build a new dataset which contains APT attack traffic. Using the campus network to gather attack data does not guarantee capturing any APT attack traffic against the monitored network.

The aim of the correlation framework is to identify different alerts raised by the various detection modules, which could be correlated and belong to one APT attack scenario. To effectively evaluate the FCI correlation framework, a dataset containing many of the detection modules alerts, in which some of those alerts belong to APT attack scenarios, has been built. The data is generated to appear as APT attack scenarios were simulated on the campus network, the techniques used in the APT life cycle were identified

by the detection modules, and all generated alerts were written into the *simulation dataset*. That dataset also contains many of the generated alerts which do not belong to APT attack scenarios. All the detection modules have been evaluated on pcap files and on the real live traffic as well, as previously explained in Section 6.2.1 on page 116. The aim of this experiment is to test if the FCI correlation framework is able to detect those APT scenarios in the *simulation dataset*.

6.2.2.1 Data Generation

A script is written, using Python. This script generates two types of alerts: (1) Random alerts which do not relate or belong to one APT attack scenario; and (2) Related alerts which belong to a full or sub-APT attack. Each alert has seven attributes: *alert_type, timestamp, src_ip, src_port, dest_ip, dest_port* and the *infected_host*; only the *scan_alert* has the extra *scanned_host* attribute.

To generate a random alert, the *alert_type* is selected randomly from the set of all 8 detectable alerts, i.e. *disguised_exe_alert, hash_alert, domain_alert, ip_alert, ssl_alert, domain_flux_alert, scan_alert* and *tor_alert*. The *timestamp* is assigned a random value between Fri, 01 Jan 2016 00:00:01 GMT and Thu, 30 Jun 2016 23:59:59 GMT. The *src_ip* is randomly assigned an IP address on the campus network. The *src_port* is selected randomly from the 49152, 65535 range of ports, which are usually assigned dynamically to client applications when initiating a connection. The *dest_ip* value is assigned based on the selected *alert_type*: If the *alert_type* is *disguised_exe_alert*, *hash_alert* or *ssl_alert*, then the *dest_ip* can be any valid IP address which is not on the campus network; if the *alert_type* is *domain_alert* or *domain_flux_alert*, then the *dest_ip*

can use an IP address which is on the campus network; if the *alert_type* is assigned *ip_alert*, then the *dest_ip* can select a random IP address from the *ip_blacklist* described in Section 5.1.4 on page 81; if *alert_type* is *scan_alert*, the *dest_ip* is selected randomly from campus network IP addresses; and if the *alert_type* is *tor_alert*, the *dest_ip* is selected randomly from *tor_server_list* mentioned in Section 4.2.1.8 on page 59. The *dest_port* is selected based on the selected *alert_type*: if the *alert_type* is *disguise_exe_alert* or *hash_alert*, the *dest_port* should be 80; if the *alert_type* is *domain_alert* or *domain_flux_alert*, the *dest_port* should be 53; if the *alert_type* is *ip_alert*, *ssl_alert* or *tor_alert*, the *dest_port* should be 443; and if the *alert_type* is *scan_alert*, the *dest_port* is selected randomly from the 1, 1024 range of ports. The *infected_host* should be the same *src_ip* of the connection. Finally, the *scanned_host* (only if *alert_type* is *scan_alert*) should be the same *dest_ip* of the connection.

To generate an APT full-scenario (consisting of 4 correlated alerts) or sub-scenario (consisting of 2 or 3 correlated alerts), the APT life cycle should be taken into consideration. Meaning, the generated alerts' attributes of each scenario are selected to appear as an APT attack which is simulated through the campus network.

6.2.2.2 Experimental Setup

To determine the number of random alerts to be generated for the *simulation_dataset*, the experimental server, previously mentioned in Section 6.2.1 on page 116, was used to monitor part of the university campus network. All detection modules were run on the experimental server to analyse the network traffic; the monitoring period and the number of detected alerts were determined. According to the actual university network size and the

actual *simulation_ dataset* monitoring period, the number of the generated random alerts was calculated. The number of the generated APT full- and sub-scenarios should be suitable to get enough samples for each class in the *machine_ learning_ dataset* previously explained in Section 5.3.1 on page 108.

The network monitoring was conducted via the experimental server for 2 weeks and 9 different alerts were detected by the detection modules. The size of the monitored network was 550 nodes, while the whole campus network is 23500 nodes. Meaning, if the whole campus network is monitored for 6 months, 4900 alerts are expected to be detected by the detection modules. Therefore, 4900 alerts were generated for the *simulation_ dataset*, of which 100 APT full attack (each scenario is 4 correlated alerts) and 50 APT sub-attack 3 steps (each scenario is 3 correlated alerts); 50 APT sub-scenarios 2 steps (each scenario is 2 correlated alerts); and 4250 random alerts (which do not relate or belong to APT attack scenarios). The APT life cycle period was configured to be for a maximum of one week.

Figure 6.16 shows part of the *simulation_ dataset* with examples of 4 random alerts (alerts 1, 2, 3 and 4), one full APT attack (alerts 5, 6, 7 and 8), sub-APT attack with three steps (alerts 9, 10 and 11) and sub-APT attack with two steps (alerts 12 and 13).

id	alert_type	timestamp	src_ip	src_port	dest_ip	dest_port	infected_host	scanned_host
1	domain_alert	1452998107	149.170.102.163	63645	149.170.39.93	53	149.170.102.163	
2	ip_alert	1464055590	149.170.109.65	56761	84.92.85.198	443	149.170.109.65	
3	ssl_alert	1458212586	149.170.119.6	60822	139.182.102.123	443	149.170.119.6	
4	tor_alert	1455639999	149.170.146.107	51376	80.81.17.31	443	149.170.146.107	
5	domain_alert	1458564234	149.170.8.178	58562	149.170.39.93	53	149.170.8.178	
6	ssl_alert	1459065352	149.170.8.178	61127	53.107.134.191	443	149.170.8.178	
7	scan_alert	1459110382	149.170.8.178	59094	149.170.185.189	830	149.170.8.178	149.170.185.189
8	tor_alert	1459142888	149.170.8.178	60178	176.31.107.163	443	149.170.8.178	
9	ssl_alert	1459761925	149.170.205.136	58469	155.136.118.250	443	149.170.205.136	
10	scan_alert	1460181080	149.170.205.136	61518	149.170.227.158	657	149.170.205.136	149.170.227.158
11	tor_alert	1460243277	149.170.205.136	59239	65.110.100.163	443	149.170.205.136	
12	hash_alert	1457714526	149.170.233.171	56560	31.189.78.33	80	149.170.233.171	
13	ssl_alert	1458163332	149.170.233.171	50074	159.124.176.26	443	149.170.233.171	

FIGURE 6.16: Part of the *simulation_ dataset*.

6.2.2.3 Results and Discussion

The FCI correlation framework was applied on the *simulation_ dataset*. Table 6.2 shows the FCI correlation framework detection results. This table indicates the TPR and the FPR for each studied APT attack, both full and partial attacks. Among all studied APT attacks, the best TPR results were for the APT sub-attack two steps scenario, followed by the APT sub-attack three steps scenario and APT full attack, respectively. The results show that the higher the number of related alerts, the lower the TPR and the higher FPR. This is due to the higher possibility of the random alerts to be incorrectly clustered when more alerts are to be correlated for APT. By manual analysis for the results, the incorrect alerts clustering was the main reason of the false alarms. Some APT attacks were not detected due to some of the random alerts which were incorrectly clustered and correlated. This can happen if those random alerts, by chance, meet the clustered and correlation rules, so one random alert can interfere with a running APT scenario (if the random alert is triggered for the missed scenario step, for the same infected host, and within the correlation time) and cause the false positive detection of APT and false negative detection of the random alert. Besides, a very rare

case can cause the wrong detection is when two, three or four random alerts can meet the correlation rules, by chance, and are correlated incorrectly.

TABLE 6.2: Correlation framework detection results.

APT attack scenario	**Detection result**	**TP**	**FP**	**FN**	**TN**	**P**	**N**	**TPR**	**FPR**
APT full scenario (4 steps)	90*4	78*4	12*4	88	4452	400	4500	78%	1%
APT sub-scenario (3 steps)	65*3	42*3	23*3	24	4681	150	4750	84%	1.4%
APT sub-scenario (2 steps)	85*2	47*2	38*2	6	4724	100	4800	94%	1.6%
APT full and sub-scenario	725	532	193	118	4132	650	4250	81.8%	4.5%

Figure 6.17 indicates part of the *correlation_ dataset* showing examples of the clustered alerts, clustered by the clustering algorithm, and the correlation index of each cluster, calculated by the correlation index algorithm.

cluster_id	alert_type_1	alert_type_2	alert_type_3	alert_type_4	correlation_index
1	disguised_exe_alert	ip_alert	scan_alert	tor_alert	3
2		ssl_alert	scan_alert	tor_alert	2
3	hash_alert	domain_flux_alert	scan_alert		2
4	hash_alert	ssl_alert			1
5	domain_alert	ssl_alert	scan_alert	tor_alert	3
6	domain_alert				0
7	hash_alert				0
8	domain_alert	ip_alert	scan_alert	tor_alert	3
9		domain_flux_alert	scan_alert		1
10		ssl_alert			0

FIGURE 6.17: Part of the *correlation_ dataset*.

Due to the space limitation, the attributes columns of each alert in Figure 6.17 are hidden. For example, for the first cluster in Figure 6.17, the alerts' attributes and the APT life cycle are shown in Figure 6.18.

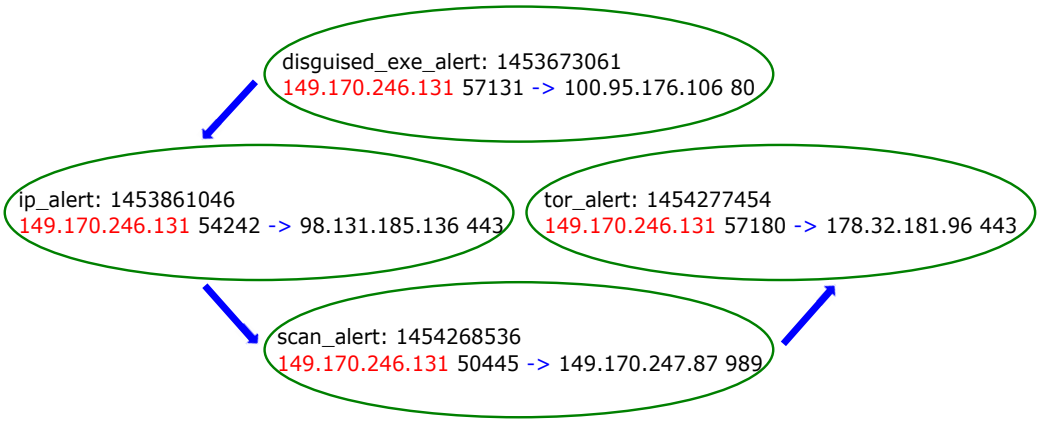

FIGURE 6.18: The APT life cycle and alerts' attributes of the first cluster.

6.2.3 Evaluation of the APT Prediction Module (PM)

To evaluate the PM module, three main steps were followed: (1) Preparing the *machine_learning_dataset*; (2) Training the prediction model; and (3) Saving the model for prediction.

Using the *correlation_dataset*, which is the output of the FCI correlation framework over a period of six months, the *machine_learning_dataset* is prepared as explained in Section 5.3.1 on page 108. Figure 6.19 shows part of *machine_learning_dataset*, the total number of samples in the dataset is 125: of which 78 samples of class 1, and 47 samples of class 0.

alert_ type_1	timestamp_1	src_ ip_1	src_ port_1	dest_ ip_1	dest_ port_1	infected_ host_1	alert_ type_2	timestamp_2	src_ ip_2	src_ port_2	dest_ ip_2	dest_ port_2	infected_ host_2	correlation_ index
1	1451972512	2510998866	59166	663539635	80	2510998866	4	1452431929	2510998866	52552	1729321334	443	2510998866	1
3	1459297694	2510997771	51543	2510956380	53	2510997771	4	1459419359	2510997771	65444	1315724730	443	2510997771	0
1	1464747034	2511001750	59958	2496369261	80	2511001750	5	1465146197	2511001750	54214	3785245666	443	2511001750	0
1	1465378617	2510980105	62921	1334164826	80	2510980105	6	1465894243	2510980105	63348	2510956381	53	2510980105	1
2	1464709345	2510995676	49631	1130775286	80	2510995676	5	1465150591	2510995676	61950	3848195829	443	2510995676	0
3	1455456975	2510950995	55490	2510956381	53	2510950995	6	1455925288	2510950995	60046	2510956381	53	2510950995	1
1	1463023917	2510970230	64874	1069783341	80	2510970230	5	1463103576	2510970230	64419	2704841754	443	2510970230	0
1	1461079375	2510946533	52595	2469707780	80	2510946533	5	1461174545	2510946533	52075	1760499160	443	2510946533	1
3	1458631211	2511002012	65007	2510956380	53	2511002012	5	1459140469	2511002012	62902	347320708	443	2511002012	0
3	1463609057	2511005628	57336	2510956380	53	2511005628	5	1464027890	2511005628	54879	1451821046	443	2511005628	1
3	1458826774	2510980218	51231	2510956380	53	2510980218	4	1459295521	2510980218	62341	3571669840	443	2510980218	0
1	1458435055	2510998731	63978	980331439	80	2510998731	5	1458487612	2510998731	65412	1807446700	443	2510998731	1
3	1454464461	2511001596	54751	2510956380	53	2511001596	5	1454836598	2511001596	60542	1370165524	443	2511001596	0
2	1460834388	2510965488	51541	710144137	80	2510965488	6	1460954836	2510965488	53692	2510956380	53	2510965488	0
3	1453194277	2510986100	52066	2510956381	53	2510986100	6	1453387501	2510986100	58253	2510956380	53	2510986100	1
2	1466974319	2510982787	60999	1414713034	80	2510982787	4	1467245625	2510982787	56217	2732547640	443	2510982787	1
3	1454504844	2510946305	62680	2510956380	53	2510946305	5	1455081205	2510946305	57790	1402391355	443	2510946305	0
2	1452892572	2510964657	53714	809457463	80	2510964657	5	1453174962	2510964657	50086	120125080	443	2510964657	0
3	1461563791	2510950685	57486	2510956380	53	2510950685	5	1461634522	2510950685	53365	4116324364	443	2510950685	1
2	1458860662	2510950995	61634	489438816	80	2510950995	6	1459094688	2510950995	54017	2510956380	53	2510950995	0
2	1462750618	2510950995	58735	409891058	80	2510950995	5	1463056367	2510950995	49466	1878323348	443	2510950995	0

FIGURE 6.19: Part of machine_learning_dataset.

As there is no machine learning algorithm which can be regarded as the best or the optimal one, various experiments should be performed on the *machine_learning_dataset* using several machine learning algorithms, and then a comparison between the trained models is made.

The Matlab's Classification Learner application [144] is used to train models to classify the *machine_learning_dataset*. Automated training is performed to search for the best classification model type, including decision trees, support vector machines, nearest neighbours, and ensemble classification. Those classification algorithms are previously explored in Section 2.3.1 on page 23, and the characteristics of each classifier type can be found in [145]. Cross-validation is used as a validation scheme to examine the prediction accuracy of each trained model.

Cross-validation is a model assessment technique used to evaluate a machine learning algorithm's performance in making predictions on new datasets which has not been trained on [146]. This is done by partitioning a dataset and using a subset to train the algorithm and the remaining data for testing. Each round of cross-validation involves randomly partitioning the original dataset into a *training set* and a *testing set*. The training set is then used to train a supervised learning algorithm and the testing set is used to evaluate its performance. This process is repeated several times and the average accuracy is used as a performance indicator. Table 6.3 shows the prediction accuracy for all investigated classification algorithms used to train the classification models.

Experimental results show that the best classification algorithm is the *Linear SVM*, with a prediction accuracy of 84.8%. This trained model can be saved by the network security team to be applied on real time traffic when a new real time *apt_sub_scenario_two_steps_alert* is triggered, as previously

TABLE 6.3: Classification algorithms and the prediction accuracy of the trained models.

Classification algorithms		**Prediction accuracy**
Decision trees	Complex tree	83.0%
	Medium tree	83.0%
	Simple tree	84.4%
Support vector machines	Linear SVM	84.8%
	Quadratic SVM	81.6%
	Cubic SVM	76.9%
	Fine Gaussian SVM	69.4%
	Medium Gaussian SVM	80.3%
	Coarse Gaussian SVM	81.0%
Nearest neighbour classifiers	Fine KNN	76.2%
	Medium KNN	80.3%
	Coarse KNN	68.0%
	Cosine KNN	82.3%
	Cubic KNN	78.9%
	Weighted KNN	78.2%
Ensemble classifiers	Boosted trees	83.7%
	Bagged trees	82.3%
	Subspace discriminant	81.6%
	Subspace KNN	72.8%
	RUSBoosted trees	81.0%

explained in Section 5.3.3 on page 110. Table 6.4 shows the confusion matrix for the *Linear SVM* prediction model with a prediction accuracy of 88.4% for class 1 and 78.7% for class 0.

TABLE 6.4: Confusion matrix for *Linear SVM* prediction model.

	Predicted 0	**Predicted 1**
Actual 0	37	10
Actual 1	9	69

6.3 A Performance Analysis of Existing APT Detection Systems

This section presents a performance analysis of four existing APT detection systems, and provides a comparison between the developed system MLAPT and these current systems, as shown in Table 6.5.

TABLE 6.5: A comparison between MLAPT and other existing systems.

APT detection system	**Autonomy**	**APT steps**	**speed**	**TPR**	**FPR**	**Prediction accuracy**
MLAPT	Autonomous	4	Real time	81.8%	4.5%	84.8%
TerminAPTor	Agent-based	4	Real time	100%	high	No
C&C-based	Autonomous	1	Off-line	83.3%	0%	No
Spear phishing based	Autonomous	1	Real time	97.2%	14.2%	No
Context-based	Agent-based	4	Real time	?	27.88%	No

The most effective system in terms of true positive rate is TerminAPTor [70] with a TPR of 100%, previously mentioned in Section 3.3 on page 41. However, the developers mentioned that TerminAPTor has a high rate of false positives (although they did not mention the figure of FPR) and needs to be improved by filtering the false positives. Moreover, this detector requires the alerts to be provided by other systems (agent-based) and cannot work autonomously. Despite having the lowest false positive rate of 0%, the C&C-based system [71], presented previously in Section 3.3 on page 42, does not achieve the real time detection. Furthermore, the authors stated that the detection can be easily evaded when the infected hosts connect to the C&C domains while users are surfing the Internet. Additionally, missing the detection of C&C domains leads to failure in APT detection since this

system depends on detecting only one step of the APT life cycle. Whilst the spear phishing based system [72], explored earlier in Section 3.3 on page 43, has a TPR of 97.2%, the FPR of 14.2% is considerably high. In addition, depending on one step for APT detection leads the system to fail when missing the spear phishing email detection. This missing can happen when the spear phishing email does not include any of the tokens which are necessary for the algorithm process. The context-based system [77], already stated in Section 3.3 on page 47, has a significantly high FPR of 27.88% while the TPR was not provided by the authors. Besides, this framework requires significant expert knowledge to set up and maintain; and similar to TerminAPTor, it is an agent-based system and cannot work autonomously.

Having a high rate of true positives is significant. Nevertheless, increasing the amount of true positives means that the false positive rate also increases. Thus, the balance between TPR and FPR is an essential requirement for any detection system. The developed system MLAPT has a suitable balance between the two values of TPR and FPR with 81.8% and 4.5% respectively. MLAPT can also work autonomously and generate the required events based on its own detection modules. The generated events covers four detectable steps of the APT life cycle which reduces the false positives and gives more possibility of APT detection in case one of the steps is missed. Furthermore, this system can achieve the real time detection, so it can be much easier to trace back to the attacker, minimise the damage and prevent further break-ins. Moreover, to the author's knowledge, MLAPT is the only system which can predict APT in its early steps with a prediction accuracy of 84.8%, which prevents the attacker from achieving the goal of data exfiltration.

6.4 Summary

This chapter presents the evaluation of the MLAPT phases. When evaluating the detection modules, all the MLAPT detection modules, apart from MDND, were able to detect the malicious connections with a true positive rate (TPR) of 100% and a false positive rate (FPR) of 0%. The MDND module was able to detect the malicious domain connections with a TPR of 83% and FPR of 0%. In addition, all the MLAPT detection modules were able to detect and report the malicious connections in real time. When evaluating the FCI correlation framework, among all studied APTs, the best TPR results were for the APT sub-attack two steps scenario with a value of 94%, followed by the APT sub-attack three steps scenario with a value of 84% and APT full attack with a value of 78%, respectively. FCI was able to detect APTs, both full and partial scenarios, with a TPR of 81.8% and FPR of 4.5%. When evaluating the APT prediction module, the best classification algorithm was the *Linear SVM*, with a prediction accuracy of 84.8%.

Chapter 7

Conclusions and Future Work

Although virus scanners, firewalls and IDPSs are able to detect and prevent many types of cyber attacks, cyber-criminals in turn have developed more advanced methods and techniques to intrude into the target's network and exploit its resources. APTs target specific organisations, this class of attack are composed of various stages. The main aim of the APT attack is espionage and then data exfiltration. Therefore, the APT attack is considered as a new and more complex version of multi-step attack. Most of the research in the area of APT detection has focused on analysing already identified APTs or detecting a particular APT attack uses a specific malware. Some previous approaches have attempted to detect potential APT attacks. However, the current APT detection systems face serious shortcomings in several aspects such as achieving real time detection, detecting all APT attack steps, having a suitable balance between false positive and false negative rates, and correlating of events spanning over a long period of time.

This research aims to develop a novel system to detect and predict APT attacks. A Machine-Learning-based APT detection system, called MLAPT,

has been developed. MLAPT runs through three main phases: threat detection, alert correlation and attack prediction. In the first phase, the sniffed data traffic is scanned to detect possible techniques used in the APT life cycle. To this end, eight detection modules have been developed and tested; each module implements a method to detect one technique used in one of APT steps. These detection modules are as follows: disguised exe file detection (DeFD), malicious file hash detection (MFHD), malicious domain name detection (MDND), malicious IP address detection (MIPD), malicious SSL certificate detection (MSSLD), domain flux detection (DFD), scan detection (SD), and Tor connection detection (TorCD). The output of this phase is alerts, also known as events, triggered by the individual modules. In the second phase, the alerts raised by the individual detection modules are fed to the newly designed (FCI) correlation framework. The aim of the correlation framework is to find alerts could be related and belong to one APT attack scenario. The process in this phase undergoes three main steps: alerts filter (AF), to identify redundant or repeated alerts; clustering of alerts (AC), which most likely belong to the same APT attack scenario; and correlation indexing (CI), to evaluate the degree of correlation between alerts of each cluster. The main objective of using the correlation framework is to reduce the false positive rate of the MLAPT detection system. In the final phase, a machine-learning-based prediction module (PM) is designed and implemented based on a historical record of the monitored network. This module can be used by the network security team to determine the probability of the early alerts to develop a complete APT attack.

In spite of the fact that the detection modules methodologies exist in the literature, their implementation and validation in real traffic are significant contributions to the field. The correlation framework and prediction module are two other major contributions in this work.

Bro scripting language is used in the implementation of all detection modules. The FCI correlation framework is implemented in two versions. The first one is implemented on top of Bro to be used on live traffic for real time detection; it can also be used offline on PCAP files. The second version is built in Python to be used offline on saved alerts' logs. The prediction module PM makes use of Python and Matlab to achieve its functionality.

All blacklists of blacklist-based detection modules are automatically updated based on different intelligence feeds at once. The MLAPT detection modules can generate eight events (alerts), send alert emails to RT and write alerts information into specific logs. Those detection modules' alerts are fed to the FCI correlation framework to be filtered, clustered, and the $Corr_{id}$ is calculated for each cluster. FCI can generate three types of alerts: *apt_full_scenario_alert*, *apt_sub_scenario_two_steps_alert* and *apt_sub_scenario_three_steps_alert*. FCI writes all the correlated clusters along with the $Corr_{id}$ of each cluster into the *correlation_dataset*. Based on the *correlation_dataset*, the PM module prepares *machine_learning_dataset*, applies different machine learning algorithms to find the best model and saves the prediction model to be used by the network security team.

All the MLAPT detection modules, apart from MDND, were able to detect the malicious connections with a true positive rate (TPR) of 100% and a false positive rate (FPR) of 0%. The MDND module was able to detect the malicious domain connections with a TPR of 83% and FPR of 0%. In addition, all the MLAPT detection modules were able to detect and report the malicious connections in real time. Regarding the FCI correlation framework, among all studied APTs, the best TPR results were for the APT

sub-attack two steps scenario with a value of 94%, followed by the APT sub-attack three steps scenario with a value of 84% and APT full attack with a value of 78%, respectively. MLAPT was able to detect the APT attacks with a TPR of 81.8% and FPR of 4.5%. The APT prediction module, built based on the *Linear SVM* algorithm, has a prediction accuracy of 84.8%.

For future work, a number of improvements within the system could be made. First, it is suggested that more detection modules are added to detect other techniques used in the APT attack life cycle. Furthermore, if MLAPT were able to monitor the internal network traffic, other detection modules could be added to dtect brute force and pass the hash attacks, increasing the detectable steps of the the system. Second, it is also recommended that more than one detection module for the same technique are developed. Third, it is advised that alerts from external IDSs deployed on the network are received and fed to MLAP, which can reduce the false positive rate of the system. Fourth, MLAP detection modules were evaluated on real traffic and pcap files contain real attacks. However, the FCI framework was validated on synthetic data. Therefore, it would be beneficial to test MLAP on real APTs. Nevertheless, obtaining such data is not easy, and the lack of relevant publicly available data sources was the main reason for using the synthetic data when evaluating the correlation framework.

Appendix A

Author's Contributions

A.1 Journals

- Ibrahim Ghafir, Vaclav Prenosil, and Mohammad Hammoudeh. *Botnet Command and Control Traffic Detection Challenges: A Correlation-based Solution.* International Journal of Advances in Computer Networks and Its Security (IJCNS), New York, USA: theIRED, 2017, vol. 7, Issue 2, p. 27-31. ISSN 2250-3757 [147].
  [Author's contribution: 80%]

- Ibrahim Ghafir, Jakub Svoboda, Vaclav Prenosil. *A Survey on Botnet Command and Control Traffic Detection.* International Journal of Advances in Computer Networks and Its Security (IJCNS), New York, USA: theIRED, 2015, vol. 5, Issue 2, p. 75-80. ISSN 2250-3757 [148].
  [Author's contribution: 80%]

- Jakub Svoboda, Ibrahim Ghafir, Vaclav Prenosil. *Network Monitoring Approaches: An Overview.* International Journal of Advances in Computer Networks and Its Security (IJCNS), New York, USA: theIRED,

2015, vol. 5, Issue 2, p. 88-93. ISSN 2250-3757 [149].
[Author's contribution: 60%]

- Ibrahim Ghafir and Vaclav Prenosil. *Advanced Persistent Threat Attack Detection: An Overview.* International Journal of Advances in Computer Networks and Its Security (IJCNS), New York, USA: theIRED, 2014, Volume 4, Issue 4, p. 50-54. ISSN: 2250-3757 [24].
[Author's contribution: 80%]

- Ibrahim Ghafir, Mohammad Hammoudeh, Vaclav Prenosil, Liangxiu Han and Robert Hegarty. *MLAPT: A Correlation-Based System for Real-Time Advanced Persistent Threat Detection and Prediction.* Journal paper, under review.
[Author's contribution: 80%]

A.2 Book Chapters

- Ibrahim Ghafir and Vaclav Prenosil. *Malicious File Hash Detection and Drive-by Download Attacks.* In Suresh Chandra Satapathy, K. Srujan Raju, Jyotsna Kumar Mandal, Vikrant Bhateja. Proceedings of the Second International Conference on Computer and Communication Technologies, series Advances in Intelligent Systems and Computing. Hyderabad: Springer, 2016. p. 661-669, 9 pp. Vol. 379. ISBN 978-81-322-2516-4. doi:10.1007/978-81-322-2517-1_63 [150].
[Author's contribution: 80%]

- Ibrahim Ghafir and Vaclav Prenosil. *Proposed Approach for Targeted Attacks Detection.* In Sulaiman, H.A., Othman, M.A., Othman, M.F.I., Rahim, Y.A., Pee, N.C.. Advanced Computer and Communication Engineering Technology, Lecture Notes in Electrical Engineering.

Phuket: Springer International Publishing, 2016. p. 73-80, 9 pp. Vol. 362. ISBN 978-3-319-24582-9. doi:10.1007/978-3-319-24584-3 [151].
[Author's contribution: 80%]

A.3 Conferences

- Ibrahim Ghafir, Vaclav Prenosil, Mohammad Hammoudeh and Umar Raza. *Malicious SSL Certificate Detection: A Step Towards Advanced Persistent Threat Defence.* In Proceedings of International Conference on Future Networks and Distributed Systems. Cambridge, United Kingdom: ACM Digital Library, 2017. ISBN 978-1-4503-4844-7. doi:10.475/123_4 [152].
[Author's contribution: 80%]

- Ibrahim Ghafir, Vaclav Prenosil, Ahmad Alhejailan and Mohammad Hammoudeh. *Social Engineering Attack Strategies and Defence Approaches.* In Proceedings of International Conference on Future Internet of Things and Cloud. Vienna, Austria: IEEE Xplore Digital Library, 2016. p. 145-149, 5 pp. ISBN 978-1-5090-4052-0. doi:10.1109/FiCloud.2016.28 [10].
[Author's contribution: 80%]

- Ibrahim Ghafir, Vaclav Prenosil, Jakub Svoboda and Mohammad Hammoudeh. *A Survey on Network Security Monitoring Systems.* In Proceedings of International Conference on Future Internet of Things and Cloud. Vienna, Austria: IEEE Xplore Digital Library, 2016. p. 77-82, 6 pp. ISBN 978-1-5090-3946-3. doi:10.1109/W-FiCloud.2016.30 [153].
[Author's contribution: 80%]

- Ibrahim Ghafir and Vaclav Prenosil. *Advanced Persistent Threat and Spear Phishing Emails.* In Proceedings of International Conference Distance Learning, Simulation and Communication. Brno, Czech Republic: University of Defence, 2015. p. 34-41, 8 pp. ISBN: 978-80-7231-992-3 [154].
  [Author's contribution: 80%]

- Ibrahim Ghafir and Vaclav Prenosil. *Blacklist-based Malicious IP Traffic Detection.* In Proceedings of Global Conference on Communication Technologies (GCCT). Thuckalay, India: IEEE Xplore Digital Library, 2015. p. 229-233, 5 pp. ISBN 978-1-4799-8552-4. doi:10.1109/GCCT.2015.7342657 [155].
  [Author's contribution: 80%]

- Ibrahim Ghafir and Vaclav Prenosil. *DNS traffic analysis for malicious domains detection.* In Proceedings of International Conference on Signal Processing and Integrated networks. Noida, India: IEEE Xplore Digital Library, 2015. p. 613 - 618, 6 pp. ISBN 978-1-4799-5990-7. doi:10.1109/SPIN.2015.7095337 [156].
  [Author's contribution: 80%]

- Ibrahim Ghafir and Vaclav Prenosil. *DNS query failure and algorithmically generated domain-flux detection.* In Proceedings of International Conference on Frontiers of Communications, Networks and Applications. Kuala Lumpur, Malaysia: IEEE Xplore Digital Library, 2014. p. 1-5, 5 pp. ISBN 978-1-78561-072-1. doi:10.1049/cp.2014.1410 [157].
  [Author's contribution: 80%]

- Ibrahim Ghafir, Jakub Svoboda and Vaclav Prenosil. *Tor-based malware and Tor connection detection.* In Proceedings of International

Conference on Frontiers of Communications, Networks and Applications. Kuala Lumpur, Malaysia: IEEE Xplore Digital Library, 2014. p. 1-6, 6 pp. ISBN 978-1-78561-072-1. doi:10.1049/cp.2014.1411 [158]. [Author's contribution: 80%]

- Ibrahim Ghafir, Martin Husak and Vaclav Prenosil. *A Survey on Intrusion Detection and Prevention Systems.* In Proceedings of student conference Zvůle 2014, IEEE/UREL. Zvůle, Czech Republic: Faculty of Electrical Engineering and Communication, Brno University of Technology, 2014. p. 10-14, 5 pp. ISBN 978-80-214-5005-9 [39].
  [Author's contribution: 80%]

A.4 Posters

- Ibrahim Ghafir and Vaclav Prenosil. *POSTER: Network-based Advanced Persistent Threat Attack Detection.* In 8th International Conference on Autonomous Infrastructure, Management and Security (AIMS). 2014.
  [Author's contribution: 80%]

- Ibrahim Ghafir and Vaclav Prenosil. *POSTER: DNS Traffic Analysis for Malicious Domains Detection.* In International Conference on Signal Processing and Integrated networks. 2015.
  [Author's contribution: 80%]

Bibliography

[1] Trend Micro white paper. The custom defense against targeted attacks. http://www.trendmicro.fr/media/wp/custom-defense-against-targeted-attacks-whitepaper-en.pdf. Accessed: 10-11-2014.

[2] Maurice de Kunder. The size of the world wide web. http://www.worldwidewebsize.com/. Accessed: 07-12-2016.

[3] M Uma and G Padmavathi. A survey on various cyber attacks and their classification. *IJ Network Security*, 15(5):390–396, 2013.

[4] Robin Gandhi, Anup Sharma, William Mahoney, William Sousan, Qiuming Zhu, and Phillip Laplante. Dimensions of cyber-attacks: Cultural, social, economic, and political. *IEEE Technology and Society Magazine*, 30(1):28–38, 2011.

[5] Stephen Jackson. Nato article 5 and cyber warfare: Nato's ambiguous and outdated procedure for determining when cyber aggression qualifies as an armed attack. *The CIP Report*, 2016.

[6] Steve Morgan. Hackerpocalypse: A cybercrime revelation. *2016 Cybercrime Report, Cybersecurity Ventures*, 2016.

[7] Javed Akhtar Khan and Nitesh Jain. A survey on intrusion detection systems and classification techniques. *International Journal of Scientific Research in Science, Engineering and Technology*, 2016.

[8] Trend Micro technical report. Targeted attacks and how to defend against them. `http://www.trendmicro.co.uk/media/misc/targeted-attacks-and-how-to-defend-against-them-en.pdf`. Accessed: 05-12-2016.

[9] Zhijie Liu, Chongjun Wang, and Shifu Chen. Correlating multi-step attack and constructing attack scenarios based on attack pattern modeling. In *Information Security and Assurance, 2008. ISA 2008. International Conference on*, pages 214–219. IEEE, 2008.

[10] Ibrahim Ghafir, Vaclav Prenosil, Ahmad Alhejailan, and Mohammad Hammoudeh. Social engineering attack strategies and defence approaches. In *Future Internet of Things and Cloud (FiCloud), 2016 IEEE 4th International Conference on*, pages 145–149. IEEE, 2016.

[11] Terry R Rakes, Jason K Deane, and Loren Paul Rees. It security planning under uncertainty for high-impact events. *Omega*, 40(1):79–88, 2012.

[12] Paul Wood, Mathew Nisbet, Gerry Egan, Nicholas Johnston, Kevin Haley, Bhaskar Krishnappa, Tuan-Khanh Tran, Irfan Asrar, Orla Cox, Sean Hittel, et al. Symantec internet security threat report trends for 2011. *Volume XVII*, 2012.

[13] Mandiant Intelligence Center. Apt1: Exposing one of china's cyber espionage units. Technical report, Mandiant, Tech. Rep, 2013.

[14] Kaspersky Lab ZAO. Red october diplomatic cyber attacks investigation. `http://www.securelist.com/en/analysis/204792262/Red_`

`October_Diplomatic_Cyber_Attacks_Investigation`. Accessed: 10-11-2014.

[15] Colin Tankard. Advanced persistent threats and how to monitor and deter them. *Network security*, 2011(8):16–19, 2011.

[16] Ronald Deibert and Rafal Rohozinski. Tracking ghostnet: Investigating a cyber espionage network. *Information Warfare Monitor*, page 6, 2009.

[17] Shun-Te Liu, Yi-Ming Chen, and Shiou-Jing Lin. A novel search engine to uncover potential victims for apt investigations. In *Network and Parallel Computing*, pages 405–416. Springer, 2013.

[18] Olivier Thonnard, Leyla Bilge, Gavin O'Gorman, Seán Kiernan, and Martin Lee. Industrial espionage and targeted attacks: Understanding the characteristics of an escalating threat. In *Research in Attacks, Intrusions, and Defenses*, pages 64–85. Springer, 2012.

[19] Martin Lee and Darren Lewis. Clustering disparate attacks: Mapping the activities of the advanced persistent threat. In *Proceedings of the 21st Virus Bulletin International Conference.(October 2011) pp*, pages 122–127.

[20] Boldizsár Bencsáth, Gábor Pék, Levente Buttyán, and Márk Félegyházi. Duqu: Analysis, detection, and lessons learned. In *ACM European Workshop on System Security (EuroSec)*, volume 2012, 2012.

[21] Marco Balduzzi, Vincenzo Ciangaglini, and Robert McArdle. Targeted attacks detection with spunge. 2013.

[22] Catherine Moxey, Mike Edwards, Opher Etzion, Mamdouh Ibrahim, Sreekanth Iyer, Hubert Lalanne, Mweene Monze, Marc Peters, Yuri

Rabinovich, Guy Sharon, et al. A conceptual model for event processing systems. *IBM Redguide publication*, 2010.

[23] Lars Brenna, Alan Demers, Johannes Gehrke, Mingsheng Hong, Joel Ossher, Biswanath Panda, Mirek Riedewald, Mohit Thatte, and Walker White. Cayuga: a high-performance event processing engine. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1100–1102. ACM, 2007.

[24] Ibrahim Ghafir, Václav Přenosil, et al. Advanced persistent threat attack detection: An overview. *International Journal of Advances in Computer Networks and Its Security (IJCNS)*, (I), 2014.

[25] Marcus Butavicius, Kathryn Parsons, Malcolm Pattinson, and Agata McCormac. Breaching the human firewall: Social engineering in phishing and spear-phishing emails. *arXiv preprint arXiv:1606.00887*, 2016.

[26] Sandeep Yadav, Ashwath Kumar Krishna Reddy, AL Narasimha Reddy, and Supranamaya Ranjan. Detecting algorithmically generated domain-flux attacks with dns traffic analysis. *Networking, IEEE/ACM Transactions on*, 20(5):1663–1677, 2012.

[27] Klaas Apostol. Brute-force attack. 2012.

[28] Bashar Ewaida. Pass-the-hash attacks: Tools and mitigation. *Last accessed September*, 11, 2013.

[29] Chirag Modi, Dhiren Patel, Bhavesh Borisaniya, Hiren Patel, Avi Patel, and Muttukrishnan Rajarajan. A survey of intrusion detection techniques in cloud. *Journal of Network and Computer Applications*, 36(1):42–57, 2013.

[30] Rob Jansen, Florian Tschorsch, Aaron Johnson, and Björn Scheuermann. The sniper attack: Anonymously deanonymizing and disabling the tor network. Technical report, DTIC Document, 2014.

[31] David Mudzingwa and Rajeev Agrawal. A study of methodologies used in intrusion detection and prevention systems (idps). In *Southeastcon, 2012 Proceedings of IEEE*, pages 1–6. IEEE, 2012.

[32] Ahmed Patel, Qais Qassim, and Christopher Wills. A survey of intrusion detection and prevention systems. *Information Management & Computer Security*, 18(4):277–290, 2010.

[33] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24, 2013.

[34] Arun Viswanathan, Kymie Tan, and Clifford Neuman. Deconstructing the assessment of anomaly-based intrusion detectors for critical applications. *RAID 2013, LNCS 8145, pp. 286-306. 2013 Springer-Verlag Berlin Heidelber 2013*, 8145(DOE-JPL & USC-00192-128), 2013.

[35] Manoranjan Pradhan, Sateesh Kumar Pradhan, and Sudhir Kumar Sahu. A survey on detection methods in intrusion detection system. *methods*, 3(2), 2012.

[36] Min Zheng, Mingshen Sun, and John CS Lui. Droid analytics: a signature based analytic system to collect, extract, analyze and associate android malware. In *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 163–171. IEEE, 2013.

[37] Indraneel Mukhopadhyay, Mohuya Chakraborty, and Satyajit Chakrabarti. A comparative study of related technologies of intrusion detection & prevention systems. *J. Information Security*, 2(1): 28–38, 2011.

[38] Karen Scarfone and Peter Mell. Guide to intrusion detection and prevention systems (idps). *NIST Special Publication*, 800(2007):94, 2007.

[39] Ibrahim Ghafir, Martin Husák, and Václav Přenosil. A survey on intrusion detection and prevention systems. 2014.

[40] Ross Heenan and Naghmeh Moradpoor. A survey of intrusion detection system technologies. *The First Post Graduate Cyber Security Symposium*, pages 1–5, 2016.

[41] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014.

[42] Taiwo Oladipupo Ayodele. *Machine learning overview*. INTECH Open Access Publisher, 2010.

[43] Farzan Farnia and David Tse. A minimax approach to supervised learning. *arXiv preprint arXiv:1606.02206*, 2016.

[44] Gao Huang, Shiji Song, Jatinder ND Gupta, and Cheng Wu. Semi-supervised and unsupervised extreme learning machines. *IEEE transactions on cybernetics*, 44(12):2405–2417, 2014.

[45] Yves Kodratoff. *Introduction to machine learning*. Morgan Kaufmann, 2014.

[46] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

[47] Stephen Marsland. *Machine learning: an algorithmic perspective*. CRC press, 2015.

[48] A Attribute. Decision tree learning. 2015.

[49] Sotiris B Kotsiantis. Decision trees: a recent overview. *Artificial Intelligence Review*, 39(4):261–283, 2013.

[50] Shan Suthaharan. Decision tree learning. In *Machine Learning Models and Algorithms for Big Data Classification*, pages 237–269. Springer, 2016.

[51] Yan Zhang and JingTao Yao. Gini objective functions for three-way classifications. *International Journal of Approximate Reasoning*, 2016.

[52] Helmut Schmid. Probabilistic part-ofispeech tagging using decision trees. In *New methods in language processing*, page 154. Routledge, 2013.

[53] Joao Seco, Frank Verhaegen, and Matthias Fippel. Variance reduction techniques. In *Monte Carlo Techniques in Radiation Therapy*, pages 29–40. Taylor & Francis, 2013.

[54] Wei Peng, Juhua Chen, and Haiping Zhou. An implementation of id3—decision tree learning algorithm. *From web. arch. usyd. edu. au/wpeng/DecisionTree2. pdf Retrieved date: May*, 13, 2009.

[55] David Meyer and FH Technikum Wien. Support vector machines. *The Interface to libsvm in package e1071*, 2015.

[56] Fergal Lane, R Azad, and Conor Ryan. Principled evolutionary algorithm design and the kernel trick. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, pages 149–150. ACM, 2016.

[57] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. Support vector machines. In *An Introduction to Statistical Learning*, pages 337–372. Springer, 2013.

[58] Jose Luis Gomez Ortega. *Occupant Behaviour Pattern Modelling and Detection for Energy Eciency and Comfort Management in Buildings Based on Environmental Sensing.* PhD thesis, Manchester Metropolitan University, 2017.

[59] Richard A Berk. Support vector machines. In *Statistical Learning from a Regression Perspective*, pages 291–310. Springer, 2016.

[60] Oliver Sutton. Introduction to k nearest neighbour classification and condensed nearest neighbour data reduction. *University lectures, University of Leicester*, 2012.

[61] David Taniar and Wenny Rahayu. A taxonomy for nearest neighbour queries in spatial databases. *Journal of Computer and System Sciences*, 79(7):1017–1039, 2013.

[62] Jeffrey Strickland. *Predictive Analytics Using R.* Lulu. com, 2015.

[63] Ivan Dokmanic, Reza Parhizkar, Juri Ranieri, and Martin Vetterli. Euclidean distance matrices: essential theory, algorithms, and applications. *IEEE Signal Processing Magazine*, 32(6):12–30, 2015.

[64] Mohammad Norouzi, David J Fleet, and Ruslan R Salakhutdinov. Hamming distance metric learning. In *Advances in neural information processing systems*, pages 1061–1069, 2012.

[65] Xingpeng Jiang, Xiaohua Hu, and Tingting He. Identification of the clustering structure in microbiome data by density clustering on the manhattan distance. *Science China Information Sciences*, 59(7): 070104, 2016.

[66] Binbin Lu, Martin Charlton, Chris Brunsdon, and Paul Harris. The minkowski approach for choosing the distance metric in geographically weighted regression. *International Journal of Geographical Information Science*, 30(2):351–368, 2016.

[67] Zhi-Hua Zhou. Ensemble learning. *Encyclopedia of Biometrics*, pages 411–416, 2015.

[68] Gavin Brown. Ensemble learning. In *Encyclopedia of Machine Learning*, pages 312–320. Springer, 2011.

[69] Shanchieh J Yang, Adam Stotz, Jared Holsopple, Moises Sudit, and Michael Kuhl. High level information fusion for tracking and projection of multistage cyber attacks. *Information Fusion*, 10(1):107–121, 2009.

[70] Guillaume Brogi and Valérie Viet Triem Tong. Terminaptor: Highlighting advanced persistent threats through information flow tracking. In *New Technologies, Mobility and Security (NTMS), 2016 8th IFIP International Conference on*, pages 1–5. IEEE, 2016.

[71] Xu Wang, Kangfeng Zheng, Xinxin Niu, Bin Wu, and Chunhua Wu. Detection of command and control in advanced persistent threat based on independent access. In *Communications (ICC), 2016 IEEE International Conference on*, pages 1–6. IEEE, 2016.

[72] J Vijaya Chandra, Narasimham Challa, and Sai Kiran Pasupuleti. A practical approach to e-mail spam filters to protect data from advanced persistent threat. In *Circuit, Power and Computing Technologies (ICCPCT), 2016 International Conference on*, pages 1–5. IEEE, 2016.

[73] Joseph Sexton, Curtis Storlie, and Joshua Neil. Attack chain detection. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 8(5-6):353–363, 2015.

[74] Nir Nissim, Aviad Cohen, Chanan Glezer, and Yuval Elovici. Detection of malicious pdf files and directions for enhancements: a state-of-the art survey. *Computers & Security*, 48:246–266, 2015.

[75] Saranya Chandran, P Hrudya, and Prabaharan Poornachandran. An efficient classification model for detecting advanced persistent threat. In *Advances in Computing, Communications and Informatics (ICACCI), 2015 International Conference on*, pages 2001–2009. IEEE, 2015.

[76] Johan Sigholm and Martin Bang. Towards offensive cyber counterintelligence: Adopting a target-centric view on advanced persistent threats. In *Intelligence and Security Informatics Conference (EISIC), 2013 European*, pages 166–171. IEEE, 2013.

[77] Paul Giura and Wei Wang. A context-based detection framework for advanced persistent threats. In *Cyber Security (CyberSecurity), 2012 International Conference on*, pages 69–74. IEEE, 2012.

[78] Esraa Alomari, Selvakumar Manickam, BB Gupta, Mohammed Anbar, Redhwan MA Saad, and Samer Alsaleem. A survey of botnet-based ddos flooding attacks of application layer: Detection and mitigation

approaches. In *Handbook of Research on Modern Cryptographic Solutions for Computer and Cyber Security*, pages 52–79. IGI Global, 2016.

[79] Sadegh Momeni Milajerdi and Mehdi Kharrazi. A composite-metric based path selection technique for the tor anonymity network. *Journal of Systems and Software*, 103:53–61, 2015.

[80] FireEye. Mandiant apt1 report. `https://www.fireeye.com/content/dam/fireeye-www/services/pdfs/mandiant-apt1-report.pdf`. Accessed: 26-06-2016.

[81] Abuse.ch. Spyeye domain blocklist. `https://spyeyetracker.abuse.ch/blocklist.php?download=domainblocklist`, . Accessed: 10-11-2014.

[82] Abuse.ch. Palevo domain blocklist. `https://palevotracker.abuse.ch/blocklists.php?download=domainblocklist`, . Accessed: 10-11-2014.

[83] Abuse.ch. Zeus domain blocklist. `https://zeustracker.abuse.ch/blocklist.php?download=domainblocklist`, . Accessed: 10-11-2014.

[84] Malware domain list. `http://www.malwaredomainlist.com/hostslist/hosts.txt`, . Accessed: 10-11-2014.

[85] Blade defender. `http://www.blade-defender.org/eval-lab/blade.csv`, . Accessed: 10-11-2014.

[86] Malware domains. `http://www.malware-domains.com/files/`, . Accessed: 10-11-2014.

[87] Abuse.ch. Spyeye ip blocklist. `https://spyeyetracker.abuse.ch/blocklist.php?download=ipblocklist`, . Accessed: 10-11-2014.

[88] Abuse.ch. Palevo c&c ip blocklist. `https://palevotracker.abuse.ch/blocklists.php?download=ipblocklist`, . Accessed: 10-11-2014.

[89] Abuse.ch. Zeus ip blocklist. `https://zeustracker.abuse.ch/blocklist.php?download=ipblocklist`, . Accessed: 10-11-2014.

[90] Malware domain list. `http://www.malwaredomainlist.com/hostslist/ip.txt`, . Accessed: 10-11-2014.

[91] Abuse.ch. Ssl blacklist a new weapon to fight malware and botnet. `http://securityaffairs.co/wordpress/26672/cyber-crime/ssl-blacklist-new-weapon-fight-malware-botnet.html`, . Accessed: 10-11-2014.

[92] Mandiant. Mandiant apt1 report appendix f update: Ssl certificate hashes. `https://www.mandiant.com/blog/md5-sha1/`. Accessed: 10-11-2014.

[93] Brett Stone-Gross, Marco Cova, Bob Gilbert, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Analysis of a botnet takeover. *Security & Privacy, IEEE*, 9(1):64–72, 2011.

[94] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 635–647. ACM, 2009.

[95] Charles Edge and Daniel O'Donnell. Network scanning, intrusion detection, and intrusion prevention tools. In *Enterprise Mac Security*, pages 441–457. Springer, 2016.

[96] Stephen Doswell, Nauman Aslam, David Kendall, and Graham Sexton. Please slow down!: the impact on tor performance from mobility. In *Proceedings of the Third ACM workshop on Security and privacy in smartphones & mobile devices*, pages 87–92. ACM, 2013.

[97] Sambuddho Chakravarty, Georgios Portokalidis, Michalis Polychronakis, and Angelos D Keromytis. Detection and analysis of eavesdropping in anonymous communication networks. *International Journal of Information Security*, pages 1–16, 2014.

[98] A Kapadia. Analysis of the tor browser and its security vulnerabilities. 2014.

[99] Rolf Jagerman, Wendo Sabée, Laurens Versluis, Martijn de Vos, and Johan Pouwelse. The fifteen year struggle of decentralizing privacy-enhancing technology. *arXiv preprint arXiv:1404.4818*, 2014.

[100] Joseph B. Kowalski. Tor network status. `http://torstatus.blutmagie.de/`. Accessed: 07-04-2015.

[101] Vivens Ndatinya, Zhifeng Xiao, Vasudeva Rao Manepalli, Ke Meng, and Yang Xiao. Network forensics analysis using wireshark. *International Journal of Security and Networks*, 10(2):91–106, 2015.

[102] Felix Fuentes and Dulal C Kar. Ethereal vs. tcpdump: a comparative study on packet sniffing tools for educational purpose. *Journal of Computing Sciences in Colleges*, 20(4):169–176, 2005.

[103] Hao Li, Guangjie Liu, Weiwei Jiang, and Yuewei Dai. Designing snort rules to detect abnormal dnp3 network data. In *Control, Automation and Information Sciences (ICCAIS), 2015 International Conference on*, pages 343–348. IEEE, 2015.

[104] Joshua S White, Thomas Fitzsimmons, and Jeanna N Matthews. Quantitative analysis of intrusion detection systems: Snort and suricata. In *SPIE Defense, Security, and Sensing*, pages 875704–875704. International Society for Optics and Photonics, 2013.

[105] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer networks*, 31(23):2435–2463, 1999.

[106] The-Bro-Project. The bro network security monitor. `https://www.bro.org/`, . Accessed: 15-02-2015.

[107] Bro-Project. Intellegence framework. `https://www.bro.org/sphinx/frameworks/intel.html`, . Accessed: 15-02-2015.

[108] Bro Project. Input framework. `https://www.bro.org/sphinx/frameworks/input.html`, . Accessed: 01-06-2016.

[109] Best-Practical-Solutions. Rt: Request tracker. `https://www.bestpractical.com/rt/`. Accessed: 15-02-2015.

[110] Bro Project. file_over_new_connection event. `https://www.bro.org/sphinx/scripts/base/bif/event.bif.bro.html#id-file_new`, . Accessed: 01-06-2016.

[111] Ibrahim Ghafir, Mohammad Hammoudeh, and Vaclav Prenosil. Disguised executable files in spear-phishing emails: Detecting the point of entry in advanced persistent threat, 2017. `https://doi.org/10.7287/peerj.preprints.2998v1`.

[112] MrForms. Mime types list. `http://www.freeformatter.com/mime-types-list.html`. Accessed: 07-04-2015.

[113] Mcafee-report. Protecting against fileless malware. https://www.mcafee.com/uk/resources/solution-briefs/sb-quarterly-threats-nov-2015-1.pdf. Accessed: 05-02-2017.

[114] Bro Project. file_new event. https://www.bro.org/sphinx/scripts/base/bif/event.bif.bro.html#id-file_over_new_connection, . Accessed: 01-06-2016.

[115] Sophos-Ltd. How large is a piece of malware. https://nakedsecurity.sophos.com/2010/07/27/large-piece-malware/. Accessed: 15-03-2016.

[116] Bro Project. Intel::match event. https://www.bro.org/sphinx-git/scripts/base/frameworks/intel/main.bro.html#id-Intel::match, . Accessed: 01-06-2016.

[117] Bro Project. new_connection event. https://www.bro.org/sphinx/scripts/base/bif/event.bif.bro.html#id-new_connection, . Accessed: 01-06-2016.

[118] Bro Project. x509_certificate event. https://www.bro.org/sphinx/scripts/base/bif/plugins/Bro_X509.events.bif.bro.html#id-x509_certificate, . Accessed: 01-06-2016.

[119] Bro Project. dns_message event. https://www.bro.org/sphinx/scripts/base/bif/plugins/Bro_DNS.events.bif.bro.html#id-dns_message, . Accessed: 01-06-2016.

[120] Miranda Mowbray and Josiah Hagen. Finding domain-generation algorithms by looking at length distribution. In *2014 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 395–400. IEEE, 2014.

[121] Bro Project. Tcp scan detection. https://www.bro.org/sphinx/scripts/policy/misc/scan.bro.html, . Accessed: 12-01-2016.

[122] The-Bro-Project. scan.bro script. https://www.bro.org/sphinx/_downloads/scan.bro, . Accessed: 12-01-2016.

[123] Bro-Project. connection_attempt event. https://www.bro.org/sphinx/scripts/base/bif/plugins/Bro_TCP.events.bif.bro.html#id-connection_attempt, . Accessed: 15-01-2017.

[124] Bro-Project. connection_rejected event. https://www.bro.org/sphinx/scripts/base/bif/plugins/Bro_TCP.events.bif.bro.html#id-connection_rejected, . Accessed: 15-01-2017.

[125] Bro-Project. connection_reset event. https://www.bro.org/sphinx/scripts/base/bif/plugins/Bro_TCP.events.bif.bro.html#id-connection_reset, . Accessed: 15-01-2017.

[126] Bro-Project. connection_pending event. https://www.bro.org/sphinx/scripts/base/bif/plugins/Bro_TCP.events.bif.bro.html#id-connection_pending, . Accessed: 15-01-2017.

[127] Bro Project. connection_established event. https://www.bro.org/sphinx/scripts/base/bif/plugins/Bro_TCP.events.bif.bro.html#id-connection_established, . Accessed: 01-06-2016.

[128] Hidden-Wiki. List of anonymous networks. https://zqktlwi4fecvo6ri.onion.to/wiki/List_of_Anonymous_Networks. Accessed: 17-02-2017.

[129] Python-Software-Foundation. Python. https://www.python.org/, . Accessed: 12-01-2016.

[130] The MathWorks. Matlab. `http://uk.mathworks.com/products/matlab/`. Accessed: 01-06-2016.

[131] Python-Software-Foundation. socket — low-level networking interface. `https://docs.python.org/3/library/socket.html`, . Accessed: 15-03-2016.

[132] Python-Software-Foundation. struct — interpret strings as packed binary data. `https://docs.python.org/2/library/struct.html`, . Accessed: 15-03-2016.

[133] ME Elhamahmy, Hesham N Elmahdy, and Imane A Saroit. A new approach for evaluating intrusion detection system. *CiiT International Journal of Artificial Intelligent Systems and Machine Learning*, 2(11), 2010.

[134] Gulshan Kumar. Evaluation metrics for intrusion detection systems-a study. *International Journal of Computer Science and Mobile Applications, II*, 11, 2014.

[135] Lambert Schaelicke, Thomas Slabach, Branden Moore, and Curt Freeland. Characterizing the performance of network intrusion detection sensors. In *International Workshop on Recent Advances in Intrusion Detection*, pages 155–172. Springer, 2003.

[136] Network-Traffic-Analysis. Nuclear ek delivers digitally-signed cryptowall malware. `http://malware-traffic-analysis.net/2014/09/29/index.html`. Accessed: 15-02-2015.

[137] Malware Traffic Analysis. Three exploit domains and a ransomware infection. `http://malware-traffic-analysis.net/2013/12/23/index.html`. Accessed: 1-11-2014.

[138] Malware-Traffic-Analysis. 2014-10-28 - asprox botnet serving free pizza. http://malware-traffic-analysis.net/2014/10/28/index.html. Accessed: 29-11-2014.

[139] 2014-09-11 - aspros botnet phishing campaign. http://malware-traffic-analysis.net/2014/09/11/index2.html, . Accessed: 29-11-2014.

[140] 2014-10-28 - asprox botnet serving free pizza. http://malware-traffic-analysis.net/2014/09/29/index.html, . Accessed: 29-11-2014.

[141] Oracle. Network tracing. https://www.virtualbox.org/wiki/Network_tips. Accessed: 07-04-2015.

[142] ATG-CTU-University. The malware capture facility project. http://mcfp.weebly.com/. Accessed: 01-09-2014.

[143] NETRESEC. Detecting tor communication in network traffic. http://www.netresec.com/?page=Blog&month=2013-04&post=Detecting-TOR-Communication-in-Network-Traffic. Accessed: 07-04-2015.

[144] The-MathWorks. Classification learner. http://www.mathworks.com/help/stats/classificationlearner-app.html, . Accessed: 15-03-2016.

[145] The-MathWorks. Characteristics of classifier types. https://uk.mathworks.com/help/stats/choose-a-classifier.html#bunt0ky, . Accessed: 15-03-2016.

[146] The-MathWorks. Assess and improve predictive performance of models. `http://www.mathworks.com/discovery/cross-validation.html`, . Accessed: 15-03-2016.

[147] Ibrahim Ghafir, Vaclav Prenosil, and Mohammad Hammoudeh. Botnet command and control traffic detection challenges: A correlation-based solution. *International Journal of Advances in Computer Networks and Its Security (IJCNS)*, vol. 7(Issue 2):27–31, 2015.

[148] Ibrahim Ghafir, Jakub Svoboda, and Vaclav Prenosil. A survey on botnet command and control traffic detection. *International Journal of Advances in Computer Networks and its security (ICJNS)*, vol. 5 (Issue 2):75–80, 2015.

[149] Jakub Svoboda, Ibrahim Ghafir, and Vaclav Prenosil. Network monitoring approaches: An overview. *International Journal of Advances in Computer Networks and Its Security (IJCNS)*, vol. 5(Issue I), 2015.

[150] Ibrahim Ghafir and Vaclav Prenosil. Malicious file hash detection and drive-by download attacks. In *Proceedings of the Second International Conference on Computer and Communication Technologies*, pages 661–669. Springer, 2016.

[151] Ibrahim Ghafir and Vaclav Prenosil. Proposed approach for targeted attacks detection. In *Advanced Computer and Communication Engineering Technology*, pages 73–80. Springer, 2016.

[152] Ibrahim Ghafir, Mohammad Hammoudeh, and Vaclav Prenosil. Malicious ssl certificate detection: A step towards advanced persistent threat defence. In *Proceedings of International Conference on Future Networks and Distributed Systems*. ACM Digital Library, Cambridge, United Kingdom, 2017.

[153] Ibrahim Ghafir, Vaclav Prenosil, Jakub Svoboda, and Mohammad Hammoudeh. A survey on network security monitoring systems. In *IEEE International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, pages 77–82. IEEE Xplore Digital Library, 2016.

[154] Ibrahim Ghafir and Vaclav Prenosil. Advanced persistent threat and spear phishing emails. In *Proceedings of International Conference on Distance Learning, Simulation and Communication*, pages 34–41. University of Defence, 2015.

[155] Ibrahim Ghafir and Vaclav Prenosil. Blacklist-based malicious ip traffic detection. In *Global Conference on Communication Technologies (GCCT)*, pages 229–233. IEEE Xplore Digital Library, 2015.

[156] Ibrahim Ghafir and Vaclav Prenosil. Dns traffic analysis for malicious domains detection. In *2nd International Conference on Signal Processing and Integrated Networks (SPIN)*, pages 613–918. IEEE Xplore Digital Library, 2015.

[157] Ibrahim Ghafir and Vaclav Prenosil. Dns query failure and algorithmically generated domain-flux detection. In *International Conference on Frontiers of Communications, Networks and Applications (ICFCNA)*, pages 1–5. IEEE Xplore Digital Library, 2014.

[158] Ibrahim Ghafir, Jakub Svoboda, and Vaclav Prenosil. Tor-based malware and tor connection detection. In *International Conference on Frontiers of Communications, Networks and Applications (ICFCNA)*, pages 1–6. IEEE Xplore Digital Library, 2014.