



INSTITUTE FOR ADVANCED COMPUTING
AND
SOFTWARE DEVELOPMENT
AKURDI, PUNE

DOCUMENTATIONON

**“Data Protection with using
authorization,TLS,IDS,IPS and DMZ”**

PG-DITISS SEPT-2023

SUBMITTEDBY

GroupNo:07

Harsh (239413)

Hemant Bhangale(239414)

MR. ROHIT PURANIK

CENTRE CO-ORDINATOR

MR. SUSHMA HATTARKI

PROJECT GUIDE

ABSTRACT

This project focuses on enhancing the existing Demilitarized Zone (DMZ) infrastructure by implementing advanced security measures and expanding its functionality to include website hosting capabilities on the Wide Area Network (WAN). The primary objectives of the project include deploying a robust Public Key Infrastructure (PKI) setup to ensure secure communications, integrating network security with PFSense firewall and Snort as an Intrusion Detection System (IDS) and Intrusion Prevention System (IPS), and deploying a File Transfer Protocol (FTP) file server with granular permission controls for different users within the Local Area Network (LAN).

Additionally, the project encompasses the setup of a MariaDB database to support the hosted website on the WAN, ensuring efficient data storage and retrieval. By implementing these measures, the project aims to fortify the DMZ infrastructure against potential security threats while enhancing its functionality to meet the evolving demands of website hosting and secure data transmission.

INDEX

| | |
|---|----|
| 1. INTRODUCTION | 1 |
| 2. PROBLEM STATEMENT | 2 |
| 3. SURVEY OF TECHNOLOGY | 3 |
| 4.REQUIREMENT&ANALYSIS | 4 |
| 5. WEB SERVER INSTALLATION AND SETUP | 6 |
| 5.1 WEBSITE HOSTING WITH DNS | 6 |
| 5.2 PKI SETUP | 15 |
| 5.3 IPTABLES | 16 |
| 5.4 MODSECURITY WAF | 17 |
| 5.5 NAGIOS FOR MONITORING | 21 |
| 6. PFSENSE AS MAIN FIREWALL AND DEFAULT ROUTER | 23 |
| 7.DATABASE MACHINE | 24 |
| 8. SNORT AS NIDS | 25 |
| 9. FILE SERVER(VSFTPD) | 28 |
| 10.FUTURESCOPE | 29 |
| 11. CONCLUSION | 30 |

1. INTRODUCTION

In the contemporary landscape of digital connectivity, safeguarding network infrastructure against cyber threats is paramount. Demilitarized Zones (DMZs) serve as a crucial line of defense, segregating internal networks from external entities while facilitating controlled access to essential services. However, with the evolving sophistication of cyber attacks, the need to fortify DMZ infrastructure with advanced security measures has become imperative.

This project sets out to enhance the existing DMZ infrastructure by implementing a comprehensive array of security measures and expanding its capabilities to include website hosting on the Wide Area Network (WAN). By integrating cutting-edge technologies such as Public Key Infrastructure (PKI), PFSense firewall, Snort Intrusion Detection and Prevention System (IDS/IPS), and stringent access controls for FTP file servers, this project aims to establish a robust security framework that ensures secure communication channels and protects against potential breaches.

Furthermore, the deployment of a MariaDB database to support the hosted website on the WAN underscores the project's commitment to not only bolstering security but also enhancing functionality and efficiency. Through meticulous planning and strategic implementation, this project endeavors to optimize the DMZ infrastructure, aligning it with contemporary security standards and equipping it to effectively mitigate emerging cyber threats while meeting the demands of modern network environments.

2. PROBLEM STATEMENT

Enhancing DMZ infrastructure by implementing advanced security measures, integrating PfSense firewall with Snort IDS/IPS, deploying PKI for secure communications, hosting website on WAN, and FTP server with strict permission controls. Challenges include complex setup, inadequate security, limited functionality, and compliance issues, necessitating a comprehensive solution for fortified and compliant infrastructure.

3. SURVEY OF TECHNOLOGY

Sure, here's a survey of the technologies used in the project along with brief descriptions:

3.1 Demilitarized Zone (DMZ):

- Description: A DMZ is a network segment that acts as a buffer zone between the internal network and the external untrusted network, typically the internet. It allows controlled access to selected services while protecting the internal network from potential security threats.

3.2 Public Key Infrastructure (PKI):

- Description: PKI is a framework that enables secure communication by using digital certificates and public-private key pairs. It provides mechanisms for authentication, encryption, and digital signatures, ensuring the confidentiality, integrity, and authenticity of data exchanged over insecure networks.

3.3 PfSense Firewall:

- Description: PfSense is an open-source firewall and routing platform based on FreeBSD. It provides a comprehensive set of features for network security, including firewall rules, VPN support, traffic shaping, and intrusion detection/prevention capabilities.

3.4 Snort Intrusion Detection System (IDS) and Intrusion Prevention System (IPS):

- Description: Snort is an open-source network intrusion detection and prevention system that monitors network traffic for suspicious activity and can take actions to block or prevent detected intrusions. It uses signature-based detection, anomaly detection, and protocol analysis to identify and respond to potential threats.

3.5 File Transfer Protocol (FTP) Server:

- Description: FTP is a standard network protocol used for transferring files between a client and a server on a computer network. An FTP server facilitates the storage and retrieval of files, allowing users to upload and download files securely over the network.

3.6 MariaDB Database:

- Description: MariaDB is an open-source relational database management system, compatible with MySQL. It provides a robust and scalable database solution for storing, managing, and retrieving data in various applications, including web hosting and other server-based services.

By leveraging these technologies, the project aims to enhance the security, functionality, and efficiency of the DMZ infrastructure, ensuring secure communication, robust network defense, efficient data management, and compliance with relevant regulations and standards.

4. REQUIREMENT & ANALYSIS

Software Requirement:

- Virtual Environment and Network Configuration –
 - VMWARE
- WEB SERVER –
 - OS - Debian12
 - Web server - Apache
 - Event Monitoring Tool – Nagios
 - WAF – ModSecurity
 - Packet Filtering firewall – IPTABLES
- Primary Firewall and Routing Platform –
 - PFSense
- DataBase –
 - OS – Debian 12
 - Database – MariaDB
- IDS/IPS –
 - Snort
- File Server –
 - Vsftpd server
 - FTP client

Hardware Requirement:

- Minimum RAM : 8GB.
- Recommended RAM: 16GB.
- Hard Drive Space : 100 GB.

Network requirements:

- WAN (WEB server ip- 192.168.40.128/24 with gateway appropriate gateway)
- LAN(DataBase Machine ip – 192.168.50.128, Snort ip- 192.168.50.129, File Server ip - 192.168.50.127)
- PFSense[LAN adapter ip – 192.168.50.10 , WAN adapter ip – 192.168.40.10]

5. WEB SERVER INSTALLATION AND SETUP

5.1 Website Hosting

PreRequisite – Debian 12 installed in VMWARE and the non-root user has administrative privileges

Install Apache Web Server –

```
sudo apt update
sudo apt install apache2
```

Remove default apache website present in /var/www/html directory and create you website in this directory.

```
sudo rm -rf /var/www/html/
```

Install php and database dependencies

```
sudo apt install php php-mysql mariadb-client
```

Create the following Website files in /var/www/html using a text editor such as vim/nano

Files –

a.index.php

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Your Web App</title>
</head>
<body>

  <h1>Welcome to Your Web App</h1>

  <ul>
    <li><a href="signup.php">Signup</a></li>
    <li><a href="login.php">Login</a></li>
    <li><a href="admin_login.php">Admin login</a></li>
    <li><a href="content.php">Content (Requires Login)</a></li>
  </ul>

</body>
</html>
```


b.signup.php

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Signup Page</title>
</head>
<body>

  <h2>Signup</h2>
  <form action="signup_process.php" method="post">
    <label for="username">Username:</label>
    <input type="text" name="username" required><br>

    <label for="password">Password:</label>
    <input type="password" name="password" required><br>

    <input type="submit" value="Signup">
  </form>

</body>
</html>
```

c.signup_process.php

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  // Database connection parameters
  $servername = "192.168.50.128";
  $username = "webuser";
  $password = "iacsd@123";
  $dbname = "web";

  // Create connection
  $conn = new mysqli($servername, $username, $password, $dbname);

  // Check connection
  if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
  }

  // Retrieve values from the signup form
  $username = $_POST['username'];
  $password = password_hash($_POST['password'], PASSWORD_BCRYPT);

  // Perform a secure query using prepared statements
  $sql = "INSERT INTO users (username, password) VALUES (?, ?)";
```

```
$stmt = $conn->prepare($sql);
$stmt->bind_param("ss", $username, $password);

if ($stmt->execute()) {
    echo "Signup successful!";
} else {
    echo "Error: " . $stmt->error;
}

$stmt->close();
$conn->close();
}
?>
```

d.Login.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login Page</title>
</head>
<body>

    <h2>Login</h2>
    <form action="login_process.php" method="post">
        <label for="username">Username:</label>
        <input type="text" name="username" required><br>

        <label for="password">Password:</label>
        <input type="password" name="password" required><br>

        <input type="submit" value="Login">
    </form>

</body>
</html>
```

e.Login_process.php

```
<?php
session_start();

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Database connection parameters
    $servername = "192.168.50.128";
```

```

$username = "webuser";
$password = "iacsd@123";
$dbname = "web";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Retrieve values from the login form
$username = $_POST['username'];
$enteredPassword = $_POST['password'];

// Perform a secure query using prepared statements
$sql = "SELECT id, username, password FROM users WHERE username = ?";
$stmt = $conn->prepare($sql);
$stmt->bind_param("s", $username);
$stmt->execute();
$result = $stmt->get_result();

if ($result->num_rows > 0) {
    $row = $result->fetch_assoc();
    $hashedPassword = $row['password'];

    // Check if the entered password matches the stored hashed password
    if (password_verify($enteredPassword, $hashedPassword)) {
        // Successful login
        $_SESSION['user_id'] = $row['id'];
        header("Location: content.php");
        exit();
    } else {
        echo "Invalid username or password.";
    }
} else {
    echo "Invalid username or password.";
}

$stmt->close();
$conn->close();
}
?>

```

f. content.php

```

<?php
session_start();

```

```
if (isset($_SESSION['user_id'])) {  
    echo "Welcome to the content page!";  
} else {  
    header("Location: login.php");  
    exit();  
}  
?>
```

g.Admin_login.php

```
<?php  
session_start();  
  
// Handle form submission  
if ($_SERVER["REQUEST_METHOD"] == "POST") {  
    $username = $_POST['username'];  
    $password = $_POST['password'];  
  
    // Database configuration  
    $servername = "192.168.50.128";  
    $username_db = "webuser";  
    $password_db = "iacsd@123";  
    $dbname = "web";  
  
    // Create connection  
    $conn = new mysqli($servername, $username_db, $password_db, $dbname);  
  
    // Check connection  
    if ($conn->connect_error) {  
        die("Connection failed: " . $conn->connect_error);  
    }  
  
    // Query to fetch admin information  
    $sql = "SELECT * FROM admin WHERE username = ?";  
    $stmt = $conn->prepare($sql);  
    $stmt->bind_param("s", $username);  
    $stmt->execute();  
    $result = $stmt->get_result();  
  
    if ($result->num_rows == 1) {  
        $row = $result->fetch_assoc();  
        // Verify password (assuming stored in plaintext)  
        if ($password === $row['password']) {  
            $_SESSION['admin_logged_in'] = true;  
            $conn->close();  
            header("Location: admin_dashboard.php");  
            exit;  
        } else {  
            $error = "Invalid username or password";  
        }  
    }  
}
```

```

    }
} else {
    $error = "Invalid username or password";
}

$conn->close();
}
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Admin Login</title>
</head>
<body>
    <h2>Admin Login</h2>
    <?php if (isset($error)) { echo "<p>$error</p>"; } ?>
    <form
        method="post"
        action="<?php
            echo
htmlspecialchars($_SERVER["PHP_SELF"]); ?>">
        <label for="username">Username:</label><br>
        <input type="text" id="username" name="username"><br>
        <label for="password">Password:</label><br>
        <input type="password" id="password" name="password"><br><br>
        <input type="submit" value="Login">
    </form>
</body>
</html>

```

h.admin_dashboard.php

```

<?php
session_start();

// Check if admin is logged in, redirect if not
if (!isset($_SESSION['admin_logged_in']) || $_SESSION['admin_logged_in'] !== true) {
    header("Location: admin_login.php");
    exit;
}

// Database configuration
$servername = "192.168.50.128";
$username = "webuser";
$password = "iacsd@123";
$dbname = "web";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

```

```

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Example: Update username and password in the users table
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $usernameToUpdate = $_POST['username'];
    $passwordToUpdate = $_POST['password'];
    $idToUpdate = $_POST['id'];

    // Example SQL update query for username and password
    $sql = "UPDATE users SET username = ?, password = ? WHERE id = ?";
    $stmt = $conn->prepare($sql);
    $stmt->bind_param("ssi", $usernameToUpdate, $passwordToUpdate, $idToUpdate);
    $stmt->execute();
    // Handle success or error
}

// Fetch complete users table
$sql = "SELECT * FROM users";
$result = $conn->query($sql);
$users = [];
if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        $users[] = $row;
    }
}
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Admin Dashboard</title>
</head>
<body>
    <h2>Welcome to Admin Dashboard</h2>

    <!-- Example form to update username and password -->
    <form
        method="post"
        action="<?php
            echo
htmlspecialchars($_SERVER["PHP_SELF"]); ?>">
        <label for="id">User ID:</label><br>
        <input type="text" id="id" name="id"><br>
        <label for="username">Username:</label><br>
        <input type="text" id="username" name="username"><br>
        <label for="password">Password:</label><br>
        <input type="password" id="password" name="password"><br><br>
        <input type="submit" value="Update">
    </form>

```

```
<!-- Display complete users table -->
<h3>Users Table</h3>
<table border="1">
  <tr>
    <th>ID</th>
    <th>Username</th>
    <th>Password</th>
  </tr>
  <?php foreach ($users as $user) : ?>
    <tr>
      <td><?php echo $user['id']; ?></td>
      <td><?php echo $user['username']; ?></td>
      <td><?php echo $user['password']; ?></td>
    </tr>
  <?php endforeach; ?>
</table>

<br>
<a href="logout.php">Logout</a> <!-- Add a logout link -->
</body>
</html>
```

Now let's setup dns for our website

1. Install DNS server

```
sudo apt-get update
sudo apt install bind9 bind9utils bind9-doc
```

2. Setting bind to IPv4 Mode

```
Sudo nano /etc/default/named
```

```
sudo systemctl restart bind9
```

3. Configuring the options file

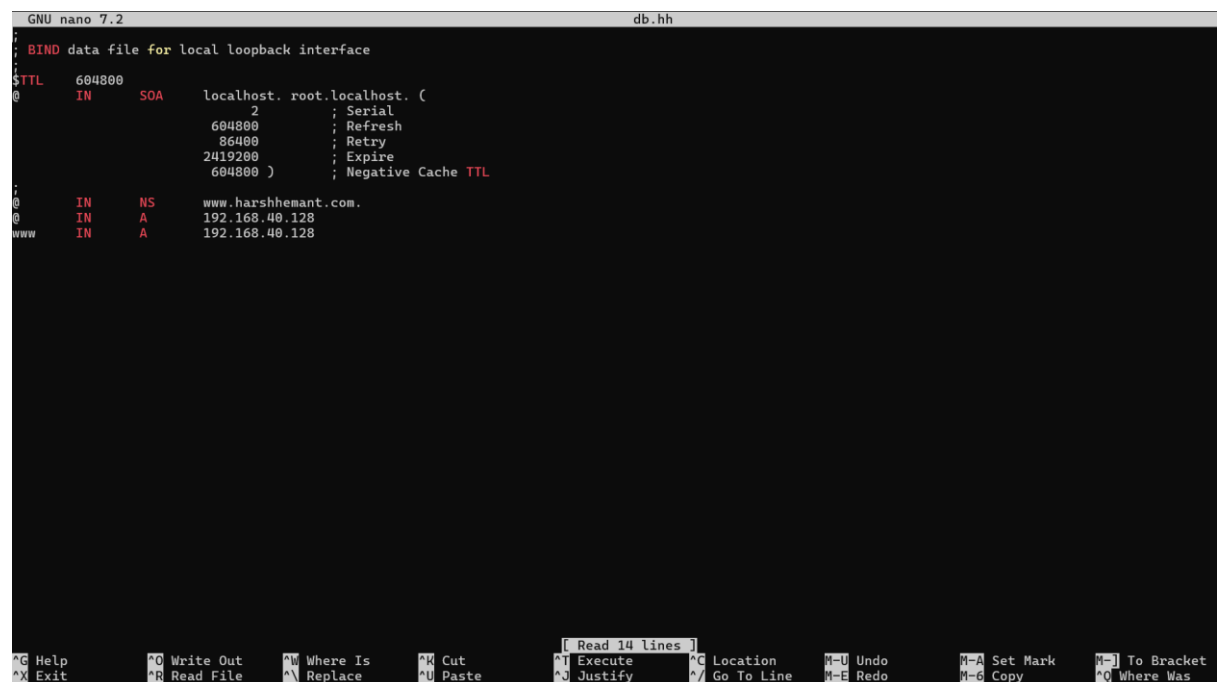
```
sudo nano /etc/bind/named.conf.options
```

4. Configuring the Local file

```
sudo nano /etc/bind/named.conf.local
```

5. Creating the Forward and Reverse Zone File

```
sudo mkdir /etc/bind/zones
sudo cp /etc/bind/db.local /etc/bind/zones/db.hh
sudo nano /etc/bind/zones/db.hh
```



```
GNU nano 7.2 db.hh
BIND data file for local loopback interface
;
$TTL 604800
@ IN SOA localhost. root.localhost. (
; Serial
604800 ; Refresh
86400 ; Retry
2419200 ; Expire
604800 ) ; Negative Cache TTL
;
@ IN NS www.harshhemant.com.
@ IN A 192.168.40.128
www IN A 192.168.40.128
```

```
sudo cp /etc/bind/db.127 /etc/bind/zones/db.128
sudo nano /etc/bind/zones/db.128
```




```
GNU nano 7.2 db.128
; BIND reverse data file for local loopback interface
;
$TTL 604800
@ IN SOA localhost. root.localhost. (
    2 ; Serial
    604800 ; Refresh
    86400 ; Retry
    2419200 ; Expire
    604800 ) ; Negative Cache TTL
;
@ IN NS www.harshhemant.com.
128 IN PTR harshhemant.com.
```

1. Checking the BIND Configuration Syntax

```
sudo named-checkconf
```

```
sudo named-checkzone www.harshhemant.com /etc/bind/zones/db.harshhemant
```

```
sudo named-checkzone 40.168.192.in-addr.arpa /etc/bind/zones/db.128
```

5.2 PKI SETUP

Now you have setup a typical login website using php but for it to work it properly you have to setup database which we will mentioned later in this report.

Now let's prepare certification/full pki setup for our web server

```
sudo apt install openssl
```

To do this we can use a software called xca:

- Download and Install XCA from SourceForge in your host machine
- Open XCA and Create a Database
- Create private keys for RootCA, SubCA, Web Server in the database
- Now create RootCA self-signed certificate using CA template extension and RootCA key that we created and fill the necessary details in Subject
- Now similarly create SubCA certificate signed by RootCA using CA extensions, and SubCA key and fill necessary subject fields
- Now create Web Server Certificate signed by SubCA using TLS Server extension and Web Server Key and necessary Subject fields (remember to mention you Website Name in Common Name field)
- Now export RootCA cert with PEM extension and WEB Server cert using PEM Chain extension
- Also export Web Server Cert. Key with PEM extension
- Open certmgr using run/search bar in windows and RootCA cert into Trusted Root Certification Authorities
- Now send the private key and certificate of Web Server from host machine to Web Server Machine
- In Web Server Machine move the certificate and key file to /etc/apache2/ssl/ directory

```
Sudo mkdir /etc/apache2/ssl
Sudo chmod 600 /etc/apache2/ssl
sudo mv certificate_file_path /etc/apache2/ssl/
sudo mv PrivateKey_file_path /etc/apache2/ssl/
sudo chmod 400 /etc/apache2/ssl/PrivateKey_file
sudo chmod 644 /etc/apache2/ssl/Certificate_file
```
- Open default-ssl.conf file and change the directory path of certificate and key mentioned in file to your path

```
sudo nano /etc/apache2/sites-available/default-ssl.conf
```
- Now enable ssl mode

```
Sudo a2enmode ssl
Sudo a2ensite default-ssl
```
- Now restart apache2

```
Sudo systemctl restart apache2
```

5.3 IPTABLES

Now to make your Web Server more Secure let's make some iptables firewall rules:

Sudo apt install iptables

```
iptables -P INPUT DROP
```

```
iptables -A INPUT -p tcp -dport 443 -j accept
```

```
iptables -A INPUT -p tcp -sport 3306 -j ACCEPT
```

```
iptables -A INPUT -p tcp -s 192.168.50.128 -j ACCEPT
```

```
iptables -A INPUT -p tcp -s 192.168.40.10 -j ACCEPT
```

```
iptables -A INPUT -m conntrack -ctstate INVALID -j DROP
```

```
iptables -A INPUT -p tcp -syn -m conntrack -ctstate NEW -m recent --update --seconds 60 --hitcount 20 -j DROP
```

```
iptables -A INPUT -p tcp -dport 22 -j ACCEPT
```

5.4 ModSecurity WAF

ModSecurity is a web application firewall (WAF). A web application firewall (WAF) is a specific form of application firewall that filters, monitors, and blocks HTTP traffic to and from a web service. By inspecting HTTP traffic, it can prevent attacks exploiting a web application's known vulnerabilities, such as SQL injection, cross-site scripting (XSS), file inclusion, and improper system configuration. With over 70% of attacks now carried out over the web application level, organizations need all the help they can get in making their systems secure. WAFs are deployed to establish an increased external security layer to detect and/or prevent attacks before they reach web applications. ModSecurity provides protection from a range of attacks against web applications and allows for HTTP traffic monitoring and real-time analysis with little or no changes to existing infrastructure

HTTP Traffic Logging

Web servers are typically well-equipped to log traffic in a form useful for marketing analyses, but fall short logging traffic to web applications. In particular, most are not capable of logging the request bodies. Your adversaries know this, and that is why most attacks are now carried out via POST requests, rendering your systems blind.

ModSecurity makes full HTTP transaction logging possible, allowing complete requests and responses to be logged. Its logging facilities also allow fine-grained decisions to be made about exactly what is logged and when, ensuring only the relevant data is recorded. As some of the request and/or response may contain sensitive data in certain fields, ModSecurity can be configured to mask these fields before they are written to the audit log.

Real-Time Monitoring and Attack Detection

In addition to providing logging facilities, ModSecurity can monitor the HTTP traffic in real time in order to detect attacks. In this case, ModSecurity operates as a web intrusion detection tool, allowing you to react to suspicious events that take place at your web systems

Flexible Rule Engine

A flexible rule engine sits in the heart of ModSecurity. It implements the ModSecurity Rule Language, which is a specialized programming language designed to work with HTTP transaction data. The ModSecurity Rule Language is designed to be easy to use, yet flexible: common operations are simple while complex operations are possible. Certified ModSecurity Rules, included with ModSecurity, contain a comprehensive set of rules that implement general-purpose hardening,

Network-based Deployment

ModSecurity works equally well when deployed as part of a reverse proxy server, and many of our customers choose to do so. In this scenario, one installation of ModSecurity can protect any number of back-end web servers.

ModSecurity Rules -----

Not every web server needs the same rules for HTTP requests. You might want to block specific traffic on one application but not another. You might want to control responses sent to

external users to add a layer of security to data. Rules can also be set up for monitoring so that administrators will be aware of ongoing attacks targeting their applications. The set of rules really depends on applications running on the server. Rules for ModSecurity can be downloaded and installed to make configuration of web server security easier, but administrators can also create their own rules. These rules will define the way a web server responds to requests. Configuration of ModSecurity rules is critical for protection of web application data, and without them or with a misconfiguration, the web application could be exploited using numerous known vulnerabilities.

Since ModSecurity is a WAF, the rules cover most of the OWASP Top 10. The OWASP Top 10 is a list of common vulnerabilities used by penetration test applications, and they also set a foundation for administrators so that they can set up WAFs such as ModSecurity to block common web-based attacks.

In the examples listed in this article, Apache .conf files will be added to the /etc/httpd/conf/modsecurity.d/rules/ directory, but you aren't limited to Apache. ModSecurity also works with NGINX and other web servers

CRS -RULE

The OWASP ModSecurity Core Rule Set (CRS) is a set of generic attack detection rules for use with ModSecurity or compatible web application firewalls. The CRS aims to protect web applications from a wide range of attacks, including the OWASP Top Ten, with a minimum of false alerts. The CRS provides protection against many common attack categories, including SQL Injection, Cross Site Scripting, Local File Inclusion, etc.

Installation: -

- ModSecurity can be installed by running the following command in your terminal:

```
sudo apt install libapache2-mod-security2 -y
```

- Alternatively, you can also build ModSecurity manually by cloning the official ModSecurity Github repository. After installing ModSecurity, enable the Apache 2 headers module by running the following command:

```
sudo a2enmod headers
```

- After installing ModSecurity and enabling the header module, you need to restart the apache2 service, this can be done by running the following command:

```
sudo systemctl restart apache2
```

- You should now have ModSecurity installed. The next steps involve enabling and configuring ModSecurity and the OWASP-CRS.

Setting Up the OWASP ModSecurity Core Rule Set: -

- First, delete the current rule set that comes prepackaged with ModSecurity by running the following command:

```
sudo rm -rf /usr/share/modsecurity-crs
```

- Clone the OWASP-CRS GitHub repository into the /usr/share/modsecurity-crs directory:

```
sudo git clone https://github.com/coreruleset/coreruleset  
/usr/share/modsecurity-crs
```

- Rename the crs-setup.conf.example to crs-setup.conf:

```
sudo mv /usr/share/modsecurity-crs/crs-setup.conf.example  
/usr/share/modsecurity-crs/crs-setup.conf
```

- Rename the default request exclusion rule file:

```
sudo mv /usr/share/modsecurity-crs/rules/REQUEST-900-EXCLUSION-  
RULES-BEFORE-CRS.conf.example /usr/share/modsecurity-  
crs/rules/REQUEST-900-EXCLUSION-RULES-BEFORE-CRS.conf
```

- Remove the .recommended extension from the ModSecurity configuration file name with the following command:

```
sudo cp /etc/modsecurity/modsecurity.conf-recommended  
/etc/modsecurity/modsecurity.conf
```

- With a text editor such as vim, open /etc/modsecurity/modsecurity.conf and change the value for SecRuleEngine to On:
- Restart Apache to apply the changes:

```
sudo systemctl restart apache2
```

Enabling ModSecurity in Apache 2: -

- Using a text editor such as vim, edit the `/etc/apache2/mods-available/security2.conf` file to include the OWASP-CRS files you have downloaded:

```
<IfModule security2_module>
    SecDataDir /var/cache/modsecurity
    IncludeOptional /etc/crs4/crs-setup.conf
    IncludeOptional /etc/crs4/plugins/*-config.conf
    IncludeOptional /etc/crs4/plugins/*-before.conf
    IncludeOptional /etc/crs4/rules/*.conf
    IncludeOptional /etc/crs4/plugins/*-after.conf
</IfModule>
```

- In `/etc/apache2/sites-enabled/000-default.conf` file VirtualHost block, include the SecRuleEngine directive set to On.

```
<VirtualHost *:80>
    ServerAdminwebmaster@localhost
```

```
    DocumentRoot /var/www/html
```

```
    ErrorLog ${APACHE_LOG_DIR}/error.log
```

```
    CustomLog ${APACHE_LOG_DIR}/access.log combined
```

```
    SecRuleEngine On
```

```
</VirtualHost>
```

- In `/etc/apache2/apache2.conf` file add VirtualHost block and include the SecRuleEngine directive set to On and below lines at the end of file.

```
    IncludeOptional /etc/crs4/crs-setup.conf
    IncludeOptional /etc/crs4/plugins/*-config.conf
    IncludeOptional /etc/crs4/plugins/*-before.conf
    IncludeOptional /etc/crs4/rules/*.conf
    IncludeOptional /etc/crs4/plugins/*-after.conf
```

- Restart the apache2 service to apply the configuration

```
sudo systemctl restart apache2
```

- ModSecurity should now be configured and running to protect your web server from attacks.

5.5 Nagios for Web Server Monitoring

Nagios Installation :

Step 1: Install Required Dependencies

```
sudo apt update && sudo apt upgrade
sudo apt install -y autoconf gcc libc6 make wget unzip apache2 php libapache2-
mod-php7.4 libgd-dev
```

Step 2: Create Nagios User and Group

Create a dedicated user and group for Nagios to run securely:-

```
sudo useradd nagios
sudo groupadd nagcmd
sudo usermod -a -G nagcmd nagios
```

Step 3: Download and Install Nagios

```
cd /tmp
wget https://assets.nagios.com/downloads/nagioscore/releases/nagios-4.4.6.tar.gz
tar xzf nagios-4.4.6.tar.gz
cd nagios-4.4.6
```

Compile and install Nagios:-

```
sudo ./configure --with-nagios-group=nagios --with-command-group=nagcmd
sudo make all
sudo make install
sudo make install-init
sudo make install-config
sudo make install-commandmode
```

Step 4: Install Nagios Web Interface

```
sudo make install-webconf
sudo htpasswd -c /usr/local/nagios/etc/htpasswd.users nagiosadmin
```

Enable the Apache rewrite and CGI modules:-

```
sudo a2enmod rewrite cgi
sudo systemctl restart apache2
```

Step 5: Install and Configure Nagios Plugins

Download and install the Nagios plugins:-

```
cd /tmp
wget https://nagios-plugins.org/download/nagios-plugins-2.3.3.tar.gz
tar xzf nagios-plugins-2.3.3.tar.gz
cd nagios-plugins-2.3.3
```

Compile and install the plugins:-

```
sudo ./configure --with-nagios-user=nagios --with-nagios-group=nagios --with-openssl
sudo make
sudo make install
```

Step 6: Configure Nagios

Edit the main Nagios configuration file to define your monitoring setup:-

```
sudo nano /usr/local/nagios/etc/nagios.cfg
```

Add your hosts, services, and commands in their respective configuration files:-

```
sudo nano /usr/local/nagios/etc/objects/commands.cfg
sudo nano /usr/local/nagios/etc/objects/hosts.cfg
sudo nano /usr/local/nagios/etc/objects/services.cfg
```

Step 7: Verify and Start Nagios

Check the Nagios configuration for any errors:

```
sudo /usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg
```

If there are no errors, start Nagios and enable it to run on boot:

```
sudo systemctl start nagios
sudo systemctl enable nagios
```

Step 8: Access Nagios Web Interface

Access the Nagios web interface by navigating to
http://your_domain_or_IP/nagios and

login with the user 'nagiosadmin' and the password you created earlier.

Now this it for Web Server Machine , let's configure Database for our WebSite

6. PFSense AS FIREWALL AND DEFAULT ROUTER

1. Download pfSense

- Visit the official pfSense website (<https://www.pfsense.org/download/>) and download the appropriate installation image.

2. Begin the installation

- Once the pfSense installer loads, chose the option to install pfSense.
- Follow the on-screen prompts to select the installation options.

3. Configure pfSense

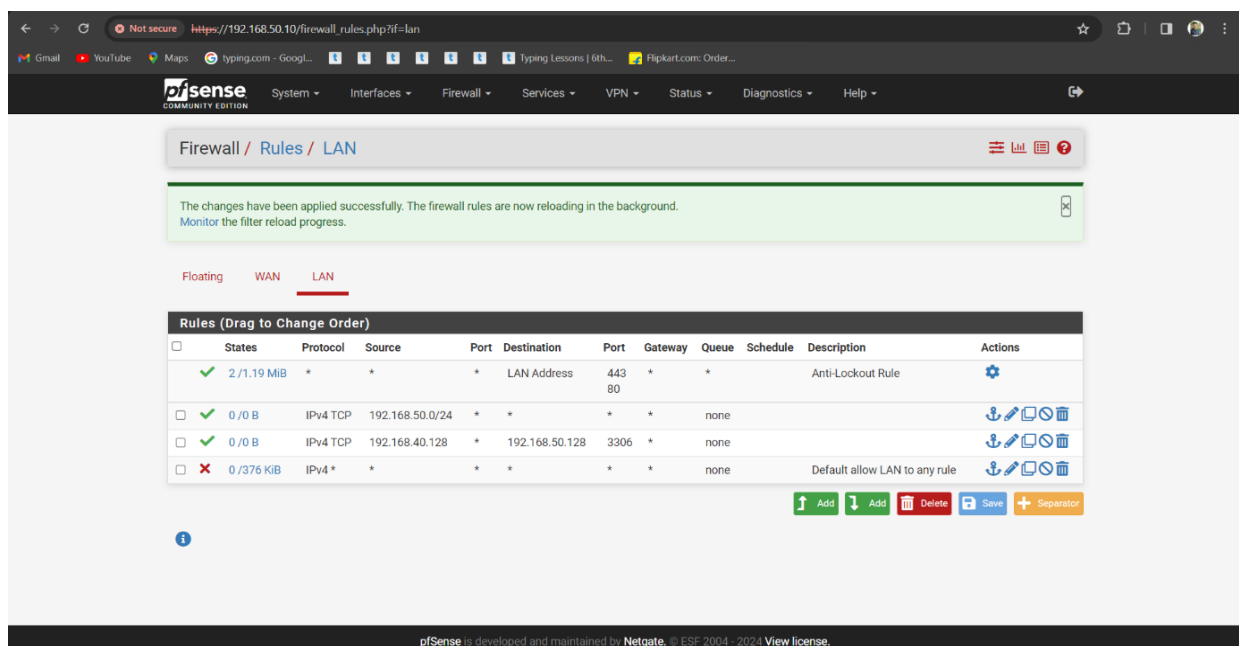
- After the installation is complete, the virtual machine will reboot. Remove the installation media (ISO) to boot from the hard disk.
- Follow the initial configuration wizard to set up network interfaces, Ip addresses, and other basic settings.

4. Access pfSense Web Interface

- Once the configuration is complete, access the pfSense web interface by entering the assigned Ip address in a web browser. The default credentials are usually “admin” for the username and “pfsense” for the password.

5. Further Configuration

- Perform additional configurations through the web interface as needed, such as setting up firewall rules



7. DATABASE MACHINE

PreRequisite-Installed Debian12 with access to non-root user with administrative Privileges

Step1: Install mariadb server

- Sudo apt install mariadb-server

- Sudo mysql -u root -p

- Create database web;

- Use web;

- Create table users (id int(5) auto_increment primary key, username varchar(255) unique, password varchar(255));

- create table admin (username varchar(255) UNIQUE, password varchar(255));

- Insert into admin values ('admin_username', 'admin_password');

- create user 'webuser'@'WebserverMachine_ip' acknowledged by 'password';

- Grant all privileges on web.* to 'webuser'@'WebserverMachine_ip'

- Flush privileges;

- Restart mariadb-server

 - sudo systemctl restart mariadb

8. INSTALLING AND CONFIGURING SNORT FOR NETWORK MONITORING ON DEBIAN

Snort is a free open source network intrusion detection system (IDS) and intrusion prevention system (IPS). Snort uses a simple, lightweight rules description language that is flexible and quite powerful.

Snort can be configured in three main modes: 1. sniffer, 2. packet logger, and 3. network intrusion detection.

Sniffer Mode

The program will read network packets and display them on the console.

Packet Logger Mode

In packet logger mode, the program will log packets to the disk.

Network Intrusion Detection System Mode

In intrusion detection mode, the program will monitor network traffic and analyze it against a rule set defined by the user. The program will then perform a specific action based on what has been identified.

There are a number of simple guidelines to remember when developing Snort rules that will help safeguard your sanity.

Most Snort rules are written in a single line.

This was required in versions prior to 1.8. In current versions of Snort, rules may span multiple lines by adding a backslash \ to the end of the line.

Snort rules are divided into two logical sections, the rule header and the rule options.

The rule header contains the information that defines the who, where, and what of a packet, as well as what to do in the event that a packet with all the attributes indicated in the rule should show up. The first item in a rule is the rule action. The rule action tells Snort what to do when it finds a packet that matches the rule criteria. There are 3 available default actions in Snort, alert, log, pass. In addition, if you are running Snort in inline mode, you have additional options which include drop, reject, and sdop.

1. alert - generate an alert using the selected alert method, and then log the packet
2. log - log the packet
3. pass - ignore the packet
4. drop - block and log the packet
5. reject - block the packet, log it, and then send a TCP reset if the protocol is TCP or an ICMP port unreachable message if the protocol is UDP.
6. sdop - block the packet but do not log it.

The Rule options form the heart of Snort's intrusion detection engine, combining ease of use with power and flexibility. All Snort rule options are separated from each other using the semicolon (;) character. Rule option keywords are separated from their arguments with a colon (:) character.

There are four major categories of rule options.

general - These options provide information about the rule but do not have any affect during detection

payload - These options all look for data inside the packet payload and can be inter-related

non-payload - These options look for non-payload data

post-detection - These options are rule specific triggers that happen after a rule has “fired.”

Uses of Snort rules

Snort uses the popular libpcap library (for UNIX/Linux) or winpcap (for Windows), the same library that tcpdump uses to perform packet sniffing. Snort’s Packet Logger feature is used for debugging network traffic. Snort generates alerts according to the rules defined in configuration file. The Snort rule language is very flexible, and creation of new rules is relatively simple. Snort rules help in differentiating between normal internet activities and malicious activities.

Installing Snort:

1. Installing Dependencies

```
sudo apt install -y gcc libpcap-dev zlib1g-dev libluajit-5.1-dev  
libpcap-dev openssl libssl-dev libnghttp2-dev libdumbnet-dev  
bison flex libdnet autoconf libtool
```

2. Installing from the source

```
wget https://www.snort.org/downloads/snort/daq-2.0.7.tar.gz  
tar -xvzf daq-2.0.7.tar.gz  
cd daq-2.0.7  
./configure && make && sudo make install  
  
./configure --enable-sourcefire && make && sudo make install  
tar -xvzf snort-2.9.16.tar.gz  
cd snort-2.9.16  
./configure --enable-sourcefire && make && sudo make install
```

3. create a symbolic link to /usr/sbin/snort

```
sudo ln -s /usr/local/bin/snort /usr/sbin/snort
```

4. create a new unprivileged user and a new user group for the daemon to run under

```
sudo groupadd snort  
sudo useradd snort -r -s /sbin/nologin -c SNORT_IDS -g snort
```

5. Setting up username and folder structure

```
sudo mkdir -p /etc/snort/rules  
sudo mkdir /var/log/snort  
sudo mkdir /usr/local/lib/snort_dynamicrules
```

6. Set the permissions for the new directories accordingly

```
sudo chmod -R 5775 /etc/snort  
sudo chmod -R 5775 /var/log/snort  
sudo chmod -R 5775 /usr/local/lib/snort_dynamicrules  
sudo chown -R snort:snort /etc/snort  
sudo chown -R snort:snort /var/log/snort
```

```
sudo chown -R snort:snort /usr/local/lib/snort_dynamicrules
```

7. Create new files for the white and blacklists as well as the local rules

```
sudo touch /etc/snort/rules/white_list.rules  
sudo touch /etc/snort/rules/black_list.rules  
sudo touch /etc/snort/rules/local.rules
```

8. Copy the configuration files from the download folder

```
sudo cp ~/snort_src/snort-2.9.16/etc/*.conf* /etc/snort  
sudo cp ~/snort_src/snort-2.9.16/etc/*.map /etc/snort
```

9. Edit configuration file of snort

```
sudo nano /etc/snort/snort.conf  
  
# Path to your rules files (this can be a relative path)  
  
var RULE_PATH /etc/snort/rules  
var SO_RULE_PATH /etc/snort/so_rules  
var PREPROC_RULE_PATH /etc/snort/preproc_rules  
  
# Set the absolute path appropriately  
var WHITE_LIST_PATH /etc/snort/rules  
var BLACK_LIST_PATH /etc/snort/rules
```

10. Add the following snort rules in /etc/snort/rules/local.rules

```
alert tcp any any -> any any (msg:"Suspicious OUTbound Traffic":  
content:"/bin/bash"; sid:100001;)  
alert tcp any any -> any 3306 (msg:"Database Accessed"; content:"SELECT";  
sid:100002;)  
alert tcp any any -> any 22 (msg:"Potential ssh brute force attack"; content:"Failed  
password"; sid:100003;)
```

11. Running snort

```
sudo snort -A console -u snort -g snort -c /etc/snort/snort.conf
```

9. FTP SERVER

Set up and configure FTP server on debian:-

1. Install vsftpd

```
sudo apt update && sudo apt upgrade  
sudo apt install vsftpd
```

2. Configure vsftpd

Open the configuration file

```
sudo nano /etc/vsftpd.conf
```

Add the following lines to this file

```
anonymous_enable=NO  
local_enable=YES  
write_enable=YES  
local_umask=022  
dirmessage_enable=YES  
use_localtime=YES  
xferlog_enable=YES  
connect_from_port_20=YES  
chroot_local_user=YES  
listen=YES
```

3. Create Directories for Users

```
sudo mkdir /ftpdata  
sudo mkdir /ftpdata/snort  
sudo mkdir /ftpdata/database
```

4. Create users corresponding to each ftp user and set their password

```
sudo adduser snort  
sudo adduser database
```

5. Set permission

```
sudo chown user1:user1 /ftpdata/user1  
sudo chown user2:user2 /ftpdata/user2  
sudo chmod 744 /ftpdata/user1  
sudo chmod 744 /ftpdata/user2
```

6. Restart vsftpd

```
sudo service vsftpd restart
```

10. FUTURE SCOPE

Future enhancements may include the implementation of advanced threat intelligence integration to bolster intrusion detection capabilities, adoption of containerization for scalable and efficient application deployment within the DMZ, incorporation of multi-factor authentication for enhanced security, and exploration of emerging technologies like Software-Defined Networking (SDN) for dynamic and flexible network management. Additionally, continuous monitoring and updates to security protocols, alongside regular audits for compliance with evolving regulatory standards, will be crucial to maintaining the integrity and effectiveness of the fortified DMZ infrastructure in the face of evolving cyber threats.

11. CONCLUSION

In conclusion, the project successfully fortified the DMZ infrastructure by implementing advanced security measures, expanding functionality to support website hosting, and deploying stringent access controls. By integrating technologies like PKI, PFsense, and Snort, the project enhances security while ensuring compliance and efficient data management. Continuous monitoring and updates will sustain the infrastructure's resilience against evolving threats. The project's success underscores the importance of proactive security measures and the adoption of robust technologies to safeguard critical network environments while meeting the demands of modern connectivity.