

Design, analysis and presentation of Intrusion Detection Systems.

by

SFAKIANAKIS IOANNIS

A THESIS

Submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

DEPARTMENT OF INFORMATICS ENGINEERING

SCHOOL OF APPLIED TECHNOLOGY

TECHNOLOGICAL EDUCATIONAL INSTITUTE OF CRETE

June 2015

Approved by:

Supervisor
Ass. Prof. Harry Manifavas

Abstract

It is very significant to maintain a high level security to ensure safe and trusted communication of information exchanged between various organizations. Computer security has become a major problem in our society. But secured transfer packets over internet and any other network are always under threats caused by intrusions and misuses. Intrusion Detection System has become a useful component in terms of computer and network security.

Security threats have become more sophisticated and are now able to pass basic security solutions such as firewalls and antivirus programs. Further protection is therefore required to enhance the overall security of the network.

This thesis thus acts as documentation for setting up an Intrusion Detection System evaluation tests. We explore IDS models that are being used for detecting attacks. We analyze two main types of IDS: the Network IDS and Host-Based IDS. The main goal of this thesis is to point how typical intrusion attacks might be detected.

Σύνοψη

Τη σημερινή εποχή είναι πολύ σημαντικό να διατηρείται ένα υψηλό επίπεδο ασφαλείας το οποίο να εξασφαλίζει την ασφάλεια και την αξιοπιστία των υπολογιστικών και δικτυακών συστημάτων. Ακόμα είναι πολύ σημαντικό να διατηρηθεί ένα υψηλό επίπεδο αξιοπιστίας στην επικοινωνία. Η μεταφορά δεδομένων μέσω τηλεπικοινωνιακών συστημάτων δημιουργεί προβλήματα καθώς αυτά καθίστανται ευπρόσβλητα σε κακόβουλες ενέργειες. Η ασφάλεια των υπολογιστικών συστημάτων και των επικοινωνιών έχει εξελιχθεί σε ένα μείζον πρόβλημα.

Μια λύση στα παραπάνω προβλήματα είναι τα συστήματα ανίχνευσης εισβολών (IDS). Τα συστήματα αυτά αποτελούν ένα χρήσιμο εργαλείο και ένα επιπλέον επίπεδο για την ασφάλεια των υπολογιστικών και δικτυακών συστημάτων.

Οι απειλές για την ασφάλεια έχουν γίνει πιο περίπλοκες και είναι πλέον σε θέση να περάσουν τις βασικές λύσεις ασφαλείας όπως τα firewalls και τα antivirus. Ως εκ τούτου, μια πρόσθετη λύση ασφαλείας απαιτείται για να ενισχυθεί η συνολική ασφάλεια του δικτύου. Μια πιθανή λύση για την ενίσχυση της ασφαλείας είναι να προσθέσουμε ένα σύστημα ανίχνευσης εισβολής (IDS) ως ένα επιπλέον επίπεδο των λύσεων ασφαλείας.

Στην παρούσα πτυχιακή γίνεται μελέτη, ανάλυση και παρουσίαση των IDS. Τα συστήματα ανίχνευσης εισβολών που χρησιμοποιήθηκαν είναι τα Snort, Suricata, Bro. Πιο συγκεκριμένα, η μελέτη και ο σχεδιασμός του συστήματος που αναπτύχθηκε αφορά δύο κατηγορίες των IDS τα HIDS και NIDS.

Acknowledgements

I would like to express my deep gratitude to Professor Harrys Manifavas, my supervisor, for his patient guidance, enthusiastic encouragement and useful critiques on this work. His willingness to give his time so generously has been very much appreciated.

Last but not least, I would like to thank my wife, Eleni, for her patience and support she has shown during the past one year it has taken me to finalize this thesis.

Contents

- Abstract 3
- Acknowledgement..... 5
- List of Tables 8
- List of Figures 9
- Chapter 1: Introduction 12
 - 1.1 Introduction 12
 - 1.2 Motivation 13
 - 1.3 Growth of the Internet 13
 - 1.4 Growth of Internet attacks 14
 - 1.5 History of Intrusion Detection Systems 15
 - 1.6 Financial risks 15
 - 1.7 Why use IDS?..... 15
 - 1.8 Limitations of Intrusion Detection System 16
- Chapter 2: Intrusion Detection Systems 17
 - 2.1 Introduction 17
 - 2.2 Free Intrusion Detection Systems 18
 - 2.3 Problems with Existing Systems 19
 - 2.4 Process model for Intrusion Detection 19
 - 2.5 Effectiveness of IDS..... 20
 - 2.6 Network Intrusion Detection Systems..... 20
 - 2.7 Host Intrusion Detection Systems 22
 - 2.8 IDS Analysis..... 23
 - 2.8.1 Misuse Detection..... 24
 - 2.8.2Anomaly Detection 24
 - 2.8.3 Specification Detection 26
 - 2.8.4 Hybrid..... 26
 - 2.9 IDS Architecture..... 26
- Chapter 3: System Model 27
 - 3.1 Overview of the proposed system 27
 - 3.2 Port mirroring 27
 - 3.2.1 Configuration set up switch..... 29

Chapter 4: Snort	31
4.1 Introduction	31
4.2 Snort Features.....	32
4.3 Architecture of Snort	32
4.3.1 Packet Decoder.....	33
4.3.2 The Preprocessors.....	33
4.3.3 Detection Engine	33
4.3.4 Snort Alerts.....	34
4.3.5 Snort Packet Data	34
4.4 Three modes of Snort	34
4.5 Snort Rules	34
4.5.1 Rules Headers.....	35
4.5.2 IP Addresses	35
4.5.3 Activate/Dynamic Rules.....	35
4.5.4 General Rule Options	36
4.6 The Snort Configuration File	38
4.7 Snort IDS mode	40
4.7.1 Test ids (ping).....	41
4.7.1.1 Rule Ping.....	42
4.8 Port Scan Detection.....	42
4.8.1 Rule Scan Fin	43
4.9 Detect SYN flood	43
4.9.1 Rule SYN Flood.....	45
4.10 Detect brute-force ftp	45
4.10.1 Rule brute-force	46
4.11 Detect UDP Flood	46
4.11.1 Rule UDP Flood	48
4.12 Detect brute-force SSH	48
4.12.1 Rule brute-force SSH	48
Chapter 5: Suricata	49
5.1 Suricata ids	49
5.1.1 Suricata configuration files	51
5.1.2 Max-pending-packets	51

5.1.3	Default-packet-size.....	51
5.1.4	Action-order	51
5.1.5	Detection engine.....	52
5.1.5.1	Inspection configuration.....	52
5.2	Suricata ids mode	54
5.2.1	Test Suricata ids.....	55
5.2.1.1	Rule Ping.....	56
5.3	Detection Port Scan.....	57
5.3.1	Rule Port Scan.....	57
5.4	Detection SYN flood.....	57
5.4.1	Rule SYN Flood.....	58
5.5	Detection UDP flood.....	58
5.5.1	Rule UDP Flood.....	59
5.6	Suricata vs Snort.....	59
Chapter 6:	Bro	61
6.1	Bro IDS	61
6.1.1	Managing Bro with Bro control	61
6.1.2	Browsing Log Files	61
6.2	Bro Scripts.....	61
6.3	Bro Log Files.....	63
6.3.1	Signature main.bro	65
6.3.2	Reporter main.bro.....	66
6.3.3	Communication main.bro.....	67
6.4	Detect Port scan.....	68
6.4.1	Scan.bro.....	71
6.5	Detect SYN Flood.....	75
6.6	Detect UDP flood.....	77
6.7	Detect Brute Force SSH.....	79
6.7.1	Brute-Force.log	81
6.8	Bro vs Snort.....	83
Chapter 7:	Conclusion.....	84
7.1	Future Work.....	84

List of Tables

Table 1-1: Snort Metadata Keys	37
Table 1-2: General rule option keywords	37
Table 5-1: Global Overview	60

List of figures

Figure 1-1:Attack Complexity	13
Figure 1-2: Internet Per 100 Inhabitants.....	14
Figure 1-3:Intrusion Detection System.....	16
Figure 2-1: Intrusion Detection System 2.....	17
Figure 2-2: IPS In Complete Deployment Mode.....	18
Figure 2-3:NIDS In Complete Deployment Mode.....	20
Figure 2-4:NIDS Architecture With Mirror Port	22
Figure 2-5: Host Intrusion Detection System	13
Figure 1-1: Attack Complexity	13
Figure 1-1: Attack Complexity	12

Chapter 1 Introduction

1.1 Introduction

In recent years, a gradually increasing number of intrusion detection systems are being in use. An intrusion detection system is a device or a software application that monitors network or system for malicious activities or policy violations that produces reports [1]. This has been driven by numerous developments, including the growing e-business paradigm, the increasing interconnection. These incidents highlight the increasing need for organizations to protect their networks from attacks. Instead of a firewall that filters bad traffic, an IDS monitored packets to detect malicious attack attempts. The use of secure protocols and the enforcement of security attributes have the potential to prevent disadvantages from being exploited and from having costly consequences [2].

IDSs are host-based, network-based and distributed IDSs [3]. Host based IDS monitors specific host machines, network-based IDS identifies intrusions on key network points and distributed IDS operates both on host as well as network [3].

Considering the damage caused by the attacks [4], it is important to detect attacks and take, if it is feasible, appropriate actions to prevent them. Efficiency of IDS can be measured by its high detection rate and a low false positive rate [5]. IDS can be correlate with fortress defense against any intrusion where as firewall can act as a first line of defense against the attackers.

Firewalls can be by passed through attacks strategies with the help of e-mail based trojan horse viruses by a concealed access through DNS or ICMP protocols [6].

We have two classes of intrusion detection systems [21]: anomaly and misuse detection. Anomaly detection systems attempt to model the usual or acceptable traffic. They have high false positive rates and usually have detection system delay. Misuse detection is an IDS technique that follows defined patterns of attack that exploit disadvantages in the system.

Unfortunately, dependable IDS should continuously provide correct services [7]. Therefore, two factors need to be considered to ensure IDS reliability. First, the IDS should deliver reliable detection results. The IDS method should be effective in discovering intrusions since poor detection performance ruins the trustiness of the IDS [7].

Furthermore, the IDS should be able to work in hostile environments or even under attack [8]. We can study also a determination based IDS that instruments a target application, and uses a scheduler to confirm timing analysis results [9].

Today's networks [10] are not only heterogeneous, but also dynamic. Therefore, intrusion detection systems need to back up mechanisms to dynamically change their configuration as the security state of the protected system evolves. Most intrusion detection systems [10] are initialized with a set of signatures at startup time.

In the end, the ad hoc nature of IDSs [11] platforms does not permit one to dynamically configure a running sensor so that a new packet stream can be used as input for the security analysis.

1.2 Motivation

A big difficulty of the existing intrusion response systems is to build a system (IDS) model [12]. This assigned value for the resources is used in the process of evaluating metrics like intrusion damage cost. Any system topology can be divided into many parts that may be a service. But it becomes a risk in general for the system administrator to assign values for a specific system model. It is not only a difficult job for the system administrators but it may not also be a secure estimate for that system [12].

We have implemented a breakthrough system model that will be a good solution to these problems. We provide a procedural method to detect network attacks.

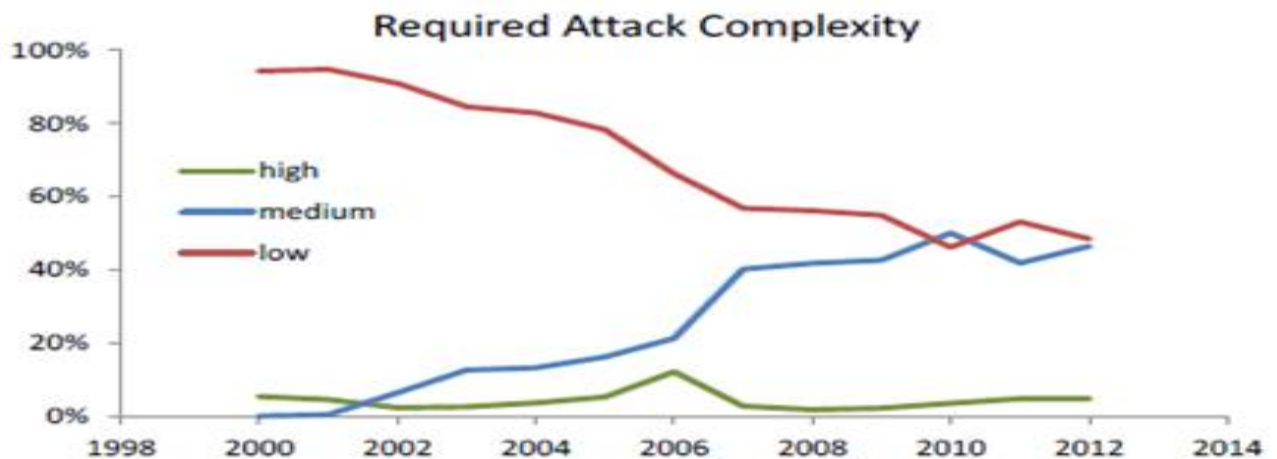


FIGURE 0-1: ATTACK COMPLEXITY

1.3 Growth of the Internet

In a matter for very few years, the internet has been a very powerful platform [13] that has changed the form of communication. It is the universal source of information and has given a globalized dimension to the word.

The Internet continues to evolve [13], driven by ever greater amounts of online information and social networking. The Internet allows greater advantage in working a lot and location, specifically when it comes to high-speed networks. The Internet can be accessed in most cases by using mobile, tablet pc and other Internet devices. Mobile phones, data cards, tablet pc, handheld game machines and routers allow users to connect to the Internet Wi-Fi. Within the constraints imposed by small screens and other limited facilities of such electronic devices, the services of the Internet, including email and the web, may be available [13].

E-commerce is trying to add revenue streams using the Internet to build and enhance relationships with clients and partners [13].

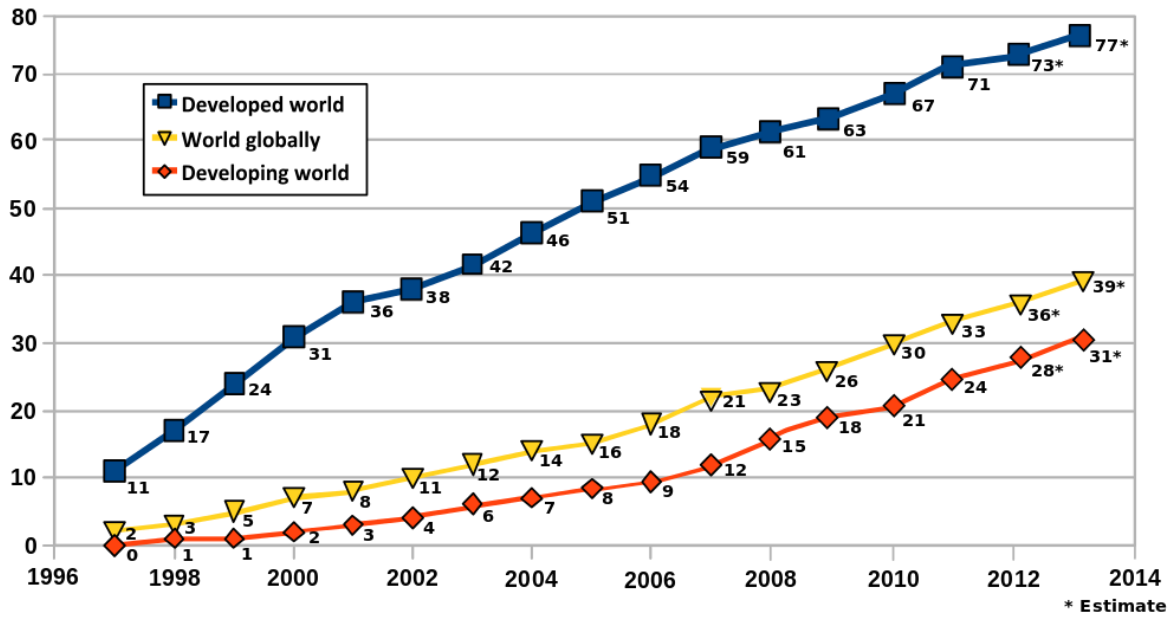


Figure 0-2: Internet Per 100 Inhabitants

1.4 Growth of Internet attacks

The attacks on the net have become both more complicated and easier to implement because of the ubiquity of the Internet and the ease of use operating systems and development environments [13].

There are multiple points for intrusions to take place in a network system. For instance, at the network level carefully created malicious IP packets can crash a victim host at the host level, disadvantages of system software can be exploited to yield an illegal root shell. The security attacks have exploited all kinds of networks ranging from traditional computers to point to point and distributed network systems. There are operating systems that regularly publish updates, but its association of administered machines, uninformed number of targets, and ever-present software bugs has allowed exploits to remain ahead of patches [13].

When attacks occur specifically against infrastructure [13] then important internet resources are being targeting. Malicious exploits are having access to web hosting and name servers, and data centers. This means that forming that seek high-reputation and resource-rich assets. Buffer errors are a top threat, at 21 percent of the Common Weakness Enumeration threat categories.

Wireless Sensor Networks [14] are vulnerable to many types of security attacks due to open wireless medium, decentralized communication and deployment physically non protected areas. In mote-class attacks, the attacker accommodates few of the sensor nodes inside a Wireless Sensor Networks. In laptop-class attacks, the attacker has more powerful devices to launch more intense attack against WSNs.

1.5 History of Intrusion Detection Systems

In the early '90s [15], researchers created real-time intrusion detection systems that reviewed audit data as it was produced.

Currently attacks on computer systems is continuously growing and in 2000 it was a fact [16] that the incident numbers of attacks reported to CERT was parallel to the growth of the internet. The Audit Data Analysis and Mining IDS in 2001 used tcp dump to create profiles of rules for classification.

In 2003, Dr. Yong guang Zhang and Dr. Wenke Lee argue about the importance of IDS in networks with mobile nodes [17, 18].

In our days, Wireless Wide Area Networks are being implemented in organizations. The usefulness of these technologies is only showing the increased need for an organization in implementing an intrusion detection system within their infrastructure [19].

1.6 Financial risks

The threats on the Internet can cause essential losses resulting from business disruption, loss of time and money, and damage the brand name. The cost of application downtime and lost productivity caused by the increasing number of attacks.

The most important cost of network viruses comes from its financial damage to company performance and to national economies. Network virus damages trade, brand name, novelty, and global economic growth.

The costs of network viruses for the world are:

- Will continue to increase as more business operates move online and as more companies and people around the world connect to the Internet.
- Losses from the theft of highbrow property will also increase.

1.7 Why use IDS?

The companies have installed IDS outside of the firewall and routers have done this in order to see the full breadth of attempted attacks against their organization. Intrusion detection allows protecting organization systems against attacks that appear with increasing network connectivity and the interdependency of information systems.

Most companies have developed IDS devices on their network. This means that the IDS exist on a shared media and captures as many traffic packets as it can handle in a mixed mode and reports this data back to a management console. One benefit of using IDS is that avoid problems by dissuading hostile individuals and detect attacks that not prevented by other protection systems.

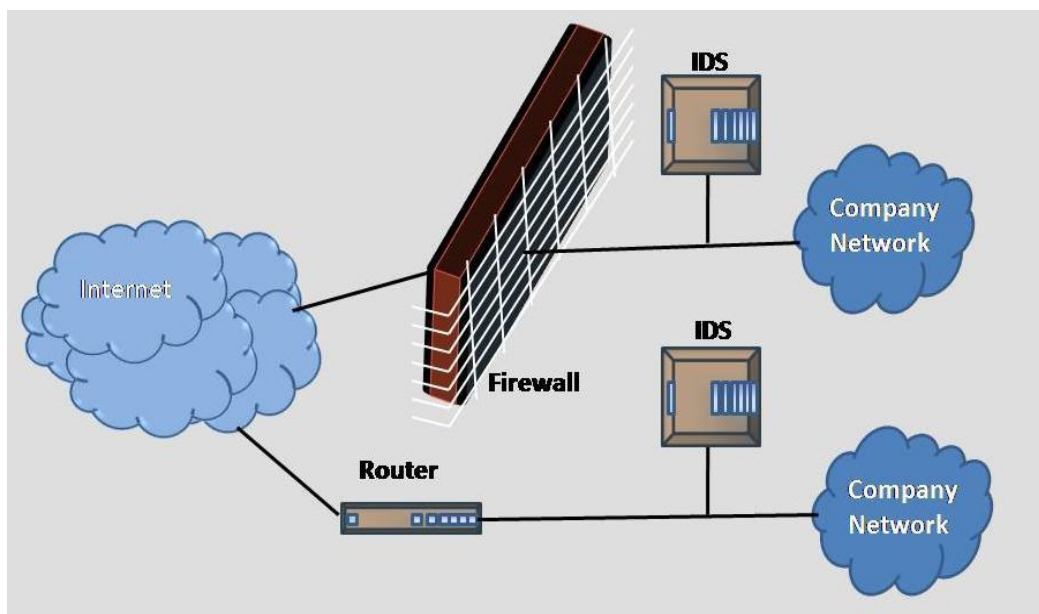


FIGURE 0-3: INTRUSION DETECTION SYSTEM

1.8 Limitations of Intrusion Detection System

- IDS cannot give a unique solution for all security problems.
- False positives [37]: False alarms when there is not real intrusion taking place.
- False negatives [37]: When a real attack or intrusion remains undetected [20].
- Resources [28]: The process of analysis and data logging, especially in real-time, makes intrusion detection systems have important requisites on system resources such as process time or stored space in data bases.
- Defense against new attacks: In most cases, intrusion detection systems cannot detect recently appeared attacks or variants of existing ones. This happens with most commercial products that have detectors based on signature technique, with attack patterns.
- Ciphering: The use of encrypted communication may disable the use of an intrusion detector based on network, because it cannot interpret what it is monitoring.

Chapter 2 Intrusion Detection Systems

2.1 Introduction

Intrusion detection [22] has the ability of checking the events occurring in a computer system or network and analyzes them for signs of reasonable incidents, which are violations attacks

of computer security rules, acceptable usage of rules, or standard security practices. An intrusion detection system is software that automates the intrusion detection process [22]. The goal of IDS is to detect intrusions. Incidents have many causes, such as malware, attackers are getting unauthorized access to systems from the Internet, and authorized users of systems who misuse their privileges or attempt earning additional privileges for which they are not authorized .

An intrusion detection system is a device or software application that observes network or system activities for malicious activities or policy violations and produces reports to a management station [22]. Detection precision and detection stability are two basic indicators to evaluate intrusion detection system [23]. Also, flow-based techniques can be used to detect scans, worms, Botnets and Denial of Service attacks [24]. Security always remains a challenge in Ad Hoc Networks [56] and becomes a goal to detect the activities of those networks.

There are network [22] (NIDS) and host (HIDS) intrusion detection systems. Intrusion detection and prevention systems [1, 22, 25] (IDPS) are mainly in used on identifying possible incidents, logging information about them, and reporting attempts. In addition, organizations use IDPSs for various reasons, such as detecting problems with security policies, documenting existing threats. IDPSs have become necessarily addition to the security infrastructure of nearly every organization [22].

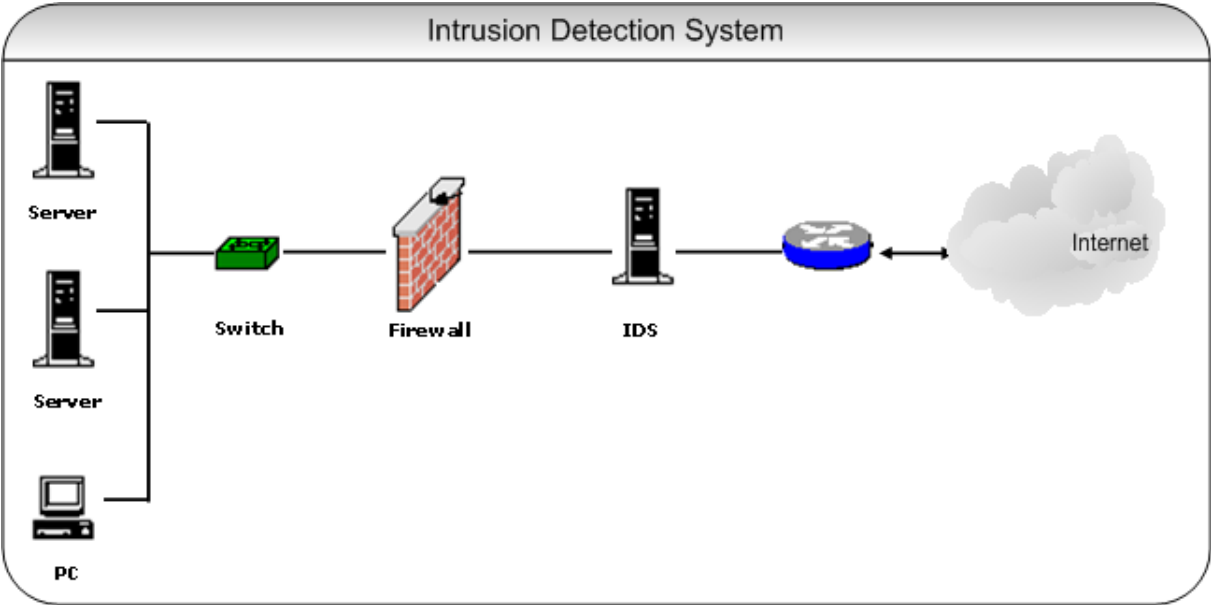


FIGURE 2-1: INTRUSION DETECTION SYSTEM 2

An intrusion prevention system [25] (IPS) has all the functions of an intrusion detection system and can also attempt to prevent possible threats. IDS and IPS technologies offer many of the same functions, and administrators can often disable prevention function in IPS products, causing them to be IDS.

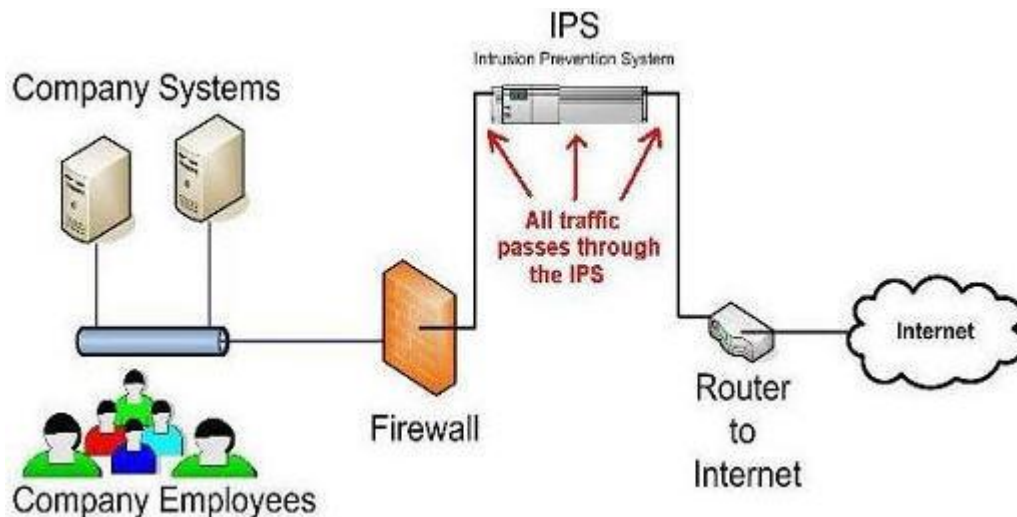


FIGURE 2-2: IPS IN COMPLETE DEPLOYMENT MODE

2.2. Free Intrusion Detection Systems

Snort: An open source and free network intrusion detection and prevention system was created by Martin Roesch in 1998 and now is developed by Sourcefire. In 2009, Snort entered InfoWorld's Open Source Hall of Fame as one of the “greatest open source software of all time” [26, 27].

Ossec: An open source host based intrusion detection system executes log analysis, integrity checking, rootkit detection, time-based alerting and active response [28, 29].

Ossim: The aim of Open Source Security Information Management is to provide an integrated compilation of tools to administrators with a detailed view over each and every aspect of networks, hosts, physical access devices, and servers [29].

Suricata: An open source based intrusion detection system was developed by the Open Information Security Foundation (OISF) [30].

Bro: An open-source, Unix-based network intrusion detection system [31]. Bro detects intrusions by first analyze network traffic to extract its application-level semantics and then performing event-oriented analyzers that compare the activity with patterns deemed troublesome [31].

Base: The Basic Analysis and Security Engine, BASE [28] is a PHP-based analysis engine to search and procedure a database of security events generated by various IDSs, firewalls and network monitoring tools.

Sguil: Sguil is built by network security analysts for network security analysts [29].

Acarm-ng : ACARM-ng [28] is an alert correlation software which can significantly facilitate analyses of traffic in computer networks. It is responsible for collection and analyze of alerts sent by network and host sensors, also referred to as NIDS and HIDS respectively.

2.3 Problems with Existing Systems

Most of intrusion detection systems have at least two of the following disadvantages [32].

- First, the data used by the intrusion detection system is taken from audit trails or from network packets. Data packets have to pass through a longer path from its origin to the IDS but during this an attack may destroy them. Furthermore, the intrusion detection system has to be informed of the functionality of the system from the data collected, which can lead misconception or missed events.
- Second, the intrusion detection system continuously uses additional resources in the system and it is monitoring even when there are no intrusions happening, because the data of the intrusion detection system have to be running all the time. This is the main problem of use resource.
- Third, because the elements of the intrusion detection system are applicable as separate programs, they are not appropriate for changes. An intruder administrator can turn off or modify the programs running on a system, which can make the intrusion detection system useless or unreliable. This is the credibility problem.

2.4 Process model for Intrusion Detection

Most of IDSs can be described in terms of three fundamental functional ingredients [35]:

- Information Sources: The different systems of event information used to define whether an intrusion has taken place. These elements are from different parts of the system, with network, host, and application tracking most common.
- Analysis: The part of intrusion detection systems that decides when intrusions are occurring or have already taken place. The most common analysis approaches are misuse and anomaly detection.
- Response: The set of actions that the system doing in intrusion detection. These actions are pooled into active and passive measures, with measure actions participate in some automated intervention on the part of the system, and passive actions participate in reporting IDS findings to humans, who are then expected to take action based on those reports.

2.5 Effectiveness of IDS

The functionality of an intrusion detection system is the rate at which audit events are processed. Incompleteness happens when the intrusion detection system can't manage to detect an attack [33]. An intrusion detection system has to perform and take its analysis as quickly as possible to enable the security system. An intrusion detection system should itself

be not vulnerable, particularly denial of service, and should be designed with this aim in mind [34].

2.6 Network Intrusion Detection Systems

Network Intrusion Detection Systems (NIDS) are fitted at a basic point within the network to monitor packet traffic on the network [51]. It performs an analysis of passing packets on the entire subnet, works in a promiscuous mode, and checks the flow that is passed on the subnets to the library of known attacks. Once an attack is identified, or threat is detected, the administrator receives an alert. Ideally one would scan all incoming and outgoing traffic, however doing so might make a bottleneck that would damage the overall speed of the network [35].

In [58], they proposed a FPGA based deep packet inspection of NIDS that can support both static and dynamic patterns. Therefore [59], an intrusion detection and security system on virtual machines. For secondary users in Cognitive Radio Networks [60] to quickly detect whether they are being attacked, a simple yet effective IDS is also proposed. In [61], they prevent virtual machines from being compromised in the cloud system with a multi-phase distributed vulnerability detection, measurement and counter measure selection.

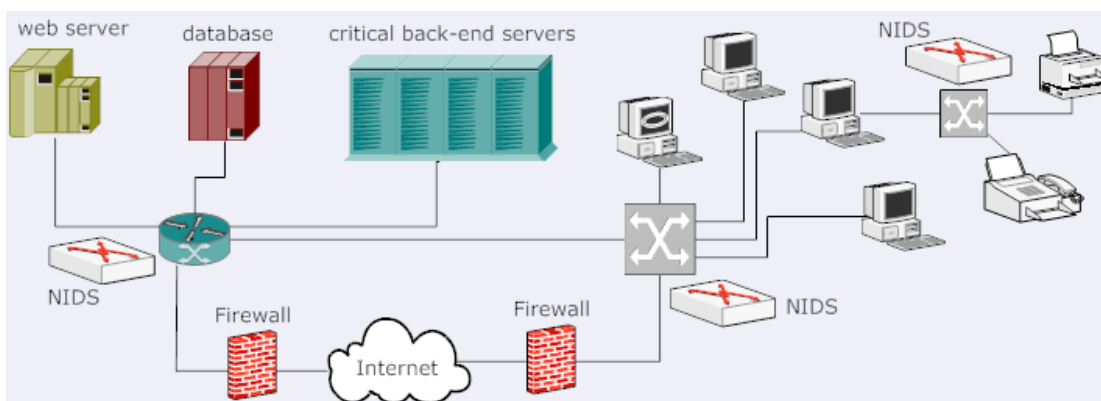


FIGURE 2-3: NIDS IN COMPLETE DEPLOYMENT MODE

A NIDS has the following advantages and disadvantages [35]:

Advantages

- Many well placed network based IDSs can monitor a huge network.
- The NIDSs have a small impact on the network, usually remaining passive and not interfering with normal operations of the latter.

- Network-based IDSs can be made very secure against attack and even made unnoticeable to many attackers.

Disadvantages

- The sensors not only analyze the headers of the packages, they also analyze their content, so they may have difficulties processing all packages in a large network or with much traffic and may fail to recognize attacks during periods of high traffic. Some vendors are trying to solve this problem by implementing IDSs completely in hardware, which makes them much faster.
- The network based IDSs do not analyze the encrypted information. In environments where communication is encrypted it is unfeasible to examine the package contents and therefore unable to evaluate whether this is a package with malicious contents or not. This problem is increased when the organization uses encryption in the network topology, but can be solved with a more relaxed security policy.
- The network-based IDSs do not know whether the attack was successful or not, the only thing known is that it was launched. This means that after a Network IDS detects an attack, administrators must manually explore every host attacked to determine if the attempt was successful or not.
- Some NIDSs have problems dealing with network-based attacks travelling in fragmented packages. These packages make the IDS not notice the attack or be volatile and may even get to fail.
- Due to their general configuration, NIDSs may have a high false acceptance or false positive rate. They may report a lot of normal activities identified as attacks. The problem comes when the number of such alarms is very high.
- Perhaps the biggest disadvantage of NIDSs is their implementation of the stack for network protocols that may differ from the stack of the systems they protect. Many servers and desktop systems do not follow in some aspects the current TCP / IP standards, thus it is possible to have them block packages the NIDS has accepted.

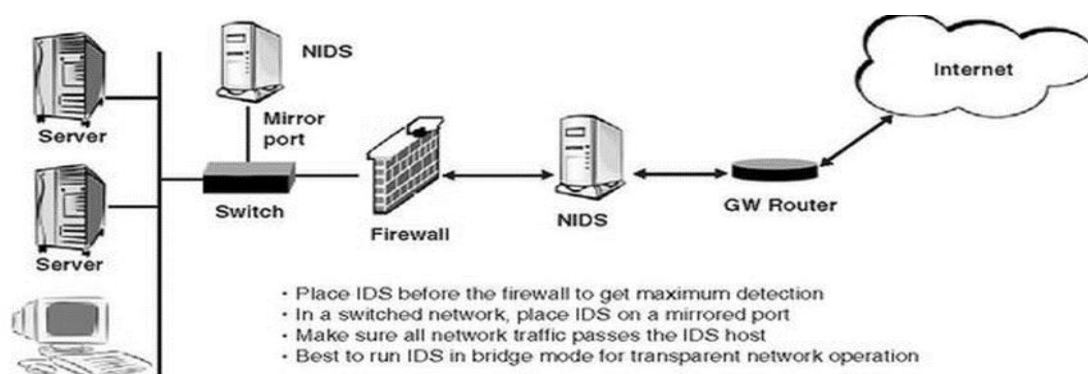


FIGURE 2-4: NIDS ARCHITECTURE WITH MIRROR PORT

2.7 Host Intrusion Detection Systems

Host Intrusion Detection Systems (HIDS) are installed on hosts or devices on the network [55]. A HIDS monitors the inbound and outbound traffic from the device only and will alert the user or the administrator if suspicious traffic is detected. It takes a part of existing system files and matches it to the previous system. If the crucial system files were modified or deleted, an alert is sent to the administrator. An example of HIDS usage can be seen on mission crucial machines, which are not expected to change their conformations [55].

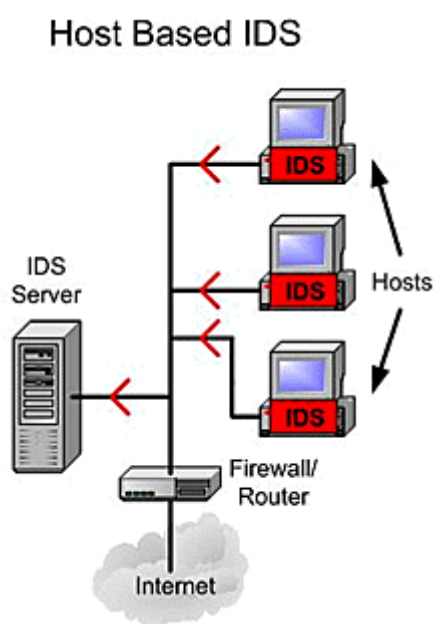


FIGURE 2-5: HOST INTRUSION DETECTION SYSTEM

HIDS has the following advantages and disadvantages [55]:

Advantages

- The HIDS, having the ability to monitor local events of a host, can detect attacks that cannot be seen by NIDS.
- HIDS can often operate in an environment in which network traffic pass encrypted, since the source of information is analyzed before the data is encrypted on the host and after the data is decrypted on the end host.
- HIDSs are uninfluenced by switched networks.
- When HIDSs operate on operating system audit trails, they can help detect attacks that involve software integrity breaches. These appear as inconsistencies in process execution.

Disadvantages

- HIDSs are more costly to administer as they must be managed and configured at each monitored host. While the NIDSs have an IDS for whole monitored systems, HIDSs have an IDS for each of them.

- If the analysis station is within the monitored host, the IDS can be disabled if an attack achieves success on the machine.
- They are not sufficient for detecting attacks on a network since the IDS only analyses those network packets sent to it.
- HIDSs use resources of the host that they are monitoring, influencing its performance.

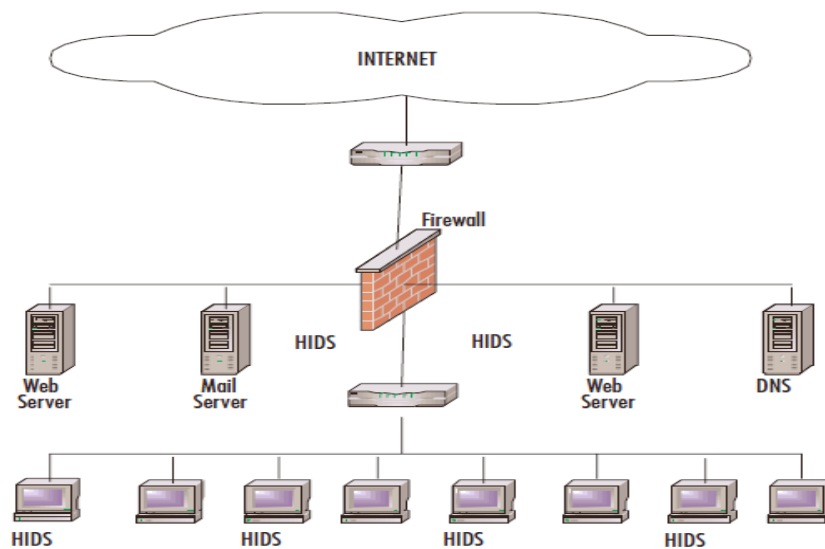


FIGURE 2-6: HOST INTRUSION DETECTION SYSTEM 2

2.8 IDS Analysis

There are two basic features [36, 40] to analyzing events to detect attacks: misuse detection and anomaly detection. Misuse detection, in which the analysis aims something known to be bad, is the technique used by most popular systems. Anomaly detection, in which the analysis searches for unusual patterns of activity, has been, and continues to be, the subject of a great research. Anomaly detection is used in specified form by a number of IDSs. There are advantages and disadvantages with each approach, and it appears that the most effective IDSs use mainly misuse detection methods with a smattering of anomaly detection components.

2.8.1 Misuse Detection

Misuse detectors [39] examine system activity, looking for functions that match a specified pattern which describe a known attack. As the patterns equivalent to known attacks are termed signatures, misuse detection is sometimes termed signature based detection. However, there are more advanced functions to doing misuse detection that can leverage a single signature to detect groups of network and host attacks. Also, Hybrid intrusion detection systems [57] use misuse detection technique.

[Advantages \[39\]:](#)

- Misuse detectors are very efficient at detecting attacks without make the vast number of false alarms.
- Misuse detectors can quickly and reliable detect the use of a specific attack tool or technique. This can help security administrators prioritize corrective measures.
- Misuse detectors can permit system administrators independently of their security level, to detect security problems on their systems, initiating handling procedures.

Disadvantages [39]:

- Misuse detectors can only detect popular and are being updated with signatures of new attacks.
- A lot of misuse detectors are designed to use signatures that prevent them from detecting variations of popular attacks.

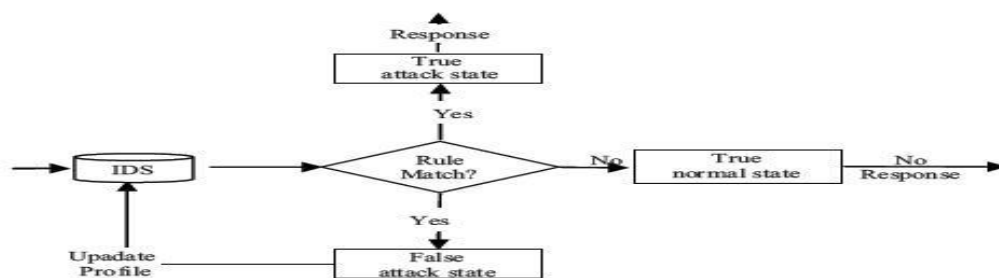


Fig.2: Misuse Detection Model [1]

FIGURE 2-7: MISUSE DETECTION MODEL

2.8.2 Anomaly Detection

Anomaly detectors [38] identify unusual traffic on a host or network. Anomaly intrusion detection identifies differences from the normal usage conduct patterns to identify the intrusion. There are two types of anomaly detection [39]. The first is static anomaly detection, which supposes that the behavior of monitored targets never change, the second type is dynamic anomaly detection. They function on the case that attacks are different from normal activity and can therefore be detected by systems that detect these differences. Anomaly detectors make profiles representing normal behavior of hosts or networks traffic. These profiles are making from data collected over a period of usual function. The detectors then gather event data and use some measurements to determine when monitored activity deviates from the usual.

Anomaly detection uses some measures and techniques, which include:

- Threshold detection, in which some features of user and system behavior are denominated in terms of counts, with some level established as allowable. Such behavior trait can include the number of files accessed by a user in a certain period of time, the number of unsuccessful system logins, the amount of CPU utilized by a process. This level can be static.

- Statistical measures, in which allotment of the profiled features is supposed to fit a specific pattern, and non-parametric, where the distributions of the profiled features are drawn from a set of historical values, observed over time.
- Rule based measurements, which are similar to non-parametric statistical measurements in that observed data defines eligible usage patterns.
- Other measurements, including neural networks, genetic algorithms, and immune system models. Only the first two measures are used in current commercial IDSs.

Unfortunately, in some cases [39] a number of false alarms are produced by anomaly detectors and the IDSs based on them, due to variation of a system behavior or normal patterns of users. Despite this drawback, unlike signature-based IDSs that rely on matching patterns of past attacks, researchers affirm that anomaly-based IDSs are able to detect new attack forms.

Furthermore [39], some misuse detectors may acquire information sources which are produced from certain forms of anomaly detection. For example, a threshold-based anomaly detector can generate a figure representing the number of files accessed by a certain user. The misuse detector can use this figure as an element of a detection signature.

Advantages [39]:

- IDSs using anomaly detection detect unusual traffic and thus have the ability to detect attacks without specific knowledge of details.
- Anomaly detectors can generate elements that can in turn be used to define signatures for misuse detectors.

Disadvantages [39]:

- Anomaly detection usually triggers a large number of false alarms due to the unusual behaviors of users and networks.
- Anomaly detection often requires extensive training sets of system event files in order to characterize normal behavior patterns.

2.8.3 Specification Detection

Specification approaches [33] takes the middle ground between misuse and anomaly detection. The aim is to create a system behavioral determination under the affair that a rightful and well-behaved system will only operate within these confines, and any outside traffic can be considered an intrusion. This is functionally different from anomaly detection as it identifies a list of functions a system may not do, rather than identifying unusual activities.

2.8.4 Hybrid

Due to the advantages of each of these hosts and networks systems, it is clear that a combination of misuse and anomaly would provide better detection results, for example, allowing anomaly detection to manage unknown events while misuse detection specifies known attack signatures [57]. Such an approach should reduce the level of false positives if an appropriate method of checking conflicting decisions from multiple detection approaches can be properly managed. Some approaches have also two anomaly detection engines together in order to try to balance the false positive rate of one against the other [57].

2.9 IDS ARCHITECTURE

All intrusion detection systems have some well-known elements that are described more detailed below [39]:

- Application data collection sources: The place where the collection of data for current or later analysis are gathered.
- Rules: These rules are often those that describe the violations that may be bound and which the data obtained in the previous point are compared to.
- Filter: This part handles the applied rules concerning the obtained data.
- Anomaly detectors: When in use of an IDSs based on anomaly analysis, they are those that detect threats in the system or monitored resources.
- Alarm or report generator: Once the data have been processed with the filter rules, if there is any situation that gives the impression that the system security has been compromised, this part of the intrusion detector informs the administrator about this fact.

Chapter 3

System Model

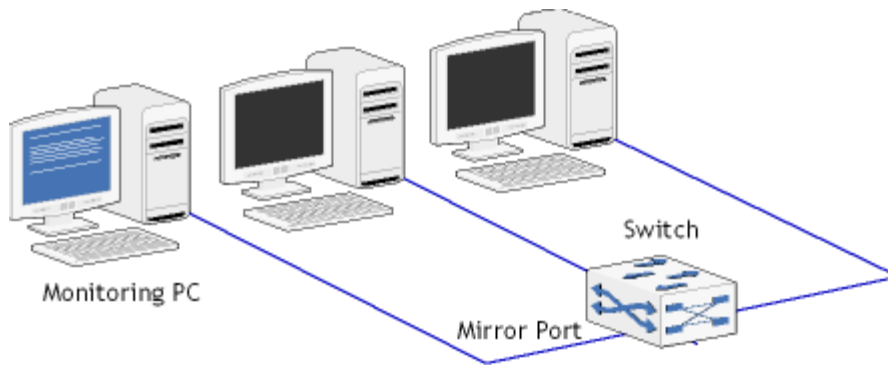


Figure 3-1: IDS Scenario

3.1 Overview of the proposed system

Setup involves three hosts generating the different kinds of application and system log traffic and sending it to the central log collector (Monitoring PC). Specifically, IDS is installed on the Monitoring PC which logs the network inbound and outbound traffic into the database.

Furthermore, we use three PCs (virtual machines). The packets are examined in real-time by the intrusion detection system. The switch has been configured with a mirrored port for the detect operability. Tests run on my local network.

3.2 Port mirroring

Port mirroring is configured on a network switch to send a copy of network packets seen on one switch port to a network monitoring connection on another switch port [41]. This is ordinarily used for network hosts that require monitoring of network traffic like an intrusion detection system technology that is used to support application performance management. Port mirroring enables the system manager to keep close track of switch performance by placing a protocol analyzer on the port that's receiving mirroring data [42].

An administrator configures port mirroring by assigning a port from which will send a copy of all packets and another port in which those packets will be sent [41]. A packet bound for heading away from the first port will be sending to the second port as well.

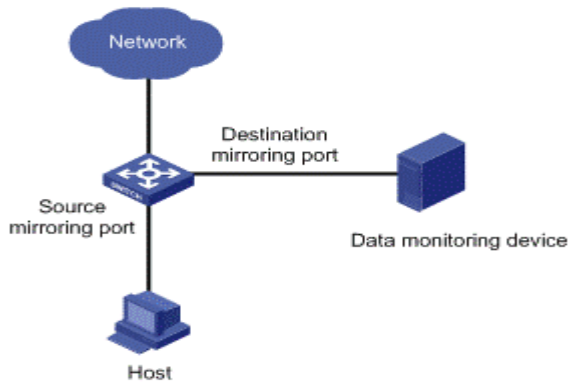


FIGURE 3-2: MIRRORED PORT

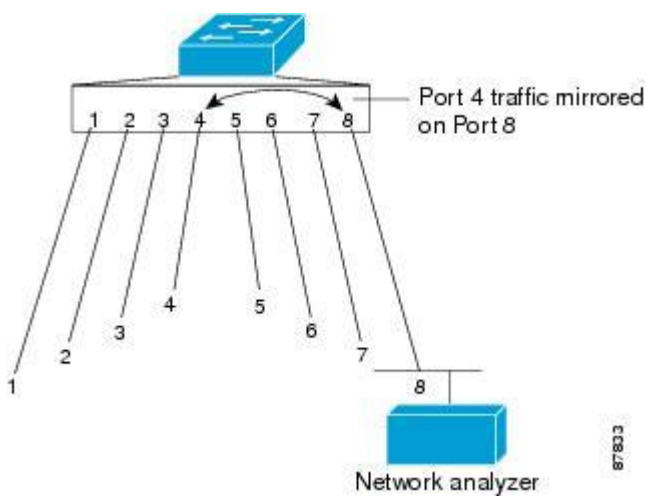


FIGURE 3-3: NETWORK ANALYZER

In this figure, the sniffer is attached to a port that is configured to receive a copy of every packet that host ascends. This port is called a SPAN port.

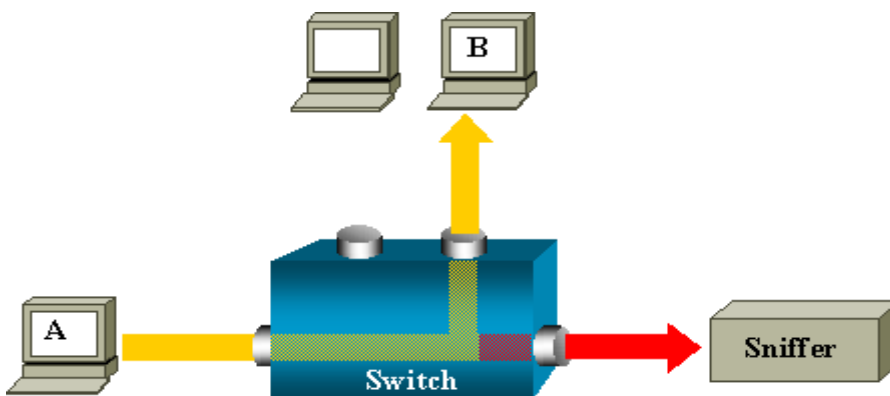


FIGURE 3-4: SNIFFER MODE

A monitored port has these characteristics [41] [42]:

- It can be any port type.
- It can be tracked in multiple SPAN sessions.
- It cannot be a destination port.
- Each port of switch can be configured with a direction to monitor.
- Source ports can be in the same or not the same VLANs.
- All active ports in the source VLAN are included as source ports.

3.2.1 Configuration set up switch

```

File Edit Setup Control Window Help
IDS#
IDS#
IDS#conf
IDS#configure te
IDS#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
IDS(config)#monitor session 2 source interface Fa0/1 - 8
IDS(config)#no sh
IDS(config)#no shutdown
% Incomplete command.

IDS(config)#sion 2 destination interface Gi0/1 encapsulation replicate
IDS(config)#end
IDS#
00:14:15: %SYS-5-CONFIG_I: Configured from console by consoleh
IDS#show sh
IDS#show sho
IDS#show no
IDS#show monitor se
IDS#show monitor session 2
Session 2
-----
Type                : Local Session
Source Ports        :
Both                : Fa0/1-8
Destination Ports   : Gi0/1
Encapsulation       : Replicate
Ingress             : Disabled

IDS#con
IDS#conf
IDS#configure te
IDS#configure terminal int
IDS#configure terminal inter
IDS#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
IDS(config)#int
IDS(config)#interface v1
IDS(config)#interface v1an 1
IDS(config-if)#ip ad
IDS(config-if)#ip add
IDS(config-if)#exit
IDS(config)#sh
IDS(config)#show
IDS(config)#show v1
IDS(config)#show v1
IDS(config)#show v1
IDS(config)#show v1
IDS(config)#exit
IDS#sh
IDS#show
00:19:11: %SYS-5-CONFIG_I: Configured from console by consolev1
IDS#show vlan
VLAN Name                Status      Ports

```

FIGURE 3-5: CONFIGURATION SWITCH

Main Code

```

switch>enable
switch#configure terminal
Enter configuration commands, one per line. End with
CNTL/Z.

switch(config) #monitor session 1 source interface
fastEthernet 0/1
switch(config)#monitor session 1 source interface
fastEthernet 0/2
switch(config)#monitor session 1 source interface
fastEthernet 0/3

```

```
switch(config)#monitor session 1 source interface
fastEthernet 0/4
switch(config)#monitor session 1 source interface
fastEthernet 0/5
switch(config)#monitor session 1 source interface
fastEthernet 0/6
switch(config)#monitor session 1 source interface
fastEthernet 0/7
switch(config)#monitor session 1 source interface
fastEthernet 0/8
switch(config)#monitor session 1 destination interface
gigabit Ethernet 0/1
```

```
switch#show monitor session 1
```

```
Session 1
```

```
-----
```

```
Type: Local Session
```

```
Source Ports :
```

```
Both : Fa0/1-8
```

```
Destination Ports : Gi0/1
```

```
Encapsulation: Native
```

Chapter 4 Snort

4.1 Introduction

Snort is a signature based IDS [46] that allows to manage the status of a network topology. Its operation has some common functions with sniffers, because Snort assays all the network traffic looking for any type of intrusion. Snort is a detection machine that allows registering, warning, and responding to any attack previously defined. It is one of the most defaults used, has a large number of preset signatures and constantly updated.

The basic data of its architecture are [46]:

- The module of capture of traffic that allows capturing all the network packages the decoder, which is reliable of creating data structures with the packages and identifying the network protocols.
- The pre-processors that allow extending the system parts.
- The detection engine that analyzes the packages pursuant the signatures.
- The file of signatures where the popular attacks are defined for their detection.
- The detection plugins that allow modifying the functionality of the detection engine and finally, the output plugins for determining what, how and where the alerts are saved.

In last years, some significantly projects have been proposed to extend the abilities of Snort [43, 44, 45]. For instance, [43] models only the http traffic, [44] models the network traffic as a set of events and look for disadvantages in these events, [45] enhance the functionalities of Snort automatically create patterns of misuse from attack data, and the ability of detecting successive intrusion behaviors, that is a pre-processor based on studying the reconstruction of package in the network to avoid popular attacks in the IDS.

Snort [46] is the most popular open source detection intrusion system. It is able to analyze the TCP/IP datagram traffic on a network in real time. It is network based and can be used either as a sniffer or as IDS. It is flexible software which can be connected to the most important databases such as Oracle, MySQL. It is consists of an attack detection engine as well as a port scanner, which allows alerting or responding to any kind of previously defined attack.

Furthermore, Snort [46] has other possible supplements to make the analysis easier to the user. There can be found GUI interfaces such as IDS center or a web application such ACID or BASE that will get data from the database and will show it in a friendlier html format. Snort implements an easy rule creation language, powerful and clean.

Snort can work [46] as a sniffer so the traffic in the network can be shown. But the side we want to use is the IDS. When a packet matches some rule pattern it is logged. Afterwards or at that moment the user knows when, how and from where the attack was performed.

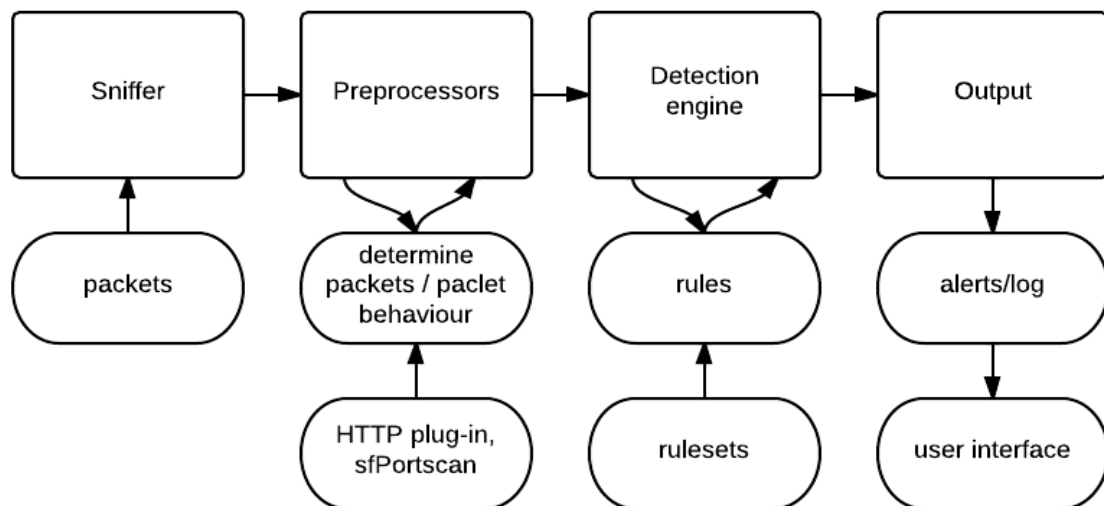


FIGURE 4-1: SNORT

4.2 Snort Features

Snort uses an ordered set of behaviors [47] to define what network traffic matches its rules and should be alerted on. Much of this behavior is customizable. Inbound data is decoded first by the packet decoder. If we are using Snort only as a packet sniffer, the decoded data will be formatted for the console display and shown. If we're using Snort as a packet logger, the packets will be put into either ASCII format in a directory tree or a binary file, whichever one we clarified on the command line, and saved to disk. If we are using Snort as a NIDS, the function is somewhat complex. When using Snort as a NIDS, after the inbound packets are analyzed by the packet decoders, the data is then sent through any preprocessors that we may have enabled in our snort configuration file. That data are being sent to the detection machine, which matches it against the rules in any ruleset enabled in our snort configuration file. Matches are sent to the alerting and logging levels, to be passed through whatever output plug-ins we have selected [47].

4.3 Architecture of Snort

The Snort processes the data in one thread and in five stages [46]. The first step is a compilation of packets that pass through the decoder of Snort, and with suitable adjustment can be made and detects attacks using the decoder alerts. Then activated all preprocessors that beyond decoding and further processing of the packets may also detect attacks and send alerts. This is the point where the Snort, although it belongs to the IDS signatures, enters with its own way the concept of detecting abnormalities. Then, the intrusion detection mechanism

is activated by applying signatures and Snort rules in processed packets. Finally, made known to the user the results of the operation of Snort from different output units.

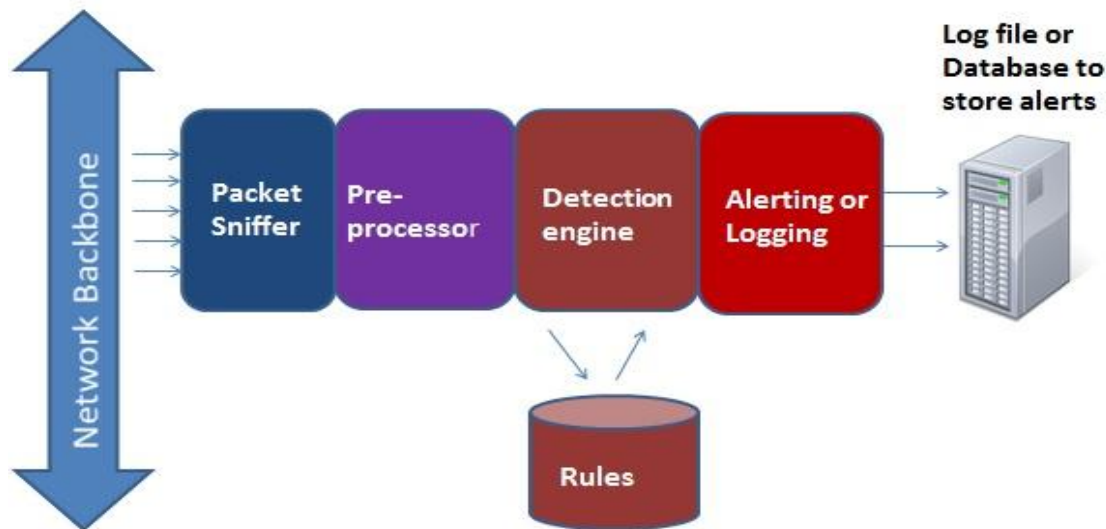


FIGURE 4-2: ARCHITECTURE OF SNORT

4.3.1 Packet Decoder

The packets [46] enter through the network card and are decoded off the wire by the packet decoder, which defines which protocol is in use for a given packet and fits the data against allow able behavior for packets of their protocol. The packet decoder can generate alerts of its own based on malformed protocol headers, exceedingly long packets, unusual or incorrect TCP options that are contained in the headers, and other such behavior. We can enable or disable more verbose alerting for all of these fields in your snort.conf file.

4.3.2 The Preprocessors

Preprocessors [46] are plug-ins to Snort that allows us to parse incoming data in different ways that may be useful. If we run Snort without any preprocessors given in our snort.conf configuration file, we will only look at each packet as it comes in over the wire. This is probably going to cause missing some attacks, since many popular attacks aim to functions like overwriting data in overlapping fragments, purposeful IDS evasion techniques such as putting part of a malicious application request in one packet and the rest in a different packet, and other such practices.

4.3.3 Detection Engine

The Detection Engine [46] achieves the actual attack detection by matching various values taken in the previous steps against a set of rules that encodes patterns of known attacks. If a

match is found, the corresponding action that is defined in rule will be executed, e.g. drop and log the packet, generate alert to system administrator.

4.3.4 Snort Alerts

In most cases, the first part of information [46] that an analyst reviews is an alert. An alert packet passed from a detection machine when it matches an event to a well-known pattern. This message can take many forms: pager message, syslog entry, ticket system entry.

4.3.5 Snort Packet Data

Snort can use packet data in three base formats [47]: ASCII, Pcap binary format, and Unified binary format. ASCII logs, are easier to read using a text editor, are not as useful as the binary logs for analysis. Pcap binary logs can be processed by many tools that have been designed with analysis network traffic. A few examples of tools that can read Pcap format files are tcpdump, ethereal, ngrep, tcpreplay, logsorter, ethereape, and many, many more.

4.4 Three modes of Snort

- Sniffer mode, which analyze the packets off of the network traffic and displays them for a continuous Stream on the console [47].
- Packet Logger mode, which logs the packets to disk [47].
- Network Intrusion Detection System (NIDS) mode, which does detection and analysis functions on network traffic. This is the most complex and configurable mode [47].

4.5 Snort Rules

Snort uses straightforward rules [48] description language that is versatile and quite useful. There are a number of straightforward guidelines when writing Snort rules that will help safeguard our logic.

Most Snort rules are written [48] in a single line. A snort rule has two parts, the header and the logical. The rule header includes the rule's action, protocol, source and destination IP addresses and netmasks, and the source and destination ports details. The rule option section includes alert messages and information on which parts of the packet should be shall inspect to determine if the rule action should be used.

```

alert tcp any any -> 192.168.1.0/24 111 \
  (content:"|00 01 86 a5|"; msg:"mountd access");)

```

Figure 4-3: Sample Snort Rules

4.5.1 Rules Headers

The rule header [46, 48] contains the information that defines the function of a packet, as well as what to do in the event that a packet with all the properties indicated in the rule should show up. The first part in a rule is the rule action. The action of rule tells Snort what to do when it finds a packet that meets the rule criteria. There are five known standard actions in Snort, alert, log, pass, activate, and dynamic. In addition, if we are running Snort in inline mode, we have additional options which include drop, reject, and sdrop.

- alert
- pass
- activate
- dynamic

4.5.2 IP Addresses

A rule header [46, 48] contains IP address and port information for a given rule. The keyword could be used to describe any address. Snort does not have a function to provide host name lookup for the IP address fields in the configuration file. The addresses are created by a straight numeric IP address and a CIDR block. The CIDR block includes the net mask that should be applied to the rule's address and any inbound packets that are tested against the rule.

```

alert tcp !192.168.1.0/24 any -> 192.168.1.0/24 111 \
  (content:"|00 01 86 a5|"; msg:"external mountd access");)

```

FIGURE 4-4: EXAMPLE IP ADDRESS NEGATION RULE

```

alert tcp ![192.168.1.0/24,10.1.1.0/24] any -> \
  [192.168.1.0/24,10.1.1.0/24] 111 (content:"|00 01 86 a5|"; \
  msg:"external mountd access");)

```

FIGURE 4-5: EXAMPLE IP ADDRESS LIST

4.5.3 Activate/Dynamic Rules

Activate/dynamic rule [46, 48] pairs give Snort a powerful function. We can now have one rule activate another when its action is executed for a set number of packets. This is very useful if we want to set Snort up to execute follow on recording when a particular rule does not work. Activate rules operate just like alert rules, unless they have a required option field activate.

```

activate tcp !$HOME_NET any -> $HOME_NET 143 (flags:PA; \
  content:"|E8C0FFFFFF|/bin"; activates:1; \
  msg:"IMAP buffer overflow!");
dynamic tcp !$HOME_NET any -> $HOME_NET 143 (activated_by:1; count:50;)

```

FIGURE 4-6: ACTIVATE DYNAMIC RULES

4.5.4 General Rule Options

The msg rule [46, 48] option tells the logging and alerting engine the message to print along with a packet dump or to an alert. It is only a text string that utilizes the \ as an escape character to show a discrete character that might otherwise confuse Snort's rules parser.

```
msg:"<message text>";
```

The reference keyword [46, 48] permit rules to include references to outer attack identification systems. The plugin currently supports different specific systems as well as unique URLs. This plugin is to be used by outer plugins to provide a link to additional information about the alert packets.

The gid keyword [46, 48] is used to recognize what parts of Snort produce the event when a particular rule happens. For example gid 1 is associated with the rules subsystem and some gids over 100 are designated for specific preprocessors and the decoder. To evade difficult conflict with gids defined in Snort, it is proposed that values starting at 1,000,000 be used. For overall rule writing, it is not proposed that the gid keyword be utilized. This option should be utilized with the sid keyword.

```
gid:<generator id>;
```

The sid [46,48] keyword is used to uniquely recognize Snort rules. This information allows output plugins to recognize rules easily. This option should be used with the rev keyword.

```
sid:<snort rules id>;
```

The rev keyword [46, 48] is used to uniquely recognize revisions of Snort rules. Revisions, along with Snort rule ids, allow signatures and characteristics to be refined and replaced with updated information. This option should be used with the sid keyword.

```
rev:<revision integer>;
```

The class type keyword [46, 48] is used to classify a rule as detecting an attack that is part of a more overall type of attack class. Snort provides a standard set of attack classes that are used by the standard set of rules it provides. Defining classifications for rules provides a way to better organize the event data Snort produces.

The class type [46, 48] option can only use registrations that have been defined in snort.conf by using the config registration option. Snort provides a standard set of classifications in classification. Config that are used by the rules it provides.

The metadata tag [46, 48] allows a rule writer to incorporate more information about the rule, typically in a key-value format. Some metadata keys and values have meaning to Snort and are classify in Table. Keys other than those listed in the table are efficiently ignored by Snort and can be free form.

Table: Snort Metadata Keys

Key	Description	Value Format
engine	Indicate a Shared Library Rule	"shared"
soid	Shared Library Rule Generator and SID	gid sid
service	Target-Based Service Identifier	"http"

Table 1-1: Snort Metadata Keys

The examples [46, 48] below show a stub rule from a shared library rule. The first uses multiple metadata keywords, the second a unified metadata keyword, with keys divided by commas.

```

metadata:key1 value1;
metadata:key1 value1, key2 value2;

alert tcp any any -> any 80 (msg:"Shared Library Rule Example"; \
  metadata:engine shared; metadata:soid 3|12345;)

alert tcp any any -> any 80 (msg:"Shared Library Rule Example"; \
  metadata:engine shared, soid 3|12345;)

```

Table: General rule option keywords [48]

Keyword	Description
msg	The msg keyword tells the logging and alerting engine the message to print with the packet dump or alert.
reference	The reference keyword allows rules to include references to external attack identification systems.

gid	The gid keyword (generator id) is used to identify what part of Snort generates the event when a particular rule fires.
sid	The sid keyword is used to uniquely identify Snort rules.
rev	The rev keyword is used to uniquely identify revisions of Snort rules.
classtype	The classtype keyword is used to categorize a rule as detecting an attack that is part of a more general type of attack class.
priority	The priority keyword assigns a severity level to rules.
metadata	The metadata keyword allows a rule writer to embed additional information about the rule, typically in a key-value format.

Table 1-2: General rule option keywords

4.6 The Snort Configuration File

Snort uses a configuration file [46, 48] at begin time. A sample configuration file is included in the snort program.

There are other benefits to using the configuration file name as a command line argument to snort. For example, it is feasible to invoke multiple Snort instances on different network interfaces with different configuration. This configuration file of snort contains six basic parts [46, 48]:

- Variable definitions, where we define different variables. These variables are used in snort rules as well as for other aims, like specifying the location of rule files.
- Config parameters. These parameters recognize different snort configuration options. Some of them can also be used on the command line.

- Preprocessor configuration file. Preprocessors are performing a few actions before a packet is operated by the main snort detection engine.
- Output module configuration. Output modules check how snort data will be logged.
- Defining new action parts. If the default action parts are not adequate for our environment, we can define custom action types in the configuration file of snort.
- Rules files and configurations. Although we can add any rules in the main file, the agreement is to use different files for rules. These files are then included inside the main configuration file using the include keyword.

Using a List of Networks in Variables.

We can also define variables [46, 48] that contain different items. Consider that we have multiple networks in the company. Intrusion detection system is right back of the company firewall connecting to the Internet.

```
Var HOME_NET []
```

Using Interface Names in Variables.

We can also use interface names in defining variables [46, 48]. The following two statements define HOME_NET and EXTERNAL_NET variables on a Linux unit.

```
var HOME_NET $eth0_ADDRESS
```

```
var EXTERNAL_NET $eth1_ADDRESS
```

The any keyword could also be a variable [46, 48]. It fits to everything, just as it does in.

```
var EXTERNAL_NET any
```

The config directives [46, 48] in the snort.conf file permit a user to configure many general settings for snort.

```
config directive name[: value]
```

Preprocessors or input plug-ins [46, 48] operate on received packets before snort rules being in use. The preprocessor configuration is the second important part of the configuration file. This part provides basic details about adding or removing Snort preprocessors. The general format of configuring a preprocessor is for example:

```
preprocessor <preprocessor_name>[: <configuration_options>]
```

Output modules [46, 48], also called output plug-ins, manipulate output from snort rules. For example, if we want to log information to a database or send SNMP traps, we need output modules.

```
output <output_module_name>[: <configuration_options>]
```

4.7 Snort IDS mode

```
sudo /usr/local/snort/bin/snort -A console -c /usr/local/snort/etc/snort.conf -i eth0
```

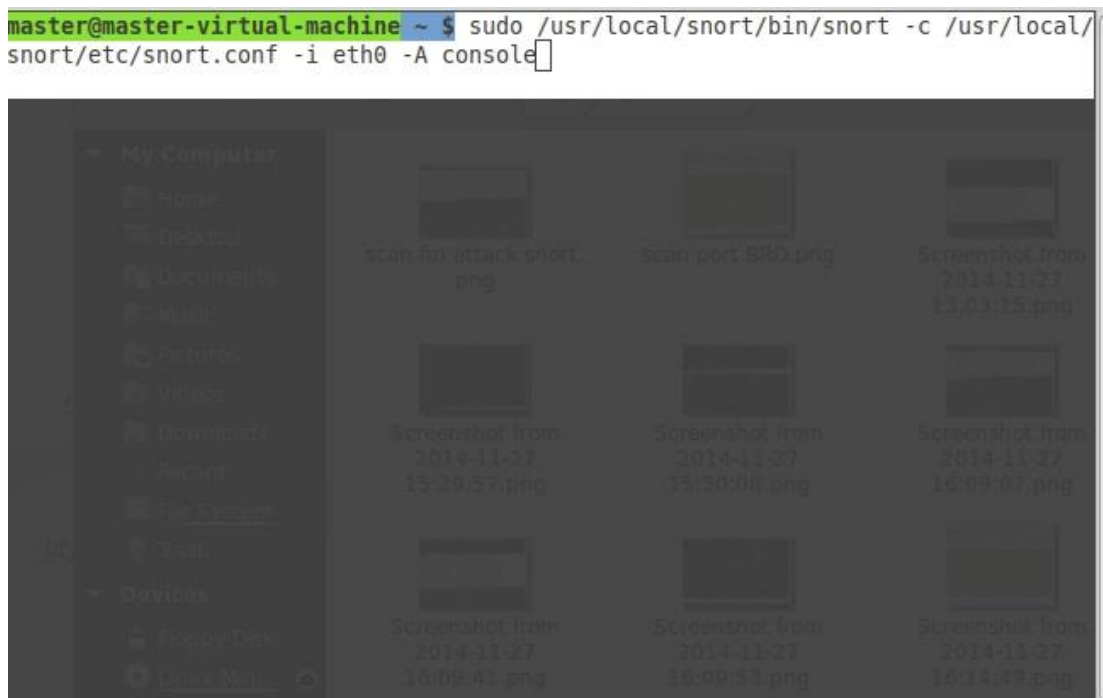


FIGURE 4-7: START SNORT

```

Terminal
'''' By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
Copyright (C) 2014 Cisco and/or its affiliates. All rights reserved.
Copyright (C) 1998-2013 Sourcefire, Inc., et al.
Using libpcap version 1.5.3
Using PCRE version: 8.31 2012-07-06
Using ZLIB version: 1.2.8

Rules Engine: SF_SNORT DETECTION ENGINE Version 2.4 <Build 1>
Preprocessor Object: SF_FTPTTELNET Version 1.2 <Build 13>
Preprocessor Object: SF_SDF Version 1.1 <Build 1>
Preprocessor Object: SF_SSH Version 1.1 <Build 3>
Preprocessor Object: SF_SIP Version 1.1 <Build 1>
Preprocessor Object: SF_GTP Version 1.1 <Build 1>
Preprocessor Object: SF_DNS Version 1.1 <Build 4>
Preprocessor Object: SF_DCERPC2 Version 1.0 <Build 3>
Preprocessor Object: SF_POP Version 1.0 <Build 1>
Preprocessor Object: SF_MODBUS Version 1.1 <Build 1>
Preprocessor Object: SF_SMTP Version 1.1 <Build 9>
Preprocessor Object: SF_SSLPP Version 1.1 <Build 4>
Preprocessor Object: SF_IMAP Version 1.0 <Build 1>
Preprocessor Object: SF_REPUTATION Version 1.1 <Build 1>
Preprocessor Object: SF_DNP3 Version 1.1 <Build 1>
Commencing packet processing (pid=3120)

```

FIGURE 4-8: SNORT IDS MODE

4.7.1 Test ids (ping)

Tests run on my local network.

```

Terminal
ICMP} 192.168.1.9 -> 192.168.1.3
11/27-13:02:28.035466 [**] [1:1000001:1] ICMP Testing Rule [**] [Priority: 0] {
ICMP} 192.168.1.3 -> 192.168.1.9
11/27-13:02:29.032146 [**] [1:1000001:1] ICMP Testing Rule [**] [Priority: 0] {
ICMP} 192.168.1.9 -> 192.168.1.3
11/27-13:02:29.042285 [**] [1:1000001:1] ICMP Testing Rule [**] [Priority: 0] {
ICMP} 192.168.1.3 -> 192.168.1.9
11/27-13:02:30.034535 [**] [1:1000001:1] ICMP Testing Rule [**] [Priority: 0] {
ICMP} 192.168.1.9 -> 192.168.1.3
11/27-13:02:30.040410 [**] [1:1000001:1] ICMP Testing Rule [**] [Priority: 0] {
ICMP} 192.168.1.3 -> 192.168.1.9
11/27-13:02:31.036503 [**] [1:1000001:1] ICMP Testing Rule [**] [Priority: 0] {
ICMP} 192.168.1.9 -> 192.168.1.3
11/27-13:02:31.042178 [**] [1:1000001:1] ICMP Testing Rule [**] [Priority: 0] {
ICMP} 192.168.1.3 -> 192.168.1.9
11/27-13:02:32.040020 [**] [1:1000001:1] ICMP Testing Rule [**] [Priority: 0] {
ICMP} 192.168.1.9 -> 192.168.1.3
11/27-13:02:32.046175 [**] [1:1000001:1] ICMP Testing Rule [**] [Priority: 0] {
ICMP} 192.168.1.3 -> 192.168.1.9
11/27-13:02:33.043680 [**] [1:1000001:1] ICMP Testing Rule [**] [Priority: 0] {
ICMP} 192.168.1.9 -> 192.168.1.3
11/27-13:02:33.056740 [**] [1:1000001:1] ICMP Testing Rule [**] [Priority: 0] {
ICMP} 192.168.1.3 -> 192.168.1.9
11/27-13:02:34.046051 [**] [1:1000001:1] ICMP Testing Rule [**] [Priority: 0] {
ICMP} 192.168.1.9 -> 192.168.1.3
11/27-13:02:34.094535 [**] [1:1000001:1] ICMP Testing Rule [**] [Priority: 0] {
ICMP} 192.168.1.3 -> 192.168.1.9
11/27-13:02:35.048011 [**] [1:1000001:1] ICMP Testing Rule [**] [Priority: 0] {
ICMP} 192.168.1.9 -> 192.168.1.3
11/27-13:02:35.055464 [**] [1:1000001:1] ICMP Testing Rule [**] [Priority: 0] {
ICMP} 192.168.1.3 -> 192.168.1.9
11/27-13:02:36.050015 [**] [1:1000001:1] ICMP Testing Rule [**] [Priority: 0] {

```

FIGURE 4-9: ICMP ALERT

4.7.1.1 Rule Ping

```
alert icmp any any -> any any (msg:"ICMP Testing Rule"; sid:1000001; rev:1;)
```

4.8 Port Scan Detection

The part when scanning a computer's ports [64]. Port is used as a mean of transferring data in and out of a computer system, port scanning recognizes open doors to a computer. Also, port scanning has legitimate uses in managing networks, but port scanning also can be malicious by default if someone is looking for a weakened access point to break into our computer.

The attacker checks if there are any open ports with the help of a particular software tool, a port scanner [64]. This program tries to connect with several ports on the destination computer. If it is successful, the tool informs about the specific ports as open and the attacker has the necessary information, showing which network services are available on the destination computer.

Tests run on my local network.

```
root@kali:~# nmap -sF 192.168.1.9
```

```
Starting Nmap 6.47 ( http://nmap.org ) at 2014-11-26 15:43 EST  
Nmap scan report for 192.168.1.9
```

FIGURE 4-10: PORT SCAN

The following figure shows the alerts that have produced the snort with message SCAN FIN.

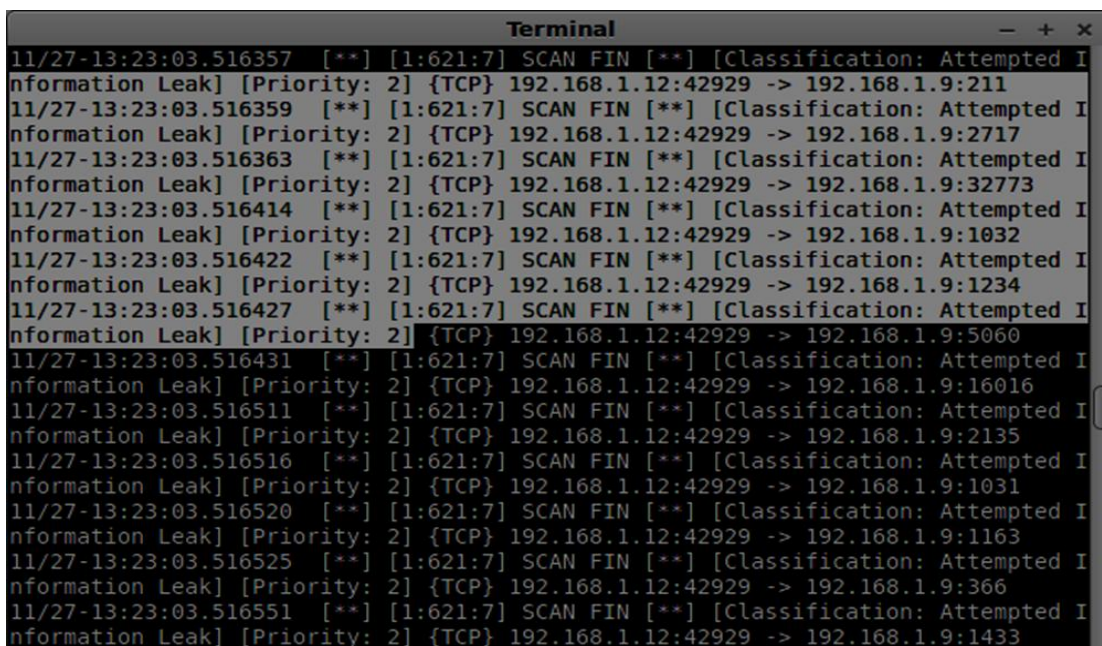


FIGURE 4-11: ALERT PORT SCAN

4.8.1 Rule Scan Fin

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN FIN"; flow:stateless; flags:F,12; classtype:attempted-recon; sid:621; rev:7;)
```

4.9 Detect SYN flood

A SYN flood [65] is a type of denial-of-service attack in which an attacker transmits a sequence of SYN requests to a target's system in an effort to use enough server resources to make the system unresponsive to network traffic.

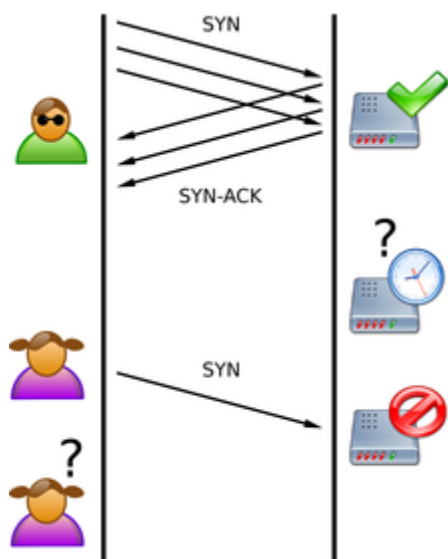


FIGURE 4-12: SYN FLOOD

In a normal [65] three-way handshake, the client would return an ACK (acknowledged) packet to confirm that the server's SYN/ACK packet was received, in order that communications could begin. Although, in a SYN flood, the ACK packet is never sent back by the enemy client. Instead, the client program sends repeated SYN requests to all the server's ports. An enemy client always knows a port is open when the server responds with a SYN/ACK packet.

The enemy [65] client makes the SYN requests all appear reliable, but because the IP addresses are fake ones, it is difficult for the server to terminate the connection by sending RST packets back to the client. Instead, the connection stays open. Before time-out can occur, another SYN packet comes from the inimical client. A connection of this type is called a half-open connection. Under these circumstances, the server becomes completely or almost completely busy with the enemy client and communications with rightful clients is difficult or impossible.

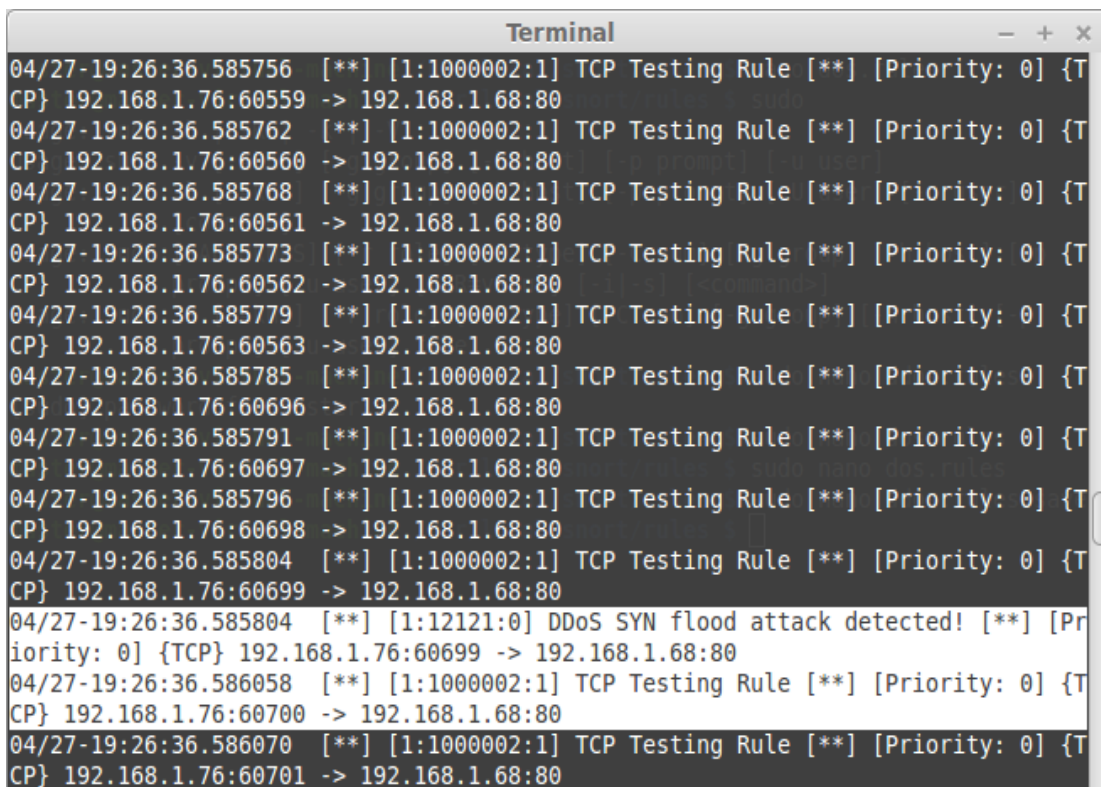
```
hping3 --flood -S -p 80 192.168.1.68
```

- Flood: sends as many packets as can the network card.
- -S: The TCP packet has flag SYN.
- -p 80: Packets are sent to port 80.

Tests run on my local network.

```
root@kali:~# hping3 --flood -S -p 80 192.168.1.68
HPING 192.168.1.68 (eth0 192.168.1.68): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
--- 192.168.1.68 hping statistic ---
97308 packets transmitted, 0 packets received, 100% packet loss
round-trip_min/avg/max = 0.0/0.0/0.0 ms
```

FIGURE 4-13: SYN FLOOD



```
Terminal
04/27-19:26:36.585756  [**] [1:1000002:1] TCP Testing Rule [**] [Priority: 0] {T
CP} 192.168.1.76:60559 -> 192.168.1.68:80
04/27-19:26:36.585762  [**] [1:1000002:1] TCP Testing Rule [**] [Priority: 0] {T
CP} 192.168.1.76:60560 -> 192.168.1.68:80
04/27-19:26:36.585768  [**] [1:1000002:1] TCP Testing Rule [**] [Priority: 0] {T
CP} 192.168.1.76:60561 -> 192.168.1.68:80
04/27-19:26:36.585773  [**] [1:1000002:1] TCP Testing Rule [**] [Priority: 0] {T
CP} 192.168.1.76:60562 -> 192.168.1.68:80
04/27-19:26:36.585779  [**] [1:1000002:1] TCP Testing Rule [**] [Priority: 0] {T
CP} 192.168.1.76:60563 -> 192.168.1.68:80
04/27-19:26:36.585785  [**] [1:1000002:1] TCP Testing Rule [**] [Priority: 0] {T
CP} 192.168.1.76:60696 -> 192.168.1.68:80
04/27-19:26:36.585791  [**] [1:1000002:1] TCP Testing Rule [**] [Priority: 0] {T
CP} 192.168.1.76:60697 -> 192.168.1.68:80
04/27-19:26:36.585796  [**] [1:1000002:1] TCP Testing Rule [**] [Priority: 0] {T
CP} 192.168.1.76:60698 -> 192.168.1.68:80
04/27-19:26:36.585804  [**] [1:1000002:1] TCP Testing Rule [**] [Priority: 0] {T
CP} 192.168.1.76:60699 -> 192.168.1.68:80
04/27-19:26:36.585804  [**] [1:12121:0] DDoS SYN flood attack detected! [**] [Pr
iority: 0] {TCP} 192.168.1.76:60699 -> 192.168.1.68:80
04/27-19:26:36.586058  [**] [1:1000002:1] TCP Testing Rule [**] [Priority: 0] {T
CP} 192.168.1.76:60700 -> 192.168.1.68:80
04/27-19:26:36.586070  [**] [1:1000002:1] TCP Testing Rule [**] [Priority: 0] {T
CP} 192.168.1.76:60701 -> 192.168.1.68:80
```

FIGURE 4-14: ALERT SYN FLOOD

4.9.1 Rule Syn Flood

```
alert tcp any any -> $HOME_NET any (flags:S; threshold: type threshold, track by_dst, count
20, seconds 3; msg:"DDoS SYN flood attack detected!";sid:12121;)
```

4.10 Detect brute-force FTP

In cryptography, a brute-force attack [53] is a cryptanalytic attack that can, theoretically, use any encrypted data except when data are encrypted in an information-theoretically secure way. An attack like this might be used when it is hard to exploit other weaknesses in an encryption system that would make the work easier. It consists of systematically checking all possible keys or passwords until the right one is found.

Brute-force attacks [54] might not be that efficient when obfuscating the data to be encoded, something that makes it more difficult for an attacker to acknowledge whether the code has been cracked. An encryption system can also calculate the time an attacker takes to successfully mount a brute-force attack against it.

Brute-force attacks are an application of brute-force search, the common problem-solving technique used for enumerating all candidates and checking each one [54].

Tests run on my local network.

```
giannios1983@kali:~$ ncrack -p 21 -user root -P 500-worst-password.txt 192.168.0.222
```

FIGURE 4-15 BRUTE FORCE

```

11/28-17:41:20.918830  [**] [1:3441:1] FTP Brute force attackgumenORT/smi [**] [
Classification: Misc Attack] [Priority: 2] {TCP} 192.168.0.18:34977 -> 192.168.0
.222:21
11/28-17:41:20.918838  [**] [1:3441:1] FTP Brute force attackgumenORT/smi [**] [
Classification: Misc Attack] [Priority: 2] {TCP} 192.168.0.18:34977 -> 192.168.0
.222:21
11/28-17:41:21.914617  [**] [1:3441:1] FTP Brute force attackgumenORT/smi [**] [
Classification: Misc Attack] [Priority: 2] {TCP} 192.168.0.18:34977 -> 192.168.0
.222:21
11/28-17:41:21.914626  [**] [1:3441:1] FTP Brute force attackgumenORT/smi [**] [
Classification: Misc Attack] [Priority: 2] {TCP} 192.168.0.18:34977 -> 192.168.0
.222:21
11/28-17:41:23.912401  [**] [1:3441:1] FTP Brute force attackgumenORT/smi [**] [
Classification: Misc Attack] [Priority: 2] {TCP} 192.168.0.18:34977 -> 192.168.0
.222:21
11/28-17:41:23.912410  [**] [1:3441:1] FTP Brute force attackgumenORT/smi [**] [
Classification: Misc Attack] [Priority: 2] {TCP} 192.168.0.18:34977 -> 192.168.0
.222:21

```

FIGURE 4-16 ALERT BRUTE FORCE FTP

4.10.1 Rule Brute Force Ftp

```

alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP Brute force attack";
content: "PASS"; nocase; offset:0; depth:4; content:"|0a|"; within:3;
flow:from_client,established; sid:10491;)

```

```

alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP Brute force
attackgumenORT/smi"; classtype:misc-attack; sid:3441; rev:1;)

```

4.11 Detect UDP Flood

A UDP flood [53] attack is a denial-of-service (DoS) attack using the UDP protocol, a connectionless computer networking protocol.

UDP when it comes to denial-of-service attacks [53] can be more complicated than TCP protocol. However, a UDP flood attack sends a large number of UDP network packets to random ports on a remote host.

For numerous UDP packets [53], the system that is under attack will be forced into sending many ICMP packets, thus leading it to be unreachable by other clients. The attackers might also be able to spoof the IP address of the UDP packets, securing that the ICMP return packets do not arrive them, and hiding their network locations [49].

```

hping3 --udp --flood -p 80 192.168.1.68

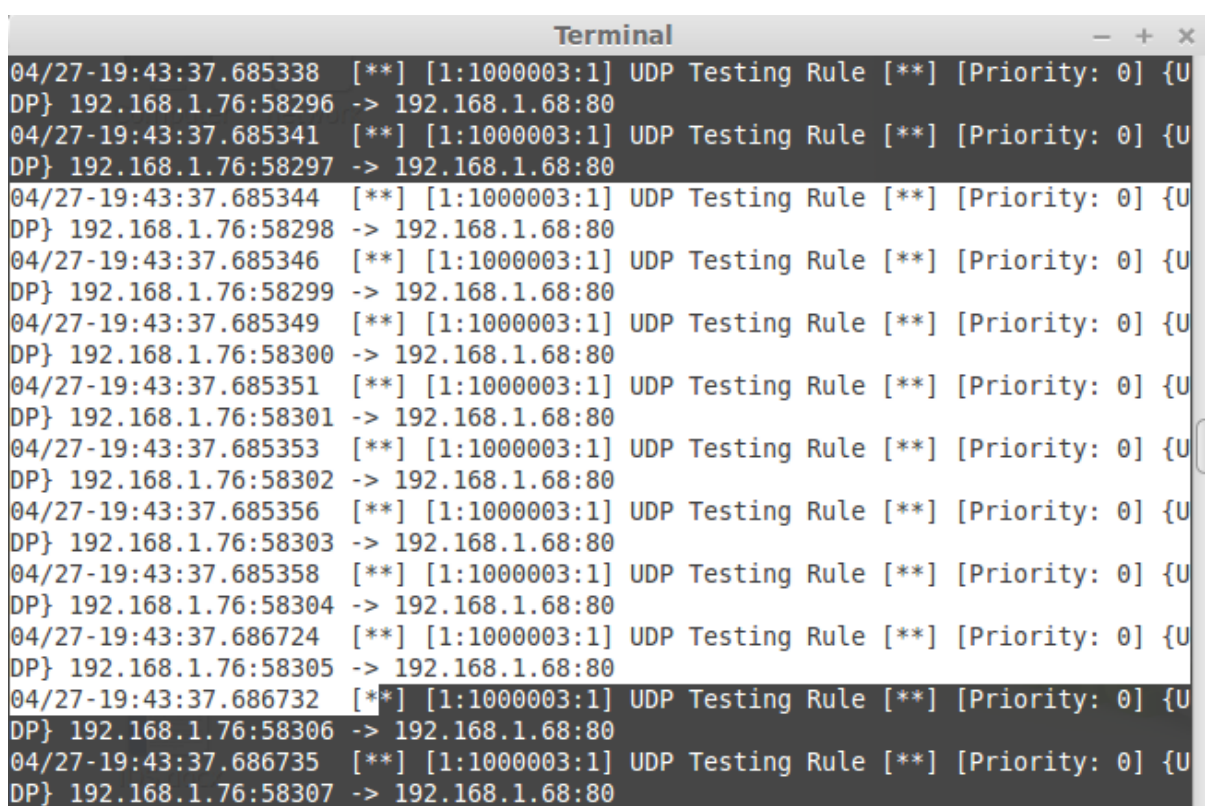
```

- Udp: sends udp packets.
- -p 80: Packets are sent to port 80.

Tests run on my local network.

```
root@kali:~# hping3 --udp --flood -p 80 192.168.1.68
HPING 192.168.1.68 (eth0 192.168.1.68): udp mode set, 28 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
--- 192.168.1.68 hping statistic ---
65860 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

FIGURE 4-17 UDP FLOOD



```
Terminal
04/27-19:43:37.685338  [**] [1:1000003:1] UDP Testing Rule [**] [Priority: 0] {U
DP} 192.168.1.76:58296 -> 192.168.1.68:80
04/27-19:43:37.685341  [**] [1:1000003:1] UDP Testing Rule [**] [Priority: 0] {U
DP} 192.168.1.76:58297 -> 192.168.1.68:80
04/27-19:43:37.685344  [**] [1:1000003:1] UDP Testing Rule [**] [Priority: 0] {U
DP} 192.168.1.76:58298 -> 192.168.1.68:80
04/27-19:43:37.685346  [**] [1:1000003:1] UDP Testing Rule [**] [Priority: 0] {U
DP} 192.168.1.76:58299 -> 192.168.1.68:80
04/27-19:43:37.685349  [**] [1:1000003:1] UDP Testing Rule [**] [Priority: 0] {U
DP} 192.168.1.76:58300 -> 192.168.1.68:80
04/27-19:43:37.685351  [**] [1:1000003:1] UDP Testing Rule [**] [Priority: 0] {U
DP} 192.168.1.76:58301 -> 192.168.1.68:80
04/27-19:43:37.685353  [**] [1:1000003:1] UDP Testing Rule [**] [Priority: 0] {U
DP} 192.168.1.76:58302 -> 192.168.1.68:80
04/27-19:43:37.685356  [**] [1:1000003:1] UDP Testing Rule [**] [Priority: 0] {U
DP} 192.168.1.76:58303 -> 192.168.1.68:80
04/27-19:43:37.685358  [**] [1:1000003:1] UDP Testing Rule [**] [Priority: 0] {U
DP} 192.168.1.76:58304 -> 192.168.1.68:80
04/27-19:43:37.686724  [**] [1:1000003:1] UDP Testing Rule [**] [Priority: 0] {U
DP} 192.168.1.76:58305 -> 192.168.1.68:80
04/27-19:43:37.686732  [**] [1:1000003:1] UDP Testing Rule [**] [Priority: 0] {U
DP} 192.168.1.76:58306 -> 192.168.1.68:80
04/27-19:43:37.686735  [**] [1:1000003:1] UDP Testing Rule [**] [Priority: 0] {U
DP} 192.168.1.76:58307 -> 192.168.1.68:80
```

FIGURE 4-18 ALERT UDP FLOOD

4.11.1 Rule UDP Flood

alert udp \$EXTERNAL_NET any -> \$HOME_NET any (msg:"UDP Testing Rule ";
threshold: type threshold, track by_src, count 10000, seconds 5; sid: 10000002; rev: 1;)

4.12 Detect Brute Force ssh

In cryptography, a brute-force attack [53] is a cryptanalytic attack that can, theoretically, use any encrypted data except when data are encrypted in an information-theoretically secure way. Such an attack might be used when it is not easy to take advantage of other inability in an encryption system that would make the task easier. It consists of systematically checking all possible keys or passwords until the right one is found.

Tests run on my local network.

```
06/13-11:33:32.567931  [**] [1:2001219:4] Potential SSH Brute Force Attack [**]  
[Classification: Attempted Denial of Service] [Priority: 2] {TCP} 192.168.0.14:6  
0916 -> 192.168.0.11:22  
06/13-11:33:33.195454  [**] [1:2001219:4] Potential SSH Brute Force Attack [**]  
[Classification: Attempted Denial of Service] [Priority: 2] {TCP} 192.168.0.14:6  
0919 -> 192.168.0.11:22
```

FIGURE 4-19 ALERT BRUTE FORCE SSH

4.12.1 Rule Brute-Force SSH

```
alert tcp any any -> $HOME_NET (msg:"Potential SSH Brute Force Attack"; flow:to_server;  
flags:S; threshold:type threshold, track by_src, count 3, seconds 60; classtype:attempted-dos;  
sid:2001219; \ rev:4; resp:rst_all; )
```

Chapter 5 Suricata

5.1 Suricata ids

Suricata is a rule-based IDS/IPS [50] program that uses externally developed rule sets to monitor network traffic and warns the admin by using alerts when suspicious events occur. Designed to be compatible with existing network security components, The Suricata Engine is a fairly new open-source intrusion detection and prevention engine. It is developed by Open Information Security Foundation. Suricata features unified output functionality and pluggable library options to accept calls from other applications. As a multi-threaded engine, Suricata offers increased speed and efficiency in network traffic analysis. Furthermore to hardware acceleration the engine is build to utilize the increased processing power offered by the latest multi-core CPU chip sets.

The operation modes of Suricata [50] are the same as Snort's. It can be used either as an IDS or IPS system. There are no differences when connecting Suricata to the network. Suricata even has basically the same rule syntax as Snort, which means that both systems can use more or less the same rules.

The general data flow through Suricata [50] is similar to Snort. Packets are captured, decoded, processed and analyzed. However, when it comes to the internals of the Suricata Engine, differences become apparent.

Suricata also features the HTP Library [50] that is a HTTP normalizer. This incorporates and provides advanced processing of HTTP streams for Suricata.

Suricata [50] uses a multi-threaded approach opposed to the Snort's single threaded engine. Threads use one or more thread modules for this. Threads have an inbound queue handler and an outbound queue handler. These are used to get packets from other threads, or from the global packet pool.

Taking these few [50], but significant differences into account, it is probable that Snort and Suricata perform differently when it comes to the speed and efficiency of network traffic analysis.

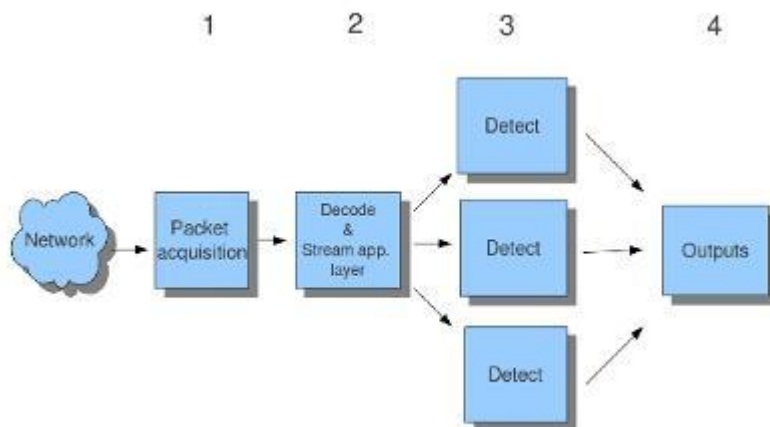


FIGURE 5-1 MULTI-THREAD DESIGN

CPU/CPU core-threads set_cpu_affinity: yes

Core	0	PAQ	DECODE	STREAM	DETECT-	OUTPUT
	1				DETECT	
	2				DETECT	
	3				DETECT	

set_cpu_affinity: no
Example

Core	0	PAQ			DETECT	
	1		DECODE			
	2			STREAM	DETECT X2	
	3				DETECT	OUTPUT

FIGURE 5-2 MULTI-CPU AFFINITY

Suricata Features [50]:

- High rate of performance, expandable through multi-threading.
- Protocol recognition.
- File recognition, extraction, on the fly MD5 calculation.
- TLS handshake analysis, detect/prevent operate like Diginotar.
- Rules and outputs compliant to Snort syntax.
- Helpful logging like HTTP request log, TLS certificate log, DNS logging.

5.1.1 Suricata configuration file

Suricata [50] uses the yaml format for configuration.

5.1.2 Max-pending-packets

With the max-pending-packets [50] setting we can set the number of packets we allow Suricata to process at the same time. This can range from one packet to tens of thousands/hundreds of thousands of packets. As a result have higher performance and more use of memory, or lower performance and less use of memory. Numerous packets being processed results in a higher performance and the use of more memory. A low number of packets, leads to lower performance and less use of memory.

```
max-pending-packets: 1024
```

5.1.3 Default-packet-size

For the max-pending-packets option [50], Suricata has to retain packets in memory. With the default-packet-size option, we can regulate the size of the packets on our network level. The computer machine can still process these bigger packets, but processing it will lower the performance.

```
default-packet-size: 1514
```

5.1.4 Action-order

All signatures [50] have different properties. Action property is one of those. A summary of what will happen when a signature fits and includes one of those actions [50]:

Pass

If a signature contains the action pass, Suricata stops checking the packet and skips to the end of all rules.

Drop

This action only uses the mode of IPS/inline. If the program checks a signature that matches, containing drop, it stops instantly and the packet will not be sent any more.

Reject

This is an active dismiss of the packet. Both receiver and sender receive a dismiss packet.

Alert

If a signature fits include alert, the packet will be dealt like any other non-threatening packet, excluding this one an alert will be created by Suricata.

5.1.5 Detection engine

5.1.5.1 Inspection configuration

The detection engine [51] builds internal groups of signatures. Suricata loads signatures comparing all the network traffic. The truth is that many rules probably will not be needed.

For that reason, all signatures will be categorized in groups [51]. Although, a distribution containing a lot of groups will make use of a certain amount of memory. Not every type of signature will be categorized in the same group. There is a possibility that different signatures with common properties will be placed together in a group. The number of groups [51] will define the balance between memory and performance. A low amount of groups will lower the performance yet uses a low amount of memory. The opposite counts for a higher number of groups. The engine allows us to control the balance between memory and performance.

```
detect-engine:
  -profile: medium          #The balance between performance and memory usage.
  This is the default setting.
  - custom-values:
    toclient_src_groups: 2
    toclient_dst_groups: 2
```

```

toclient_sp_groups: 2

toclient_dp_groups: 3

toserver_src_groups: 2

toserver_dst_groups: 4

toserver_sp_groups: 2

toserver_dp_groups: 25

- sgh-mpm-context: auto

- inspection-recursion-limit: 3000

```

At all of these options, we can add a value. Most signatures have the ability to focus on one direction, meaning focusing on the server, or focusing on the client [51].

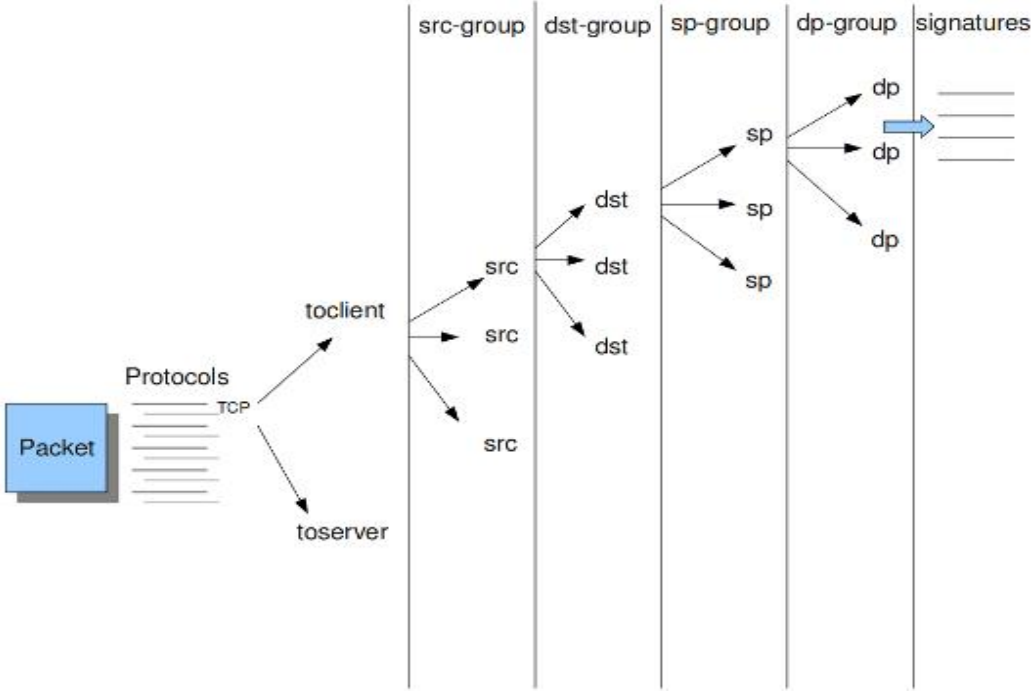


FIGURE 5-3 DETECTION ENGINE GROUPING TREE

src	Stands for source IP-address.
dst	Stands for destination IP-address.
sp	Stands for source port.
dp	Stands for destination port.

5.2 Suricata ids mode

```

[ advertise %x ]
[ tx-lpi on|off ]
[ tx-timer %d ]
ethtool -h|--help          Show this help
ethtool --version          Show version number
master@master-virtual-machine ~ $ ethtool -k eth0 rx off
ethtool: bad command line argument(s)
For more information run ethtool -h
master@master-virtual-machine ~ $ sudo suricata -c /etc/suricata/suricata.yaml -
i eth0
27/11/2014 -- 15:27:06 - <Notice> - This is Suricata version 2.0.1 RELEASE
27/11/2014 -- 15:27:10 - <Error> - [ERRCODE: SC_ERR_REFERENCE_UNKNOWN(150)] - un
known reference key "osvdb". Supported keys are defined in reference.config file
. Please have a look at the conf param "reference-config-file"
27/11/2014 -- 15:27:10 - <Error> - [ERRCODE: SC_ERR_INVALID_SIGNATURE(39)] - err
or parsing signature "alert http $EXTERNAL_NET any -> $HOME_NET any (msg:"ET WEB
CLIENT Samsung Galaxy Knox Android Browser RCE smdm attempt"; flow:to_client,es
tablished; content:"|0d 0a 0d 0a|"; content:"smdm|3a|//"; nocase; distance:0; re
ference:url,blog.quarkslab.com/abusing-samsung-knox-to-remotely-install-a-malici
ous-application-story-of-a-half-patched-vulnerability.html; reference:osvdb,1145
90; reference:url,cxsecurity.com/issue/WLB-2014110124; classtype:web-application
-activity; sid:2019750; rev:4;)" from file /etc/suricata/rules/emerging-web_clie
nt.rules at line 874
27/11/2014 -- 15:27:22 - <Warning> - [ERRCODE: SC_ERR_FOPEN(44)] - Error opening

```

FIGURE 5-4 START SURICATA

```
Terminal
i eth0
27/11/2014 -- 15:27:06 - <Notice> - This is Suricata version 2.0.1 RELEASE
27/11/2014 -- 15:27:10 - <Error> - [ERRCODE: SC_ERR_REFERENCE_UNKNOWN(150)] - un
known reference key "osvdb". Supported keys are defined in reference.config file
. Please have a look at the conf param "reference-config-file"
27/11/2014 -- 15:27:10 - <Error> - [ERRCODE: SC_ERR_INVALID_SIGNATURE(39)] - err
or parsing signature "alert http sEXTERNAL_NET any -> $HOME_NET any (msg:"ET WEB
_CLIENT Samsung Galaxy Knox Android Browser RCE smdm attempt"; flow:to client,es
tablished; content:"|0d 0a 0d 0a|"; content:"smdm|3a|//"; nocase; distance:0; re
ference:url,blog.quarkslab.com/abusing-samsung-knox-to-remotely-install-a-malici
ous-application-story-of-a-half-patched-vulnerability.html; reference:osvdb,1145
90; reference:url,cxsecurity.com/issue/WLB-2014110124; classtype:web-application
-activity; sid:2019750; rev:4;)" from file /etc/suricata/rules/emerging-web_clie
nt.rules at line 874
27/11/2014 -- 15:27:22 - <Warning> - [ERRCODE: SC_ERR_FOPEN(44)] - Error opening
file: "/etc/suricata/threshold.config": No such file or directory
27/11/2014 -- 15:27:22 - <Warning> - [ERRCODE: SC_ERR_NOT_SUPPORTED(225)] - Eve-
log support not compiled in. Reconfigure/recompile with libjansson and its devel
opment files installed to add eve-log support.
27/11/2014 -- 15:27:22 - <Warning> - [ERRCODE: SC_ERR_PCAP_CREATE(21)] - Using P
cap capture with GRO or LRO activated can lead to capture problems.
27/11/2014 -- 15:27:22 - <Notice> - all 2 packet processing threads, 3 managemen
t threads initialized, engine started.
```

FIGURE 5-5 SURICATA IDS MODE

5.2.1 Test suricata ids

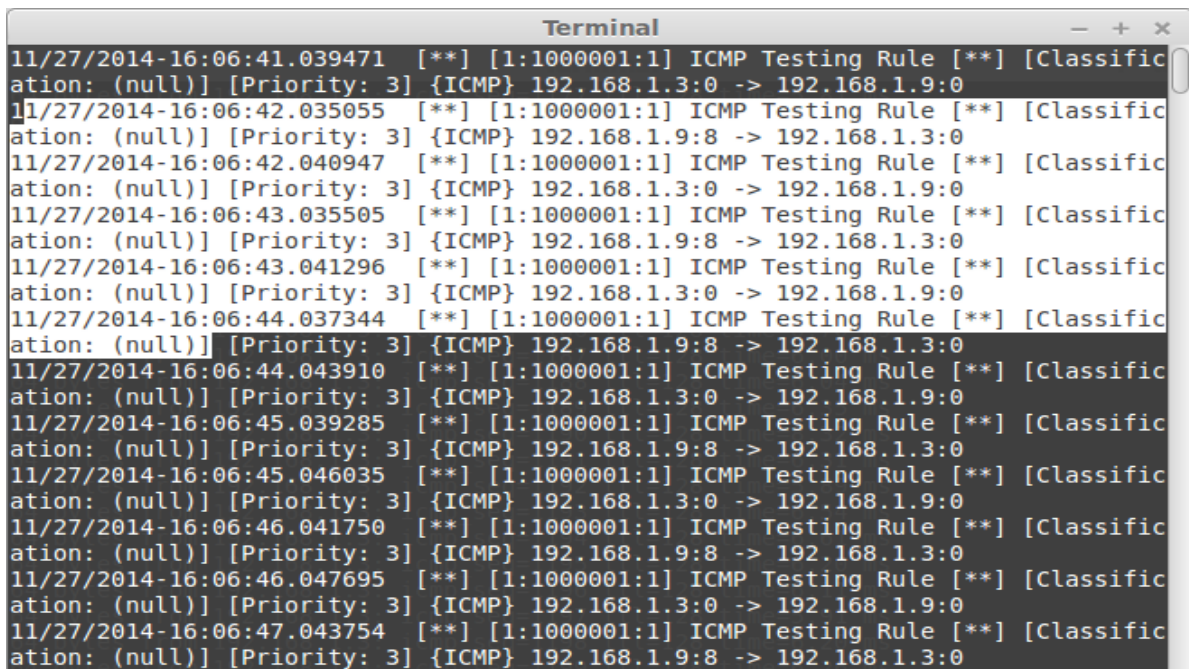
Tests run on my local network.

```
Terminal
master@master-virtual-machine ~ $ cd /var/lo
local/ lock/ log/
master@master-virtual-machine ~ $ cd /var/log/suricata/
master@master-virtual-machine /var/log/suricata $ tail -f fast.log http.log
==> fast.log <==
11/27/2014-15:27:35.007034  [**] [1:2012648:3] ET POLICY Dropbox Client Broadcas
ting [**] [Classification: Potential Corporate Privacy Violation] [Priority: 1]
{UDP} 192.168.1.5:17500 -> 255.255.255.255:17500
11/27/2014-15:27:59.317472  [**] [1:2100498:7] GPL ATTACK RESPONSE id check retu
rned root [**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP} 82.
165.177.154:80 -> 192.168.1.9:59613

==> http.log <==
11/27/2014-15:27:59.213192 www.testmyids.com [**] / [**] Wget/1.15 (linux-gnu) [
**] 192.168.1.9:59613 -> 82.165.177.154:80
```

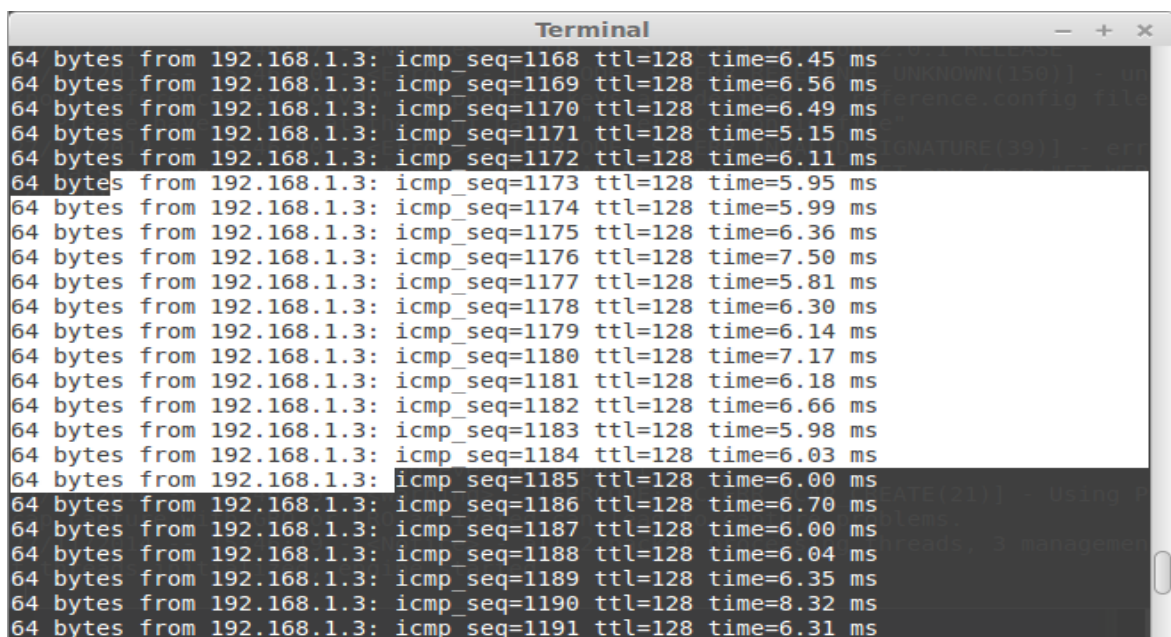
FIGURE 5-6 TEST SURICATA IDS

Test ping



```
Terminal
11/27/2014-16:06:41.039471  [**] [1:1000001:1] ICMP Testing Rule [**] [Classification: (null)] [Priority: 3] {ICMP} 192.168.1.3:0 -> 192.168.1.9:0
11/27/2014-16:06:42.035055  [**] [1:1000001:1] ICMP Testing Rule [**] [Classification: (null)] [Priority: 3] {ICMP} 192.168.1.9:8 -> 192.168.1.3:0
11/27/2014-16:06:42.040947  [**] [1:1000001:1] ICMP Testing Rule [**] [Classification: (null)] [Priority: 3] {ICMP} 192.168.1.3:0 -> 192.168.1.9:0
11/27/2014-16:06:43.035505  [**] [1:1000001:1] ICMP Testing Rule [**] [Classification: (null)] [Priority: 3] {ICMP} 192.168.1.9:8 -> 192.168.1.3:0
11/27/2014-16:06:43.041296  [**] [1:1000001:1] ICMP Testing Rule [**] [Classification: (null)] [Priority: 3] {ICMP} 192.168.1.3:0 -> 192.168.1.9:0
11/27/2014-16:06:44.037344  [**] [1:1000001:1] ICMP Testing Rule [**] [Classification: (null)] [Priority: 3] {ICMP} 192.168.1.9:8 -> 192.168.1.3:0
11/27/2014-16:06:44.043910  [**] [1:1000001:1] ICMP Testing Rule [**] [Classification: (null)] [Priority: 3] {ICMP} 192.168.1.3:0 -> 192.168.1.9:0
11/27/2014-16:06:45.039285  [**] [1:1000001:1] ICMP Testing Rule [**] [Classification: (null)] [Priority: 3] {ICMP} 192.168.1.9:8 -> 192.168.1.3:0
11/27/2014-16:06:45.046035  [**] [1:1000001:1] ICMP Testing Rule [**] [Classification: (null)] [Priority: 3] {ICMP} 192.168.1.3:0 -> 192.168.1.9:0
11/27/2014-16:06:46.041750  [**] [1:1000001:1] ICMP Testing Rule [**] [Classification: (null)] [Priority: 3] {ICMP} 192.168.1.9:8 -> 192.168.1.3:0
11/27/2014-16:06:46.047695  [**] [1:1000001:1] ICMP Testing Rule [**] [Classification: (null)] [Priority: 3] {ICMP} 192.168.1.3:0 -> 192.168.1.9:0
11/27/2014-16:06:47.043754  [**] [1:1000001:1] ICMP Testing Rule [**] [Classification: (null)] [Priority: 3] {ICMP} 192.168.1.9:8 -> 192.168.1.3:0
```

FIGURE 5-7 ICMP ALERT



```
Terminal
64 bytes from 192.168.1.3: icmp_seq=1168 ttl=128 time=6.45 ms
64 bytes from 192.168.1.3: icmp_seq=1169 ttl=128 time=6.56 ms
64 bytes from 192.168.1.3: icmp_seq=1170 ttl=128 time=6.49 ms
64 bytes from 192.168.1.3: icmp_seq=1171 ttl=128 time=5.15 ms
64 bytes from 192.168.1.3: icmp_seq=1172 ttl=128 time=6.11 ms
64 bytes from 192.168.1.3: icmp_seq=1173 ttl=128 time=5.95 ms
64 bytes from 192.168.1.3: icmp_seq=1174 ttl=128 time=5.99 ms
64 bytes from 192.168.1.3: icmp_seq=1175 ttl=128 time=6.36 ms
64 bytes from 192.168.1.3: icmp_seq=1176 ttl=128 time=7.50 ms
64 bytes from 192.168.1.3: icmp_seq=1177 ttl=128 time=5.81 ms
64 bytes from 192.168.1.3: icmp_seq=1178 ttl=128 time=6.30 ms
64 bytes from 192.168.1.3: icmp_seq=1179 ttl=128 time=6.14 ms
64 bytes from 192.168.1.3: icmp_seq=1180 ttl=128 time=7.17 ms
64 bytes from 192.168.1.3: icmp_seq=1181 ttl=128 time=6.18 ms
64 bytes from 192.168.1.3: icmp_seq=1182 ttl=128 time=6.66 ms
64 bytes from 192.168.1.3: icmp_seq=1183 ttl=128 time=5.98 ms
64 bytes from 192.168.1.3: icmp_seq=1184 ttl=128 time=6.03 ms
64 bytes from 192.168.1.3: icmp_seq=1185 ttl=128 time=6.00 ms
64 bytes from 192.168.1.3: icmp_seq=1186 ttl=128 time=6.70 ms
64 bytes from 192.168.1.3: icmp_seq=1187 ttl=128 time=6.00 ms
64 bytes from 192.168.1.3: icmp_seq=1188 ttl=128 time=6.04 ms
64 bytes from 192.168.1.3: icmp_seq=1189 ttl=128 time=6.35 ms
64 bytes from 192.168.1.3: icmp_seq=1190 ttl=128 time=8.32 ms
64 bytes from 192.168.1.3: icmp_seq=1191 ttl=128 time=6.31 ms
```

FIGURE 5-8 ICMP ALERT2

5.2.1.1 Rule Ping

```
#alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"GPL ICMP Information Request undefined code"; icode:>0; itype:15; classtype:misc-activity; sid:2100418; rev:8;)
```

5.3 Detect Port Scan

Tests run on my local network.

```
root@kali:~# nmap -sF 192.168.1.68

Starting Nmap 6.47 ( http://nmap.org ) at 2015-04-27 14:49 EDT
Nmap scan report for 192.168.1.68
```

FIGURE 5-9 PORT SCAN

Tests run on my local network.

```
root@kali:~# nmap -sX -p 80 192.168.1.68

Starting Nmap 6.47 ( http://nmap.org ) at 2015-04-27 14:54 EDT
```

FIGURE 5-10 PORT SCAN 2

```
==> fast.log <==
04/28/2015-21:03:59.740530  [**] [1:1000003:1] UDP Testing Rule [**] [Classifica
tion: (null)] [Priority: 3] {UDP} 192.168.1.69:137 -> 192.168.1.255:137
04/28/2015-21:04:02.460443  [**] [1:1000003:1] UDP Testing Rule [**] [Classifica
tion: (null)] [Priority: 3] {UDP} 192.168.1.76:52915 -> 192.168.1.1:53
04/28/2015-21:04:07.482336  [**] [1:1000002:1] TCP Testing Rule [**] [Classifica
tion: (null)] [Priority: 3] {TCP} 192.168.1.76:62002 -> 192.168.1.68:80
04/28/2015-21:04:08.591982  [**] [1:1000002:1] TCP Testing Rule [**] [Classifica
tion: (null)] [Priority: 3] {TCP} 192.168.1.76:62003 -> 192.168.1.68:80
```

FIGURE 5-11 ALERT PORT SCAN

5.3.1 Rule Port Scan

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"TCP Testing Rule";
flow:stateless; flags:F,12; classtype:attempted-recon; sid:621; rev:7;)
```

5.4 Detect Syn flood

Tests run on my local network.


```
Terminal
NACK resend with different ack [**] [Classification: (null)] [Priority: 3] {TCP}
192.168.1.11:80 -> 192.168.1.14:33258
12/02/2014-16:46:53.452087 [**] [1:2210004:1] SURICATA STREAM 3way handshake SY
NACK resend with different ack [**] [Classification: (null)] [Priority: 3] {TCP}
192.168.1.11:80 -> 192.168.1.14:33256
12/02/2014-16:46:53.453960 [**] [1:2210004:1] SURICATA STREAM 3way handshake SY
NACK resend with different ack [**] [Classification: (null)] [Priority: 3] {TCP}
192.168.1.11:80 -> 192.168.1.14:33316
12/02/2014-16:46:53.649608 [**] [1:2210004:1] SURICATA STREAM 3way handshake SY
NACK resend with different ack [**] [Classification: (null)] [Priority: 3] {TCP}
192.168.1.11:80 -> 192.168.1.14:33251
12/02/2014-16:46:53.650692 [**] [1:2210004:1] SURICATA STREAM 3way handshake SY
NACK resend with different ack [**] [Classification: (null)] [Priority: 3] {TCP}
192.168.1.11:80 -> 192.168.1.14:35120
12/02/2014-16:46:53.652102 [**] [1:2210004:1] SURICATA STREAM 3way handshake SY
NACK resend with different ack [**] [Classification: (null)] [Priority: 3] {TCP}
192.168.1.11:80 -> 192.168.1.14:33253
12/02/2014-16:46:53.652173 [**] [1:2210004:1] SURICATA STREAM 3way handshake SY
NACK resend with different ack [**] [Classification: (null)] [Priority: 3] {TCP}
192.168.1.11:80 -> 192.168.1.14:33259
12/02/2014-16:46:53.652491 [**] [1:2210004:1] SURICATA STREAM 3way handshake SY
NACK resend with different ack [**] [Classification: (null)] [Priority: 3] {TCP}
192.168.1.11:80 -> 192.168.1.14:33255
12/02/2014-16:46:53.655071 [**] [1:2210004:1] SURICATA STREAM 3way handshake SY
```

FIGURE 5-12 ALERT SYN FLOOD

5.4.1 Rule Syn Flood

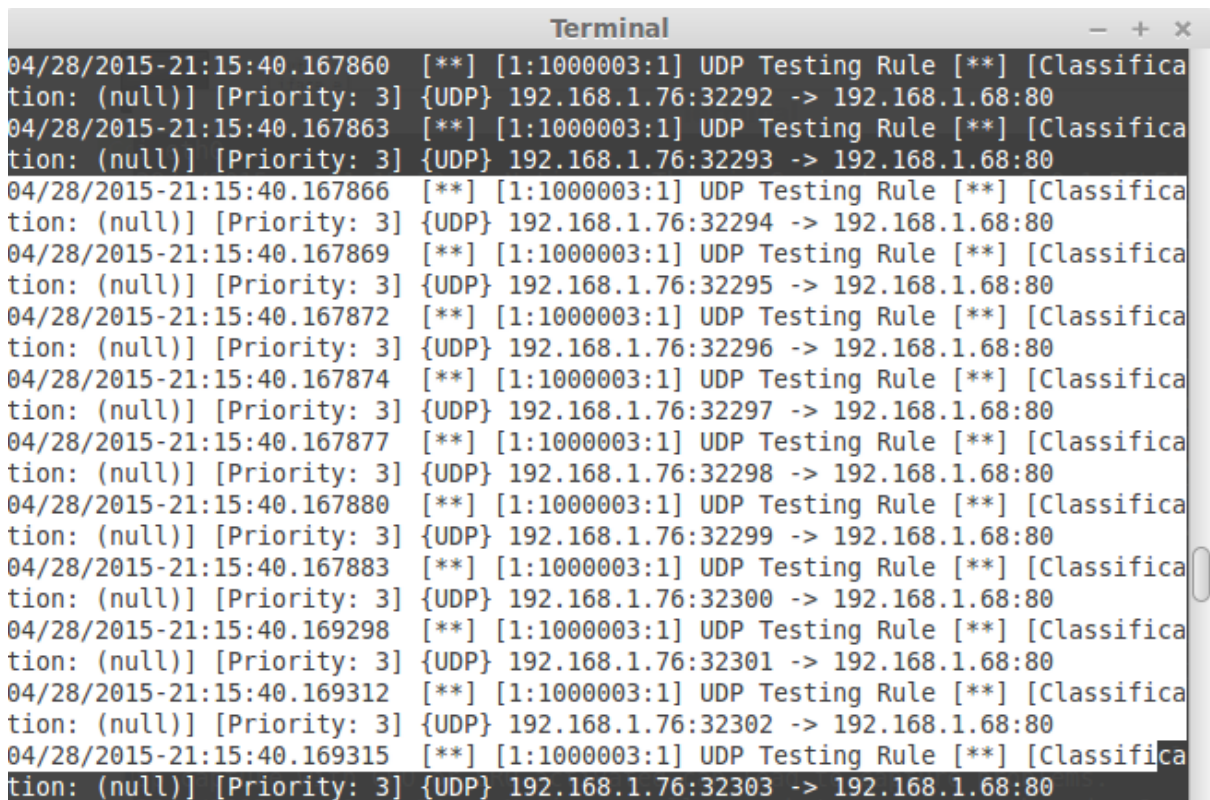
alert tcp any any -> any any (msg:"SURICATA STREAM 3way handshake SYNACK resend with different ack"; stream-event:3whs_synack_resend_with_different_ack; sid:2210004; rev:1;)

5.5 Detect UDP flood

Tests run on my local network.

```
root@kali:~# hping3 --udp --flood -p 80 192.168.1.68
HPING 192.168.1.68 (eth0 192.168.1.68): udp mode set, 28 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
--- 192.168.1.68 hping statistic ---
305017 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

FIGURE 5-13 UDP FLOOD

A terminal window titled "Terminal" with standard window controls (-, +, x) in the top right. The terminal displays a continuous stream of alert messages. Each message line contains a timestamp (e.g., 04/28/2015-21:15:40.167860), a severity level (**), a rule ID ([1:1000003:1]), the rule name (UDP Testing Rule), another severity level (**), and classification details ([Classification: (null)] [Priority: 3]). The payload of each alert is {UDP} 192.168.1.76:32292 -> 192.168.1.68:80, with the destination port varying slightly (e.g., 32292, 32293, 32294, etc.). The messages are repeated rapidly, illustrating a flood of alerts.

```
04/28/2015-21:15:40.167860  [**] [1:1000003:1] UDP Testing Rule [**] [Classification: (null)] [Priority: 3] {UDP} 192.168.1.76:32292 -> 192.168.1.68:80
04/28/2015-21:15:40.167863  [**] [1:1000003:1] UDP Testing Rule [**] [Classification: (null)] [Priority: 3] {UDP} 192.168.1.76:32293 -> 192.168.1.68:80
04/28/2015-21:15:40.167866  [**] [1:1000003:1] UDP Testing Rule [**] [Classification: (null)] [Priority: 3] {UDP} 192.168.1.76:32294 -> 192.168.1.68:80
04/28/2015-21:15:40.167869  [**] [1:1000003:1] UDP Testing Rule [**] [Classification: (null)] [Priority: 3] {UDP} 192.168.1.76:32295 -> 192.168.1.68:80
04/28/2015-21:15:40.167872  [**] [1:1000003:1] UDP Testing Rule [**] [Classification: (null)] [Priority: 3] {UDP} 192.168.1.76:32296 -> 192.168.1.68:80
04/28/2015-21:15:40.167874  [**] [1:1000003:1] UDP Testing Rule [**] [Classification: (null)] [Priority: 3] {UDP} 192.168.1.76:32297 -> 192.168.1.68:80
04/28/2015-21:15:40.167877  [**] [1:1000003:1] UDP Testing Rule [**] [Classification: (null)] [Priority: 3] {UDP} 192.168.1.76:32298 -> 192.168.1.68:80
04/28/2015-21:15:40.167880  [**] [1:1000003:1] UDP Testing Rule [**] [Classification: (null)] [Priority: 3] {UDP} 192.168.1.76:32299 -> 192.168.1.68:80
04/28/2015-21:15:40.167883  [**] [1:1000003:1] UDP Testing Rule [**] [Classification: (null)] [Priority: 3] {UDP} 192.168.1.76:32300 -> 192.168.1.68:80
04/28/2015-21:15:40.169298  [**] [1:1000003:1] UDP Testing Rule [**] [Classification: (null)] [Priority: 3] {UDP} 192.168.1.76:32301 -> 192.168.1.68:80
04/28/2015-21:15:40.169312  [**] [1:1000003:1] UDP Testing Rule [**] [Classification: (null)] [Priority: 3] {UDP} 192.168.1.76:32302 -> 192.168.1.68:80
04/28/2015-21:15:40.169315  [**] [1:1000003:1] UDP Testing Rule [**] [Classification: (null)] [Priority: 3] {UDP} 192.168.1.76:32303 -> 192.168.1.68:80
```

FIGURE 5-14 ALERT UDP FLOOD

5.5.1 Rule UDP Flood

```
alert udp $EXTERNAL_NET any -> $HOME_NET any (msg:"UDP Testing Rule ";
threshold: type threshold, track by_src, count 10000, seconds 5; sid: 10000002; rev: 1;)
```

5.6 Suricata vs Snort

For a long time, Snort has been the standard for open source Intrusion Detection Systems (IDS/IPS) [62]. Its engine combines the advantages of signatures, protocols, and anomaly-based inspection and has become the most common deployed IDS/IPS in the world.

Suricata, a new and less widely product developed by the Open Information Security Foundation (OISF) [62]. It is based on signatures but incorporates revolutionary techniques. This engine embeds an http normalizer and parser that provide very advanced processing of HTTP streams.

Both Snort and Suricata [62] are based on sets of rules. Most of the tests have shown that VRT Snort and Emerging Threats rules are complementary and are both needed to optimize the detection of all attack types. Furthermore, both Snort and Suricata have demonstrated their ability to detect attacks based on signatures from rules.

Suricata offers new features that Snort could implement in the future like multi-threading support, capture accelerators but suffers from a lack of documentation [62]. In addition, Suricata doesn't accept some rules from VRT::Snort and Emerging Threats due to incompatibilities. The support of these missing keywords should be implemented in future versions of Suricata.

On the other hand, Snort is mature [62]. It remains a very powerful and effectiveness IDS/IPS, very well documented over the net and that properly detects most of the malwares and evasion techniques. Its preprocessors are very useful powerful for reassembling fragmented packets.

Param	Suricata	Snort
IPS feature	optional while compiling (--enable- nfqueue)	Snort inline or snort used with -Q option
Rules	<ul style="list-style-type: none"> • VRT::Snort rules • EmergingThreats rules 	<ul style="list-style-type: none"> • VRT::Snort rules • SO rules • EmergingThreats rules
Threads	Multi-thread	Single-thread
Ease of install	Not available from packages. Manual installation.	Relatively straightforward. Installation also available from packages.
Documentation	Few resources on the Internet	Well documented on the official website and over the Internet
Event logging	Flat file, database, unified2 logs for barnyard	
IPv6 support	Fully supported	Supported when compiled with -- enable-ipv6 option.
Capture accelerators	PF_RING, packet capture accelerator	None, use of libpcap
Configuration file	suricata.yaml , classification.config, reference.config, threshold.config	snort.conf, threshold.conf
Offline analysis (pcap file)	yes	
Frontends	Sguil, Aanval, BASE, FPCGUI (Full Packet Capture GUI), Snortsnarf	

Table 5-1: Global Overview

Chapter 6 Bro

6.1 Bro IDS

Bro is an open-source, [36, 63] Unix-based Network Intrusion Detection System (NIDS) that passively monitors network traffic and looks for an abnormal action. Bro detects intrusions by first parsing network traffic to extract its application level significant and then executing event-oriented analyzers that correlate the activity with patterns deemed troublesome.

Bro uses a particular policy language [63] that permits a site to tailor Bro's operation, both as site policies develop and as new attacks are discovered. If Bro detects something of interest, it can be commandment to either produce a log entry, alert the operator in real-time.

A bro script [63] could be written to keep track of user attempts against the application and create an alert if it overdraws a threshold value. This requires the intrusion detection system to not only comprehend the protocol but also keep track of failed user sessions against the application. This crucial feature of Bro to understand the higher order application details gives it a distinct advantage against signature based intrusion detection systems.

6.1.1 Managing Bro with Bro control

Bro Control [63] is an interactive shell for easily operating or managing Bro installations on a system or even across multiple systems in a traffic-monitoring cluster.

6.1.2 Browsing Log Files

By default, logs [63] are written out in human-readable (ASCII) format and data is organized into columns.

6.2 Bro Scripts

Bro includes an event-driven scripting language [63] that provides the primary means for an organization to extend and customize Bro's operability. Virtually all of the output generated by Bro is, in fact, generated by Bro scripts. It's almost easier to think about that Bro will be an entity behind the scenes processing connections and generating facts while Bro's scripting language is the medium through which we can succeed communication. Bro scripts effectively shall notify Bro that should there be an event of a type we define, and then let us

have the information about the connection so we can execute some function on. .

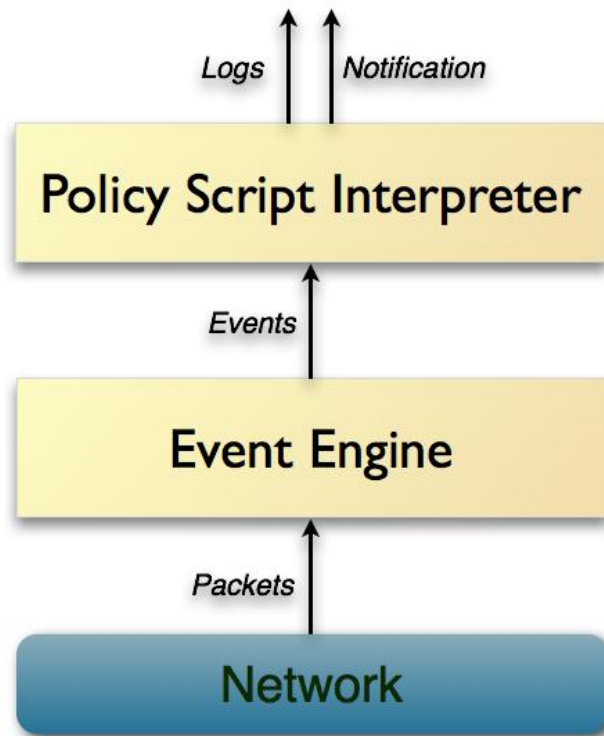
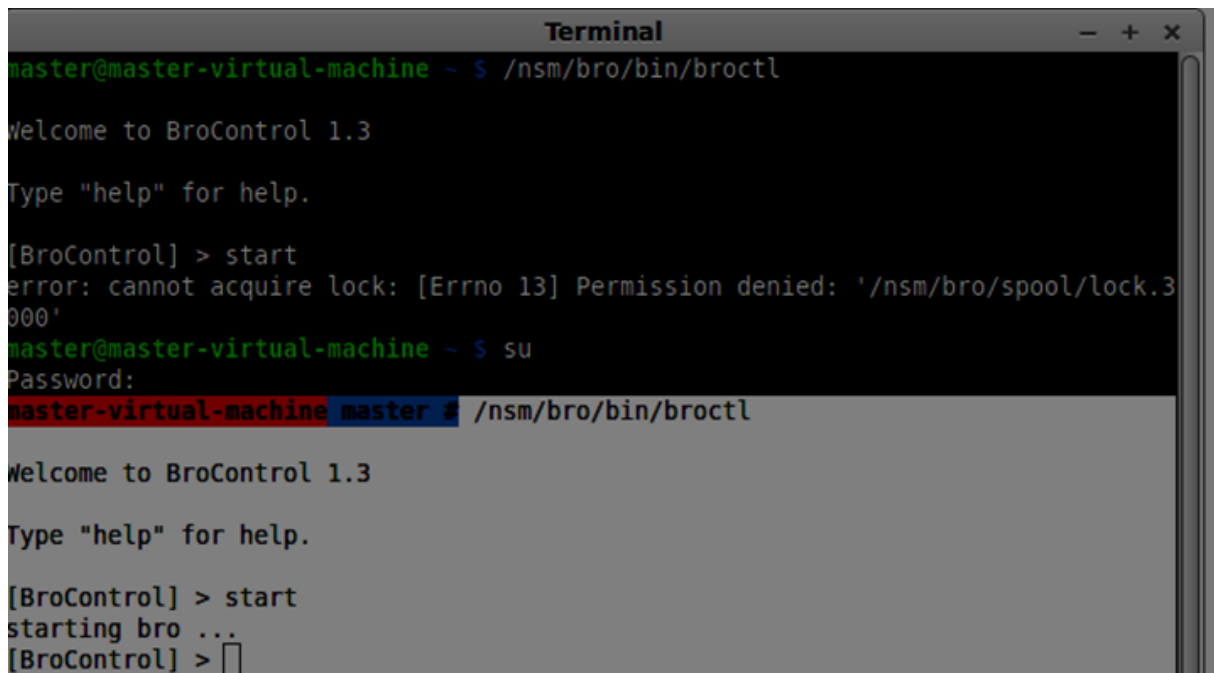


FIGURE 6-1 BRO ARCHITECTURE

Architecturally, Bro is layered into two major levels [63]. Its event engine decreases the inbound traffic stream into a series of higher-level events. These events represent network function in policy neutral terms, they describe what has been seen, but not why, or whether it is considerable.

Such semantics [63] are instead derived by Bro's second main element, the script interpreter, which performs a set of event users written in Bro's custom scripting language. These scripts can implement a site's security policy, i.e., what activities to take when the monitor detects different types of activity. More generally they can log any desired properties and statistics from the input traffic.

Start Bro



```
Terminal
master@master-virtual-machine ~ $ /nsm/bro/bin/broctl
Welcome to BroControl 1.3
Type "help" for help.
[BroControl] > start
error: cannot acquire lock: [Errno 13] Permission denied: '/nsm/bro/spool/lock.3000'
master@master-virtual-machine ~ $ su
Password:
master-virtual-machine master # /nsm/bro/bin/broctl
Welcome to BroControl 1.3
Type "help" for help.
[BroControl] > start
starting bro ...
[BroControl] > █
```

FIGURE 6-2 START BRO

6.3 Bro Log Files

Bro is shipped with an interactive shell for management purpose: Bro Control [63]. This application is able to control and monitor the Bro installation. In a cluster and multi Bro installation case Bro Control is crucial. When using Bro Control, Bro creates logs in the directory \$BROHOME/log. The directory is \$BROHOME/log/current but logs are often moved to \$BROHOME/log/YYYY-MM-DD. These log files are in clear text ASCII unless default configuration is changed [63]. When running from CLI, all log files are created in actual directory where we start Bro. The following log files are always created: conn.log, loaded_scripts.log and notice_policy.log. These filenames reveal much of the actual log file content, but some more description is necessary [63]:

- Conn.log consists of the complete connection log during Bro's run time.
- Loaded_scripts.log shows Bro scripts that were loaded during Bro startup.
- Notice_policy.bro shows the current Bro Notice policy.

Bro create several new log files during run time. This overview shows more general and internal log files [63]:

- Communication.log logs for Bro's internal communication between remote and central instances, clusters etc.
- Conn-summary.log generated when Bro is terminated. Post processing connection summaries.

- Known_hosts.log hosts that have performed complete TCP handshake.
- Notice.log notices that Bro rises.
- Reporter.log internal messages and warnings errors for troubleshooting.

Bro also creates a lot of log files that are protocol/service specific [63]:

- Dns.log log over DNS queries.
- Dpd.log log over what port/service dependent dynamic protocol detection analysis that has been activated.
- Http.log log over http request and responses including metadata.
- Software.log reports known and recognized software detected from protocol analyzers.
- Weird.log notices that Bro has tagged as weird. Odd protocol behavior will be logged here. A log of unexpected protocol-level activity.

```

loaded_scripts.19:21:50-19:22:24.log (/nsm/bro/logs/2015-04-29) - gedit
File Edit View Search Tools Documents Help
loaded_scripts.19:21:50-19:22:24.log x
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path _loaded_scripts
#open 2015-04-29-19-21-50
#fields name
#types string
/nsm/bro/share/bro/base/init-bare.bro
/nsm/bro/share/bro/base/bif/const.bif.bro
/nsm/bro/share/bro/base/bif/types.bif.bro
/nsm/bro/share/bro/base/bif/strings.bif.bro
/nsm/bro/share/bro/base/bif/bro.bif.bro
/nsm/bro/share/bro/base/bif/reporter.bif.bro
/nsm/bro/share/bro/base/bif/plugins/Bro_SNMP.types.bif.bro
/nsm/bro/share/bro/base/bif/event.bif.bro
/nsm/bro/share/bro/base/bif/plugins/_load_.bro
/nsm/bro/share/bro/base/bif/plugins/Bro_ARP.events.bif.bro
/nsm/bro/share/bro/base/bif/plugins/Bro_AYIYA.events.bif.bro
/nsm/bro/share/bro/base/bif/plugins/Bro_BackDoor.events.bif.bro
/nsm/bro/share/bro/base/bif/plugins/Bro_BitTorrent.events.bif.bro
/nsm/bro/share/bro/base/bif/plugins/Bro_ConnSize.events.bif.bro
/nsm/bro/share/bro/base/bif/plugins/Bro_DCE_RPC.events.bif.bro
/nsm/bro/share/bro/base/bif/plugins/Bro_DHCP.events.bif.bro
/nsm/bro/share/bro/base/bif/plugins/Bro_DNP3.events.bif.bro

```

FIGURE 6-3 BRO SCRIPTS

6.3.1 Signature main.bro

Actions for a signature [63]:

```
const actions: table[string] of Action = {
    ["unspecified"] = SIG_IGNORE, # place-holder
} &redef &default = SIG_ALARM;
## Signature IDs that should always be ignored.
const ignored_ids = /NO_DEFAULT_MATCHES/ &redef;
## Generate a notice if, for a pair [orig, signature], the number of
## different responders has reached one of the thresholds.
const horiz_scan_thresholds = { 5, 10, 50, 100, 500, 1000 } &redef;
## Generate a notice if, for a pair [orig, resp], the number of
## different signature matches has reached one of the thresholds.
const vert_scan_thresholds = { 5, 10, 50, 100, 500, 1000 } &redef;
## Generate a notice if a :bro:enum:`Signatures::SIG_COUNT_PER_RESP`
## signature is triggered as often as given by one of these thresholds.
const count_thresholds = { 5, 10, 50, 100, 500, 1000, 10000, 1000000, }
&redef;
## The interval between when :bro:enum:`Signatures::Signature_Summary`
## notices are generated.
const summary_interval = 1 day &redef;
## This event can be handled to access/alter data about to be logged
## to the signature logging stream.
## rec: The record of signature data about to be logged.
global log_signature: event(rec: Info);
}
if ( action == SIG_ALARM_ONCE )
{
    if ( [sig_id] !in did_sig_log )
    {
        notice = T;
        add did_sig_log[sig_id];
    }
}
if ( notice )
    NOTICE([$note=Sensitive_Signature,
            $conn=state$conn, $src=src_addr,
            $dst=dst_addr, $msg=fmt("%s: %s", src_addr, msg),
            $sub=data]);
Log::write(Signatures::LOG,
           [$ts=network_time(),
            $note=Multiple_Signatures,
            $src_addr=orig,
```



```
$dst_addr=resp, $sig_id=sig_id,
```

6.3.2 Reporter main.bro

This framework is intended to create an output and filtering path internal messages/warnings/errors. It should typically be loaded to log such messages to a file in a standard way [63]:

```
export {
    ## The reporter logging stream identifier.
    redef enum Log::ID += { LOG };
    ## An indicator of reporter message severity.
    type Level: enum {
        ## Informational, not needing specific attention.
        INFO,
        ## Warning of a potential problem.
        WARNING,
        ## A non-fatal error that should be addressed, but doesn't
        ## terminate program execution.
        ERROR
    };
    ## The record type which contains the column fields of the reporter log.
    type Info: record {
        ## The network time at which the reporter event was generated.
        ts:      time    &log;
        ## The severity of the reporter message.
        level:   Level  &log;
        ## An info/warning/error message that could have either been
        ## generated from the internal Bro core or at the scripting-layer.
        message: string &log;
        ## This is the location in a Bro script where the message originated.
        ## Not all reporter messages will have locations in them though.
        location: string &log &optional;
    };
}
event bro_init() &priority=5
{
    Log::create_stream(Reporter::LOG, [$columns=Info]);
}
event reporter_info(t: time, msg: string, location: string) &priority=-5
{

```

```

        Log::write(Reporter::LOG,      [$ts=t,      $level=INFO,      $message=msg,
$location=location]);
    }
event reporter_warning(t: time, msg: string, location: string) &priority=-5
    {
        Log::write(Reporter::LOG,      [$ts=t,      $level=WARNING,      $message=msg,
$location=location]);

event reporter_error(t: time, msg: string, location: string) &priority=-5
    {
        Log::write(Reporter::LOG,      [$ts=t,      $level=ERROR,      $message=msg,
$location=location]);
    }

```

6.3.3 Communication main.bro

Main.bro [63]:

```

module Communication;
export {
    ## The communication logging stream identifier.
    redef enum Log::ID += { LOG };
    ## Which interface to listen on. The addresses ``0.0.0.0`` and ``[:]:``
    ## are wildcards.
    const listen_interface = 0.0.0.0 &redef;
    ## Which port to listen on. Note that BroControl sets this
    ## automatically.
    const listen_port = 47757/tcp &redef;
    ## This defines if a listening socket should use SSL.
    const listen_ssl = F &redef;
    ## Defines if a listening socket can bind to IPv6 addresses.
    ## Defines the interval at which to retry binding to
    ## :bro:id:`Communication::listen_interface` on
    ## :bro:id:`Communication::listen_port` if it's already in use.
    const listen_retry = 30 secs &redef;
    ## Default compression level. Compression level is 0-9, with 0 = no
    ## compression.
    global compression_level = 0 &redef;
    ## A record type containing the column fields of the communication log.
    event bro_init() &priority=5

```

```

    {
    Log::create_stream(Communication::LOG, [$columns=Info]);
    }

function do_script_log_common(level: count, src: count, msg: string)
    {
    Log::write(Communication::LOG, [$ts = network_time(),
    $level = (level == REMOTE_LOG_INFO ? "info" :
"error"),
    $src_name = src_names[src],
    $peer = get_event_peer()$descr,
    $message = msg]);
    }
# This is a core generated event.
event remote_log(level: count, src: count, msg: string)
    {
    do_script_log_common(level, src, msg);
    }

# Actually initiate the connections that need to be established.
event bro_init() &priority = -10 # let others modify nodes
    {
    if ( |nodes| > 0 )
        enable_communication();

    for ( tag in nodes )
        {
        if ( ! nodes[tag]$connect )
            next;

        connect_peer(tag);
        }
    }

```

6.4 Detect Port scan

Tests run on my local network.

Protocols	States	Sources	Destinations	Services
6	84.5% REJ	192.168.1.14#1	71.9% 192.168.1.11#2	75.6%
17	15.5% SF	192.168.1.11#3	16.8% ff02::1:3#4	13.9%
1	0.1% S0	192.168.1.6#5	5.5% 224.0.0.252#6	5.5%
ssl	53	3.9% fe80::9de0:3a80:63a9:20e8#7	4.4% 192.168.1.1#8	3.8%
	137	1.9% 192.168.1.3#9	0.5% 70.42.23.121#10	2.8%
	138	0.6% 192.168.1.15#11	0.4% 192.168.1.255#12	2.6%
	1900	0.4% 192.168.1.5#13	0.4% 192.150.187.43#14	1.7%
	17500	0.2% 0.0.0.0#15	0.1% 131.243.2.77#16	1.3%
	547	0.2%	199.96.57.7#17	0.9%
	67	0.1%	216.58.208.99#18	0.5%

FIGURE 6-4 CONN-SUMMARY.LOG

```

conn.19:26:34-19:50:38.log [Read-Only] (/nsm/bro/logs/2014-12-01) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
conn.19:26:34-19:50:38.log x
D      2      144      0      0      (empty)
1417454780.446524 CXfHq82nSQWT26pDUa 192.168.1.6 137 192.168.1.255 137
udp dns 5.998821 200 0 S0 T 0 D 4 312
0 0 (empty)
1417454798.685329 C6FzlfFYPeALQsax 192.168.1.11 41227 199.16.156.201 443
tcp ssl 0.385833 517 137 SF T 0 ShADFadRf 6
813 4 353 (empty)
1417454795.573376 CBZZg73vDJGqiIqQP7 192.168.1.11 57409 192.150.187.43 80
tcp http 5.485343 313 533 SF T 0 ShADadFf 5
581 5 801 (empty)
1417454796.224523 CFfVCHPP1LPhynWV3 192.168.1.11 37311 192.150.187.43 443
tcp ssl 6.004940 709 20849 SF T 0 ShADadFf 18
1653 21 21949 (empty)
1417454797.746559 CCZLwL1n0qwCLfSSb 192.168.1.11 47325 192.168.1.1 53
udp dns 0.024494 49 329 SF T 0 Dd 1 77
1 357 (empty)
1417454799.709849 CivTIJ1VaLuf6dSqk7 192.168.1.11 54342 199.96.57.7 443
tcp ssl 5.439424 599 141 SF T 0 ShADadFf 8
1023 7 513 (empty)
1417454799.708920 Cw9GSl2lUuyP3zeu43 192.168.1.11 54341 199.96.57.7 443
tcp ssl 5.443043 599 141 SF T 0 ShADadFf 8
1023 7 513 (empty)
1417454799.708070 CeXGmw49Am7m6fHqKl 192.168.1.11 54340 199.96.57.7 443
tcp ssl 5.446864 599 141 SF T 0 ShADadFf 8

```

FIGURE 6-5 CONN.LOG

```

communication.19:26:20-19:50:38.log [Read-Only] (/nsm/bro/logs/2014-12-01) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
communication.19:26:20-19:50:38.log x
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path communication
#open 2014-12-01-19-26-20
#fields ts peer src_name connected_peer_desc connected_peer_addr
connected_peer_port level message
#types time string string string addr port string string
0.000000 bro parent - - - info raised pipe's socket buffer size
from 208K to 1024K
0.000000 bro parent - - - info communication started, parent
pid is 26096, child pid is 26098
1417454780.111839 bro child - - - info listening on
0.0.0.0:47760 (clear)
1417454780.111839 bro child - - - info listening on [::]:47760
(clear)
1417454783.341049 bro child - - - info [#10000/127.0.0.1:59472]
accepted clear connection
1417454783.342245 bro parent - - - info [#10000/127.0.0.1:59472]
added peer
1417454783.342245 bro parent - - - info [#10000/127.0.0.1:59472]
peer connected
1417454783.342245 bro parent - - - info [#10000/127.0.0.1:59472]
phase: version

```

FIGURE 6-6 COMMUNICATION.LOG

```

notice.19:45:21-19:50:38.log (/nsm/bro/logs/2014-12-01) - gedit
File Edit View Search Tools Documents Help
Open Save Undo Cut Copy Paste Find
notice.19:45:21-19:50:38.log x
#separator \x09
#set separator ,
#empty_field (empty)
#unset_field -
#path notice
#open 2014-12-01-19-45-21
#fields ts uid id.orig_h id.orig_p id.resp_h id.resp_p fuid
file_mime_type file_desc proto note msg sub src dst p n
peer_descr actions suppress_for dropped remote_location.country_code
remote_location.region remote_location.city remote_location.latitude
remote_location.longitude
#types time string addr port addr port string string string enum enum
string string addr addr port count string set[enum] interval bool
string string string double double
1417455921.272559 - - - - - -
Scan::Port_Scan 192.168.1.14 scanned at least 15 unique ports of host 192.168.1.11 in 0m0s
local 192.168.1.14 192.168.1.11 - - bro Notice::ACTION_LOG
3600.000000 F - - - - -
#close 2014-12-01-19-50-38

```

FIGURE 6-7 NOTICE.LOG

```

weird.19:26:34-19:50:38.log [Read-Only] (/nsm/bro/logs/2014-12-01) - gedit
File Edit View Search Tools Documents Help
Open Save Undo Cut Copy Paste Find
weird.19:26:34-19:50:38.log x
1417455531.450072 CFvTFHPQyo18fM3ub 192.168.1.11 56127 94.31.29.192 80
above_hole_data_without_any_acks - F bro
1417455598.046253 C39WB53toBLAdh9TY 192.168.1.11 50072 129.241.16.59 80
above_hole_data_without_any_acks - F bro
1417455605.002113 CDjYUVB7U1nqozKk 192.168.1.11 41591 70.42.23.121 80
window_recision - F bro
1417455612.678090 CUFqVo2vnXkdzpBvN2 192.168.1.11 41595 70.42.23.121 80
window_recision - F bro
1417455616.738420 CyiZHJ22iFbXhhNEH4 192.168.1.11 41602 70.42.23.121 80
window_recision - F bro
1417455616.878073 Cs3JYtuJuSrg2Dtbk 192.168.1.11 41626 70.42.23.121 80
window_recision - F bro
1417455617.242115 CMCMQrAvRiAl0JjE1 192.168.1.11 41617 70.42.23.121 80
window_recision - F bro
1417455617.394258 CelpYI2iqcG10NTbXf 192.168.1.11 41623 70.42.23.121 80
window_recision - F bro
1417455720.719327 - - - - dns_unmatched_msg -
F bro
1417455725.830221 CK30Bl159oTIHHGX5k 192.168.1.11 37472 192.150.187.43 443
above_hole_data_without_any_acks - F bro
1417455726.410344 C8jTL91f4KJRsqbEv4 192.168.1.11 37471 192.150.187.43 443
above_hole_data_without_any_acks - F bro
1417455915.641937 - - - - dns_unmatched_msg -
F bro
1417455984.145880 - - - - dns_unmatched_msg -

```

FIGURE 6-8 WEIRD.LOG

6.4.1 Scan.bro

Scan.bro [63]:

```
@load base/frameworks/notice
@load base/frameworks/sumstats
@load base/utils/time
module Scan;
export {
    redef enum Notice::Type += {
        ## Address scans detect that a host appears to be scanning some
        ## number of destinations on a single port. This notice is
        ## generated when more than :bro:id:`Scan::addr_scan_threshold`
        ## unique hosts are seen over the previous
        ## :bro:id:`Scan::addr_scan_interval` time range.
        Address_Scan,
        ## :bro:id:`Scan::port_scan_threshold`
        ## unique ports on a single host over the previous
        ## :bro:id:`Scan::port_scan_interval` time range.
        Port_Scan,
    };
    ## Failed connection attempts are tracked over this time interval for
    ## the address scan detection. A higher interval will detect slower
    ## scanners, but may also yield more false positives.
    const addr_scan_interval = 5min &redef;
    ## Failed connection attempts are tracked over this time interval for
    ## the port scan detection. A higher interval will detect slower
    ## scanners, but may also yield more false positives.
    const port_scan_interval = 5min &redef;
    ## The threshold of the unique number of hosts a scanning host has to
    ## have failed connections with on a single port.
    const addr_scan_threshold = 25.0 &redef;
    ## The threshold of the number of unique ports a scanning host has to
    ## have failed connections with on a single victim host.
    const port_scan_threshold = 15.0 &redef;
    global Scan::addr_scan_policy: hook(scanner: addr, victim: addr,
scanned_port: port);
    global Scan::port_scan_policy: hook(scanner: addr, victim: addr,
scanned_port: port);
```

```

}

event bro_init() &priority=5
{
    local      r1:      SumStats::Reducer      =      [stream="scan.addr.fail",
$apply=set(SumStats::UNIQUE), $unique_max=double_to_count(addr_scan_threshold+2)];
    SumStats::create([$name="addr-scan",
                    $epoch=addr_scan_interval,
                    $reducers=set(r1),
                    $threshold_val(key:      SumStats::Key,      result:
SumStats::Result) =
                        {
                            return result["scan.addr.fail"]$unique+0.0;
                        },
                    # $threshold_func=check_addr_scan_threshold,
                    $threshold=addr_scan_threshold,
                    $threshold_crossed(key:      SumStats::Key,      result:
SumStats::Result) =
                        {
                            local r = result["scan.addr.fail"];
                            local side = Site::is_local_addr(key$host) ? "local" :
"remote";
                            local dur = duration_to_mins_secs(r$end-r$begin);
                            local message=fmt("%s scanned at least %d unique hosts on
port %s in %s", key$host, r$unique, key$str, dur);
                            NOTICE([$note=Address_Scan,
                                    $src=key$host,
                                    $p=to_port(key$str),
                                    $sub=side,
                                    $msg=message,
                                    $identifier=cat(key$host)]);
                        }]);

    # Note: port scans are tracked similar to: table[src_ip, dst_ip] of
set(port);

    local      r2:      SumStats::Reducer      =      [stream="scan.port.fail",
$apply=set(SumStats::UNIQUE), $unique_max=double_to_count(port_scan_threshold+2)];
    SumStats::create([$name="port-scan",
                    $epoch=port_scan_interval,
                    $reducers=set(r2),
                    $threshold_val(key:      SumStats::Key,      result:
SumStats::Result) =

```

```

        {
            return result["scan.port.fail"]$unique+0.0;
        },
        $threshold=port_scan_threshold,
        $threshold_crossed(key:          SumStats::Key,          result:
SumStats::Result) =
        {
            local r = result["scan.port.fail"];
            local side = Site::is_local_addr(key$host) ? "local" :
"remote";

            local dur = duration_to_mins_secs(r$end-r$begin);
            local message = fmt("%s scanned at least %d unique ports
of host %s in %s", key$host, r$unique, key$str, dur);
            NOTICE([$note=Port_Scan,
                    $src=key$host,
                    $dst=to_addr(key$str),
                    $sub=side,
                    $msg=message,
                    $identifier=cat(key$host)]);
        });
    }

function add_sumstats(id: conn_id, reverse: bool)
    {
        local scanner      = id$orig_h;
        local victim      = id$resp_h;
        local scanned_port = id$resp_p;

        if ( reverse )
            {
                scanner      = id$resp_h;
                victim      = id$orig_h;
                scanned_port = id$orig_p;
            }

        if ( hook Scan::addr_scan_policy(scanner, victim, scanned_port) )
            SumStats::observe("scan.addr.fail",          [$host=scanner,
$str=cat(scanned_port)], [$str=cat(victim)]);

        if ( hook Scan::port_scan_policy(scanner, victim, scanned_port) )

```



```

        SumStats::observe("scan.port.fail", [$host=scanner, $str=cat(victim)],
[$str=cat(scanned_port)]);
    }

function is_failed_conn(c: connection): bool
{
    # Sr || ( (hR || ShR) && (data not sent in any direction) )
    if ( (c$orig$state == TCP_SYN_SENT && c$resp$state == TCP_RESET) ||
        ((c$orig$state == TCP_RESET && c$resp$state == TCP_SYN_ACK_SENT) ||
         (c$orig$state == TCP_RESET && c$resp$state == TCP_ESTABLISHED && "S"
in c$history )
         ) && /[Dd]/ !in c$history )
        )
        return T;
    return F;
}

function is_reverse_failed_conn(c: connection): bool
{
    # reverse scan i.e. conn dest is the scanner
    # sR || ( (Hr || sHr) && (data not sent in any direction) )
    if ( (c$resp$state == TCP_SYN_SENT && c$orig$state == TCP_RESET) ||
        ((c$resp$state == TCP_RESET && c$orig$state == TCP_SYN_ACK_SENT) ||
         (c$resp$state == TCP_RESET && c$orig$state == TCP_ESTABLISHED && "s"
in c$history )
         ) && /[Dd]/ !in c$history )
        )
        return T;
    return F;
}

event connection_attempt(c: connection)
{
    local is_reverse_scan = F;
    if ( "H" in c$history )
        is_reverse_scan = T;
    add_sumstats(c$id, is_reverse_scan);
}

event connection_rejected(c: connection)
{
    local is_reverse_scan = F;
    if ( "s" in c$history )

```

```

        is_reverse_scan = T;

        add_sumstats(c$id, is_reverse_scan);
    }
event connection_reset(c: connection)
{
    if ( is_failed_conn(c) )
        add_sumstats(c$id, F);
    else if ( is_reverse_failed_conn(c) )
        add_sumstats(c$id, T);
}
event connection_pending(c: connection)
{
    if ( is_failed_conn(c) )
        add_sumstats(c$id, F);
    else if ( is_reverse_failed_conn(c) )
        add_sumstats(c$id, T);
}

```

6.5 Detect Syn Flood

Tests run on my local network.

```

root@kali:~# hping3 --flood -S -p 80 192.168.1.68
+PING 192.168.1.68 (eth0 192.168.1.68): S set, 40 headers + 0 data bytes
ping in flood mode, no replies will be shown
^C
--- 192.168.1.68 hping statistic ---
250386 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms

```

FIGURE 6-9 SYN FLOOD

```

conn.19:05:27-19:05:50.log x
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path conn
#open 2015-04-29-19-05-27
#fields ts uid id.orig_h id.orig_p id.resp_h id.resp_p proto
service duration orig_bytes resp_bytes conn_state local_orig
missed_bytes history orig_pkts orig_ip_bytes resp_pkts resp_ip_bytes
tunnel_parents
#types time string addr port addr port enum string interval count
count string bool count string count count count count set[string]
1430323522.076068 CX509JxoJz8KAmTUi 192.168.1.76 2967 192.168.1.68 80
tcp - 0.014410 0 0 RSTO T 0 ShR 2 80
1 44 (empty)
1430323522.076385 CqF0wR40EEQtEkloc 192.168.1.76 2968 192.168.1.68 80
tcp - 0.014108 0 0 RSTO T 0 ShR 2 80
1 44 (empty)
1430323522.076502 CfSBy01NTt2VoFXEX1 192.168.1.76 2969 192.168.1.68 80
tcp - 0.013994 0 0 RSTO T 0 ShR 2 80
1 44 (empty)
1430323522.076736 C7kTIc44nGMudkNCN7 192.168.1.76 2970 192.168.1.68 80
tcp - 0.013761 0 0 RSTO T 0 ShR 2 80
1 44 (empty)
1430323522.076854 CCX7B63TBkj2j0e345 192.168.1.76 2971 192.168.1.68 80

```

FIGURE 6-10 CONN.LOG

```

reporter.19:05:50-19:05:50.log x
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path reporter
#open 2015-04-29-19-05-50
#fields ts level message location
#types time enum string string
1430323550.450833 Reporter::INFO received termination signal (empty)
1430323550.450833 Reporter::INFO 171082 packets received on interface eth0, 353575 dropped
\x0a (empty)
#close 2015-04-29-19-05-51

```

FIGURE 6-11 REPORTER.LOG

```

weird.19:05:19-19:05:50.log x
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path weird
#open 2015-04-29-19-05-19
#fields ts uid id.orig_h id.orig_p id.resp_h id.resp_p name
addl notice peer
#types time string addr port addr port string string bool string
1430323519.929918 - - - - - truncated_IP - F
bro
1430323520.405300 - - - - - unknown_packet_type -
F bro
1430323527.966540 CQ4usB36xPqgs074r5 192.168.1.76 4270 192.168.1.68 80
active_connection_reuse - F bro
1430323527.966551 CXmPNJ3v9noaWSdRT9 192.168.1.76 4271 192.168.1.68 80
active_connection_reuse - F bro
1430323527.966557 CqZNVl24BaXz6vvyu6l 192.168.1.76 4272 192.168.1.68 80
active_connection_reuse - F bro
1430323527.966658 CUqFz7n0J5ZAnf7Xf 192.168.1.76 4273 192.168.1.68 80
active_connection_reuse - F bro
1430323527.966671 CjGwMmluFtPMStibIi 192.168.1.76 4274 192.168.1.68 80
active_connection_reuse - F bro
1430323527.966676 CfPZjMHxmQwC2NdSc 192.168.1.76 4275 192.168.1.68 80
active_connection_reuse - F bro

```

FIGURE 6-12 WEIRD.LOG

6.6 Detect UDP flood

Tests run on my local network.

```

root@kali:~# hping3 --udp --flood -p 80 192.168.1.68
HPING 192.168.1.68 (eth0 192.168.1.68): udp mode set, 28 headers + 0 data bytes
ping in flood mode, no replies will be shown
^C
--- 192.168.1.68 hping statistic ---
305017 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms

```

FIGURE 6-13 UDP FLOOD

```

reporter.19:14:40-19:14:40.log x
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path reporter
#open 2015-04-29-19-14-40
#fields ts level message location
#types time enum string string
1430324080.716244 Reporter::INFO received termination signal (empty)
1430324080.716244 Reporter::INFO 307914 packets received on interface eth0, 0 dropped
\x0a (empty)
#close 2015-04-29-19-14-40

```

FIGURE 6-14 REPORTER.LOG

```

weird.19:14:15-19:14:40.log x
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path weird
#open 2015-04-29-19-14-15
#fields ts uid id.orig_h id.orig_p id.resp_h id.resp_p name
addl notice peer
#types time string addr port addr port string string bool string
1430324055.242178 - - - - truncated_IP - F
bro
1430324057.001764 CLPIT12TZrvzypYVPV9 192.168.1.68 80 192.168.1.76 5353
DNS truncated_len_lt_hdr_len - F bro
1430324057.001764 CvQCju4pLKoR00ioe4 192.168.1.68 80 192.168.1.76 5355
DNS truncated_len_lt_hdr_len - F bro
1430324060.528547 - - - - unknown_packet_type -
F bro
1430324060.954421 CgGzln20vfpYNK3vrd 192.168.1.68 80 192.168.1.76 53
DNS truncated_len_lt_hdr_len - F bro
1430324060.954421 C7Ns911bwHTszPYGVk 192.168.1.68 80 192.168.1.76 67
binpac exception: out_of_bound: DHCP_Message:giaddr: 28 > 0 - F bro
1430324060.958416 CJZ83y2ja2tgouvrUc 192.168.1.68 80 192.168.1.76 137
DNS truncated_len_lt_hdr_len - F bro
1430324060.978230 CJciG24wkihZeq3dL7 192.168.1.68 80 192.168.1.76 514
binpac exception: out_of_bound: Syslog Priority:lt: 1 > 0 - F bro

```

FIGURE 6-15 WEIRD.LOG

```

conn.14:12:41-14:13:03.log x
#unset_field -
#path conn
#open 2015-05-17-14-12-41
#fields ts uid id.orig_h id.orig_p id.resp_h
id.resp_p proto service duration orig_bytes
resp_bytes conn_state local_orig missed_bytes history
orig_pkts orig_ip_bytes resp_pkts resp_ip_bytes tunnel_parents
#types time string addr port addr port enum string
interval count count string bool count string count
count count count set[string]
1431861151.839949 CAVMSk4SzL6ZV9qHv8 192.168.1.70 53097
224.0.0.252 5355 udp dns 0.099705 44 0
S0 T 0 D 2 100 0 0 (empty)
1431861151.839900 CbFfMA3oGERmyMu9yc
fe80::7126:cc1d:95b3:4efc 51711 ff02::1:3 5355 udp
dns 0.099732 44 0 S0 F 0 D
2 140 0 0 (empty)
1431861162.138255 CQs5tL2g1TwWpizmzg
fe80::7126:cc1d:95b3:4efc 60058 ff02::1:3 5355 udp
dns 0.100295 44 0 S0 F 0 D
2 140 0 0 (empty)
1431861162.138478 CD4vRv32ahlCW8nU74 192.168.1.70 58713

```

FIGURE 6-16 CONN.LOG

6.7 Detect Brute Force ssh

Tests run on my local network.

Timestamp	Username	Client Version	Server Version	IP	Port	Destination
1434187805.597359	CxBeNr4uKZ9a5SblK4	SSH-2.0-libssh-0.5.2	SSH-2.0-OpenSSH_6.6.1p1	192.168.0.14	33273	192.168.0.11 22
failure	OUTBOUND	-	-	-	-	-
1434187806.157551	CkzeBQ30MeFJoeKus4	SSH-2.0-libssh-0.5.2	SSH-2.0-OpenSSH_6.6.1p1	192.168.0.14	33276	192.168.0.11 22
failure	OUTBOUND	-	-	-	-	-
1434187806.233071	Cd0upz4Rwc2sJr0xAc	SSH-2.0-libssh-0.5.2	SSH-2.0-OpenSSH_6.6.1p1	192.168.0.14	33279	192.168.0.11 22
failure	OUTBOUND	-	-	-	-	-
1434187806.257459	CMvtPq41EAK006Wxwd	SSH-2.0-libssh-0.5.2	SSH-2.0-OpenSSH_6.6.1p1	192.168.0.14	33280	192.168.0.11 22
failure	OUTBOUND	-	-	-	-	-
1434187806.221096	CNFEPCL1L15t50LjEH1	SSH-2.0-libssh-0.5.2	SSH-2.0-OpenSSH_6.6.1p1	192.168.0.14	33278	192.168.0.11 22
failure	OUTBOUND	-	-	-	-	-
1434187806.213159	CV0xbZ3iC1y0V8pokf	SSH-2.0-libssh-0.5.2	SSH-2.0-OpenSSH_6.6.1p1	192.168.0.14	33277	192.168.0.11 22
failure	OUTBOUND	-	-	-	-	-
1434187806.305150	CBkoch2m4H31rYcyui	SSH-2.0-libssh-0.5.2	SSH-2.0-OpenSSH_6.6.1p1	192.168.0.14	33283	192.168.0.11 22
failure	OUTBOUND	-	-	-	-	-
1434187806.121505	CWvqGV3shj4D0RziTf	SSH-2.0-libssh-0.5.2	SSH-2.0-OpenSSH_6.6.1p1	192.168.0.14	33274	192.168.0.11 22
failure	OUTBOUND	-	-	-	-	-
1434187806.137002	CMSWEw1yEDya0sC5Q4	SSH-2.0-libssh-0.5.2	SSH-2.0-OpenSSH_6.6.1p1	192.168.0.14	33275	192.168.0.11 22

FIGURE 6-17 SSH.LOG

```

#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path software
#open 2015-06-13-12-29-43
#fields ts host host_p software_type name version.major version.minor
version.minor2 version.minor3 version.add1 unparsed_version
#types time addr port enum string count count count count string string
1434187783.781529 192.168.0.11 22 SSH::SERVER OpenSSH 6 6
1 192.168.0.11 22 SSH::SERVER OpenSSH_6.6.1p1 Ubuntu-2ubuntu2
1434187783.781529 192.168.0.14 - SSH::CLIENT libssh 0 5
2 192.168.0.14 - SSH::CLIENT libssh-0.5.2
#close 2015-06-13-12-31-42

```

FIGURE 6-18 SOFTWARE.LOG

```

known_services.12_29_43-12_31_42.log (7,8 GB Volume) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
known_services.12_29_43-12_31_42.log x
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path known services
#open 2015-06-13-12-29-43
#fields ts host port_num port_proto service
#types time addr port enum set[string]
1434187783.781529 192.168.0.11 22 tcp SSH
#close 2015-06-13-12-31-42

```

FIGURE 6-19 SERVICES.LOG

```

conn-summary.12_29_48-12_31_42.log (7,8 GB Volume) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
conn-summary.12_29_48-12_31_42.log x
>== Total === 2015-06-13-12-29-34 - 2015-06-13-12-31-36
- Connections 179.0 - Payload 338.9k -
  Ports | Sources | Destinations | Services |
Protocols | States |
22 52.0% | SF 96.1% | 192.168.0.14#1 52.0% | 192.168.0.11#2 52.0% | dns 46.4% |
6 53 45.3% | 192.168.0.11#3 45.8% | 192.168.0.1#4 45.3% | ssh 45.3% |
17 47.5% | S0 2.8% |
137 1.1% | fe80::55e0:5699:9f9c:e375#5 1.1% | ff02::1:ffdb:9217#6 0.6% |
- 8.4% | 1 0.6% | RST0 0.6% |
6771 0.6% | 192.168.0.15#7 0.6% | ff02::c#8 0.6% |
| 0TH 0.6% |
1900 0.6% | 192.168.0.7#9 0.6% | 239.192.152.143#10 0.6% |
| 136 0.6% | | 192.168.0.255#11 0.6% | |
| | | | 192.168.0.15#12 0.6% |
| | | |
| | | |
| | | |
| | | |
| | | |

```

FIGURE 6-20 CONN-SUMMARY.LOG

6.7.1 brute-force.bro

brute-force.bro [63]:

```
@load base/protocols/ssh
@load base/frameworks/sumstats
@load base/frameworks/notice
@load base/frameworks/intel
module SSH;
export {
    redef enum Notice::Type += {
        ## Indicates that a host has been identified as crossing the
        ## :bro:id:`SSH::password_guesses_limit` threshold with
        ## failed logins.
        Password_Guessing,
        ## Indicates that a host previously identified as a "password
        ## guesser" has now had a successful login
        ## attempt. This is not currently implemented.
        Login_By_Password_Guesser,
    };
    redef enum Intel::Where += {
        ## An indicator of the login for the intel framework.
        SSH::SUCCESSFUL_LOGIN,
    };
    ## The number of failed SSH connections before a host is designated as
    ## guessing passwords.
    const password_guesses_limit: double = 30 &redef;
    ## The amount of time to remember presumed non-successful logins to
    ## build a model of a password guesser.
    const guessing_timeout = 30 mins &redef;
    ## This value can be used to exclude hosts or entire networks from being
    ## tracked as potential "guessers". The index represents
    ## client subnets and the yield value represents server subnets.
    const ignore_guessers: table[subnet] of subnet &redef;
}
event bro_init()
{
    local r1: SumStats::Reducer = [$stream="ssh.login.failure",
    $apply=set(SumStats::SUM, SumStats::SAMPLE), $num_samples=5];
    SumStats::create([$name="detect-ssh-bruteforcing",
```



```

        $epoch=guessing_timeout,
        $reducers=set(r1),
        $threshold_val(key:          SumStats::Key,          result:
SumStats::Result) =
        {
            return result["ssh.login.failure"]$sum;
        },
        $threshold=password_guesses_limit,
        $threshold_crossed(key:          SumStats::Key,          result:
SumStats::Result) =
        {
            local r = result["ssh.login.failure"];
            local sub_msg = fmt("Sampled servers: ");
            local samples = r$samples;
            for ( i in samples )
            {
                if ( samples[i]?$str )
                    sub_msg = fmt("%s%s %s", sub_msg, i==0 ?
"":",", samples[i]$str);
            }
            # Generate the notice.
            NOTICE([$note=Password_Guessing,
                    $msg=fmt("%s appears to be guessing SSH passwords
(seen in %d connections).", key$host, r$num),
                    $sub=sub_msg,
                    $src=key$host,
                    $identifier=cat(key$host)]);
        }]);
    }

event ssh_auth_successful(c: connection, auth_method_none: bool)
{
    local id = c$id;

    Intel::seen([$host=id$orig_h,
                $conn=c,
                $where=SSH::SUCCESSFUL_LOGIN]);
}

event ssh_auth_failed(c: connection)
{

```

```

local id = c$id;
# Add data to the FAILED_LOGIN metric unless this connection should
# be ignored.
if ( ! (id$orig_h in ignore_guessers &&
        id$resp_h in ignore_guessers[id$orig_h]) )
    SumStats::observe("ssh.login.failure",           [ $host=id$orig_h,
[$str=cat(id$resp_h)] );
}

```

6.8 Bro vs Snort

Contrast of Snort and Bro is made on the basis of different parameters such as speed, signatures, flexibility, interface and operating system ability [52].

- a. Speed: Bro IDS has the advantage to run in high-speed networks. Bro is very effective and able to collect data from Gbps networks. This makes it suitable for Bro to run perfect in high speed networks without losing packets or slowing down the traffic.
- b. Signatures: When it comes to the signatures used for detecting intrusions, the Bro signatures are more refined than the signatures used in Snort.
- c. Flexibility: Bro is a flexible intrusion detection system with the ability of being configured and then clarified for its intended computer network. Bro comes with policy scripts which can be used right out of the box and these will detect the most well-known attacks.
- d. Interface: Snort has a graphical user platform which makes it more sophisticated. Bro's lack of a user interface (GUI) can also be regarded as a disadvantage since one should have expertise of how a UNIX system function and be able to handle shell commands to understand this system.
- e. Operating System Compatibility: The Snort can run on all of today's most well known operating systems and is not confined to a fully establishments server hardware platform whereas Bro is confined to UNIX like operating systems.

Chapter 7 Conclusion

Network intrusion models based on detection events are able to detect real-time threats. In addition, network intrusion models show a possibility of threat prediction by analyzing correlation of intrusion detection events.

The IDSs are gaining importance in the field of internet security. It is not a tool intended to replace firewalls or anti-viruses, but a basic tool for network security.

We have seen many projects of IDSs, which deduct the relevance of this tool in the computer field. Many users are covered within the development of IDSs from administrators, who install IDSs to defend its small network, to companies, who buy powerful security tools.

Although an intrusion detection system is a good way to keep our network safe from attacks, this option is not useful at all if we do not take into account other aspects much more basic such as having appropriate passwords in our systems, correct firewall settings and a backup.

IDSs are not autonomous systems but they are alert tools that must be interpreted by security expertise to get knowledge of who attacks and how the attacks are performed to apply measures so that the system cannot be compromised again.

We should accept that intrusion detection systems are not suitable for all organizations. If an organization cannot afford a specialist on attack responses, having an intrusion detection system will not provide any additional security. For the rest of the organizations, the use of IDSs should be stated in their security policy, completely coordinated with the other resources.

We have noticed that the intrusion detection systems are not perfect yet, since new malicious codes are constantly coming up just as holes in new software. Those can be exploited by the hacker to bypass the security system. Two very important recommendations can be given to avoid bad things: Developers of software must carefully follow the rules to create secure programmes and the security system must be always updated to prevent the zero day attacks.

An optimal IDS deployment should have some operational procedure behind it to gather additional information and optimize the process. Many customers think that a given security product like an IDS will protect them from 100% of the attacks. In a practical world, there are no absolutes, instead IDS can significantly reduce the risk from network attacks, but they are not perfect.

7.1 Future Work

Regarding future enhancement and development of certain IDS aspects some suggestions might concern automatic generation of the dependency for the system as well as metrics development when it comes to measuring system security policies, in general creating standards to assess the system resources in terms of security policies. Additionally a

mechanism for transferring the knowledge of one response engine to another, so that it is shared across all hosts would be essential.

Another proposal could be the development of standards to measure the success of a selected response on different environments. By this the comparison between results of response selected between one environment and another could be achieved.

Future IDS will also have to address scalability and distributed data collection issues in order to achieve the level of effectiveness that is required.

BIBLIOGRAPHY

- [1] K. Scarfone, P. Mell, "Guide to Intrusion Detection and Prevention Systems (IDPS)". Computer Security Resource Center (National Institute of Standards and Technology). February 2007.
- [2] Robin Berthier, William H. Sanders, and Himanshu Khurana, Intrusion Detection for Advanced Metering Infrastructures: Requirements and Architectural Directions,IEEE,2010
- [3] Irfan Gul, M. Hussain, Distributed Cloud Intrusion Detection Mode, International Journal of Advanced Science and Technology Vol. 34,2011
- [4] Computer Economics, "2007 malware report: The economic impact of viruses, spyware, adware, botnets, and other malicious code," 2008.
- [5] http://en.wikipedia.org/wiki/Intrusion_detection_system.
- [6] Giovanni Vigna. Fredrik Valeur Richard A. Kemmerer, Designing and Implementing a Family of Intrusion. Detection Systems, Reliable Software Group, ACM New York,2003
- [7] M. Roesch. Snort - Lightweight Intrusion Detection for Networks. In Proceedings of the USENIX LISA '99 Conference, November 1999.
- [8] C. Wang and J. C. Knight. Towards survivable intrusion detection. In Proceedings of the 3rd Information Survivability Workshop (ISW-2000), Boston, USA, October 2000.
- [9] C.Zimmer,B.Bhat,F.Mueller, and S.Mohan, "Time-based intrusion detection in cyber-physical systems,"inProc.1stACM/IEEEInt.Conf. CyberPhysicalSyst.,Stockholm,Sweden,2010,pp.109–118.
- [10] A. Avizienis, J. Laprie, and B. Randell. Fundamental concepts of dependability. Technical Report N01145, LAAS-CNRS, 2001.

- [11] Joseph S. Sherif, Tommy G. Dearmond, "Intrusion Detection: Systems and Models", Eleventh IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE) 2002.
- [12] I. Balepin, S. Maltsev, J. Rowe, and K. Levit. "Using specification-based intrusion detection for automated response," in the 6th International Symposium on Recent Advances in Intrusion Detection (RAID) 2003.
- [13] Cisco, Annual Security Report, 2014.
- [14] NabilAliAlrajeh S.Khan and BilalShams, Intrusion Detection Systems in Wireless Sensor Networks: A Review, Hindawi, 2013.
- [15] Ashara Banu Mohamed ,Norbik Bashah Idris,Bharanidharan Shanmugam, A Brief Introduction to Intrusion Detection System, First International Conference, IRAM 2012, Kuala Lumpur, Malaysia, November 28-30, 2012.
- [16] Barbara, Daniel, Couto, Julia, Jajodia, Sushil, Popyack, Leonard, and Wu, Ningning, "ADAM: Detecting Intrusions by Data Mining," Proceedings of the IEEE Workshop on Information Assurance and Security, West Point, NY, June 5–6, 2001.
- [17] Intrusion Detection Techniques for Mobile Wireless Networks, ACM WINET 2003.
- [18] McHugh, J., Christie, A. & Allen J. Defending yourself: the role of intrusion detection systems.IEEE Software, 2000.
- [19] Thamilarasu, G., Balasubramanian, A., Mishra, S. & Sridhar, R. A cross-layer based intrusion detection approach for wireless ad hoc networks.IEEE International Conference on Mobile Adhoc and Sensor Systems Conference, 2005.
- [20] Daniele Sgandurra, A Survey of Intrusion Detection Systems, Istituto di Informatica e Telematica, CNR, Pisa, Italy,2009.

- [21] Svetlana Radosavac, John S. Baras, Detection and Classification of Network Intrusions Using Hidden Markov Models, 2003 Conference on Information Sciences and Systems, The Johns Hopkins University, March 12–14, 2003
- [22] Scarfone, Karen, Mell, Peter (February 2007). "Guide to Intrusion Detection and Prevention Systems (IDPS)". Computer Security Resource Center (National Institute of Standards and Technology) (800–94). Retrieved 1 January 2010.
- [23] Silva, L. D. S., Santos, A. C., Mancilha, T. D., Silva, J. D., & Montes, A. Detecting attack signatures in the real network traffic with ANNIDA. Expert Systems with Applications, 34(4), 2326–2333.2008.
- [24] Anna Sperotto, Gregor Schaffrath, Ramin Sadre, Cristian Morariu, Aiko Pras and Burkhard Stiller, An Overview of IP Flow-Based Intrusion Detection, IEEE COMMUNICATIONS SURVEYS & TUTORIALS, VOL. 12, NO. 3, THIRD QUARTER 2010.
- [25] Karen A. Scarfone, M. Mell, Guide to Intrusion Detection and Prevention Systems IDPSACM, 2007.
- [26] Snort (software) http://en.wikipedia.org/wiki/Snort_%28software%29.
- [27] InfoWorld, The greatest open source software of all time, 2009. <http://www.infoworld.com/d/open-source/greatest-open-source-software-all-time776?source=fssr>.
- [28] Sectools.Org: 2006 Results <http://sectools.org/tools2006.html>.
- [29] SecTools.Org: Top 125 Network Security Tools; <http://sectools.org/tag/ids/>.
- [30] Suricata (software), [http://en.wikipedia.org/wiki/Suricata_\(software\)](http://en.wikipedia.org/wiki/Suricata_(software)).

- [31] The Bro Network Security Monitor, <http://bro-ids.org/>.
- [32] R. Graham, "FAQ: Network Intrusion Detection Systems". March 21, 2000.
- [33] P.A. Porras, A. Valdes, Live traffic analysis of tcp/ip gateway, Proc. ISOC Symp. on Network and Distributed System Security, Security NDSS'98, San Diego, CA, March 1998.
- [34] Hervé Debar, Marc Dacier, Andreas Wespi, Towards a taxonomy of intrusion-detection systems, ACM, 1999.
- [35] http://en.wikipedia.org/wiki/Intrusion_detection_system, 2015.
- [36] Ganesh Kumar Varadarajan, Web Application Attack Analysis Using Bro IDS, SANS, 2012.
- [37] Shevali Agarwal, Anurag Punde, Shubhi Kesharwani, Proposed Algorithm for Network Traffic Classification Based On DB Scan, IJESRT, 2013.
- [38] Sandhya Peddabachigaria, Ajith Abraham, Crina Grosan, Johnson Thomas, Modeling intrusion detection system using hybrid intelligent systems, Elsevier, 2005.
- [39] S. Chebroly, A. Abraham, and J. P. Thomas. Feature deduction and ensemble design of intrusion detection systems. Computers & Security, 24(4):295–307, 2005.
- [40] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, Kuang-Yuan Tung, Intrusion detection system: A comprehensive review, Elsevier, 2013.
- [41] <http://searchnetworking.techtarget.com/>.
- [42] http://en.wikipedia.org/wiki/Port_mirroring.

- [43] Díaz-Verdejo, J.E., García-Teodoro, P., Muñoz, P., Maciá-Fernández, G., De Toro, F.: Una aproximación basada en Snort para el desarrollo e implantación de IDS híbridos (A Snort-based approach for the development and deployment of hybrid IDS). *IEEE Latin America Transactions* 5(6), 386–392 (2007).
- [44] Hwang, K., Cai, M., Chen, Y., Qin, M.: Hybrid Intrusion Detection with Weighted Signature Generation Over Anomalous Internet Episodes. *IEEE Transactions on Dependable and Secure Computing* 4(1), 41–55 (2007).
- [45] Wu, L.C., Hung, C.H., Chen, S.F.: Building intrusion pattern miner for Snort network intrusion detection system. *Journal of Systems and Software* 80(10), 1699–1715 (2007) 12.
- [46] <http://www.snort.com>.
- [47] Jay Beale, *Snort 2.1 Intrusion Detection*, Syngress, 2004.
- [48] <http://manual.snort.org/>.
- [49] http://en.wikipedia.org/wiki/UDP_flood_attack.
- [50] <http://suricata-ids.org/>.
- [51] <https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Suricatayaml>.
- [52] Pritika Mehra, A brief study and comparison of Snort and Bro Open Source Network Intrusion Detection Systems, *International Journal of Advanced Research in Computer and Communication Engineering* Vol. 1, Issue 6, August 2012.
- [53] Paar Christof, Pelzl Jan, *Understanding Cryptography: A Textbook for Students and Practitioners*, ISBN 3-642-04100-0, Bart (2010).

- [54] http://en.wikipedia.org/wiki/Brute-force_attack.
- [55] V. Jaiganesh , S. Mangayarkarasi , Dr. P. Sumathi, Intrusion Detection Systems: A Survey and Analysis of Classification Techniques, International Journal of Advanced Research in Computer and Communication Engineering Vol. 2, Issue 4, April 2013.
- [56] Neeraj Kumar Naveen Chilamkurti, Collaborative trust aware intelligent intrusion detection in VANETs, Elsevier, August, 2014.
- [57] Gisung Kima, Seungmin Leeb,Sehun Kima, A novel hybrid intrusion detection method integrating anomaly detection with misuse detection, Elsevier 2014.
- [58] Tran Ngoc Thinh, Tran Trung Hieu, Van Quoc Dung, Kittitornkun, S, A FPGA-based deep packet inspection engine for Network Intrusion Detection System,IEEE,2014.
- [59] Junaid Arshad, Paul Townend, Jie Xu, Junaid Arshad, A Novel Intrusion Severity Analysis Approach for Clouds, Elsevier,2013.
- [60] Zubair Md. Fadlullah, Hiroki Nishiyama, Nei Kato, and Mostafa M. Fouda, Intrusion Detection System (IDS) for Combating Attacks Against Cognitive Radio Networks, IEEE Network Magazine, vol. 27, no. 3, pp. 51-56, May/June 2013.
- [61] Chun-Jen Chung, Pankaj Khatkar, Tianyi Xing, Jeongkeun Lee, Dijiang Huang , Network Intrusion Detection and Countermeasure Selection in Virtual Network Systems, IEEE,2013.
- [62] <http://www.aldeid.com/wiki/Suricata-vs-snort>.
- [63] www.bro.org
- [64] http://www.webopedia.com/TERM/P/port_scanning.html
- [65] http://en.wikipedia.org/wiki/SYN_flood