ECE 567: Project Report Part 2

Date: April 12th, 2024

By: Payton Murdoch, V00904677

&

Yun Ma, V01018599

Table Of Contents

Phase 1: Intrusion Detection

For this Phase, we decided to utilize the vulnerability shown to us through the Nessus Vulnerability Scanning tool depicted in Figure 1.

Figure 1: Cross-site Scripting Vulnerability on HTTP.

The following vulnerability depicts that given the outdated version of JQuery, the HTTP web server is susceptible to multiple Cross-Site Scripting attacks. We have examined the Mars webpage and determined an area on the page that is susceptible to the attacks, which we will describe in our general attack scenario. These attacks require deeper packet inspection levels, making for slightly more complex inspection and rule parameters. Thus, implementing SNORT rules to detect these general scenario XSS attacks is a nice challenge.

Part 1: Attack Scenario

Cross-site scripting can be categorized under Three possible attack vectors: Persistent, Reflected or DOM-based cross-site scripts. While investigating the susceptibility of the MARS website, we determined that the general attack scenario to explore is the Persistent XSS. In this case, the attacker must find a section embedded within the site that allows information to be posted or saved. Persistent XSS completely depends on the scripting attack originating from the website. In this general attack scenario, the attacker will find the area of the site that can store user input, such as comment sections of forum posts and write a script embedded within it; these scripts can use simple <script></script>, <img…src…onerror> or more. These scripts will be saved in the website database. If another user visits the site, the server loads the comments or forums that are improperly sanitized and will load and run the associated commands/payload. These scripts can contain various commands, implement objects such as keyloggers, steal session cookie data, and send the information to a server the attacker is running. The information can reveal sensitive data such as session data, key inputs for usernames and passwords and more. Figures 2 and 3 below show a brief graphical sketch of this attack's attacker and user sides.

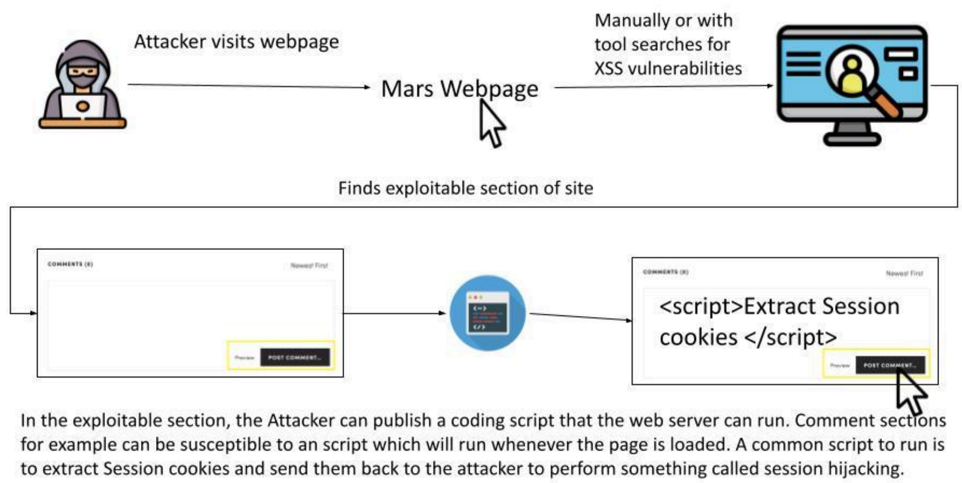


Figure 2: Attacker Side of XSS.

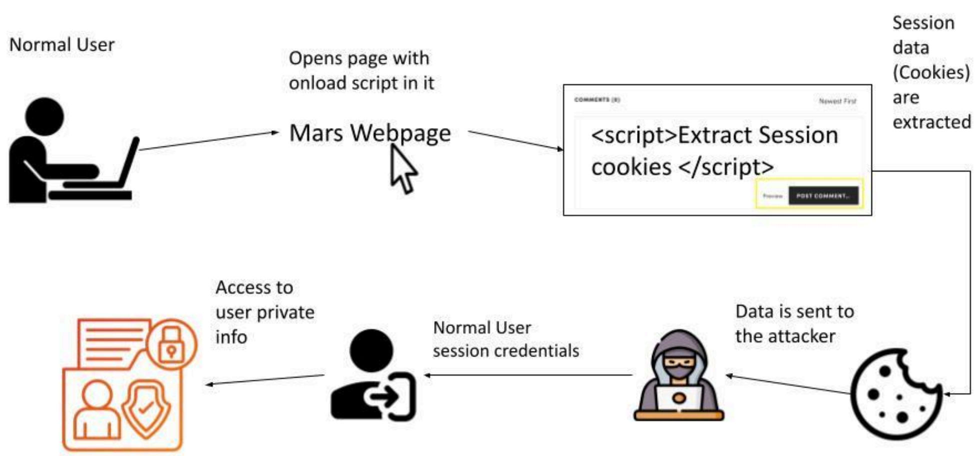


Figure 3: User side of XSS.

Concerning the specific susceptibility of the MARS webserver, we know through inspection and extensive testing that each article page embedded within the site has a comments section susceptible to Persistent XSS attacks embedded within a bolded, underlined or URL-embedded section. One of the pages referenced is shown below in Figure 4. Furthermore, through general trial and error, we decided to focus on XSS attacks with the '<...onerror='script'> flag, which narrows down the requirements for the Snort rules.

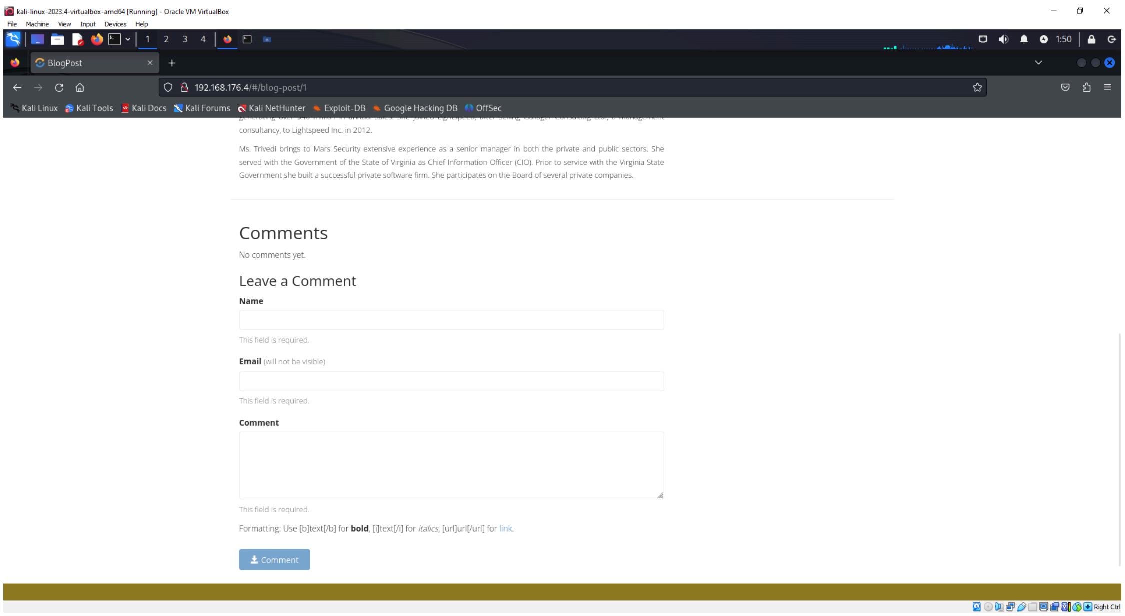


Figure 4: Exploitable XSS field on Mars Website.

Part 2: Snort Rules

Before setting up the Snort rules to attempt to detect XSS vulnerabilities, we must determine and configure the interface so that the Detection system on MarsR2.0 can trigger on the exploitation attempt. As we know, the webserver runs on the MarsN2.0 machine, which, in our case, is listed at IP 192.168.176.4. As such, we need to make sure that the snort.conf file on MarsR2.0 has additional logic implemented to surveil the neighbouring IPs within its network. To ensure this, we will open the /etc/snort/snort.conf file using VI and change the initial ipvar HOME_NET to the appropriate network, and we change the HTTP_SERVERS variable to match MarsN2.0, as shown in Figure 5.

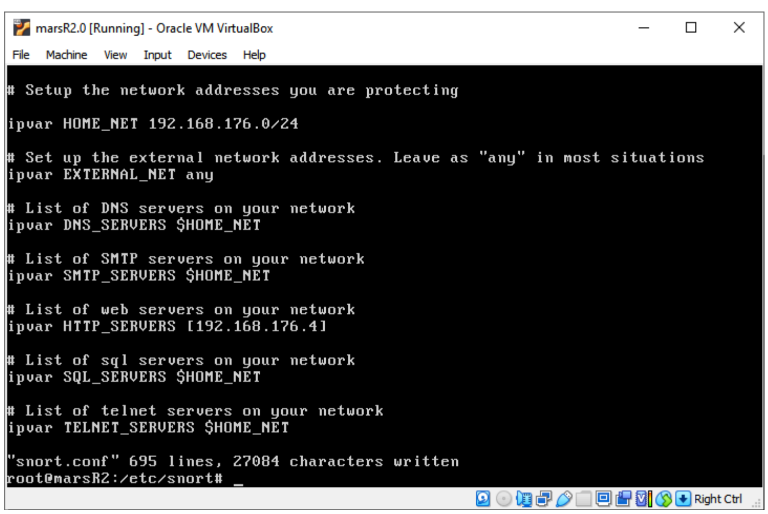


Figure 5: snort.conf on MarsR2.0 with proper HOME_NET variable.

It is important to note the various rules we attempted to implement before arriving at the final rule set, which detects the XSS attack. The first hurdle we encountered while trying to create snort rules was that Snort starts examining packets at the network level. That said, one needed to do more than add HTTP in the protocol field and immediately receive results. We had to begin with the TCP level, develop a snort rule that parses the HTTP content within it, and take it one step further; we wanted to specifically detect the HTTP post command as this signifies that something had been written to the server. Therefore, after looking through snort manuals [1], our initial rule, shown in Figure 6, is simply to log HTTP traffic with post content within it going into the server.

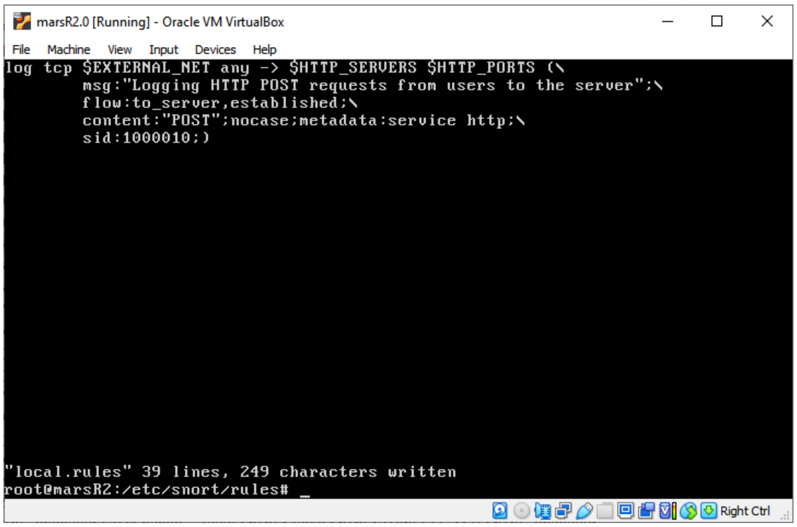


Figure 6: Log snort rule for all HTTP inbound traffic with ‘post’ content.

Unfortunately, this did not yield the correct results, as the only traffic flagged were HTTP GET commands, as the content ‘post’ is embedded in the website header in the logs. Furthermore, upon inspection with general snort rules to log all inbound traffic at the network level, we found no HTTP POST commands. Thus, we needed to switch methods and instead log bidirectional general HTTP traffic while posting a comment through the attacker to see how the snort logs the network information. By deleting all the additional flow and content conditions and switching the arrow to ‘<>’ when we posted the comment on the site with snort running for the following log in Figure 7.

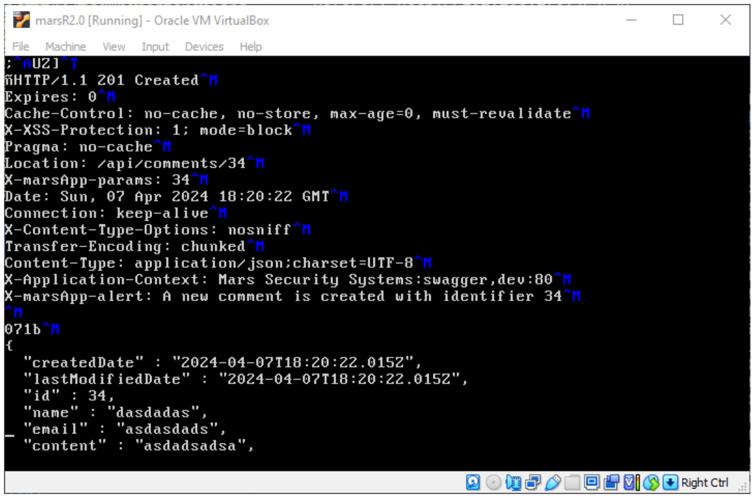


Figure 7: Snort log of general comment post on Mars Site.

This gives us all the information necessary to create our final snort rules for XSS detection. As we can see, stat code 201 is associated with making the new comment. Therefore, this will be the first condition for isolating the traffic. Additionally, this shows us that the traffic originates from the HTTP server and is going to the client utilizing it. Thus, we can switch the $HTTP_SERVERS and $EXTERNAL_NET vars and return the arrow to '->' so that snort does not need to over-examine the bi-directional traffic. Next, we must consider the different types of XSS scripts the system is vulnerable to for the alerts. Through general experimentation on the website through the attacker machine, we determined that the system is susceptible to scripts that feature 'onerror.' We will expand this to a few more generalized scripts with common elements. With all this in mind, we will use pcre and the information above to create snort alerts which flag general expressions utilized in XSS.[2] As shown in Figure 8, you can see the resulting rules. After many attempts, it was logical to simply flag regex, which contains common script elements such as '<script>...</script>', '<...onerror...>', '<...onload…>' and '<...javascript…>' hopefully isolating particular XSS attempts. Any '[^\n]+ signifies one or more repetitions of any character besides a new line, which can generally denote non-significant information to the regex.

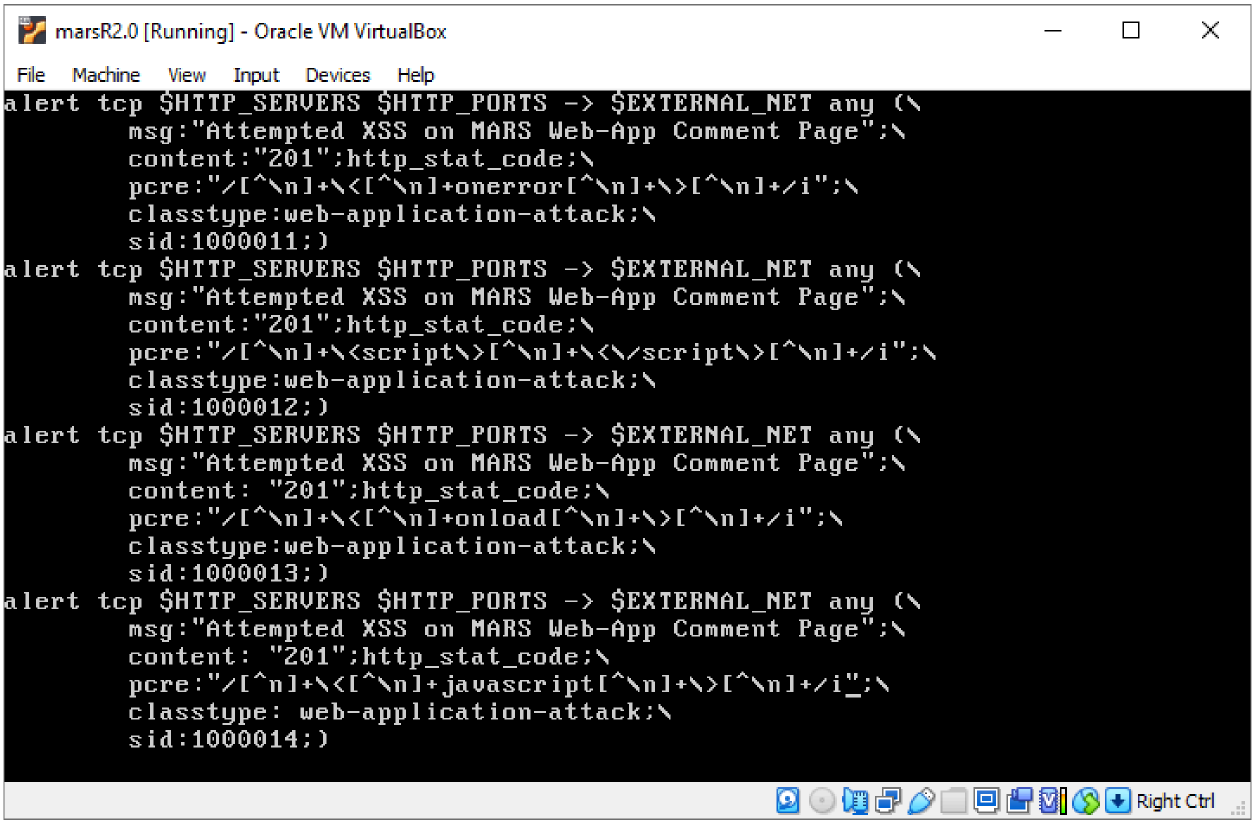
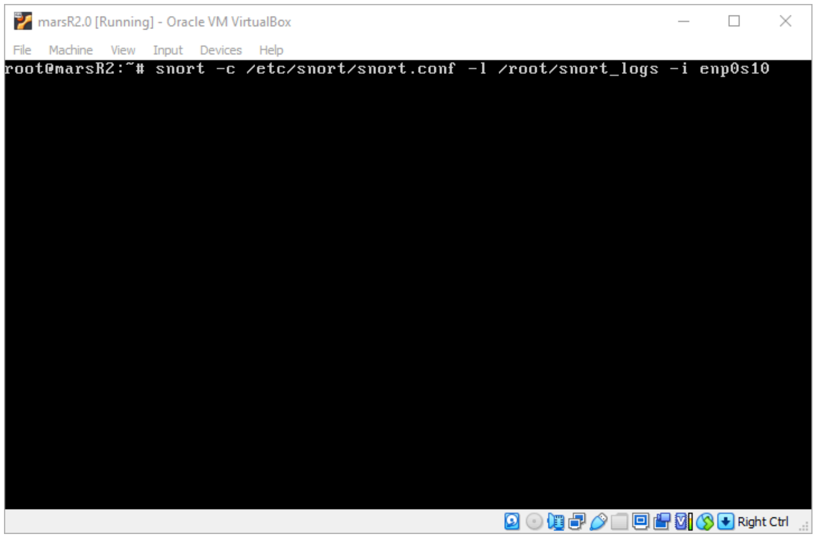


Figure 8: Snort Rules for XSS Detection.

This only covers some possible scripts within the realm of XSS, as there are thousands. Unfortunately, we do not have the time to compute thousands of rules. Furthermore, as stated in the project parameters, we do not want to overfit our rules. Thus, these rules will detect the generic attack vector we proposed, and since we tested and know that 'onerror' XSS attacks work on the MARS site, we want to test against those specifically.

Part 3: Snort Detection

To run snort in Intrusion Detection mode, we run the command shown in the first image in Figure 9. We have initialized the home network HTTP server and rules in the snort.conf file, it will operate with all the general rules in place. Therefore, the second image shows it running with no errors.



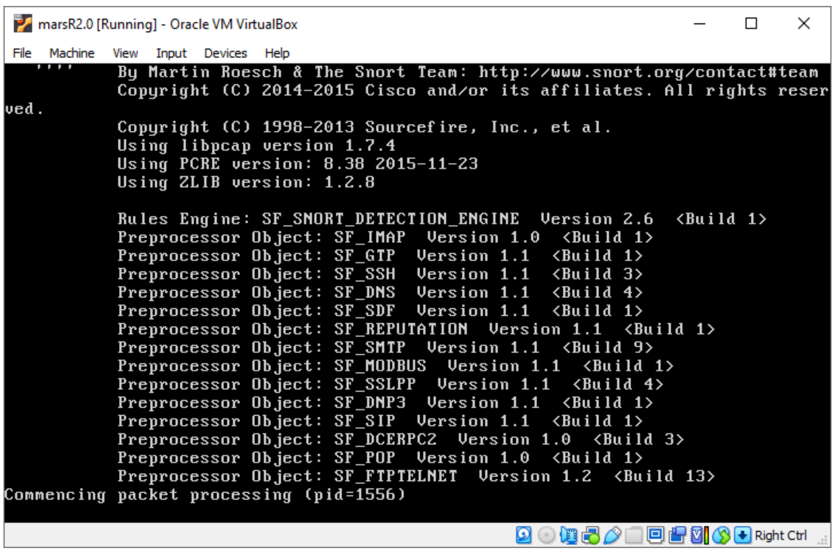


Figure 9: Initializing snort on IDS mode.

Moving on to Figure 10, we go to the Kali machine and find pages on the website with comment boxes. We will visit different media pages and add comments on each page, some of which will be embedded with XSS. Each of the XSS embedded commands will use one of the [b],[i], or [url] commands as a wrapper, as this will cause the server to run it in an attempt to transform it into the designated text type whether it be italics, bolded or added as a link. Thus, we must run snort for those three cases when considering XSS.

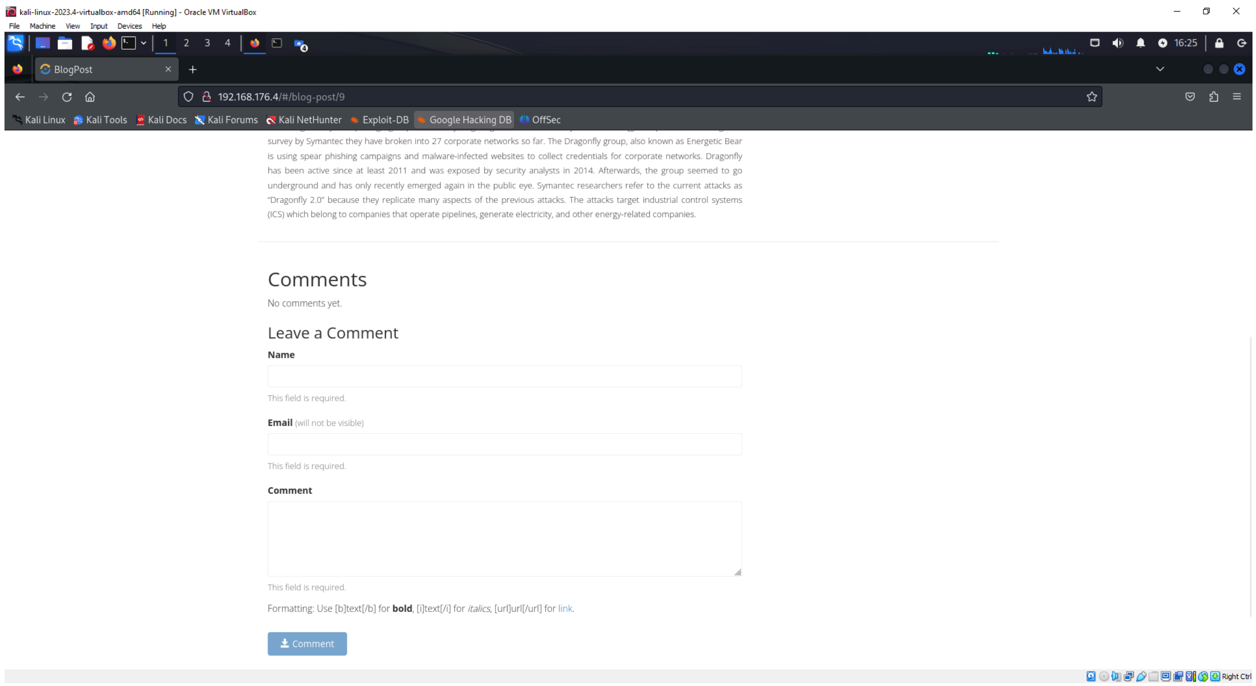


Figure 10: Kali XSS setup.

Figure 11 shows XSS attacks from hackers and non-attacking comments, demonstrating the XSS scripts that occur. It shows the example attacker inputs with the Hacker name and which command they use. Although not represented in the image, the attacking comments yield alerts denoting information, such as session cookies or default alerts.

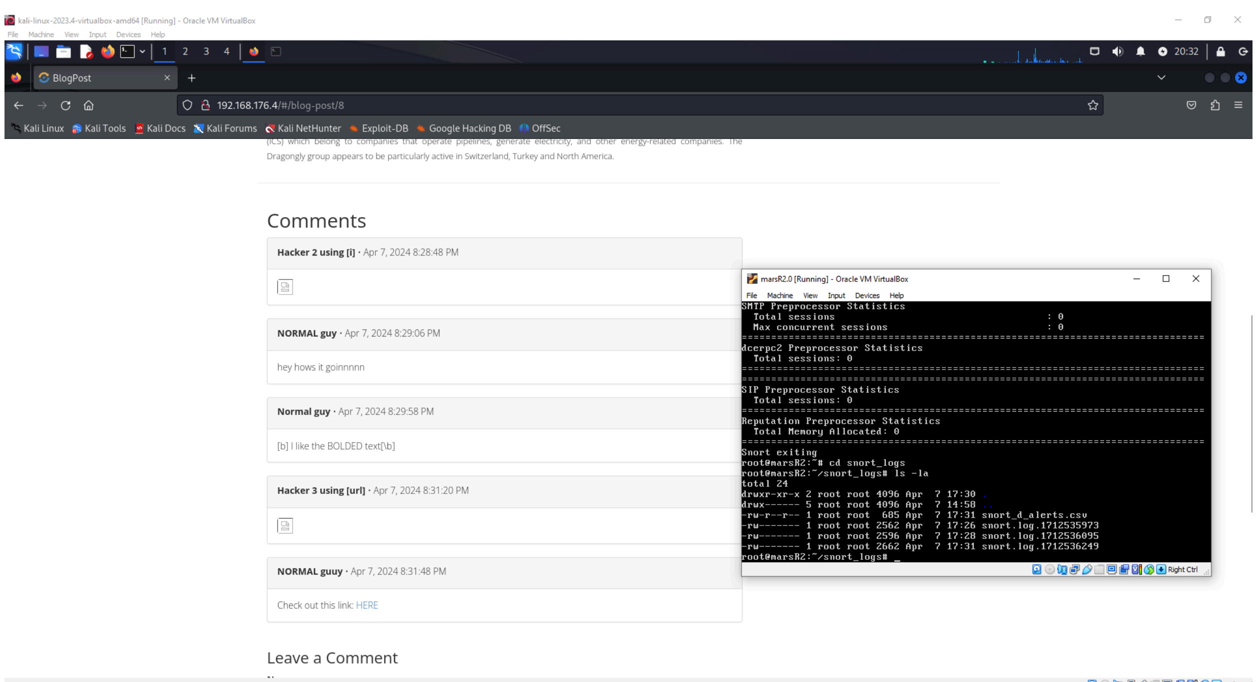


Figure 11: Comments embedded with scripts and normal comments.

After posting each comment, we halted the Snort IDS and looked to verify that the proper log files shown in Figure 11 were generated or to confirm that the normal comments did not generate any logs or alerts.

Part 4: Analysis of Results

The first step is to exfiltrate data to the Kali machine to get a higher-level view of the alerts and logs. We show the FTP process through the Kali machine in Figure 12.

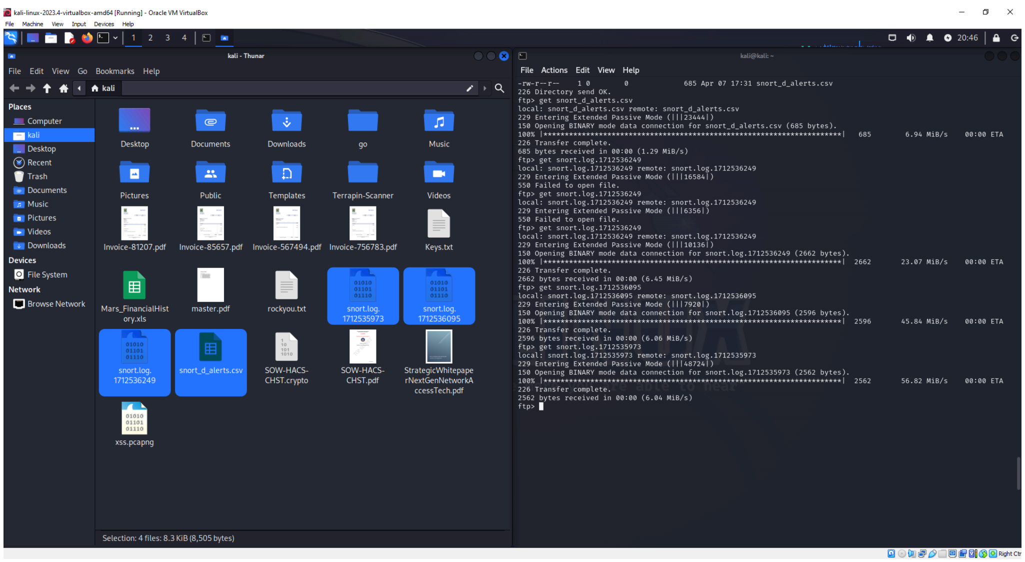


Figure 12: Retrieval of snort.log and alert files.

While this specific example yields the correct results by only flagging the comments with ‘<...onerror…>’ embedded within them, as shown in Figures 13 to 16, it is important to consider the cases where false positives and negatives can occur. As stated in the prior paragraph, these snort rules only detect specific XSS variations. Thus, any script outside the variations included may run if no other protections exist in the server's code. Furthermore, we must consider false positives. This specific variation of an XSS holds with it a very peculiar combination of characters. Thus, it is relatively rare for a normal user to type this combination of characters in a normal sentence, albeit not impossible. Furthermore, a series of relatively false positives could not run properly but could indicate an attempted XSS attack by those without sufficient knowledge. By all means, the snort rules we implemented are susceptible to evasion. However, we hope that our willingness to tackle an attack of such a complex nature shows our motivation to learn all about snort and its advantages. The rules developed suffice for cases dependent on mishandled loading errors in the code, a general script or comments embedded with javascript, but not additional XSS cases as it is a massive field as denoted by the list in [3].

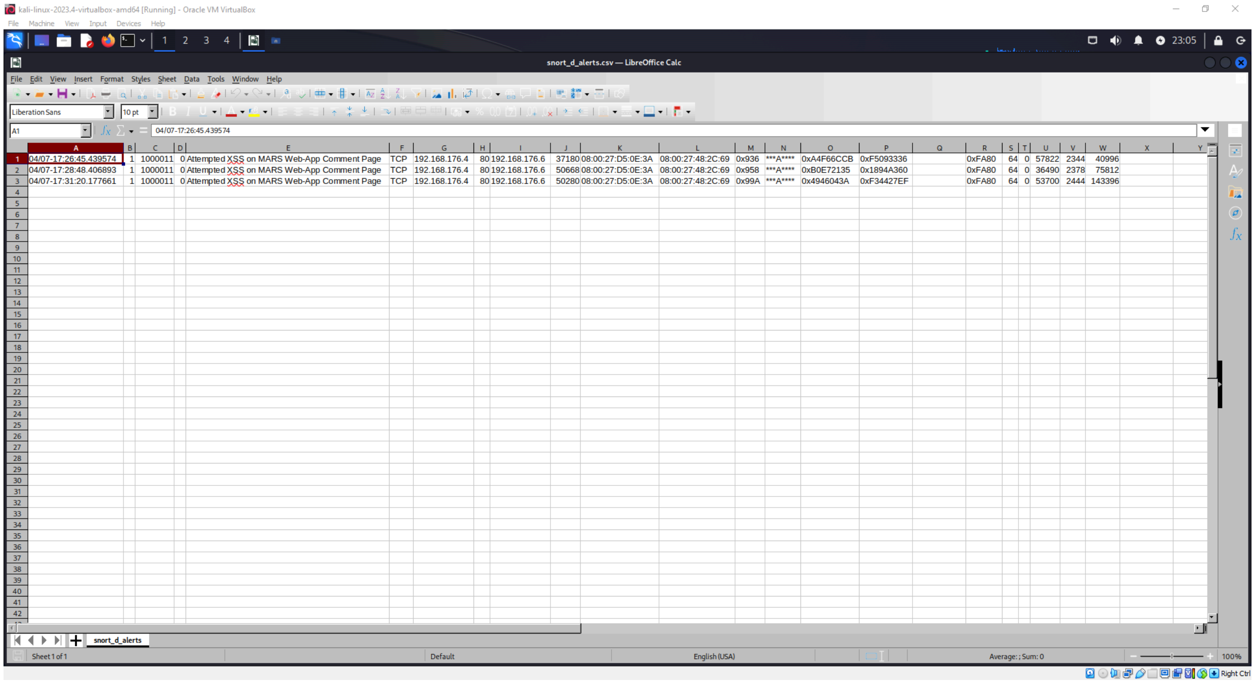


Figure 13: Three Alerts generated by SNORT.

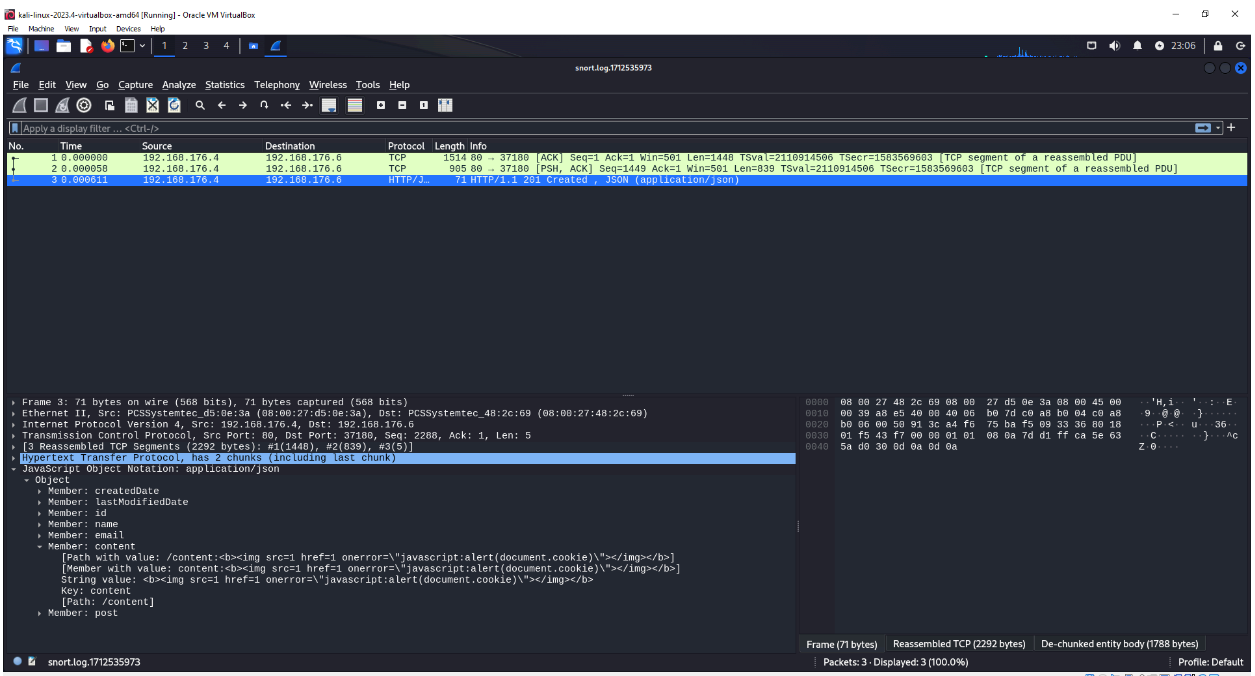


Figure 14: [b][/b] wrapped XSS lists document cookies for session theft.

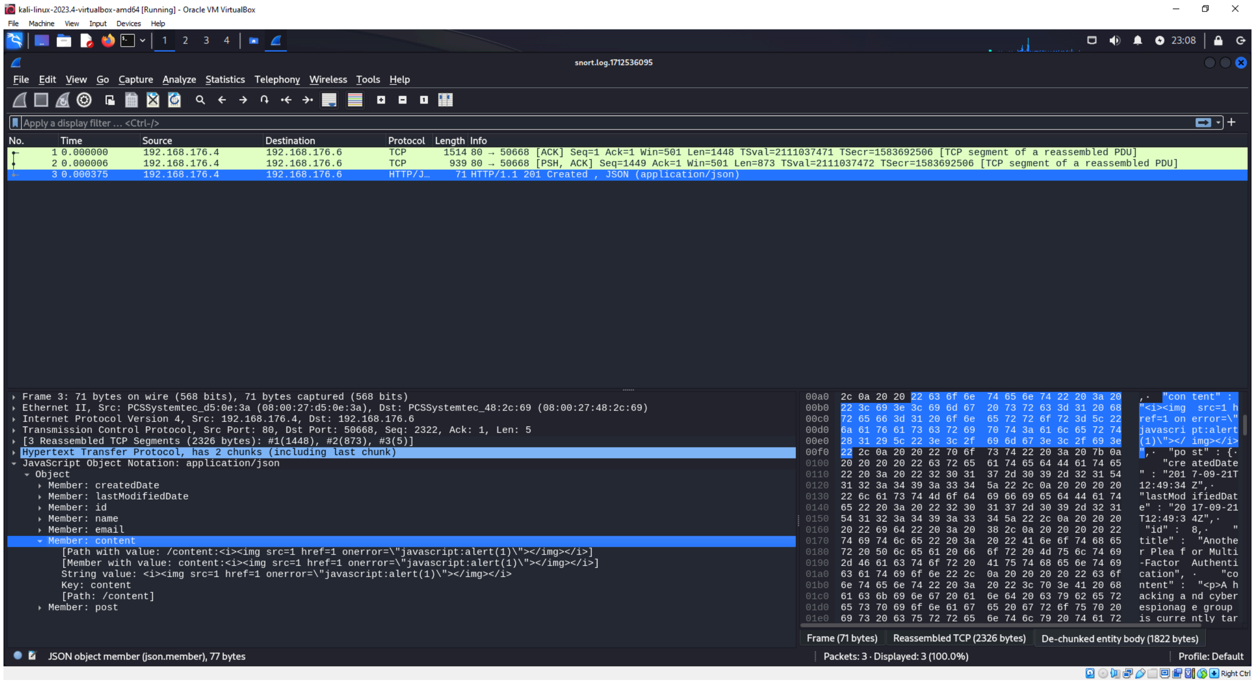


Figure 15: [i][/i] Wrapped XSS with simple alert for testing vulnerabilities.

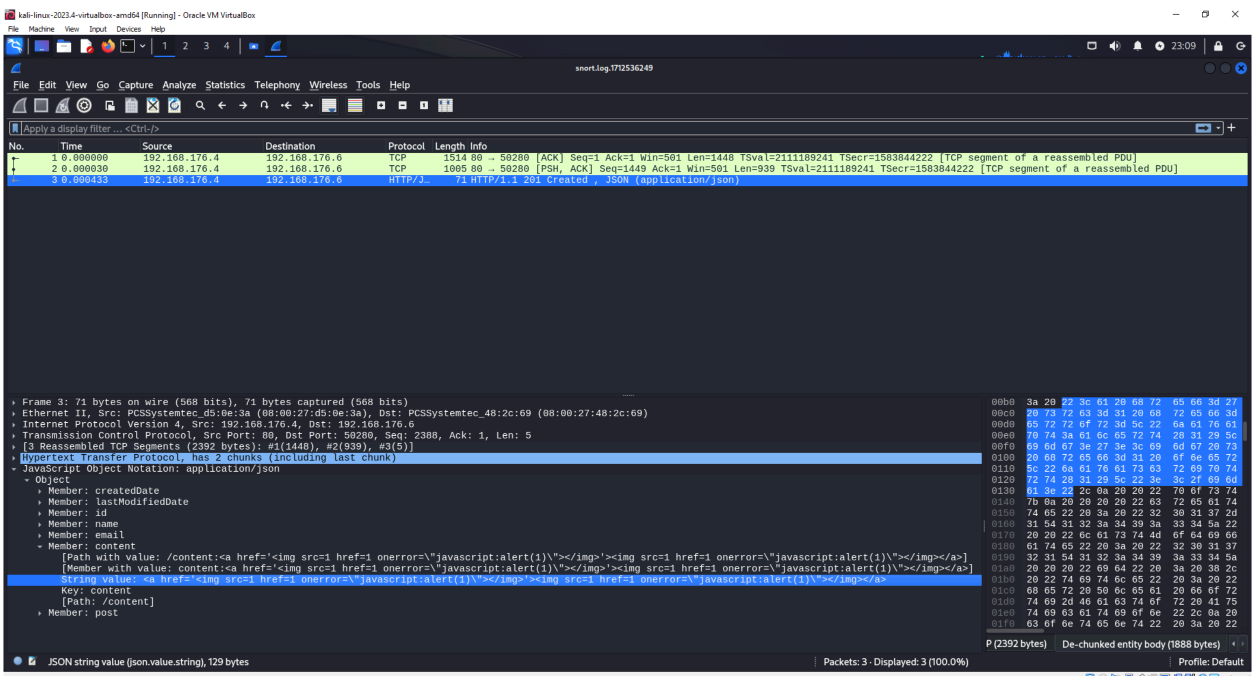


Figure 16: [url][/url] Wrapped XSS with a simple alert for testing vulnerabilities.

Phase 2: Intrusion Prevention

Part 1: Define the IPTables rules and provide the rationale for each.

In this part, we were given MarsN2.0 credentials of root/Jum!@4217 to set the IPTables firewall directly. The basic ideal to mitigate and prevent attacks is to find out what services MarsN2.0 provides and to give each service proper accessibility.

1. We used Nmap to scan how many ports and services MarsN2.0 is providing:

```
File  Actions  Edit  View  Help
┌──(root㉿kali)-[~]
└─# nmap 192.168.56.104
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-04-09 14:06 EDT
Nmap scan report for 192.168.56.104
Host is up (0.0011s latency).
Not shown: 989 closed tcp ports (reset)
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
23/tcp   open  telnet
25/tcp   open  smtp
53/tcp   open  domain
80/tcp   open  http
110/tcp  open  pop3
143/tcp  open  imap
993/tcp  open  imaps
995/tcp  open  pop3s
3306/tcp open  mysql
MAC Address: 08:00:27:DD:BF:25 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 0.52 seconds
```

Figure 17: MarsN2.0 ports scan

We also find there is a GUI interface MarsN2.0 Ubuntu provided:

```
startx
```

Figure 18: Go into the GUI of Ubuntu

2. We back up the Iptables at the beginning to mitigate unnecessary mistakes and ensure every change can be recovered.

```
sudo apt-get install iptables
sudo iptables-save > /root/iptables.backup
```

Figure 19: Install and Backup IPTables

3. To start, the default policy is to drop INPUT and FORWARD and to accept OUTPUT.

```
sudo iptables -P INPUT DROP
sudo iptables -P FORWARD DROP
sudo iptables -P OUTPUT ACCEPT
```

Figure 20: Default Policies

4. After scanning MarsN2.0, no TCP port is open under the default policies.

```
Not shown: 989 closed tcp ports (reset)
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
23/tcp   open  telnet
25/tcp   open  smtp
53/tcp   open  domain
80/tcp   open  http
110/tcp  open  pop3
143/tcp  open  imap
993/tcp  open  imaps
995/tcp  open  pop3s
3306/tcp open  mysql
MAC Address: 08:00:27:DD:BF:25 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 0.52 seconds

┌──(root㉿kali)-[~]
└─# nmap 192.168.56.104
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-04-09 14:51 EDT
Nmap scan report for 192.168.56.104
Host is up (0.00050s latency).
All 1000 scanned ports on 192.168.56.104 are in ignored states.
Not shown: 1000 filtered tcp ports (no-response)
MAC Address: 08:00:27:DD:BF:25 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 21.63 seconds
```

Figure 21: Using Nmap from Kali to test the status of ports on MarsN2.0

5. We'll allow incoming traffic on their respective ports for each public-facing service. Given that the inside IP '192.168.56.104' is for internal use and '10.0.2.15' is the outside IP, we'll focus on rules applicable to the outside interface. We do not allow port 21 for FTP and port 23 for Telnet. Instead, port 22 is allowed for SSH and SFTP.

```
# HTTP and DNS (domain)
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -p tcp --dport 53 -j ACCEPT

# Email services (SMTP, POP3, IMAP, IMAPS, POP3S)
iptables -A INPUT -p tcp --dport 25 -j ACCEPT
```

```
iptables -A INPUT -p tcp --dport 110 -j ACCEPT
iptables -A INPUT -p tcp --dport 143 -j ACCEPT
iptables -A INPUT -p tcp --dport 993 -j ACCEPT
iptables -A INPUT -p tcp --dport 995 -j ACCEPT

# Secure channels (SSH, for remote access and file transfer)
iptables -A INPUT -p tcp --dport 22 -j ACCEPT

# Database access, consider limiting to specific IPs if possible
iptables -A INPUT -p tcp --dport 3306 -j ACCEPT
```

Figure 22: Allow service's Ports

6. In this part, we limited new TCP connections to mitigate DDoS attacks. Limiting new connections to 5 per minute.

```
iptables -A INPUT -p tcp -m connlimit --connlimit-above 5
--connlimit-mask 32 --connlimit-saddr -j DROP
```

Figure 23: Mitigate DDoS

7. Incoming mail traffic must be restricted to the internal mail server. We only allow mail traffic to the internal mail server IP 192.168.56.104.

```
# Allow SMTP (25),POP3(110),IMAP(143),IMAPS(993),POP3S(995) for the
internal mail server
iptables -A INPUT -p tcp --dport 25 -d 192.168.56.104 -j ACCEPT
iptables -A INPUT -p tcp --dport 110 -d 192.168.56.104 -j ACCEPT
iptables -A INPUT -p tcp --dport 143 -d 192.168.56.104 -j ACCEPT
iptables -A INPUT -p tcp --dport 993 -d 192.168.56.104 -j ACCEPT
iptables -A INPUT -p tcp --dport 995 -d 192.168.56.104 -j ACCEPT

# Block these email service ports for other destinations
iptables -A INPUT -p tcp --dport 25 -j DROP
iptables -A INPUT -p tcp --dport 110 -j DROP
iptables -A INPUT -p tcp --dport 143 -j DROP
iptables -A INPUT -p tcp --dport 993 -j DROP
iptables -A INPUT -p tcp --dport 995 -j DROP
```

Figure 24: Allow mail server ports and drop other destinations.

8. Accept incoming traffic that is a response to legitimate requests initiated by the server itself

```
iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
```

Figure 25: Related Incoming Connections

9. Save IPtables:

```
sudo iptables-save > /etc/iptables/rules.v4
```

Part 2: Test the firewall rules and provide screenshots documenting the results.

We use Kali to test the firewall policy we set in the previous part.

10. Using the browser to access the website on MarsN2.0 through port 80. This step is to make sure the website is working well to access.

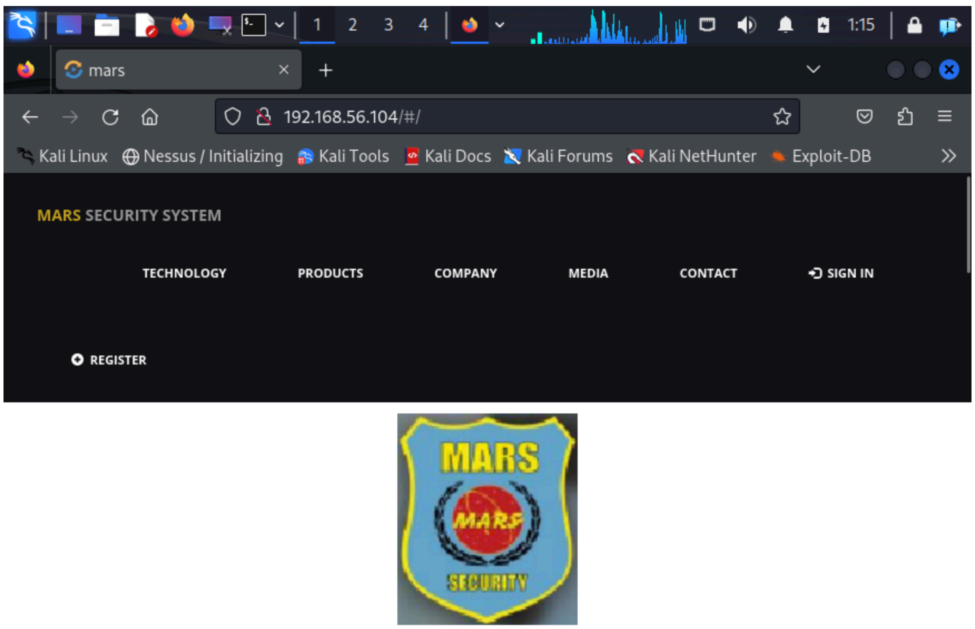


Figure 26: Testing public HTTP service

11. We use Nmap to scan if the ports related to email service are achievable.

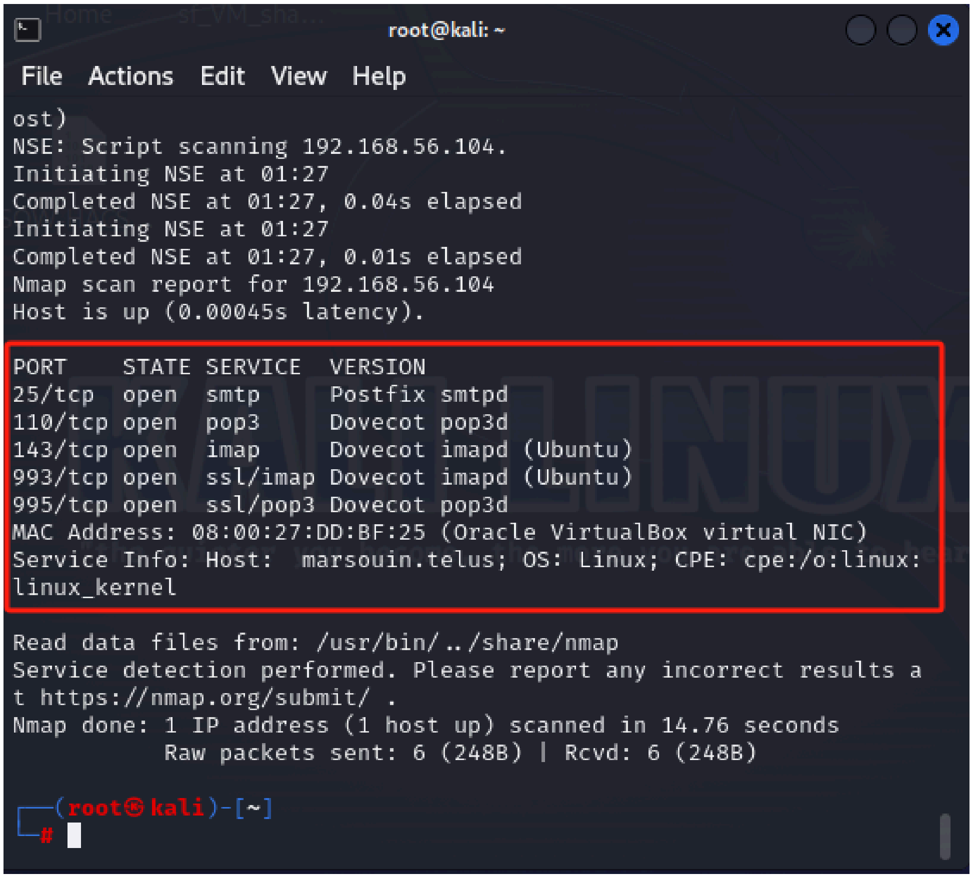


Figure 27: Testing SMTP, POP3, IMAP, IMAPS and POP3s services

12. The server should allow remote access and file transfer. We can use SSH for control and SFTP for file sharing. Logging from the Keli machine, two services work properly.

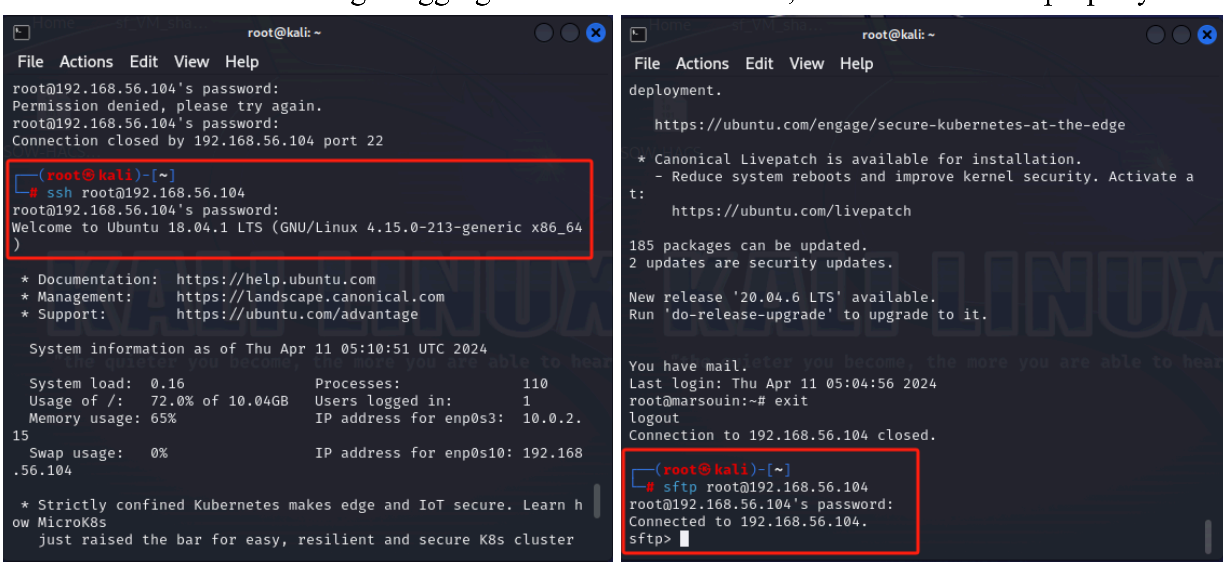


Figure 28: Testing SSH and SFTP

13. We use ‘hping3’ to generate network traffic to test firewall rules for the DDoS Mitigation test: start a TCP flood towards MarsN2.0. Sending a large number of SYN packets, simulating multiple attempts to establish a TCP connection:

```
hping3 -S -p 80 -i u1 --rand-source 192.168.56.104
```

Thanks to the IPTables, 33% of packets are dropped by the server.

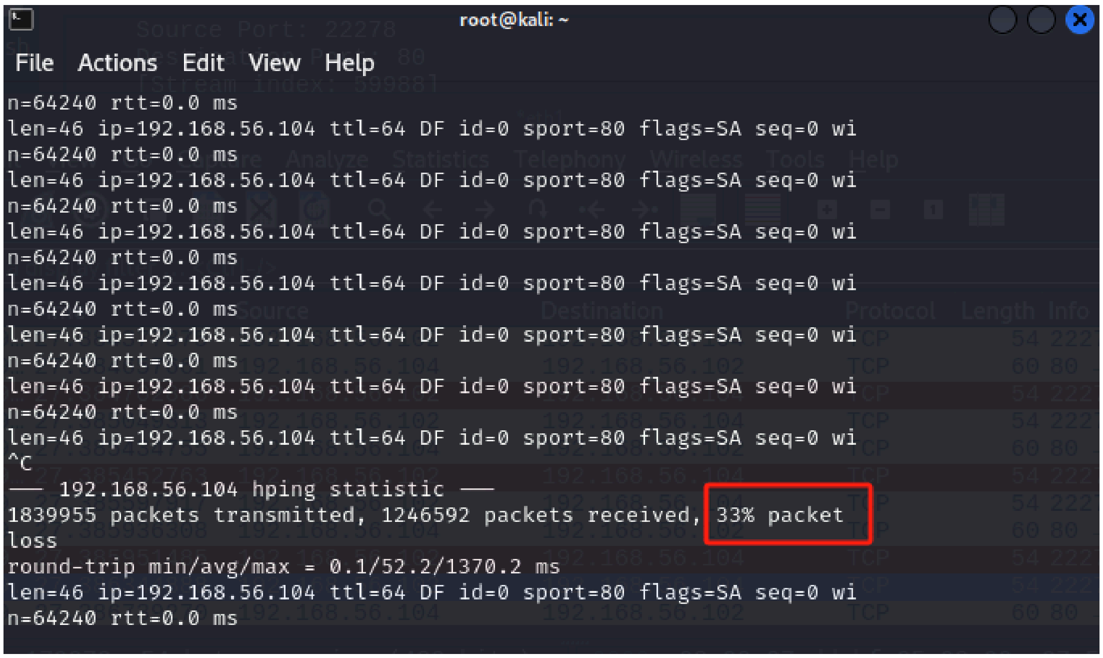


Figure 29: The result of hping3

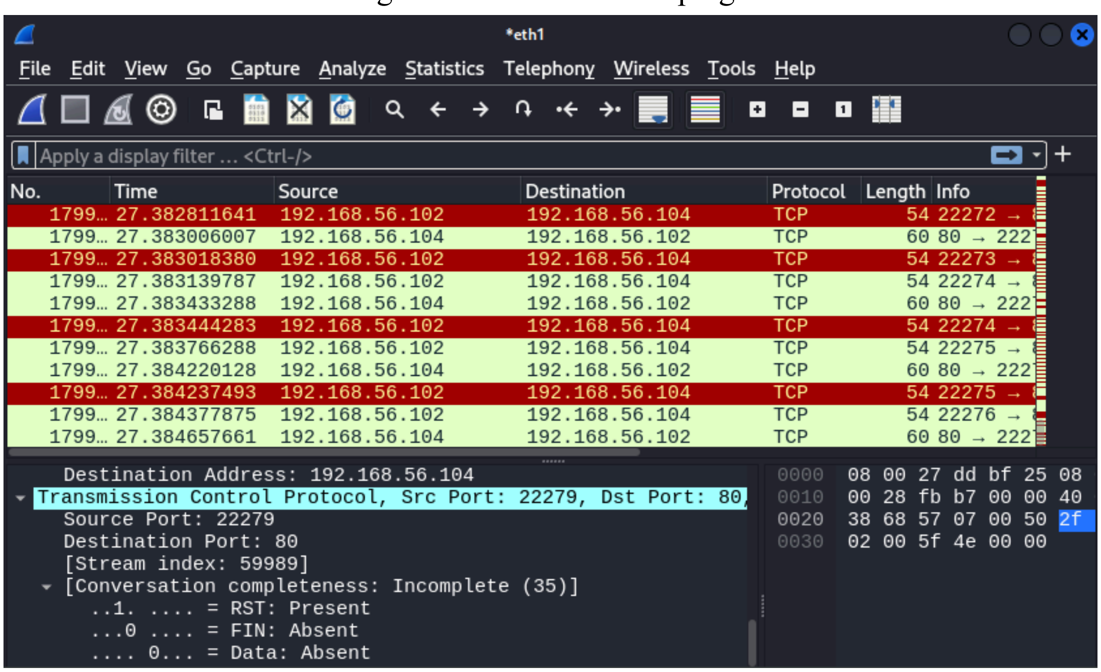


Figure 30: The traffic captured by Wireshark

Phase 3: Anomaly Intrusion Detection

From a high-end view, our code is split into two files, 'Train.py' simply trains the ML model on relevant data that we will describe below and 'Test.py,' which looks into the input data file to discern if it has valid labels and then predicts and validates the data with the loaded ML model. Luckily, a lot of research has been cited concerning these models. In [4], we gained more insight and decided to implement the Random Forest ML classifier model, which was mentioned to have a very high accuracy rate. Furthermore, we learned from [5] that embedded within the random Forest module in Scikit exists a module called .feature_importances_, which allows us to visualize the weights associated with training data to determine which vectors have minimal influence on the models' predictions. With this in mind, we outputted a visual representation of these features during model construction and determined which columns could be pruned from the training sets. After many test attempts, we pruned the information concerning forward and backward URG, RST, ECE, CWE flags, and bulk packet and byte rates. Aside from those, we incorporated all other data points, as they had some influence on the Random Forests. When considering the classes for this project, we wanted to simplify the data into simple binaries; therefore, we generalized all benign data as 0 and any DoS data as 1. We cite the Figure below to understand better how random forests work, which was taken from [6].

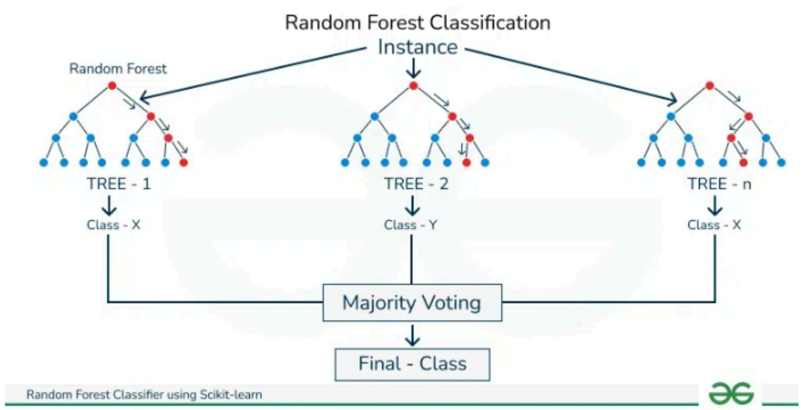


Figure 31: Random Forest Classifier from [6].

Figure: Random Forest Classifier from [6]. This model incorporates a series of randomized decision trees that consider input variables. Each results in a class designation. These classifications are amalgamated, and the majority is the resulting prediction.[6] With this in mind, we conducted thorough experiments on Random Forests regarding estimator parameters. We were determined to ensure that the forest was trained very thoroughly, as differences in DoS data and normal data are relatively minor. Therefore, we implemented the system with 50 estimators. Concerning our test parameters, to distinguish easily between the true values of the traffic and the estimations, we conducted SSH traffic while employing slowhttptest so that we could record any dest port which is not 80 as normal traffic. The result is shown in the Figure below with the ROC curve, AUC noted in the graph, and the DR and FPR indicated in the command line of this test.

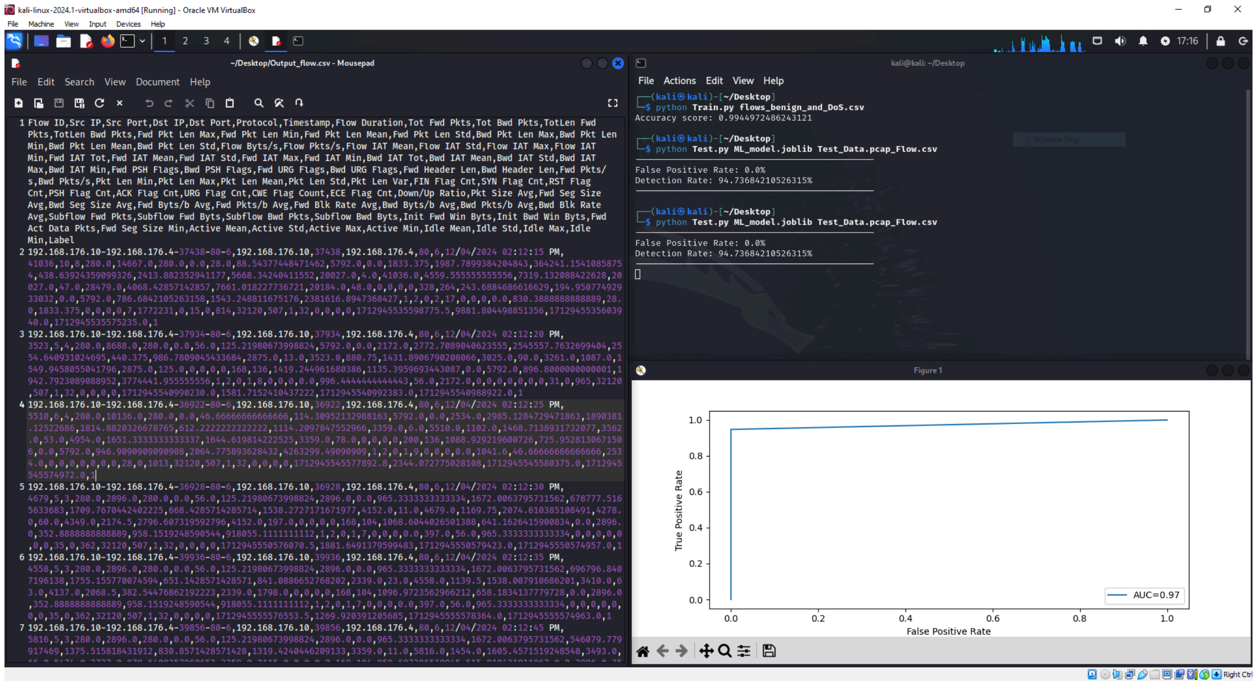


Figure 32: Best Operating Point of IDS code on Testing in Kali Environment.

References

[1] “Snort users manual 2.9.16,” SNORT Users Manual 2.9.16, http://manual-snort-org.s3-website-us-east-1.amazonaws.com/ (accessed Apr. 9, 2024).

[2] Detect SQL injection and cross-site scripting attacks, https://www.blackhat.com/presentations/bh-usa-04/bh-us-04-mookhey/old/bh-us-04-mookhey_whitepaper.pdf (accessed Apr. 9, 2024).

[3] Payloadbox, “Payloadbox/XSS-payload-list: 🎯 Cross-site scripting (XSS) Vulnerability payload list,” GitHub, https://github.com/payloadbox/xss-payload-list (accessed Apr. 9, 2024).

[4] S. Wankhede and D. Kshirsagar, "DoS Attack Detection Using Machine Learning and Neural Network," 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), Pune, India, 2018, pp. 1-5, doi: 10.1109/ICCUBEA.2018.8697702.

[5] S. Singh, “ML Beginner’s Guide to ddos attack detection model,” Labellerr, https://www.labellerr.com/blog/ddos-attack-detection/ (accessed Apr. 12, 2024).

[6] GfG, “Random Forest classifier using Scikit-learn,” GeeksforGeeks, https://www.geeksforgeeks.org/random-forest-classifier-using-scikit-learn/ (accessed Apr. 12, 2024).