



**INSTITUTE FOR ADVANCED
COMPUTING AND
SOFTWARE DEVELOPMENT
AKURDI, PUNE**

DOCUMENTATION ON

“WEB APPLICATION FIREWALL”

PG-DITISS March-2023

SUBMITTED BY:

GROUP NO: 9

ABHISHEK DUBEY (233401)

**MOHANEESH SHIRBHAYYE
(233418)**

**MR. KARTIK AWARI
PROJECT GUIDE**

**MR. ROHIT PURANIK
CENTRE CO-ORDINATOR**

ABSTRACT

In this project we propose a centralized web firewall system for web application security which will provide a new type of synchronized system, which has the ability to detect and prevent a variety of web application attacks for a wide range of hosts at the same time , using an centralized command and control system, the attacked client then sends the information to a centralized command and control server which will distribute the attack information to all of the integrated clients connected to it.

The distributed information contains all of the attack information including the type of attack, the IP address of the attacker, and the time of attack. The process of receiving the attacker's information and distributing it through the centralized web firewall is done automatically and immediately at the time of the attack. In addition, all of the receiving clients will take actions against the threat depending on the distributed information such as banning the IP address of the attacker.

The main process aims to protect multiple clients from any possible attack from the same attacker or the same type of attack. These system has been implemented to protect areal web application. Experiments showed that the attacks has been successfully prevented on multiple hosts at the time.

This protect came to provide a centralized web firewall system that connect different web firewalls in order to detect and prevent different types of web attacks and work as a fully integrated system with the different clients.

Keywords: Snort, Squid, DVWA, Distributing web-firewall, SQL Injection, XSS, DDoS Attack, Suspicious User Behavior, Web Applications

TABLE OF CONTENTS

Topics	Page No.
1. INTRODUCTION	1
1.1 Problem Statement	1
2. LITERATURE SURVEY	2
3. METHODOLOGY	3
3.1 System Architecture	4
3.2.1 Load Balancer	5
3.2.2 Snort	7
3.2.3 Wireless based IDPS	8
3.2.4 Network Behavior Analysis	9
3.2.5 Host based IDPS	10
4. REQUIREMENT SPECIFICATION	11
4.1 Software Requirement	
4.2 Hardware Requirement	
5. IMPLEMENTATION	12
5.1 Snort Configuration as Intrusion Prevention System	12
5.3 PfSense as a firewall	13
5.2 Intrusion Prevention Snort Rules	14
5.4 Load balancer configuration	15

6. DIFFERENT ATTACK TESTING	18
6.3.1 SQL Injection	18
6.3.2 Java Script	20
6.3.3 Command Injection Testing On Snort	21
6.3.4 XSS Reflected	23
7. CONCLUSION	25
8. REFERENCE	26

1. INTRODUCTION

Recently, the revolutionary growth of web application usage started to increase in a large way which made the possibility of more web application weaknesses and vulnerabilities to appear and more infiltration attempts to happen on daily basis. The development of web applications witnessed a huge change along with the event of the Internet appearance. Most companies and individuals have started to use web applications on daily basis. The web became the main link that connects all users all over the world where private information about the web users is stored in databases. Some of these activities contains sensitive data about the users such as e banking.

Many attackers may be able to compromise some web applications and get access to private data across the globe by exploiting several web application vulnerabilities. Such cyber threats can cause financial loss for many parties including private companies and any other type of infrastructure. Different types of web security mechanisms have been developed to detect and prevent multiple types of web threats. The idea of detecting web threats and passing down the information's to other web hosts plays a major role in preventing the occurrence of possible attacks performed by the same attacker or using the same technique of attack.

1.1. PROBLEM STATEMENT

This work is a review study in the context of web application security testing. In this section, to understand the rest of the paper we first provide an overview of web application security testing and then we discuss a few of related works which are existing systematic literature review papers in this area.

The web continues to grow and attacks against the web continue to increase. This paper focuses on the literature review on scanning web vulnerabilities and solutions to mitigate web attacks. Vulnerability scanning methods will be reviewed as well as frameworks for improving web security. This research is the basis for future work that will end with the elaboration of web scanning and security with the aim of proposing better innovations.

The existing research works on securing the web application showcases different approaches used such as Web Application firewall, vulnerability assessment and penetration testing. The current scenario of web application security has shortcomings as preventive mechanisms are not implemented at run-time. Also, Attackers are becoming smarter by finding new and clever ways to create malicious inputs that will bypass the Firewall input filters. Passive Approaches such as Vulnerability Assessment and Penetration Testing is effective in threat and attack detection but it's time-consuming process.

2. LITERATURE SURVEY

There is nothing that has 100% security, but we must try to provide security in various ways and tricks in order to maintain our privacy on the Internet, but the web application firewall is an important element in protecting websites and users' privacy and is an additional option on sites and this makes it easier for hackersto attack the site and steal data From the site.

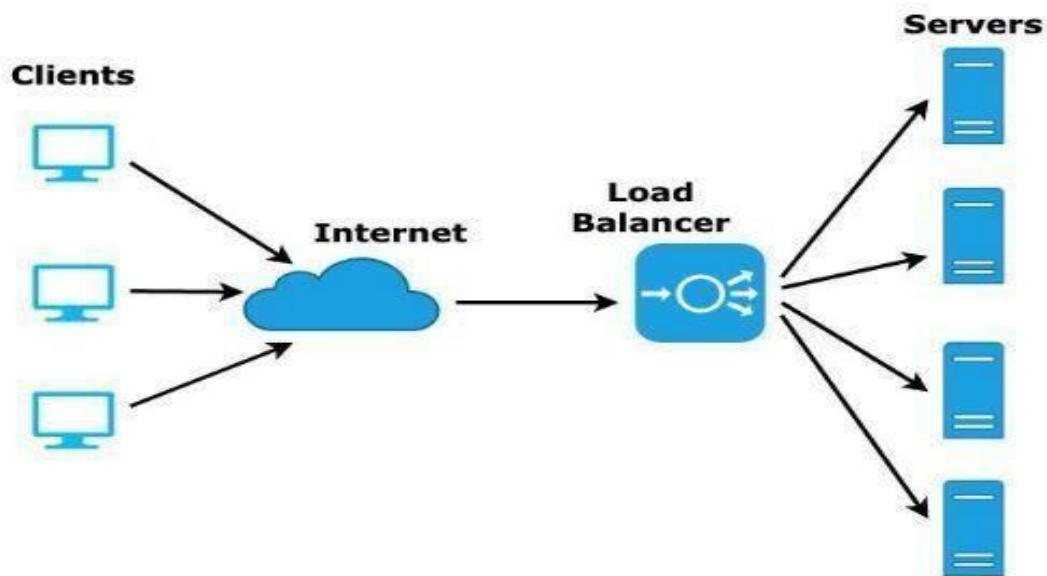
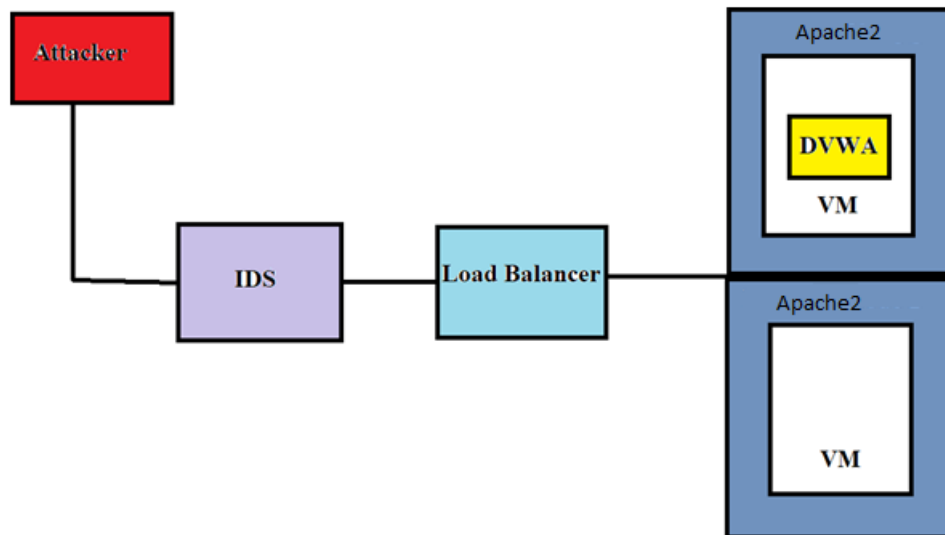
3. METHODOLOGY

Type 1 hypervisor is considered as a native, embedded, bare metal hypervisor that is in corporate and run directly in hardware to control resources. Type I hypervisor does not have an OS running below it; instead, it is fully responsible for scheduling and allocating of systems resources between VMs. It serves as OS. The open-source hypervisor as any other kind of open-source software must fulfill the definition of open-source software as defined by Stallman (1980) that include redistribute the software without restriction, access the source code, modify the source code and distribute the modified version of the software [2]. Generally, the open-source hypervisor is available under the open-source license GNU/GPL that allows free distribution of the software for use and improvements.

The hypervisor uses underlying technologies of Kernel-based Virtual Machine (KVM) hypervisor, which supports full virtualization, and LXC containers, which supports container, based virtualization. It uses both GUI and CLI to manage containers and virtual machines.

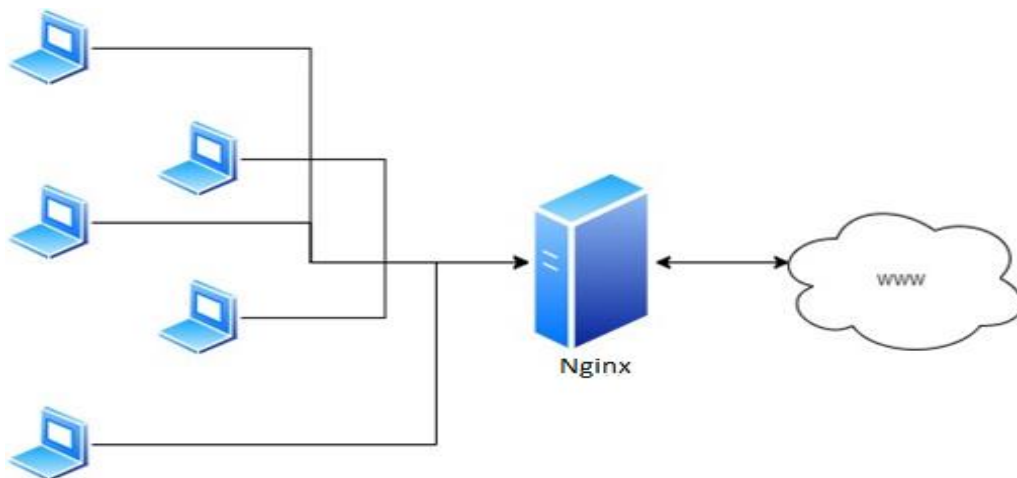
Virtual Environment is an open-source software server for virtualization management. It is a hosted Type-1 hypervisor that can run operating systems including Linux and Windows on x64 hardware. We create and configure snort as A IDS for the detection various attacks who's trying to attack on our machine, We are write rule for detection and prevention. Also we configure load balancing for managing network traffics. DVWA configure for various type of different attacks.

3.1 SYSTEM ARCHITECTURE



3.2.1 Load Balancer

Nginx is a proxy-server that caches internet data. Squid supports the HTTP, HTTPS, and FTP protocols. Using Nginx, you can improve the server response time by caching and reusing frequently-requested web pages. It improves performance without the need for greater bandwidth. Nginx is generally used as a server accelerator. You can configure Nginx to act as a caching HTTP accelerator. It improves the server performance, network performance, and security as Nginx overcomes disadvantages related with performance. Load balancing shares the network traffic between two or more servers so that a single server does not get loaded with requests. Load balancing increases performance and reliability. You can use multiple processors or multiple threads in a single processor for load balancing. Load balancing does not require dedicated software and hardware nodes. DNS servers can run the round-robin algorithm against multiple IP Addresses associated with a single domain name. Nginx is generally used to act as a caching proxy server. It sends client HTTP requests to the proxy server. The proxy server fetches web pages in accordance with the cache setup, and returns them to the client. Nginx can be used to perform basic round-robin load balancing, and to cache results based on your cache configuration



3.2.2 Snort IPS

Snort is a free open source intrusion detection system. It's very popular and powerful multi packettool run by a lot of different people and companies. It is one of the Signature based Intrusion Detection and Prevention System. The beauty of this tool lies with the formation of rules. Rules can be created/designed to block traffic or to merely send alerts; alerts can be logged to a log file, can be sent to the console or displayed on the screen. They can be configured to send an email to someone or they can be logged to database. Various options can be used for the formation of rules. Snort basically works on the three modes: Sniffer mode, Packet logger mode and NIDS mode.

It can be run as a packet sniffer mode from command line, which is simply looking at header information and printing the details on the screen. It can be used as a packet logger mode, which takes each packet and log it into the log files, which resides in the root directory. The file can be viewed later on using Snort or tcp dump. This mode is for the later use as if someone wants to view the captured packets later on. The third and the last mode is Network Intrusion Detection System mode (NIDS mode) which is the most important mode among all, considering the intrusion detection point of view. Snort as NIDS mode, uses its rules to find out if there are any intrusion activities going on the network.

Snort use NICs running in promiscuous mode to analyze and capture raw packet data in real time in NIDS mode. Snort can perform real-time packet logging, content searching/matching and protocol analysis and can detect a variety of attacks with known loopholes. It not only monitors or detects the intrusions but also can prevent it by taking various actions like _ reject, drop and block. The difference between NIDS and the first two modes of Snort is that the snort in NIDS mode is actually applying different actions to the packet content that are flowing across the network against the rule set that has indicated it is being used by the snort.

In general term, an intrusion can be said as an unauthorized access to someone's property or area, but when it comes to computer science, it is an act to compromise the basic computer network security goals viz. confidentiality, integrity, and privacy. Intrusion Detection is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of possible incidents of threats and violations of computer security practices, acceptable use policies or standard security policies. Intrusion Detection System (IDS) detects the presence of intrusion in the network. It is designed to monitor the events occurring in a computer system or network and responds to events with signs of possible incidents of violations of security policies. On the other hand, Intrusion Prevention System (IPS) is the network security system or technology that is capable of not only detecting the intrusion activities but also take required counter measures to prevent them.

Types of Technologies

3.2.3 Wireless based IDPS

A Wireless Intrusion Detection and Prevention System (IDPS) is a security solution designed to monitor and protect wireless networks from unauthorized access, malicious activities, and various types of attacks. It plays a crucial role in ensuring the security and integrity of wireless communication. Here's an overview of a wireless-based IDPS:

1. Introduction to Wireless IDPS:

Explain the importance of wireless network security due to the widespread use of Wi-Fi and mobile devices.

Introduce the concept of a Wireless IDPS as a specialized security system for monitoring wireless networks.

2. How Wireless IDPS Works:

Describe the fundamental working principle of a Wireless IDPS.

Explain how it passively or actively monitors wireless traffic, detecting anomalies and patterns associated with attacks.

3. Features and Capabilities:

Detail the features and capabilities of a Wireless IDPS, including:

Rogue AP detection: Identifying unauthorized access points.

Intrusion detection: Detecting unauthorized or suspicious devices.

Attack detection: Identifying denial-of-service (DoS) attacks, de-authentication attacks, etc.

Anomaly detection: Recognizing unusual patterns of behavior.

Encryption monitoring: Ensuring proper encryption standards are maintained.

4. Types of Wireless IDPS:

Discuss different types of Wireless IDPS based on deployment, approach, and functionality.

Mention standalone solutions, cloud-based solutions, and solutions integrated with wireless controllers.

5. Wireless IDPS Components:

Explain the components of a Wireless IDPS system, such as sensors, analyzers, and reporting interfaces.

Highlight the importance of a central management console for configuration and monitoring.

6. Detection Techniques:

Explain the various detection techniques used by Wireless IDPS, such as signature-based, anomaly-based, and heuristic-based methods.

7. Benefits of Wireless IDPS:

Discuss the benefits of deploying a Wireless IDPS:

Early threat detection: Detecting and responding to threats before they escalate.

Regulatory compliance: Meeting security standards and compliance requirements.

Data protection: Safeguarding sensitive data transmitted over wireless networks.

Network performance: Ensuring optimal wireless network performance

3.2.4 Network Behavior Analysis

NBA examines network traffic to identify threats which generate unusual traffic flows such as DDoS (Distributed Denial of Service) attack, malwares (e.g. worms, backdoors), and policy violations. These systems are deployed for monitoring the flow on an organization's internal network, sometimes it is used to monitor organization's networks and external network. Network Behavior Analysis (NBA) is a cybersecurity approach that focuses on monitoring and analyzing the behavior of network traffic to detect anomalies, unusual patterns, and potential security threats. Integrating NBA with a Web Application Firewall (WAF) can provide enhanced security for web applications by detecting and responding to malicious activities that might evade traditional rule-based security mechanisms. Here's an overview of how Network Behavior Analysis can complement a Web Application Firewall:

1. Introduction to Network Behavior Analysis (NBA) and WAF:

Explain the significance of NBA in identifying advanced and evolving cyber threats. Introduce WAF as a critical component for protecting web applications from various attacks.

2. The Role of NBA in Web Application Security:

Describe how Network Behavior Analysis goes beyond traditional rule-based methods by focusing on deviations from normal network behavior.

Highlight how NBA can identify zero-day attacks and insider threats that might not be captured by signature-based detection.

3. How NBA Works:

Explain the process of collecting and analyzing network traffic data to establish a baseline of normal behavior.

Discuss how machine learning algorithms and statistical methods are used to detect deviations from this baseline.

4. Integrating NBA with WAF:

Describe the integration of NBA and WAF to create a comprehensive security solution. Emphasize how NBA can provide contextual insights that help WAF make more informed decisions.

5. Benefits of NBA for WAF:

Discuss the advantages of combining NBA with WAF:

Improved threat detection: Identifying attacks that bypass traditional rule-based mechanisms.

Reduced false positives: Enhancing accuracy by considering behavioral context.

Real-time monitoring: Detecting anomalies in real-time to minimize potential damage.

3.2.5 Host based IDPS

A Host-Based Intrusion Detection and Prevention System (HIDPS) combined with a Web Application Firewall (WAF) creates a comprehensive security solution that protects both the host system and the web applications running on it. Here's an overview of how a Host-Based IDPS and WAF can work together:

1. Introduction to Host-Based IDPS and WAF:

Define what a Host-Based IDPS is and how it focuses on monitoring and protecting individual systems.

Introduce the concept of a Web Application Firewall and its role in securing web applications from various attacks.

2. Host-Based IDPS for System Protection:

Explain how a Host-Based IDPS works by monitoring system logs, processes, file integrity, and other host-specific data.

Highlight its role in detecting and preventing unauthorized access, malware infections, and system-level attacks.

3. WAF for Web Application Security:

Discuss the purpose of a WAF in safeguarding web applications from attacks like SQL injection, cross-site scripting (XSS), and more.

Emphasize the need for specialized security mechanisms to protect against application-layer vulnerabilities.

4. REQUIREMENT SPECIFICATION

4.1 Software Requirements:

The software configurations used are: Snort
Squid.

Operating System: Debian 10, Pfsense, kali Linux, ubuntu.

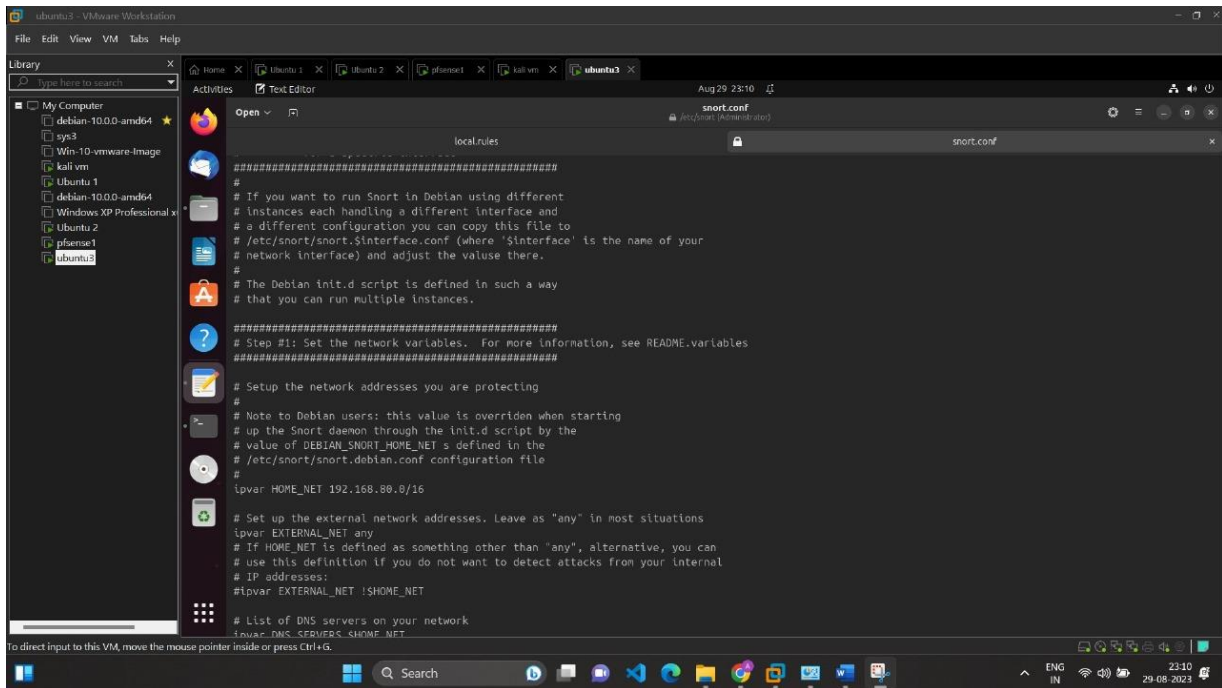
4.2 Hardware Requirements:

The Hardware requirements are: Processor: INTEL
RAM: 8 GB
HDD: 500 GB or Higher

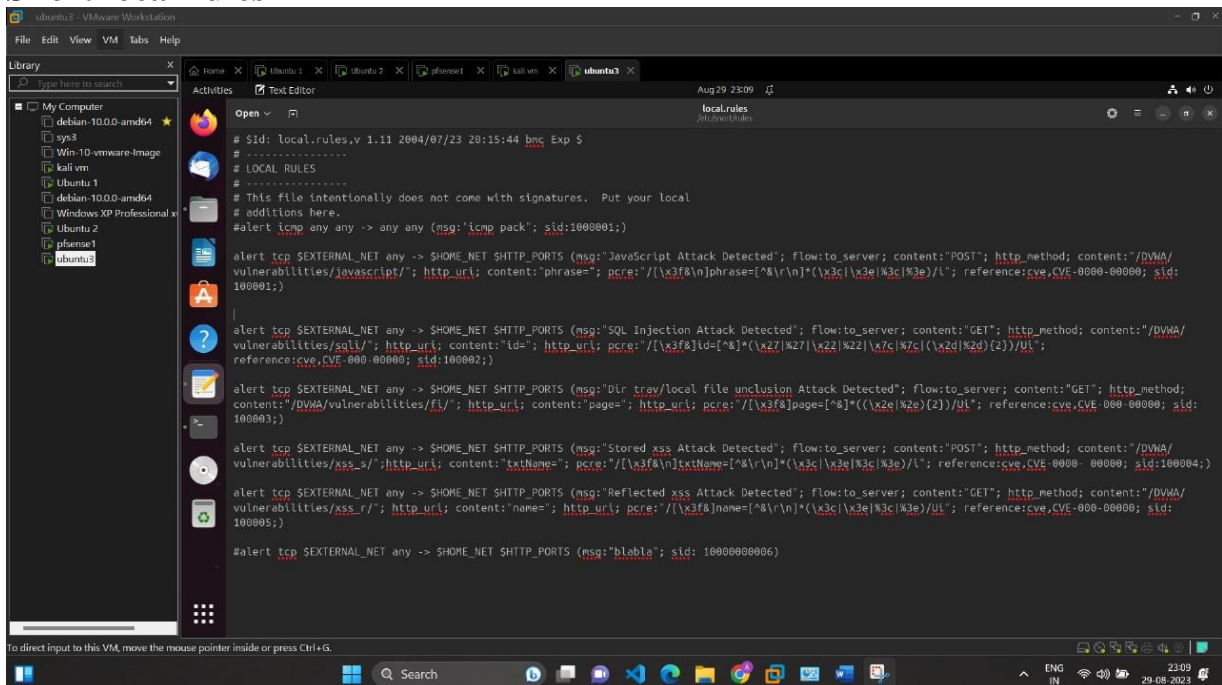
5. IMPLEMENTATION

5.1 Snort Configuration as Intrusion Prevention System (IPS)

Snort Configuration as Intrusion Prevention system:

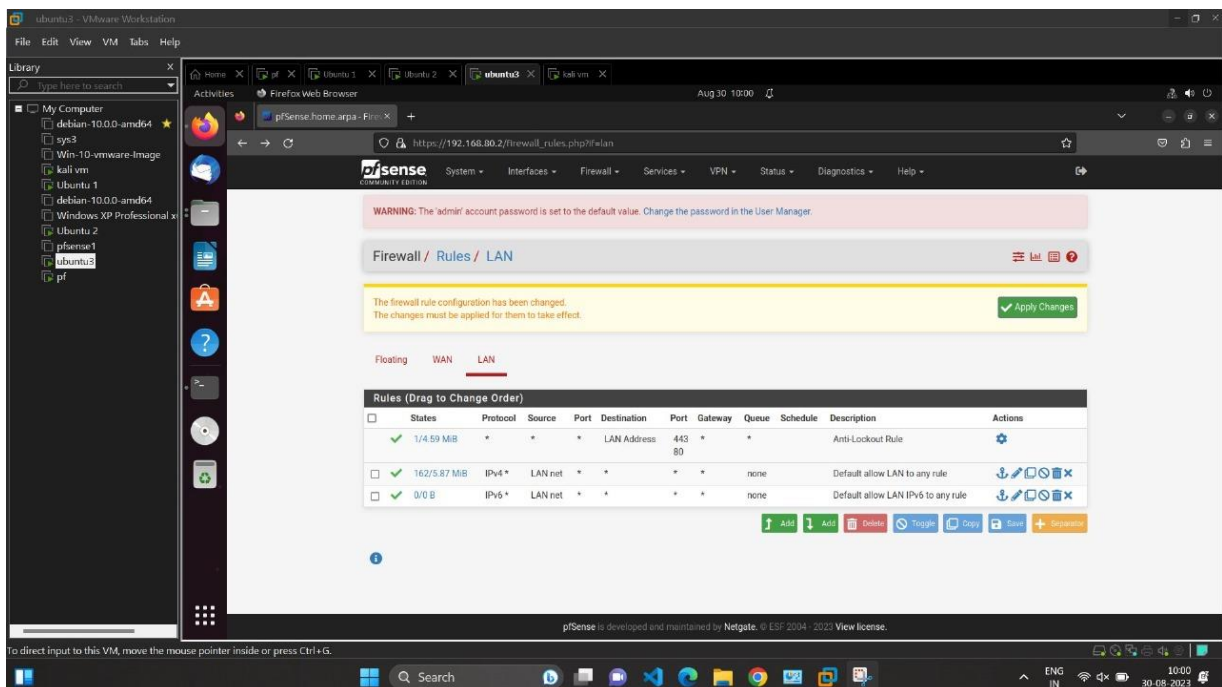
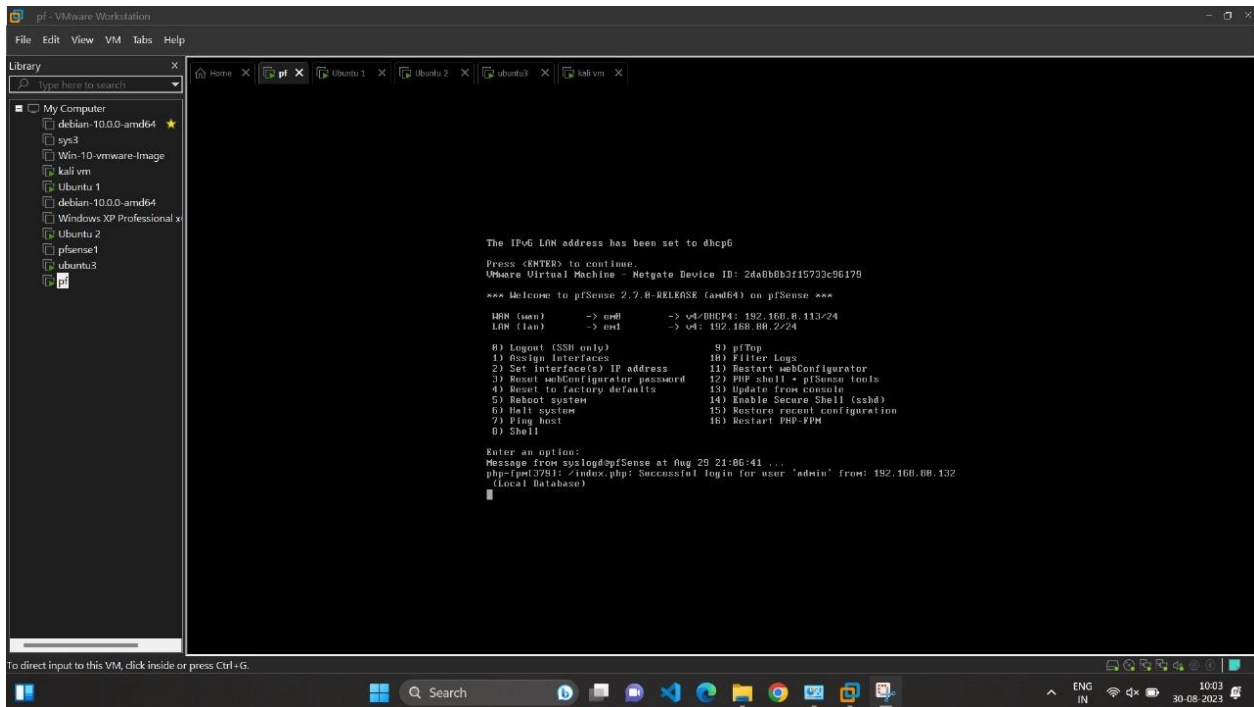


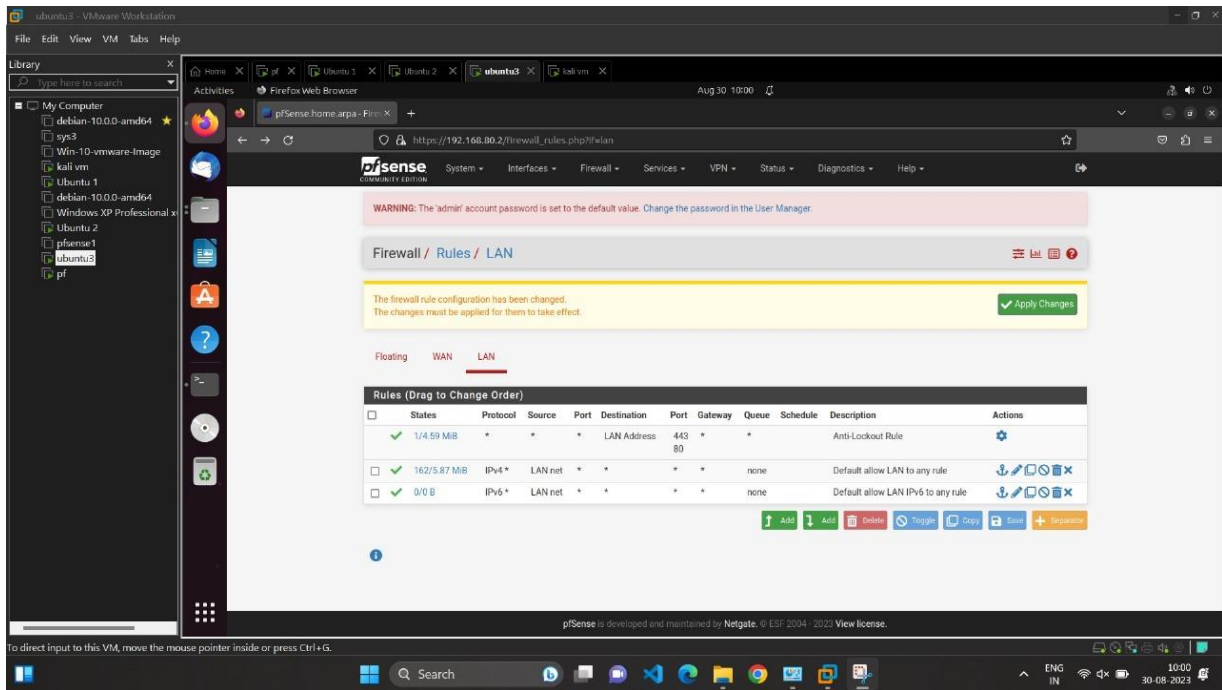
Snort local rules



5.2 PfSense as a firewall

pfSense is a popular open-source firewall and routing software that is based on FreeBSD. It's designed to provide advanced firewalling, routing, and network security features, making it suitable for both home and enterprise environments. Here are some key features and aspects of pfSense as a firewall





5.3 Intrusion Prevention Snort Rules -

A. Java Script

```
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"JavaScript AttackDetected"; flow:to_server; content:"POST";
http_method; content:"/DVWA/vulnerabilities/javascript/"; http_uri;
content:"phrase="; pcre:"/[\\x3f&\\n]phrase=[^&\\r\\n]*(\\x3c|\\x3e|%3c|%3e)/i";
reference:cve,CVE-0000-00000; sid:100001;)
```


B. SQL Injection

```
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"SQL
Injection Attack Detected"; flow:to_server; content:"GET"; http_method;
content:"/DVWA/vulnerabilities/sqli/"; http_uri;content:"id="; http_uri;
pcre:"/[\\x3f&]id=[^&]*(\\x27|%27|\\x22|%22|\\x7c|%7c|(\\x2d|%2d){2})/Ui";
reference:cve,CVE-000-00000; sid:100002;)
```


C. Directory Traversal / local file inclusion

```
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"Dir
trav/local fileunclusion Attack Detected"; flow:to_server; content:"GET";
http_method; content:"/DVWA/vulnerabilities/fi/"; http_uri; content:"page=";
```

```
http_uri; pcre:"/[x3f&]page=[^&]*((\x2e|\%2e){2})/Ui"; reference:cve,CVE-000-00000; sid:100003;)
```

D. Stored XSS

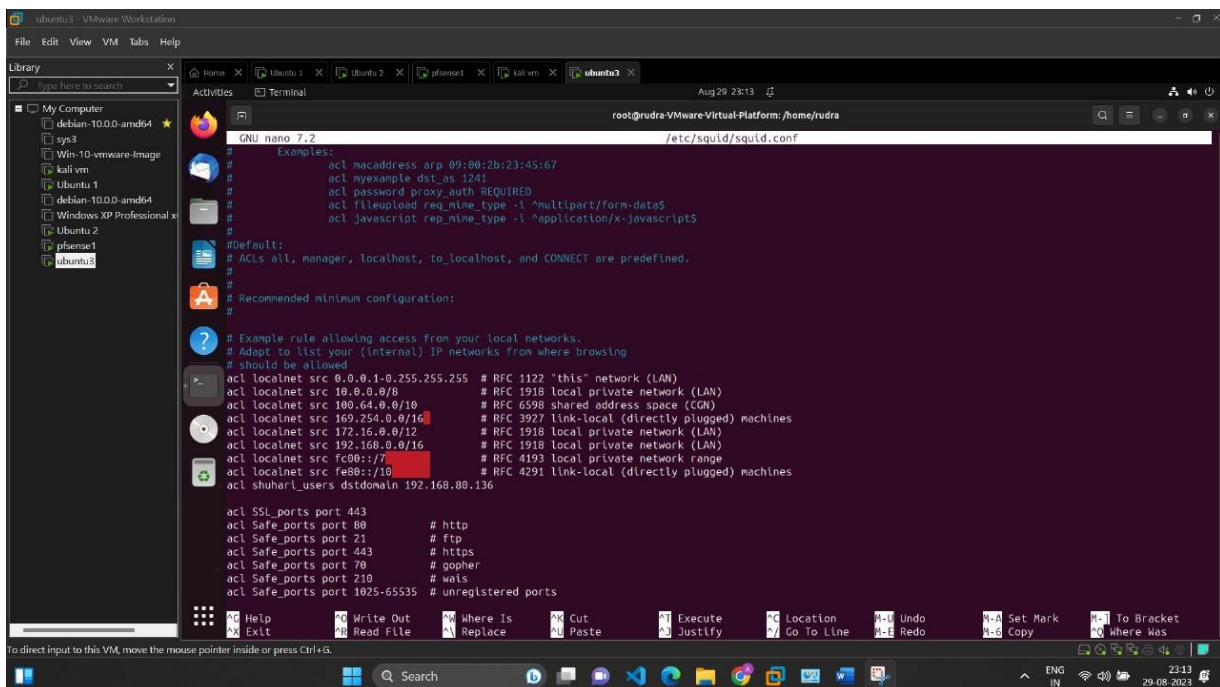
```
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"Stored xss Attack Detected"; flow:to_server; content:"POST"; http_method; content:"/DVWA/vulnerabilities/xss_s/";http_uri; content:"txtName="; pcre:"/[x3f&\n]txtName=[^&\r\n]*(\x3c\x3e|\%3c\%3e)/i"; reference:cve,CVE-0000-00000; sid:100004;)
```

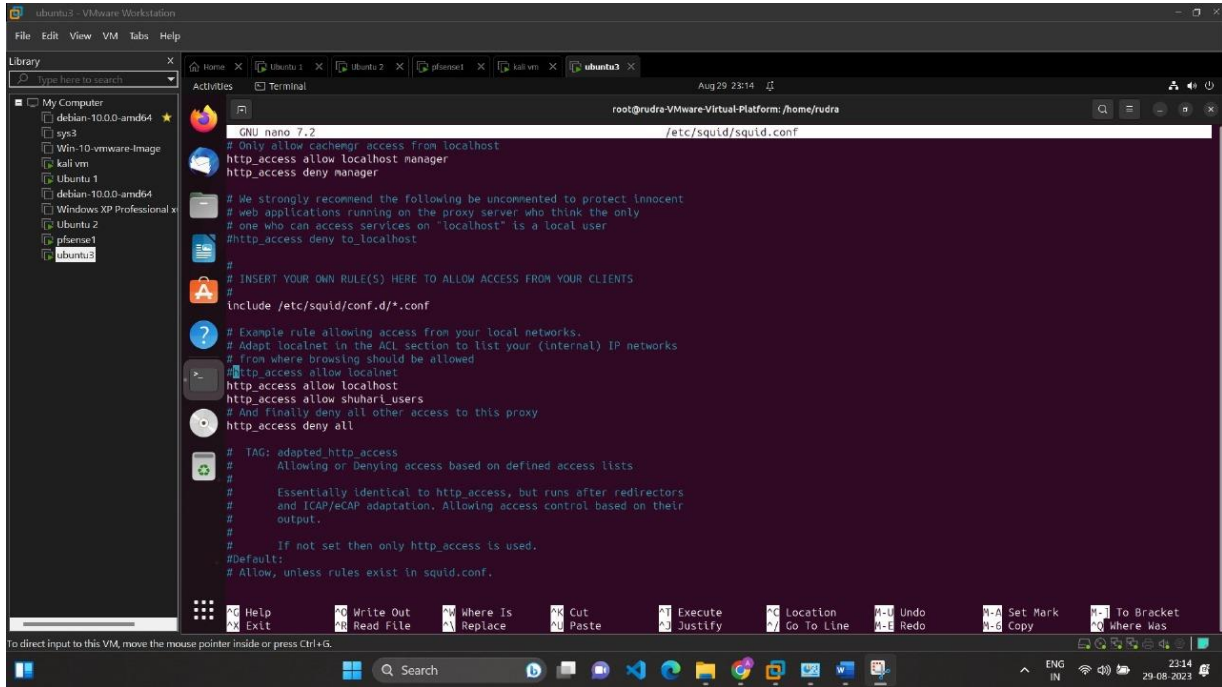
E. Reflected XSS

```
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"Reflected xss Attack Detected"; flow:to_server; content:"GET"; http_method; content:"/DVWA/vulnerabilities/xss_r/";http_uri; content:"name="; http_uri; pcre:"/[x3f&]name=[^&\r\n]*(\x3c\x3e|\%3c\%3e)/Ui"; reference:cve,CVE-000-00000; sid:100005;)
```

5.4 Load Balancer Configuration

Squid configuration





```
root@rudra:VMware-Virtual-Platform:/home/rudra
GNU nano 7.2 /etc/squid/squid.conf
# Only allow cachmgr access from localhost
http_access allow localhost manager
http_access deny manager

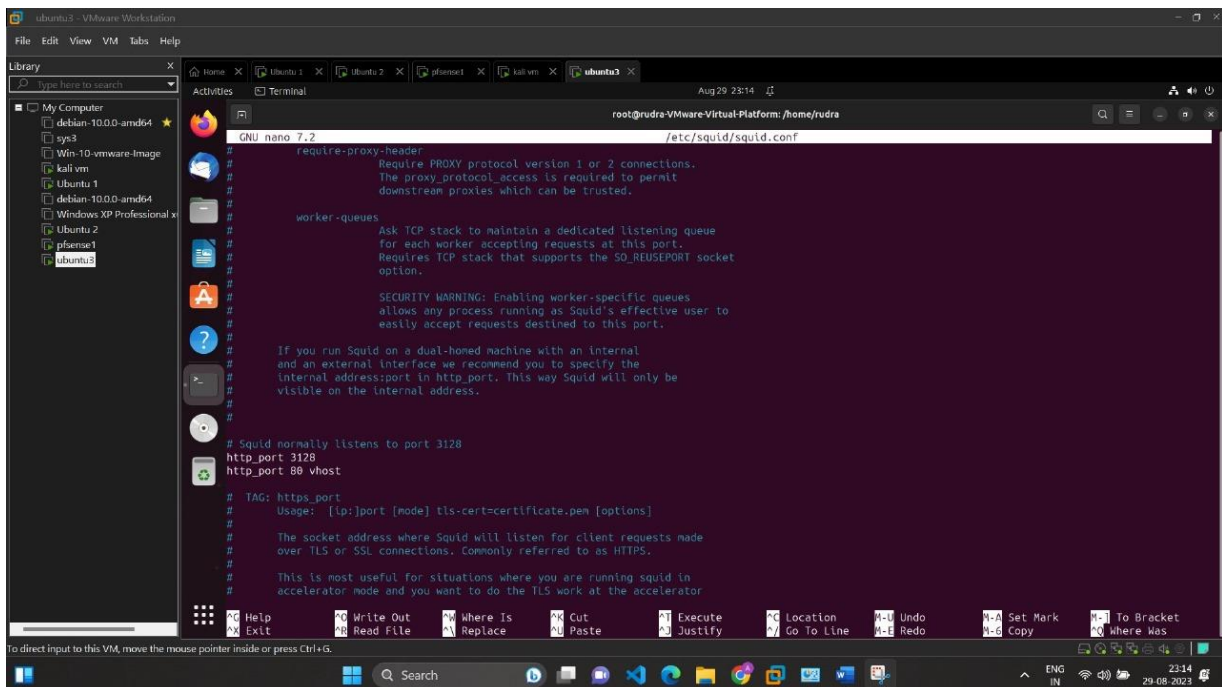
# We strongly recommend the following be uncommented to protect innocent
# web applications running on the proxy server who think the only
# one who can access services on "localhost" is a local user
#http_access deny to_localhost

#
# INSERT YOUR OWN RULE(S) HERE TO ALLOW ACCESS FROM YOUR CLIENTS
#

include /etc/squid/conf.d/*.conf

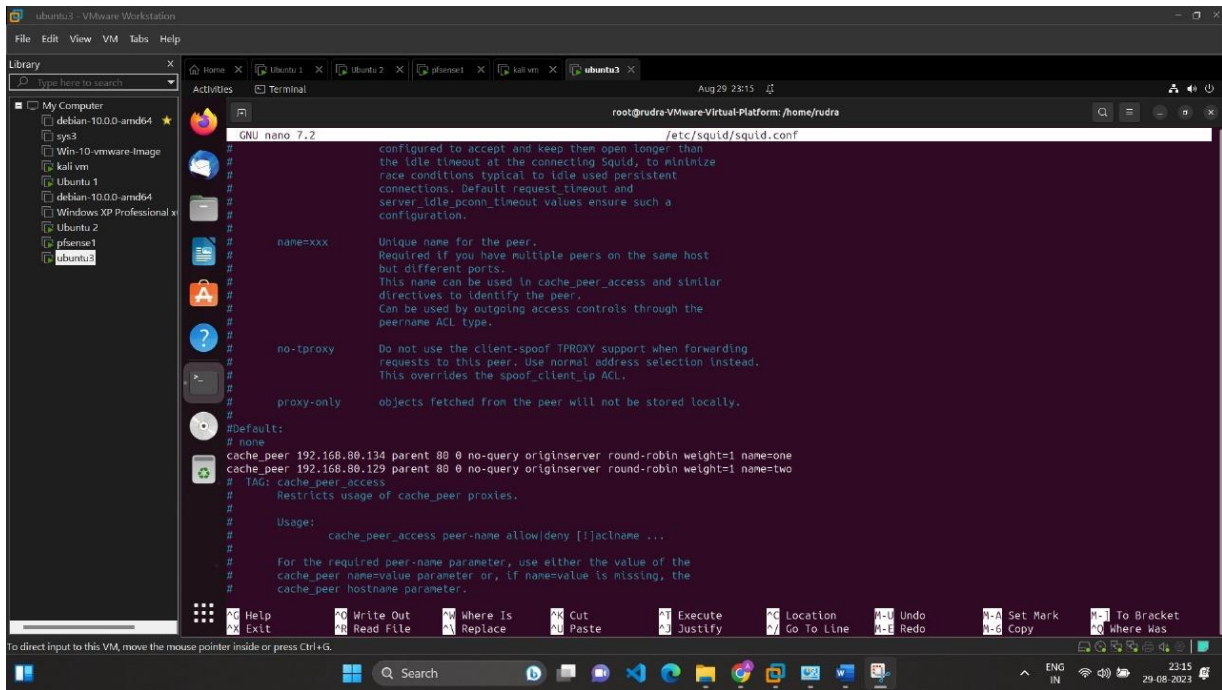
# Example rule allowing access from your local networks.
# Adapt localnet in the ACL section to list your (internal) IP networks
# from where browsing should be allowed
http_access allow localnet
http_access allow localhost
http_access allow shuhari_users
# And finally deny all other access to this proxy
http_access deny all

# TAG: adapted_http_access
# Allowing or Denying access based on defined access lists
#
# Essentially identical to http_access, but runs after redirectors
# and ICAP/eCAP adaptation. Allowing access control based on their
# output.
#
# If not set then only http_access is used.
#Default:
# Allow, unless rules exist in squid.conf.
```



```
root@rudra:VMware-Virtual-Platform:/home/rudra
GNU nano 7.2 /etc/squid/squid.conf
#
# require-proxy-header
#
# Require PROXY protocol version 1 or 2 connections.
# The proxy_protocol.access is required to permit
# downstream proxies which can be trusted.
#
# worker-queues
#
# Ask TCP stack to maintain a dedicated listening queue
# for each worker accepting requests at this port.
# Requires TCP stack that supports the SO_REUSEPORT socket
# option.
#
# SECURITY WARNING: Enabling worker-specific queues
# allows any process running as Squid's effective user to
# easily accept requests destined to this port.
#
# If you run Squid on a dual-homed machine with an internal
# and an external interface we recommend you to specify the
# internal address:port in http_port. This way Squid will only be
# visible on the internal address.
#
# Squid normally listens to port 3128
http_port 3128
http_port 88 vhost

# TAG: https_port
# Usage: [ip:]port [mode] tls-cert=certificate.pem [options]
#
# The socket address where Squid will listen for client requests made
# over TLS or SSL connections. Commonly referred to as HTTPS.
#
# This is most useful for situations where you are running squid in
# accelerator mode and you want to do the TLS work at the accelerator
```



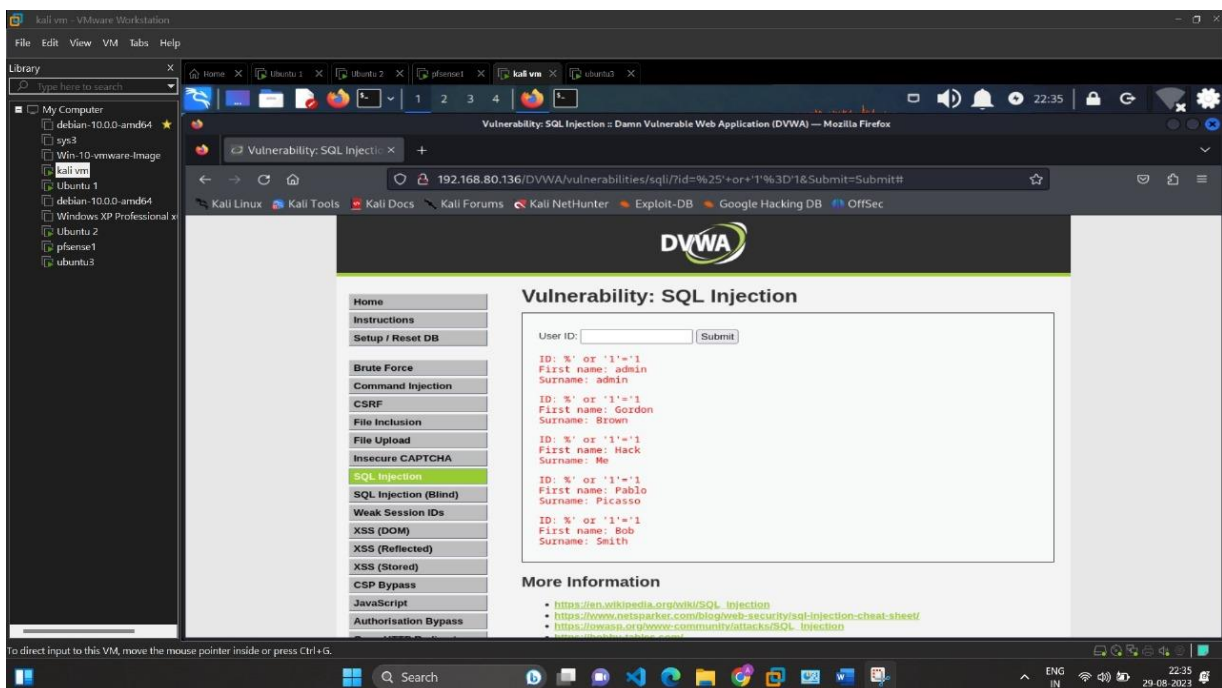
```
GNU nano 7.2 /etc/squid/squid.conf
#
# configured to accept and keep then open longer than
# the idle timeout at the connecting Squid, to minimize
# race conditions typical to idle used persistent
# connections. Default request timeout and
# server_idle_pconn_timeout values ensure such a
# configuration.
#
# name=xxx
# Unique name for the peer.
# Required if you have multiple peers on the same host
# but different ports.
# This name can be used in cache_peer_access and similar
# directives to identify the peer.
# Can be used by outgoing access controls through the
# peername ACL type.
#
# no-iproxy
# Do not use the client-spoof IPROXY support when forwarding
# requests to this peer. Use normal address selection instead.
# This overrides the spoof_client_ip ACL.
#
# proxy-only
# objects fetched from the peer will not be stored locally.
#
#Default:
# none
cache_peer 192.168.80.134 parent 80 0 no-query originserver round-robin weight=1 name=one
cache_peer 192.168.80.129 parent 80 0 no-query originserver round-robin weight=1 name=two
# IAG: cache_peer_access
# Restricts usage of cache_peer proxies.
#
# Usage:
# cache_peer_access peer-name allow|deny [|aclname ...
#
# For the required peer-name parameter, use either the value of the
# cache_peer name=value parameter or, if name=value is missing, the
# cache_peer hostname parameter.
```

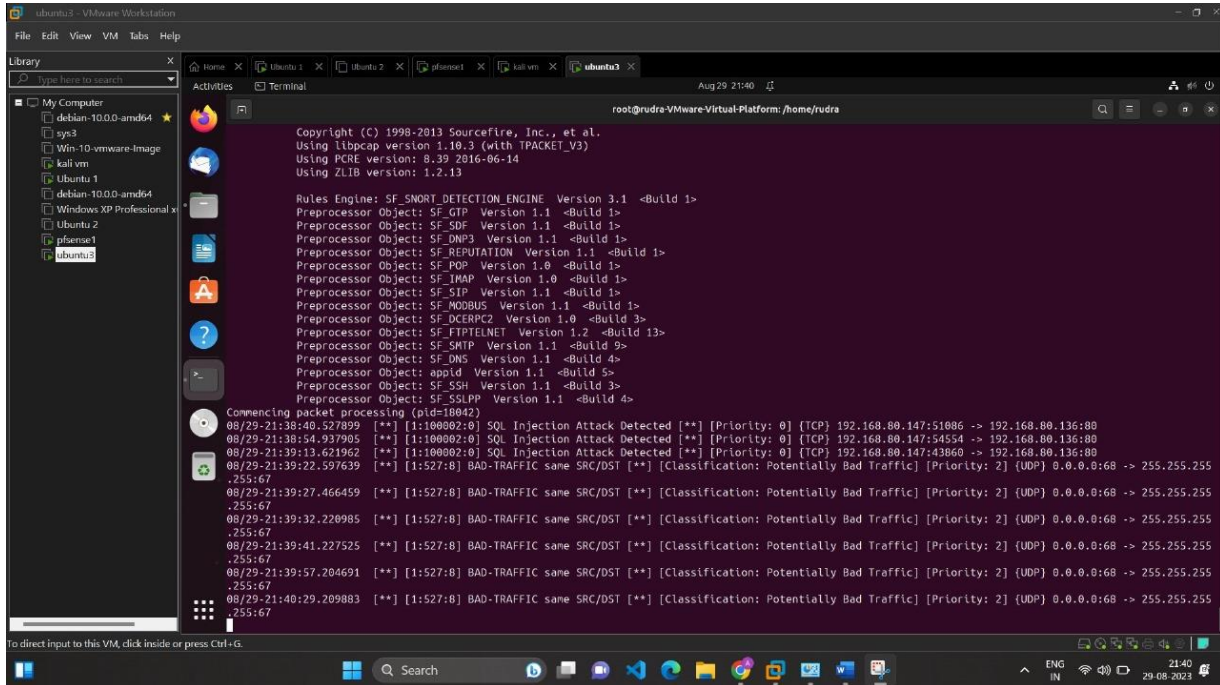

6. DIFFERENT ATTACKS TESTING

6.3.1 SQL Injection

SQL Injection is a type of cyber attack that targets vulnerabilities in web applications by injecting malicious SQL code into input fields or other user-controlled areas. A Web Application Firewall (WAF) is a security solution designed to protect web applications from various types of attacks, including SQL Injection. Here's how a WAF can detect and mitigate SQL Injection attacks:

SQL Attack Testing On Snort:

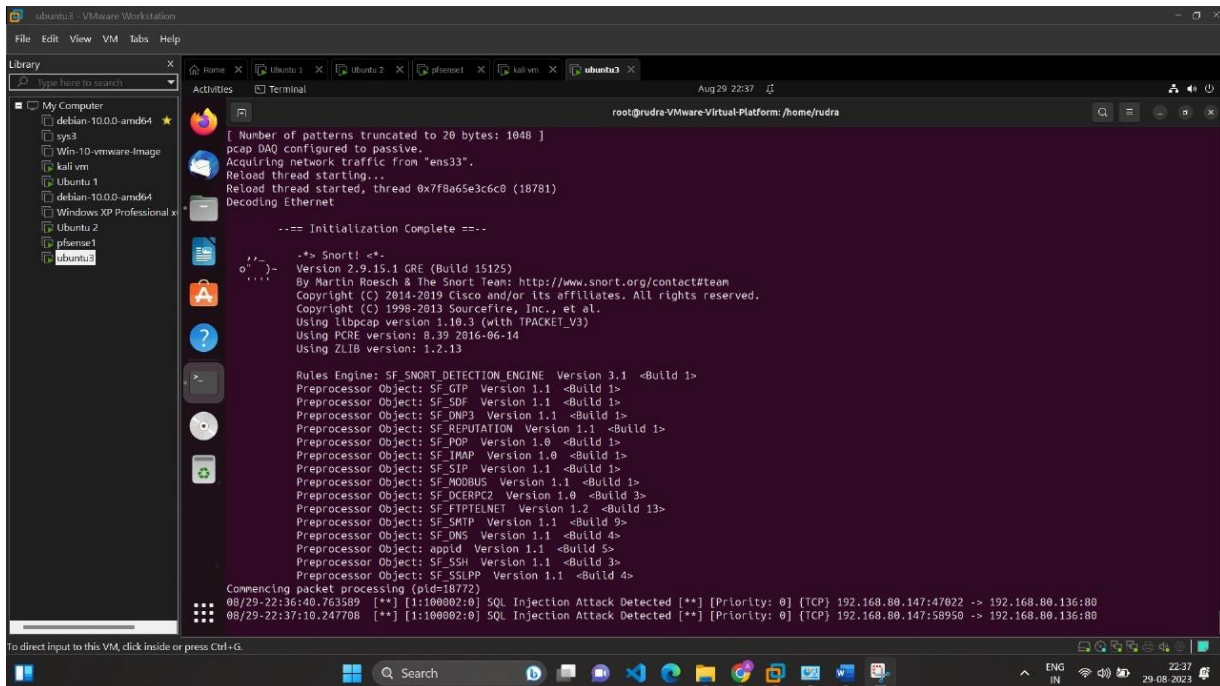




```
Copyright (C) 1998-2013 Sourcefire, Inc., et al.
Using libpcap version 1.10.3 (with TPACKET_V3)
Using PCRE version: 8.39 2016-06-14
Using ZLIB version: 1.2.13

Rules Engine: SF_SNORT_DETECTION_ENGINE Version 3.1 <Build 1>
Preprocessor Object: SF_GTP Version 1.1 <Build 1>
Preprocessor Object: SF_SDF Version 1.1 <Build 1>
Preprocessor Object: SF_DNP3 Version 1.1 <Build 1>
Preprocessor Object: SF_REPUTATION Version 1.1 <Build 1>
Preprocessor Object: SF_POP Version 1.0 <Build 1>
Preprocessor Object: SF_IMAP Version 1.0 <Build 1>
Preprocessor Object: SF_SIP Version 1.1 <Build 1>
Preprocessor Object: SF_MODBUS Version 1.1 <Build 1>
Preprocessor Object: SF_DCERPC2 Version 1.0 <Build 3>
Preprocessor Object: SF_FTPTELNET Version 1.2 <Build 13>
Preprocessor Object: SF_SMTP Version 1.1 <Build 9>
Preprocessor Object: SF_DNS Version 1.1 <Build 4>
Preprocessor Object: SF_APPID Version 1.1 <Build 5>
Preprocessor Object: SF_SSH Version 1.1 <Build 3>
Preprocessor Object: SF_SSLPP Version 1.1 <Build 4>

Commencing packet processing (pid=10042)
08/29-21:38:40.527899 [**] [1:100002:0] SQL Injection Attack Detected [**] [Priority: 0] [TCP] 192.168.80.147:51086 -> 192.168.80.136:80
08/29-21:38:54.937905 [**] [1:100002:0] SQL Injection Attack Detected [**] [Priority: 0] [TCP] 192.168.80.147:54554 -> 192.168.80.136:80
08/29-21:39:13.621962 [**] [1:100002:0] SQL Injection Attack Detected [**] [Priority: 0] [TCP] 192.168.80.147:43860 -> 192.168.80.136:80
08/29-21:39:22.597639 [**] [1:527:0] BAD-TRAFFIC same SRC/DST [**] [Classification: Potentially Bad Traffic] [Priority: 2] [UDP] 0.0.0.0:68 -> 255.255.255.255:67
08/29-21:39:27.466459 [**] [1:527:0] BAD-TRAFFIC same SRC/DST [**] [Classification: Potentially Bad Traffic] [Priority: 2] [UDP] 0.0.0.0:68 -> 255.255.255.255:67
08/29-21:39:32.220985 [**] [1:527:0] BAD-TRAFFIC same SRC/DST [**] [Classification: Potentially Bad Traffic] [Priority: 2] [UDP] 0.0.0.0:68 -> 255.255.255.255:67
08/29-21:39:41.227525 [**] [1:527:0] BAD-TRAFFIC same SRC/DST [**] [Classification: Potentially Bad Traffic] [Priority: 2] [UDP] 0.0.0.0:68 -> 255.255.255.255:67
08/29-21:39:57.204691 [**] [1:527:0] BAD-TRAFFIC same SRC/DST [**] [Classification: Potentially Bad Traffic] [Priority: 2] [UDP] 0.0.0.0:68 -> 255.255.255.255:67
08/29-21:40:29.209883 [**] [1:527:0] BAD-TRAFFIC same SRC/DST [**] [Classification: Potentially Bad Traffic] [Priority: 2] [UDP] 0.0.0.0:68 -> 255.255.255.255:67
```



```
[ Number of patterns truncated to 20 bytes: 1048 ]
pcap DAG configured to passive.
Acquiring network traffic from "ens33".
Reload thread starting...
Reload thread started, thread 0x7f8a65e3c6c0 (18781)
Decoding Ethernet

--- Initialization Complete ---

--* Snort! *-
Version 2.9.15.1 GRE (Build 15125)
By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
Copyright (C) 2014-2019 Cisco and/or its affiliates. All rights reserved.
Copyright (C) 1998-2013 Sourcefire, Inc., et al.
Using libpcap version 1.10.3 (with TPACKET_V3)
Using PCRE version: 8.39 2016-06-14
Using ZLIB version: 1.2.13

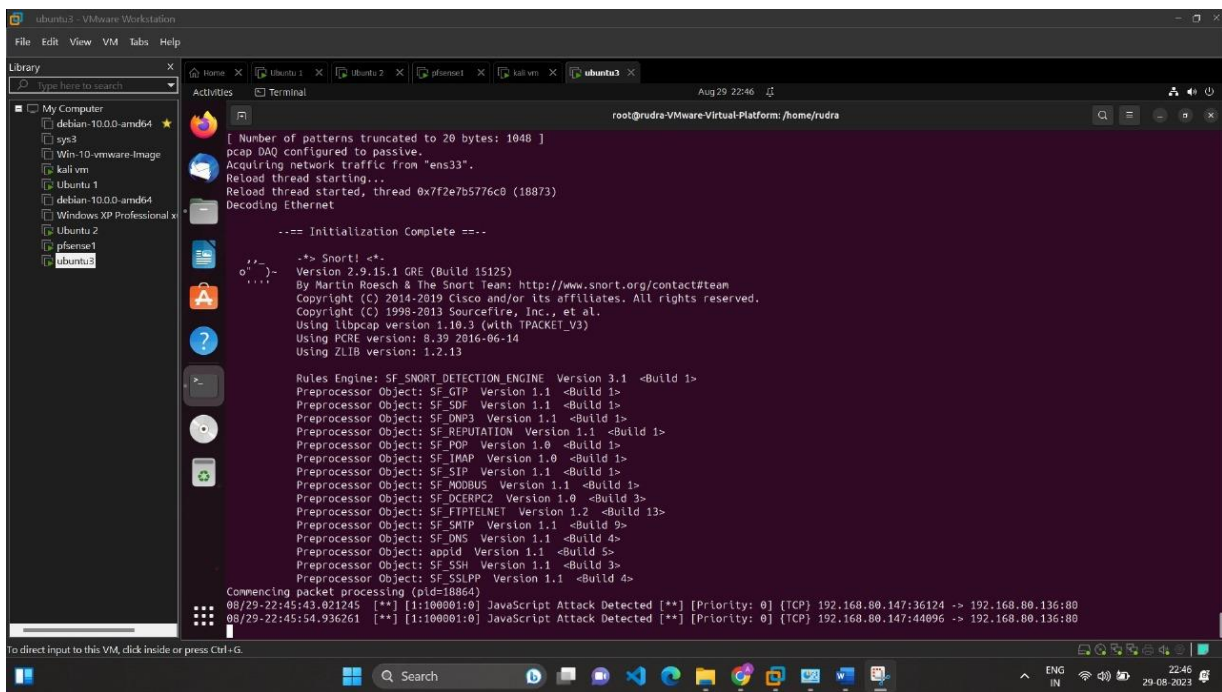
Rules Engine: SF_SNORT_DETECTION_ENGINE Version 3.1 <Build 1>
Preprocessor Object: SF_GTP Version 1.1 <Build 1>
Preprocessor Object: SF_SDF Version 1.1 <Build 1>
Preprocessor Object: SF_DNP3 Version 1.1 <Build 1>
Preprocessor Object: SF_REPUTATION Version 1.1 <Build 1>
Preprocessor Object: SF_POP Version 1.0 <Build 1>
Preprocessor Object: SF_IMAP Version 1.0 <Build 1>
Preprocessor Object: SF_SIP Version 1.1 <Build 1>
Preprocessor Object: SF_MODBUS Version 1.1 <Build 1>
Preprocessor Object: SF_DCERPC2 Version 1.0 <Build 3>
Preprocessor Object: SF_FTPTELNET Version 1.2 <Build 13>
Preprocessor Object: SF_SMTP Version 1.1 <Build 9>
Preprocessor Object: SF_DNS Version 1.1 <Build 4>
Preprocessor Object: SF_APPID Version 1.1 <Build 5>
Preprocessor Object: SF_SSH Version 1.1 <Build 3>
Preprocessor Object: SF_SSLPP Version 1.1 <Build 4>

Commencing packet processing (pid=18772)
08/29-22:36:40.763589 [**] [1:100002:0] SQL Injection Attack Detected [**] [Priority: 0] [TCP] 192.168.80.147:47022 -> 192.168.80.136:80
08/29-22:37:10.247708 [**] [1:100002:0] SQL Injection Attack Detected [**] [Priority: 0] [TCP] 192.168.80.147:58950 -> 192.168.80.136:80
```

6.3.2 JavaScript Attack Testing On Snort:

JavaScript is a widely used programming language that adds interactivity and dynamic behavior to websites. It's mainly executed within web browsers, allowing developers to create responsive and interactive user experiences. JavaScript enables tasks like form validation, animations, and updating content without needing to reload the entire web page. It's a crucial component in modern web development and supports both client-side and server-side applications through platforms like Node.js.

Testing JavaScript-based attacks on Snort, an Intrusion Detection and Prevention System (IDPS), involves creating scenarios that simulate real-world attacks and observing how Snort detects and responds to them. Here's an outline of how you can conduct JavaScript attack testing using Snort:



```
root@rudra-VMware-Virtual-Platform: /home/rudra

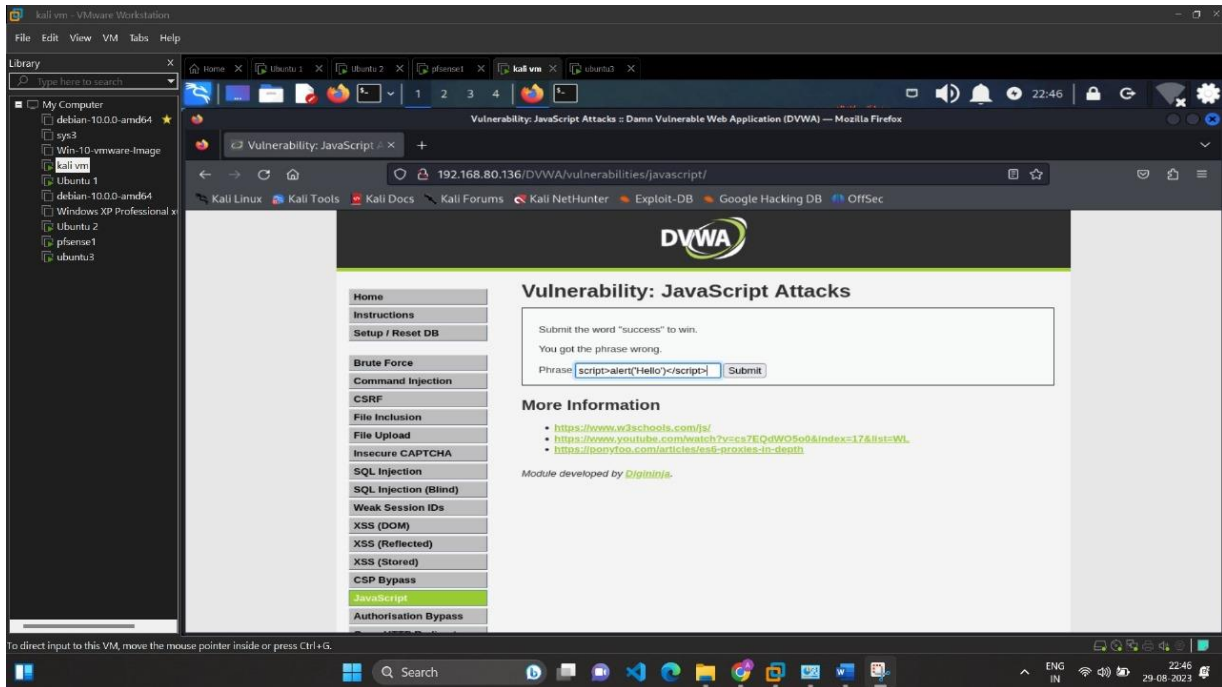
[ Number of patterns truncated to 20 bytes: 1048 ]
pcap DAQ configured to passive.
Acquiring network traffic from "ens33".
Reload thread starting...
Reload thread started, thread 0x7f2e7b5776c0 (18873)
Decoding Ethernet

---- Initialization Complete ----

--> Snort! <--
Version 2.9.15.1 GRE (Build 15125)
By Martin Roesch & The Snort Team: http://www.snort.org/contact@team
Copyright (C) 2014-2019 Cisco and/or its affiliates. All rights reserved.
Copyright (C) 1998-2013 Sourcefire, Inc., et al.
Using libpcap version 1.10.3 (with TPACNET_V3)
Using PCRE version: 8.39 2016-06-14
Using ZLIB version: 1.2.13

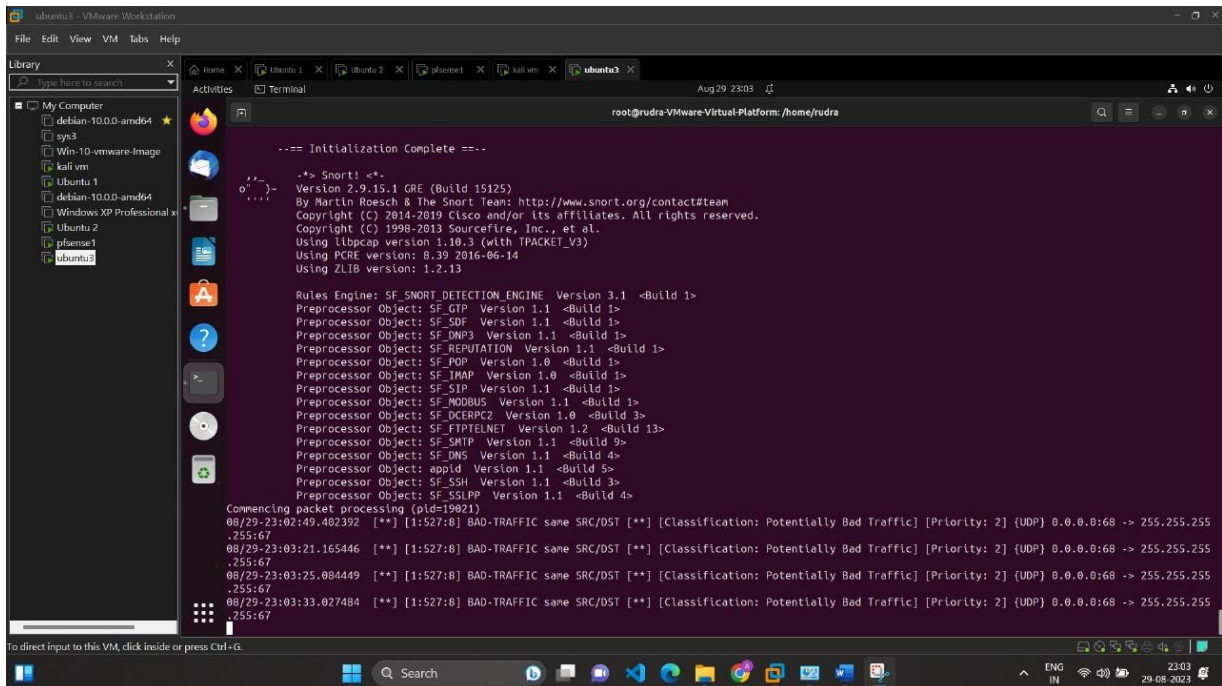
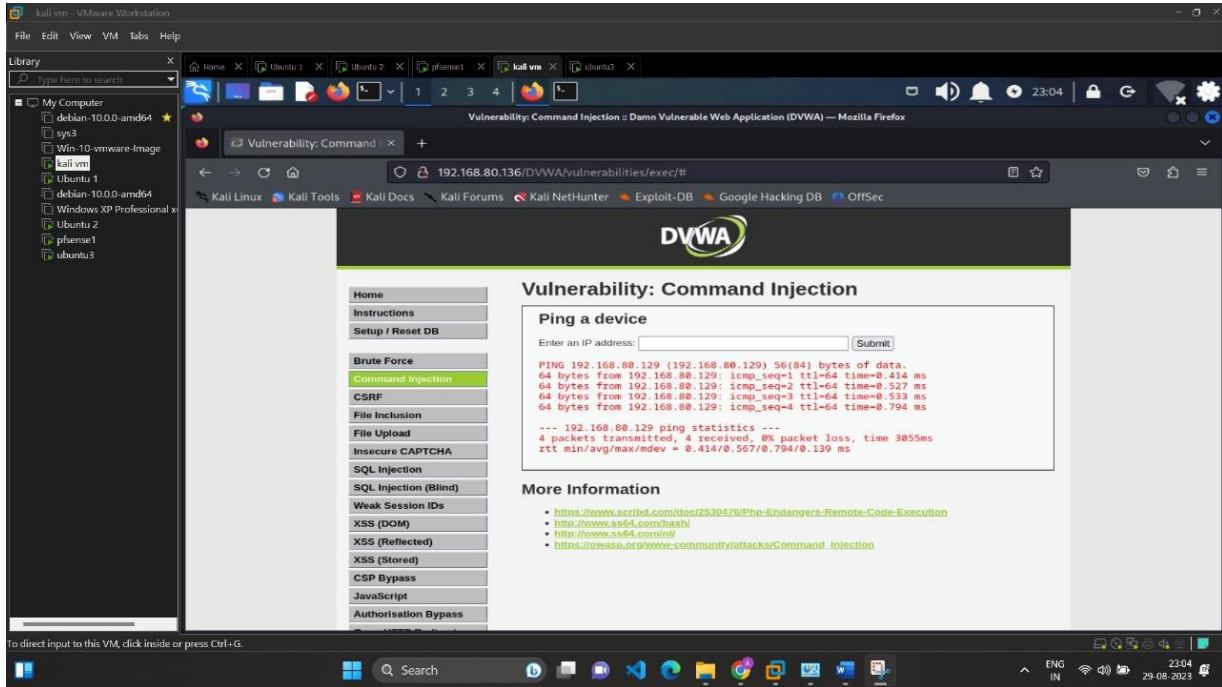
Rules Engine: SF_SNORT_DETECTION_ENGINE Version 3.1 <Build 1>
Preprocessor Object: SF_GTP Version 1.1 <Build 1>
Preprocessor Object: SF_SDF Version 1.1 <Build 1>
Preprocessor Object: SF_DNP3 Version 1.1 <Build 1>
Preprocessor Object: SF_REPUTATION Version 1.1 <Build 1>
Preprocessor Object: SF_POP Version 1.0 <Build 1>
Preprocessor Object: SF_IMAP Version 1.0 <Build 1>
Preprocessor Object: SF_SIP Version 1.1 <Build 1>
Preprocessor Object: SF_MQEMIBUS Version 1.1 <Build 1>
Preprocessor Object: SF_DCERPC2 Version 1.0 <Build 3>
Preprocessor Object: SF_FTPTELNET Version 1.2 <Build 13>
Preprocessor Object: SF_SRP Version 1.1 <Build 9>
Preprocessor Object: SF_DNS Version 1.1 <Build 4>
Preprocessor Object: apdld Version 1.1 <Build 5>
Preprocessor Object: SF_SSH Version 1.1 <Build 3>
Preprocessor Object: SF_SSLLPP Version 1.1 <Build 4>

Commencing packet processing (pid=18864)
08/29-22:45:43.021245 [**] [1:100001:0] JavaScript Attack Detected [**] [Priority: 0] [TCP] 192.168.80.147:36124 -> 192.168.80.136:80
08/29-22:45:54.936261 [**] [1:100001:0] JavaScript Attack Detected [**] [Priority: 0] [TCP] 192.168.80.147:44096 -> 192.168.80.136:80
```

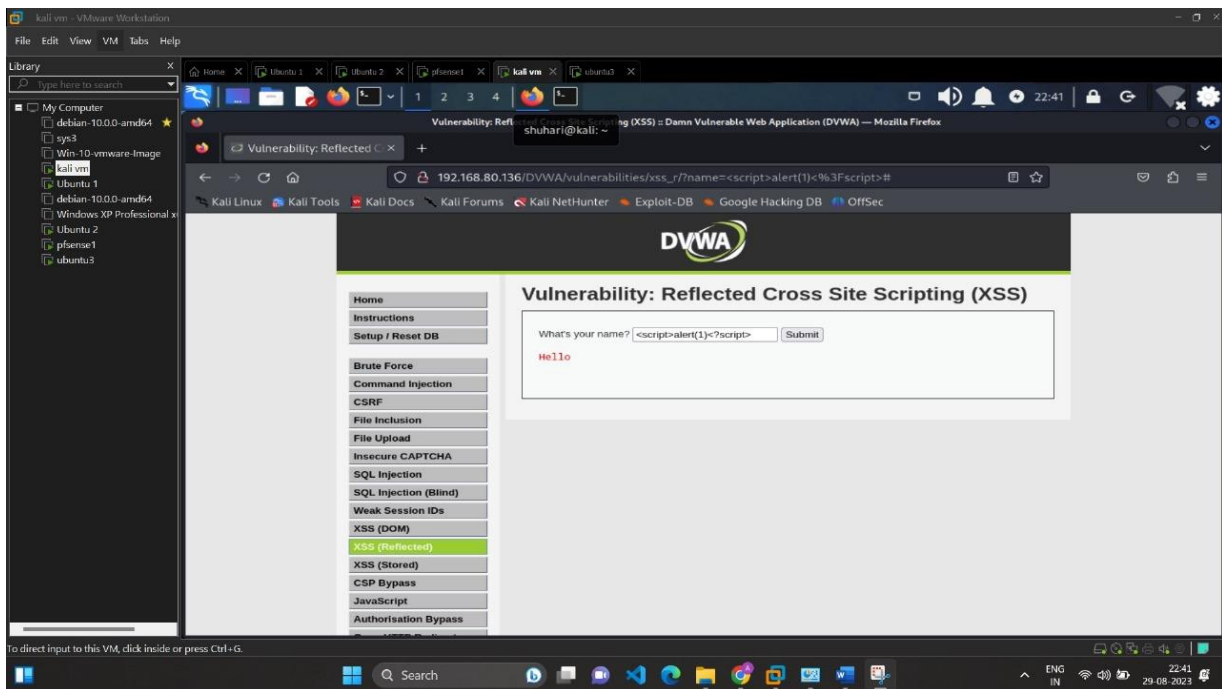
6.3.3 Command Injection Testing On Snort:

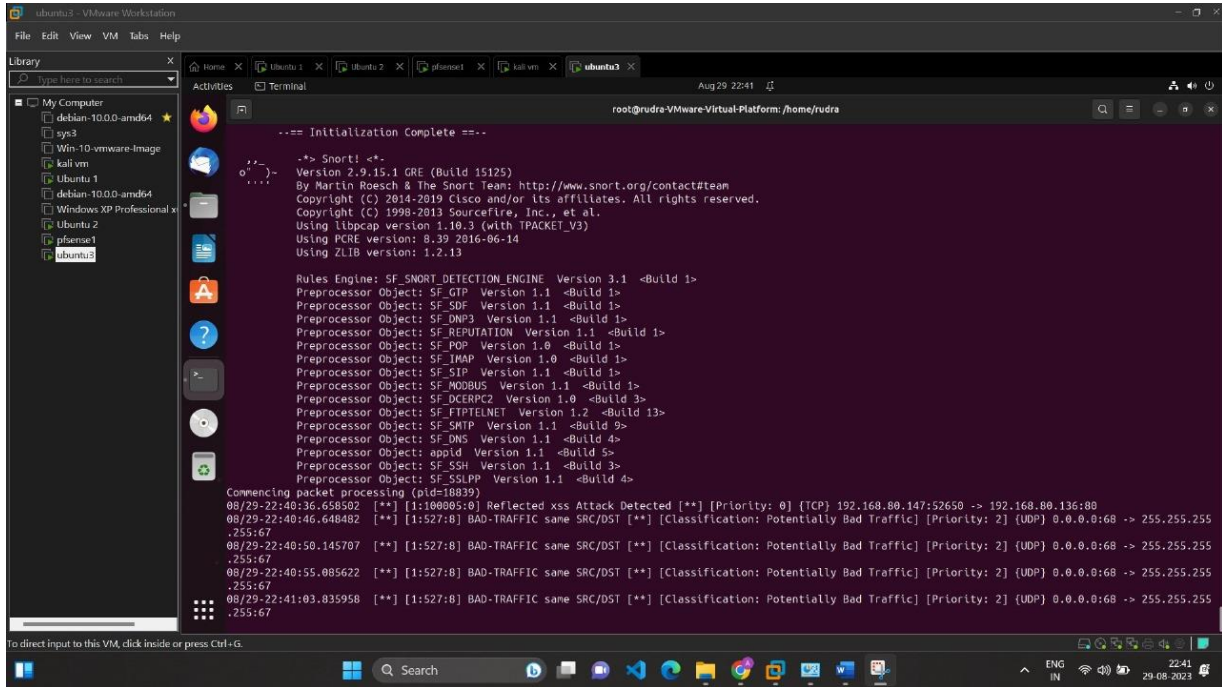
Command Injection testing on Snort involves evaluating how well the Intrusion Detection and Prevention System (IDPS) can detect and prevent command injection attacks. Command injection is a type of security vulnerability where an attacker manipulates input fields to execute arbitrary system commands. Here's an outline of how you can perform Command Injection testing using Snort:



6.3.4 XSS Reflected

Cross-Site Scripting (XSS) Reflected is a type of cyber attack where a malicious script is injected into a web application, and when a user interacts with a specially crafted link or input, the script is executed within the context of the victim's browser. This allows the attacker to steal sensitive data, hijack user sessions, or perform other malicious actions. XSS Reflected attacks typically occur when an application fails to properly sanitize user inputs before displaying them to other users. To defend against XSS Reflected attacks, web developers should implement proper input validation and output encoding, as well as use security mechanisms like Content Security Policy (CSP)





The screenshot shows a VMware Workstation interface with a terminal window open. The terminal displays the output of the Snort initialization process and subsequent log entries. The logs indicate a reflected XSS attack detected on 08/29-22:40:36.658502. Subsequent log entries on 08/29-22:40:46.648482, 08/29-22:40:50.145707, 08/29-22:40:55.085622, and 08/29-22:41:03.835958 all classify the traffic as 'Potentially Bad Traffic' with a priority of 2.

```
root@rudra-VMware-Virtual-Platform: /home/rudra

=== Initialization Complete ===

--> Snort! <*.
Version 2.9.15.1 GRE (Build 15125)
By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
Copyright (C) 2014-2019 Cisco and/or its affiliates. All rights reserved.
Copyright (C) 1998-2013 Sourcefire, Inc., et al.
Using libpcap version 1.10.3 (with TPACKET_V3)
Using PCRE version: 8.39 2016-06-14
Using ZLIB version: 1.2.13

Rules Engine: SF_SNORT_DETECTION_ENGINE Version 3.1 <Build 1>
Preprocessor Object: SF_GTP Version 1.1 <Build 1>
Preprocessor Object: SF_SDF Version 1.1 <Build 1>
Preprocessor Object: SF_DNP3 Version 1.1 <Build 1>
Preprocessor Object: SF_REPUTATION Version 1.1 <Build 1>
Preprocessor Object: SF_POP Version 1.0 <Build 1>
Preprocessor Object: SF_IMAP Version 1.0 <Build 1>
Preprocessor Object: SF_SIP Version 1.1 <Build 1>
Preprocessor Object: SF_MODBUS Version 1.1 <Build 1>
Preprocessor Object: SF_DCERPC2 Version 1.0 <Build 3>
Preprocessor Object: SF_FTPTELNET Version 1.2 <Build 13>
Preprocessor Object: SF_SMTP Version 1.1 <Build 9>
Preprocessor Object: SF_DNS Version 1.1 <Build 4>
Preprocessor Object: appid Version 1.1 <Build 5>
Preprocessor Object: SF_SSH Version 1.1 <Build 3>
Preprocessor Object: SF_SSLPP Version 1.1 <Build 4>

Commencing packet processing (pid=10839)
08/29-22:40:36.658502  [**] [1:100005:0] Reflected xss Attack Detected [**] [Priority: 0] (TCP) 192.168.80.147:52650 -> 192.168.80.136:80
08/29-22:40:46.648482  [**] [1:527:0] BAD-TRAFFIC same SRC/DST [**] [Classification: Potentially Bad Traffic] [Priority: 2] (UDP) 0.0.0.0:68 -> 255.255.255
.255:67
08/29-22:40:50.145707  [**] [1:527:0] BAD-TRAFFIC same SRC/DST [**] [Classification: Potentially Bad Traffic] [Priority: 2] (UDP) 0.0.0.0:68 -> 255.255.255
.255:67
08/29-22:40:55.085622  [**] [1:527:0] BAD-TRAFFIC same SRC/DST [**] [Classification: Potentially Bad Traffic] [Priority: 2] (UDP) 0.0.0.0:68 -> 255.255.255
.255:67
08/29-22:41:03.835958  [**] [1:527:0] BAD-TRAFFIC same SRC/DST [**] [Classification: Potentially Bad Traffic] [Priority: 2] (UDP) 0.0.0.0:68 -> 255.255.255
.255:67
```

7. CONCLUSION

The above project discusses potential benefits of introducing a Web Application Firewall for the purpose of detecting, preventing and eliminating malicious users' behavior from external network. Making use of these learning processes for the benefit of computer security is recently gaining audience among researchers and, with the growing number and sophistication of the web attack attempts, it appears to be a logical approach to address these attacks and prevent them. The obvious benefit of such a WAF is the ability to fend off a large class of dangerous attack patterns directed at the web application, with a very limited set-up time.

8. References

- Aljawarneh, S., Laing, C., & Paul, V. (2013). *Verification of Web Content Integrity: A new approach to protecting servers against tampering. School of Computing, Engineering & Information Sciences Northumbria University, Newcastle upon Tyne*, 1-6. <https://doi.org/10.13140/2.1.4225.6648>.
- Ashwini, R., et al. (November 2014). *Enhanced Security Approach for Detecting Intrusions in Multitier Web Applications. International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, 3(11), 3850 -3854.
- Chavan, S., & Meshram, B. (March2015). *Classification of Web Application Vulnerabilities. International Journal of Engineering Science and Innovative Technology (IJESIT)*, 2(2), 226-234
- Chou, T. (June 2013). *security threats on cloud computing vulnerabilities. International Journal of Computer Science & Information Technology (IJCSIT)*, 5(3), 79-88. <https://doi.org/10.5121/ijcsit.2013.5306>
- Garg, S., & Narula, P. (2014). *A novel Approach and Implementation Of Secured Algorithm Against SQL Injections. International Journal of Enterprise Computing and Business Systems*, 4(1), 1-7

