Rerverse Engineer: Lab 1 Report

[Redacted]

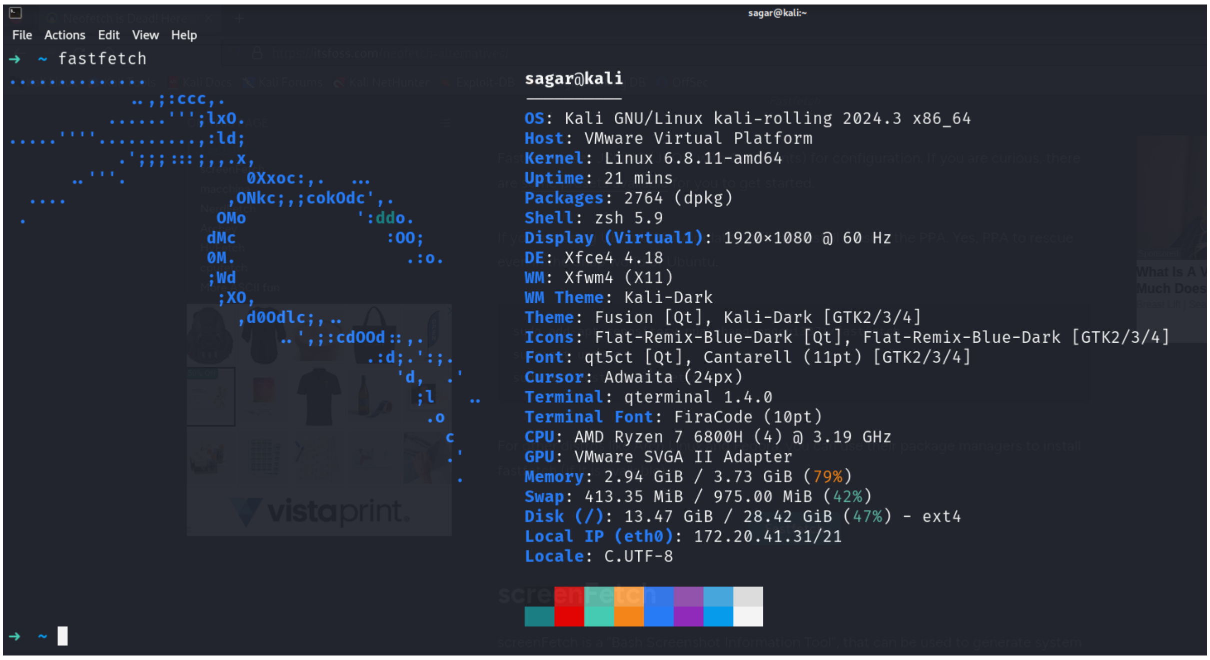
September 6, 2024

Contents

Chapter 1

Getting Started

In this lab we used a virtual machine (VM) Running Kali linux, These are the specifications:



```
➜  ~ fastfetch
sagar@kali
----------
OS: Kali GNU/Linux kali-rolling 2024.3 x86_64
Host: VMware Virtual Platform
Kernel: Linux 6.8.11-amd64
Uptime: 21 mins
Packages: 2764 (dpkg)
Shell: zsh 5.9
Display (Virtual1): 1920×1080 @ 60 Hz
DE: Xfce4 4.18
WM: Xfwm4 (X11)
WM Theme: Kali-Dark
Theme: Fusion [Qt], Kali-Dark [GTK2/3/4]
Icons: Flat-Remix-Blue-Dark [Qt], Flat-Remix-Blue-Dark [GTK2/3/4]
Font: qt5ct [Qt], Cantarell (11pt) [GTK2/3/4]
Cursor: Adwaita (24px)
Terminal: qterminal 1.4.0
Terminal Font: FiraCode (10pt)
CPU: AMD Ryzen 7 6800H (4) @ 3.19 GHz
GPU: VMware SVGA II Adapter
Memory: 2.94 GiB / 3.73 GiB (79%)
Swap: 413.35 MiB / 975.00 MiB (42%)
Disk (/): 13.47 GiB / 28.42 GiB (47%) - ext4
Local IP (eth0): 172.20.41.31/21
Locale: C.UTF-8

➜  ~ 
```

The tools that where used are the following:

- Wireshark
- YARA
- Ghidra
- **Bonus**: Radare2

Chapter 2

Wireshark

2.1 Basic Packet Capture and Analysis

2.1.1 Objective

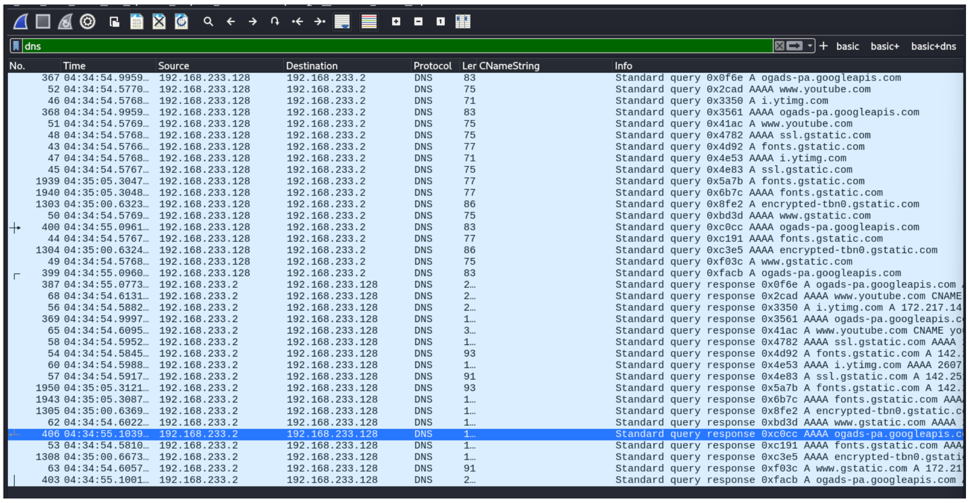
Capture live network traffic on your local machine and analyze the captured data.

2.1.2 Steps

1. **Start a Capture:** Open Wireshark and select the appropriate network interface.
2. **Generate Traffic:** Open a web browser and visit several websites and perform actions like downloading files or streaming a video.
3. **Stop the Capture:** Stop capturing traffic and save the captured traffic by going to File > Save As.
4. **Analyze the Capture Data:**
   - Use the filter bar to search for HTTP traffic (http) or DNS traffic (dns).
   - Right-click on a TCP packet and choose Follow > TCP Stream to view the data exchange.

2.1.3 Results

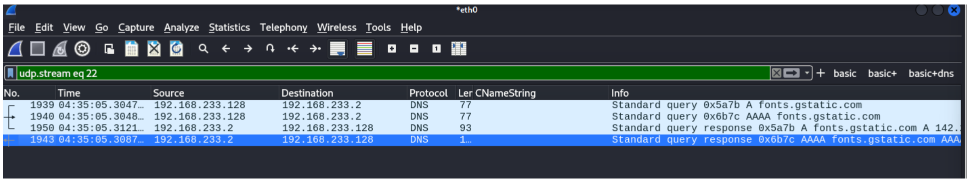
This is the dns traffic:



dns

No.	Time	Source	Destination	Protocol	Ler	CNameString	Info
367	04:34:54.9959…	192.168.233.128	192.168.233.2	DNS	83		Standard query 0x0f6e A ogads-pa.googleapis.com
52	04:34:54.5770…	192.168.233.128	192.168.233.2	DNS	75		Standard query 0x2cad AAAA www.youtube.com
46	04:34:54.5768…	192.168.233.128	192.168.233.2	DNS	71		Standard query 0x3350 A i.ytimg.com
368	04:34:54.9959…	192.168.233.128	192.168.233.2	DNS	83		Standard query 0x3561 AAAA ogads-pa.googleapis.com
51	04:34:54.5769…	192.168.233.128	192.168.233.2	DNS	75		Standard query 0x41ac A www.youtube.com
48	04:34:54.5768…	192.168.233.128	192.168.233.2	DNS	75		Standard query 0x4782 AAAA ssl.gstatic.com
43	04:34:54.5766…	192.168.233.128	192.168.233.2	DNS	77		Standard query 0x4d92 A fonts.gstatic.com
47	04:34:54.5768…	192.168.233.128	192.168.233.2	DNS	71		Standard query 0x4e53 AAAA i.ytimg.com
45	04:34:54.5767…	192.168.233.128	192.168.233.2	DNS	75		Standard query 0x4e83 A ssl.gstatic.com
1939	04:35:05.3047…	192.168.233.128	192.168.233.2	DNS	77		Standard query 0x5a7b A fonts.gstatic.com
1940	04:35:05.3048…	192.168.233.128	192.168.233.2	DNS	77		Standard query 0x6b7c AAAA fonts.gstatic.com
1303	04:35:00.6323…	192.168.233.128	192.168.233.2	DNS	86		Standard query 0x8fe2 A encrypted-tbn0.gstatic.com
50	04:34:54.5769…	192.168.233.128	192.168.233.2	DNS	75		Standard query 0xbd3d AAAA www.gstatic.com
400	04:34:55.0961…	192.168.233.128	192.168.233.2	DNS	83		Standard query 0xc0cc AAAA ogads-pa.googleapis.com
44	04:34:54.5767…	192.168.233.128	192.168.233.2	DNS	77		Standard query 0xc191 AAAA fonts.gstatic.com
1304	04:35:00.6324…	192.168.233.128	192.168.233.2	DNS	86		Standard query 0xc3e5 AAAA encrypted-tbn0.gstatic.com
49	04:34:54.5768…	192.168.233.128	192.168.233.2	DNS	75		Standard query 0xf03c A www.gstatic.com
399	04:34:55.0960…	192.168.233.128	192.168.233.2	DNS	83		Standard query 0xfacb A ogads-pa.googleapis.com
387	04:34:55.0773…	192.168.233.2	192.168.233.128	DNS	2…		Standard query response 0x0f6e A ogads-pa.googleapis.com
68	04:34:54.6131…	192.168.233.2	192.168.233.128	DNS	2…		Standard query response 0x2cad AAAA www.youtube.com CNAME
56	04:34:54.5882…	192.168.233.2	192.168.233.128	DNS	2…		Standard query response 0x3350 A i.ytimg.com A 172.217.14
369	04:34:54.9997…	192.168.233.2	192.168.233.128	DNS	1…		Standard query response 0x3561 AAAA ogads-pa.googleapis.c
65	04:34:54.6095…	192.168.233.2	192.168.233.128	DNS	3…		Standard query response 0x41ac A www.youtube.com CNAME yo
58	04:34:54.5952…	192.168.233.2	192.168.233.128	DNS	1…		Standard query response 0x4782 AAAA ssl.gstatic.com AAAA
54	04:34:54.5845…	192.168.233.2	192.168.233.128	DNS	93		Standard query response 0x4d92 A fonts.gstatic.com A 142.
60	04:34:54.5988…	192.168.233.2	192.168.233.128	DNS	1…		Standard query response 0x4e53 AAAA i.ytimg.com AAAA 2607
57	04:34:54.5917…	192.168.233.2	192.168.233.128	DNS	91		Standard query response 0x4e83 A ssl.gstatic.com A 142.25
1950	04:35:05.3121…	192.168.233.2	192.168.233.128	DNS	93		Standard query response 0x5a7b A fonts.gstatic.com A 142.
1943	04:35:05.3087…	192.168.233.2	192.168.233.128	DNS	1…		Standard query response 0x6b7c AAAA fonts.gstatic.com AAA
1305	04:35:00.6369…	192.168.233.2	192.168.233.128	DNS	1…		Standard query response 0x8fe2 A encrypted-tbn0.gstatic.c
62	04:34:54.6022…	192.168.233.2	192.168.233.128	DNS	1…		Standard query response 0xbd3d AAAA www.gstatic.com AAAA
406	04:34:55.1039…	192.168.233.2	192.168.233.128	DNS	1…		Standard query response 0xc0cc AAAA ogads-pa.googleapis.c
53	04:34:54.5810…	192.168.233.2	192.168.233.128	DNS	1…		Standard query response 0xc191 AAAA fonts.gstatic.com AAA
1308	04:35:00.6673…	192.168.233.2	192.168.233.128	DNS	1…		Standard query response 0xc3e5 AAAA encrypted-tbn0.gstati
63	04:34:54.6057…	192.168.233.2	192.168.233.128	DNS	91		Standard query response 0xf03c A www.gstatic.com A 172.21
403	04:34:55.1001…	192.168.233.2	192.168.233.128	DNS	2…		Standard query response 0xfacb A ogads-pa.googleapis.com

If we right click and press follow package this happens


*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
+ basic basic+ basic+dns

udp.stream eq 22

```
No.   Time            Source           Destination      Protocol Ler CNameString  Info
 1939 04:35:05.3047… 192.168.233.128  192.168.233.2    DNS      77               Standard query 0x5a7b A fonts.gstatic.com
 1940 04:35:05.3048… 192.168.233.128  192.168.233.2    DNS      77               Standard query 0x6b7c AAAA fonts.gstatic.com
 1950 04:35:05.3121… 192.168.233.2    192.168.233.128  DNS      93               Standard query response 0x5a7b A fonts.gstatic.com A 142.
 1943 04:35:05.3087… 192.168.233.2    192.168.233.128  DNS      1…               Standard query response 0x6b7c AAAA fonts.gstatic.com AAA
```

2.2 Analyzing Malicious Network Traffic

2.2.1 Objective

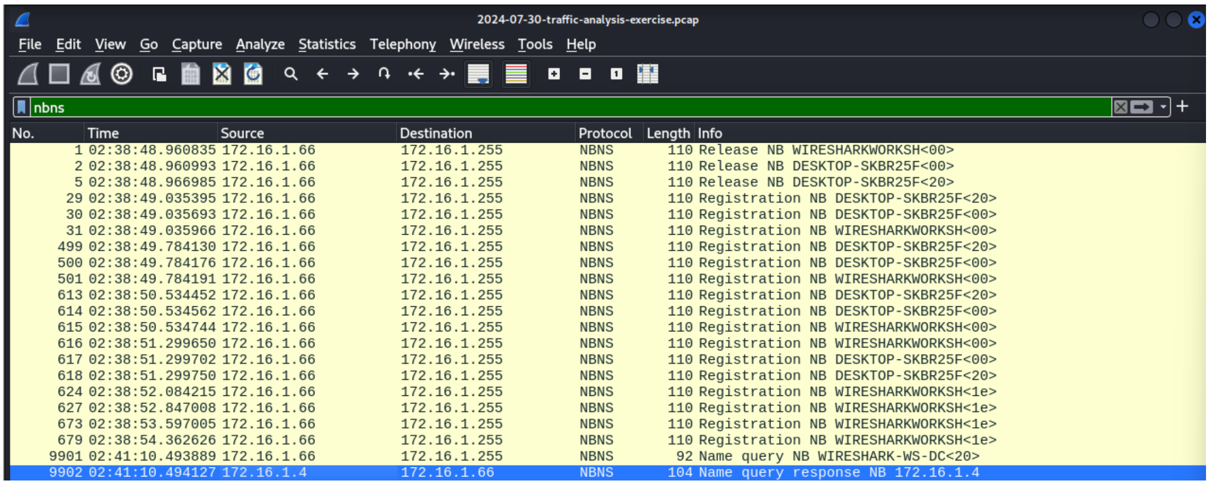
Analyze a pre-captured pcap file containing malicious traffic.

2.2.2 Steps

1. **Obtain the pcap File:** Obtain the pcap file from blackboard. Password to open the file ***infected_20240730***
2. **Load the pcap File in Wireshark**
3. **Scenario:**
   - LAN segment range: 172.16.1[.]0/24 (172.16.1[.]0 through 172.16.1[.]255)
   - Domain: wiresharkworkshop[.]online
   - Domain controller: 172.16.1[.]4 - WIRESHARK-WS-DC
   - LAN segment gateway: 172.16.1[.]1
   - LAN segment broadcast address: 172.16.1[.]255

2.2.3 Results

In the filter we have to use **nbns**, this way we can identify hotsnames for computers running Microsoft Windows.

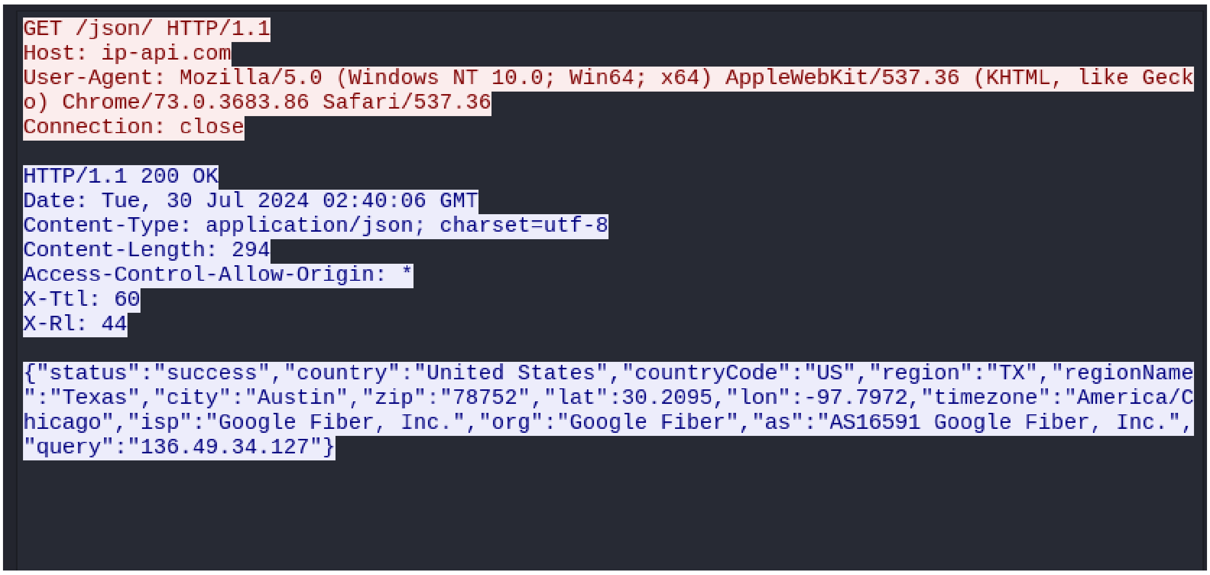

2024-07-30-traffic-analysis-exercise.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

nbns

```
No.  Time            Source       Destination  Protocol Length Info
   1 02:38:48.960835 172.16.1.66  172.16.1.255 NBNS     110 Release NB WIRESHARKWORKSH<00>
   2 02:38:48.960993 172.16.1.66  172.16.1.255 NBNS     110 Release NB DESKTOP-SKBR25F<00>
   5 02:38:48.966985 172.16.1.66  172.16.1.255 NBNS     110 Release NB DESKTOP-SKBR25F<20>
  29 02:38:49.035395 172.16.1.66  172.16.1.255 NBNS     110 Registration NB DESKTOP-SKBR25F<20>
  30 02:38:49.035693 172.16.1.66  172.16.1.255 NBNS     110 Registration NB DESKTOP-SKBR25F<00>
  31 02:38:49.035966 172.16.1.66  172.16.1.255 NBNS     110 Registration NB WIRESHARKWORKSH<00>
 499 02:38:49.784130 172.16.1.66  172.16.1.255 NBNS     110 Registration NB DESKTOP-SKBR25F<20>
 500 02:38:49.784176 172.16.1.66  172.16.1.255 NBNS     110 Registration NB DESKTOP-SKBR25F<00>
 501 02:38:49.784191 172.16.1.66  172.16.1.255 NBNS     110 Registration NB WIRESHARKWORKSH<00>
 613 02:38:50.534452 172.16.1.66  172.16.1.255 NBNS     110 Registration NB DESKTOP-SKBR25F<20>
 614 02:38:50.534562 172.16.1.66  172.16.1.255 NBNS     110 Registration NB DESKTOP-SKBR25F<00>
 615 02:38:50.534744 172.16.1.66  172.16.1.255 NBNS     110 Registration NB WIRESHARKWORKSH<00>
 616 02:38:51.299650 172.16.1.66  172.16.1.255 NBNS     110 Registration NB WIRESHARKWORKSH<00>
 617 02:38:51.299702 172.16.1.66  172.16.1.255 NBNS     110 Registration NB DESKTOP-SKBR25F<00>
 618 02:38:51.299750 172.16.1.66  172.16.1.255 NBNS     110 Registration NB DESKTOP-SKBR25F<20>
 624 02:38:52.084215 172.16.1.66  172.16.1.255 NBNS     110 Registration NB WIRESHARKWORKSH<1e>
 627 02:38:52.847008 172.16.1.66  172.16.1.255 NBNS     110 Registration NB WIRESHARKWORKSH<1e>
 673 02:38:53.597005 172.16.1.66  172.16.1.255 NBNS     110 Registration NB WIRESHARKWORKSH<1e>
 679 02:38:54.362626 172.16.1.66  172.16.1.255 NBNS     110 Registration NB WIRESHARKWORKSH<1e>
9901 02:41:10.493889 172.16.1.66  172.16.1.255 NBNS      92 Name query NB WIRESHARK-WS-DC<20>
9902 02:41:10.494127 172.16.1.4   172.16.1.66  NBNS     104 Name query response NB 172.16.1.4
```

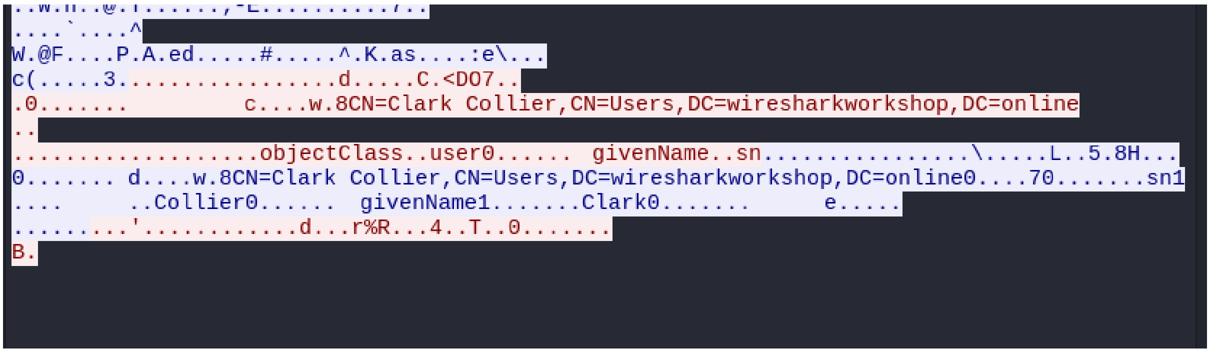
Now we notice that a computer **Desktop-SKBR25F**.

After following the tcp.stream eq 84. (we can get this by (http.request) and !(ssdp))



```
GET /json/ HTTP/1.1
Host: ip-api.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Geck
o) Chrome/73.0.3683.86 Safari/537.36
Connection: close

HTTP/1.1 200 OK
Date: Tue, 30 Jul 2024 02:40:06 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 294
Access-Control-Allow-Origin: *
X-Ttl: 60
X-Rl: 44

{"status":"success","country":"United States","countryCode":"US","region":"TX","regionName
":"Texas","city":"Austin","zip":"78752","lat":30.2095,"lon":-97.7972,"timezone":"America/C
hicago","isp":"Google Fiber, Inc.","org":"Google Fiber","as":"AS16591 Google Fiber, Inc.",
"query":"136.49.34.127"}
```

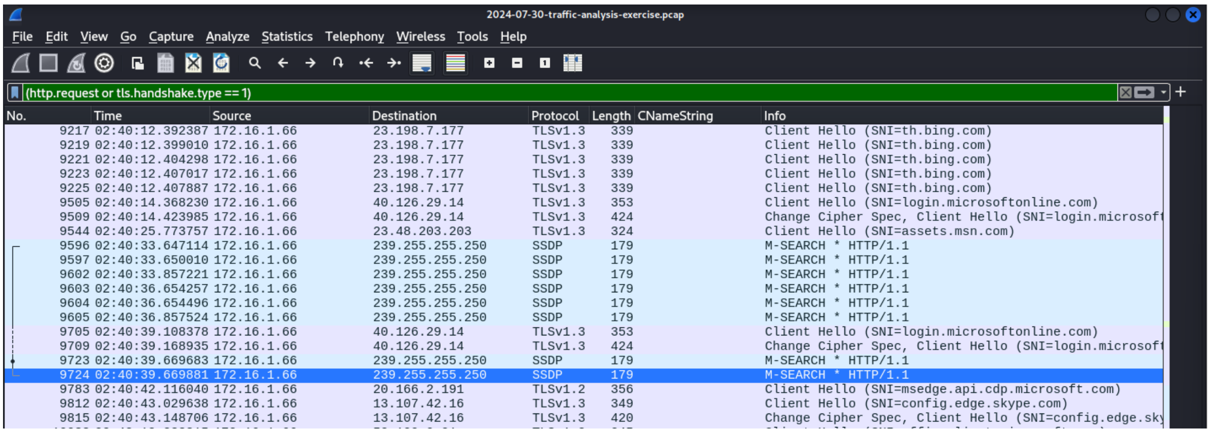
Basically this is the information of the victim



```
..w.h..@.T......,-L..........7..
....`....^
W.@F....P.A.ed.....#.....^.K.as....:e\...
c(.....3.................d.....C.<DO7..
.0.......         c....w.8CN=Clark Collier,CN=Users,DC=wiresharkworkshop,DC=online
..
...................objectClass..user0......  givenName..sn................\.....L..5.8H...
0........ d....w.8CN=Clark Collier,CN=Users,DC=wiresharkworkshop,DC=online0....70.......sn1
....     ..Collier0......  givenName1.......Clark0.......     e.....
..........'............d...r%R...4..T..0.......
B.
```

After this we are going to use **ldap contains "CN=Users"**

Now we are going to filter Web traffic, if we use (http.request or tls.handshake.type == 1) we get the following:

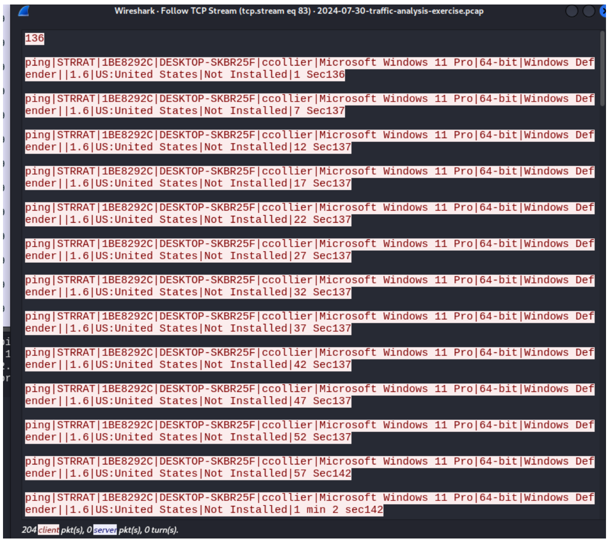


(http.request or tls.handshake.type == 1)

```
No.   Time            Source        Destination      Protocol Length CNameString Info
 9217 02:40:12.392387 172.16.1.66   23.198.7.177     TLSv1.3  339               Client Hello (SNI=th.bing.com)
 9219 02:40:12.399010 172.16.1.66   23.198.7.177     TLSv1.3  339               Client Hello (SNI=th.bing.com)
 9221 02:40:12.404298 172.16.1.66   23.198.7.177     TLSv1.3  339               Client Hello (SNI=th.bing.com)
 9223 02:40:12.407017 172.16.1.66   23.198.7.177     TLSv1.3  339               Client Hello (SNI=th.bing.com)
 9225 02:40:12.407887 172.16.1.66   23.198.7.177     TLSv1.3  339               Client Hello (SNI=th.bing.com)
 9505 02:40:14.368230 172.16.1.66   40.126.29.14     TLSv1.3  353               Client Hello (SNI=login.microsoftonline.com)
 9509 02:40:14.423985 172.16.1.66   40.126.29.14     TLSv1.3  424               Change Cipher Spec, Client Hello (SNI=login.microsof
 9544 02:40:25.773757 172.16.1.66   23.48.203.203    TLSv1.3  324               Client Hello (SNI=assets.msn.com)
 9596 02:40:33.647114 172.16.1.66   239.255.255.250  SSDP     179               M-SEARCH * HTTP/1.1
 9597 02:40:33.650010 172.16.1.66   239.255.255.250  SSDP     179               M-SEARCH * HTTP/1.1
 9602 02:40:33.857221 172.16.1.66   239.255.255.250  SSDP     179               M-SEARCH * HTTP/1.1
 9603 02:40:36.654257 172.16.1.66   239.255.255.250  SSDP     179               M-SEARCH * HTTP/1.1
 9604 02:40:36.654496 172.16.1.66   239.255.255.250  SSDP     179               M-SEARCH * HTTP/1.1
 9605 02:40:36.857524 172.16.1.66   239.255.255.250  SSDP     179               M-SEARCH * HTTP/1.1
 9705 02:40:39.108378 172.16.1.66   40.126.29.14     TLSv1.3  353               Client Hello (SNI=login.microsoftonline.com)
 9709 02:40:39.168935 172.16.1.66   40.126.29.14     TLSv1.3  424               Change Cipher Spec, Client Hello (SNI=login.microsof
 9723 02:40:39.669683 172.16.1.66   239.255.255.250  SSDP     179               M-SEARCH * HTTP/1.1
 9724 02:40:39.669881 172.16.1.66   239.255.255.250  SSDP     179               M-SEARCH * HTTP/1.1
 9783 02:40:42.116040 172.16.1.66   20.166.2.191     TLSv1.2  356               Client Hello (SNI=msedge.api.cdp.microsoft.com)
 9812 02:40:43.029638 172.16.1.66   13.107.42.16     TLSv1.3  349               Client Hello (SNI=config.edge.skype.com)
 9815 02:40:43.148706 172.16.1.66   13.107.42.16     TLSv1.3  420               Change Cipher Spec, Client Hello (SNI=config.edge.sky
```

The problem is taht we are getting SSDP traffic this can be solved by adding the following to the filter:

(http.request or tls.handshake.type == 1) && !(ssdp) now, with this filter (http.request or tls.handshake.type eq 1 or (tcp.flags.syn eq 1 and tcp.flags.ack eq 0)) and !(ssdp) we are going to get the following text

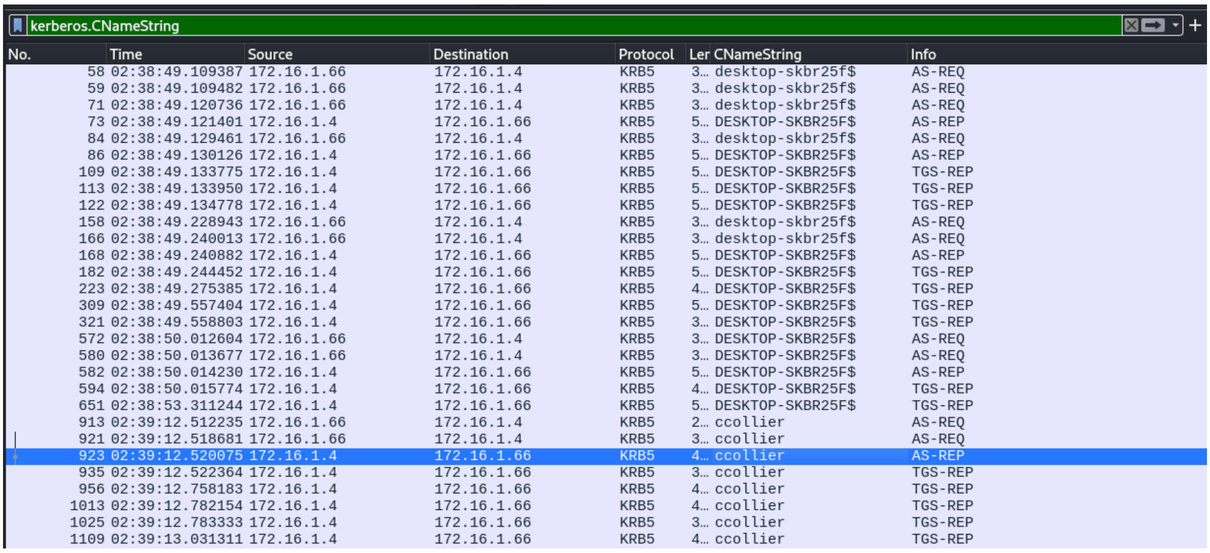


Wireshark · Follow TCP Stream (tcp.stream eq 83) · 2024-07-30-traffic-analysis-exercise.pcap

```
136

ping|STRRAT|1BE8292C|DESKTOP-SKBR25F|ccollier|Microsoft Windows 11 Pro|64-bit|Windows Def
ender||1.6|US:United States|Not Installed|1 Sec136

ping|STRRAT|1BE8292C|DESKTOP-SKBR25F|ccollier|Microsoft Windows 11 Pro|64-bit|Windows Def
ender||1.6|US:United States|Not Installed|7 Sec137

ping|STRRAT|1BE8292C|DESKTOP-SKBR25F|ccollier|Microsoft Windows 11 Pro|64-bit|Windows Def
ender||1.6|US:United States|Not Installed|12 Sec137

ping|STRRAT|1BE8292C|DESKTOP-SKBR25F|ccollier|Microsoft Windows 11 Pro|64-bit|Windows Def
ender||1.6|US:United States|Not Installed|17 Sec137

ping|STRRAT|1BE8292C|DESKTOP-SKBR25F|ccollier|Microsoft Windows 11 Pro|64-bit|Windows Def
ender||1.6|US:United States|Not Installed|22 Sec137

ping|STRRAT|1BE8292C|DESKTOP-SKBR25F|ccollier|Microsoft Windows 11 Pro|64-bit|Windows Def
ender||1.6|US:United States|Not Installed|27 Sec137

ping|STRRAT|1BE8292C|DESKTOP-SKBR25F|ccollier|Microsoft Windows 11 Pro|64-bit|Windows Def
ender||1.6|US:United States|Not Installed|32 Sec137

ping|STRRAT|1BE8292C|DESKTOP-SKBR25F|ccollier|Microsoft Windows 11 Pro|64-bit|Windows Def
ender||1.6|US:United States|Not Installed|37 Sec137

ping|STRRAT|1BE8292C|DESKTOP-SKBR25F|ccollier|Microsoft Windows 11 Pro|64-bit|Windows Def
ender||1.6|US:United States|Not Installed|42 Sec137

ping|STRRAT|1BE8292C|DESKTOP-SKBR25F|ccollier|Microsoft Windows 11 Pro|64-bit|Windows Def
ender||1.6|US:United States|Not Installed|47 Sec137

ping|STRRAT|1BE8292C|DESKTOP-SKBR25F|ccollier|Microsoft Windows 11 Pro|64-bit|Windows Def
ender||1.6|US:United States|Not Installed|52 Sec137

ping|STRRAT|1BE8292C|DESKTOP-SKBR25F|ccollier|Microsoft Windows 11 Pro|64-bit|Windows Def
ender||1.6|US:United States|Not Installed|57 Sec142

ping|STRRAT|1BE8292C|DESKTOP-SKBR25F|ccollier|Microsoft Windows 11 Pro|64-bit|Windows Def
ender||1.6|US:United States|Not Installed|1 min 2 sec142
```

which means that the user was infected with STRRAT.

Now, we can determine the user from the user account names in Kerberos traffic.



No.	Time	Source	Destination	Protocol	Ler	CNameString	Info
58	02:38:49.109387	172.16.1.66	172.16.1.4	KRB5	3…	desktop-skbr25f$	AS-REQ
59	02:38:49.109482	172.16.1.66	172.16.1.4	KRB5	3…	desktop-skbr25f$	AS-REQ
71	02:38:49.120736	172.16.1.66	172.16.1.4	KRB5	3…	desktop-skbr25f$	AS-REQ
73	02:38:49.121401	172.16.1.4	172.16.1.66	KRB5	5…	DESKTOP-SKBR25F$	AS-REP
84	02:38:49.129461	172.16.1.66	172.16.1.4	KRB5	3…	desktop-skbr25f$	AS-REQ
86	02:38:49.130126	172.16.1.4	172.16.1.66	KRB5	5…	DESKTOP-SKBR25F$	AS-REP
109	02:38:49.133775	172.16.1.4	172.16.1.66	KRB5	5…	DESKTOP-SKBR25F$	TGS-REP
113	02:38:49.133950	172.16.1.4	172.16.1.66	KRB5	5…	DESKTOP-SKBR25F$	TGS-REP
122	02:38:49.134778	172.16.1.4	172.16.1.66	KRB5	5…	DESKTOP-SKBR25F$	TGS-REP
158	02:38:49.228943	172.16.1.66	172.16.1.4	KRB5	3…	desktop-skbr25f$	AS-REQ
166	02:38:49.240013	172.16.1.66	172.16.1.4	KRB5	3…	desktop-skbr25f$	AS-REQ
168	02:38:49.240882	172.16.1.4	172.16.1.66	KRB5	5…	DESKTOP-SKBR25F$	AS-REP
182	02:38:49.244452	172.16.1.4	172.16.1.66	KRB5	5…	DESKTOP-SKBR25F$	TGS-REP
223	02:38:49.275385	172.16.1.4	172.16.1.66	KRB5	4…	DESKTOP-SKBR25F$	TGS-REP
309	02:38:49.557404	172.16.1.4	172.16.1.66	KRB5	5…	DESKTOP-SKBR25F$	TGS-REP
321	02:38:49.558803	172.16.1.4	172.16.1.66	KRB5	3…	DESKTOP-SKBR25F$	TGS-REP
572	02:38:50.012604	172.16.1.66	172.16.1.4	KRB5	3…	DESKTOP-SKBR25F$	AS-REQ
580	02:38:50.013677	172.16.1.66	172.16.1.4	KRB5	3…	DESKTOP-SKBR25F$	AS-REQ
582	02:38:50.014230	172.16.1.4	172.16.1.66	KRB5	5…	DESKTOP-SKBR25F$	AS-REP
594	02:38:50.015774	172.16.1.4	172.16.1.66	KRB5	4…	DESKTOP-SKBR25F$	TGS-REP
651	02:38:53.311244	172.16.1.4	172.16.1.66	KRB5	5…	DESKTOP-SKBR25F$	TGS-REP
913	02:39:12.512235	172.16.1.66	172.16.1.4	KRB5	2…	ccollier	AS-REQ
921	02:39:12.518681	172.16.1.66	172.16.1.4	KRB5	3…	ccollier	AS-REQ
923	02:39:12.520075	172.16.1.4	172.16.1.66	KRB5	4…	ccollier	AS-REP
935	02:39:12.522364	172.16.1.4	172.16.1.66	KRB5	3…	ccollier	TGS-REP
956	02:39:12.758183	172.16.1.4	172.16.1.66	KRB5	4…	ccollier	TGS-REP
1013	02:39:12.782154	172.16.1.4	172.16.1.66	KRB5	4…	ccollier	TGS-REP
1025	02:39:12.783333	172.16.1.4	172.16.1.66	KRB5	3…	ccollier	TGS-REP
1109	02:39:13.031311	172.16.1.4	172.16.1.66	KRB5	4…	ccollier	TGS-REP

From here we know that the username was ccollier.

Now lets summarize everything, These is the info we got from the victim:

1. **Computer Name:** Desktop-SKBR25F
2. **IP:** 172.16.1.66
3. **Username:** ccollier

The method of infection I think it was due email, because analyzing the traffic I noticed that it was on MSN before getting to github.

2.3 Network Traffic Analysis of a benign Application

2.3.1 Objective

Analyze the network traffic generated by a benign application to undestand its communication patterns.

2.3.2 Steps

1. **Choose a Benign Applicatoon:** Select a commonly used application.(e.g., a web browser, a messaging app, or a file transfer tool).

2. **Start Packet Capture:** Open Wireshark and start capturing traffic. Launch the application and perform typical actions, such as sending a message or downloading a file.

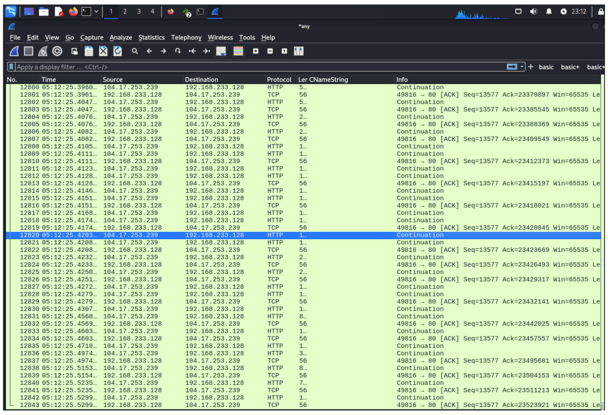
3. **Analyze the Captured Traffic:**

   - Stop the capture and filter the traffic based on the application's communication protocols (e.g., http, ftp, smtp).
   - Identify the IP addresses, ports, and protocols used by the application.
   - Follow a specific stream to understand how data is exchanged between the client and server.

2.3.3 Results

For This exercise we are going to use the browser that is included in Kali linux, which is FireFox.

we start the packet capture with any:


1 2 3 4
23:12
*any
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
basic basic+ basic+

No.	Time	Source	Destination	Protocol	Ler	CNameString	Info
12800	05:12:25.3960…	104.17.253.239	192.168.233.128	HTTP	5…		Continuation
12801	05:12:25.3961…	192.168.233.128	104.17.253.239	TCP	56		49816 → 80 [ACK] Seq=13577 Ack=23379897 Win=65535 Le
12802	05:12:25.4047…	104.17.253.239	192.168.233.128	HTTP	5…		Continuation
12803	05:12:25.4047…	192.168.233.128	104.17.253.239	TCP	56		49816 → 80 [ACK] Seq=13577 Ack=23385545 Win=65535 Le
12804	05:12:25.4076…	104.17.253.239	192.168.233.128	HTTP	2…		Continuation
12805	05:12:25.4076…	192.168.233.128	104.17.253.239	TCP	56		49816 → 80 [ACK] Seq=13577 Ack=23388369 Win=65535 Le
12806	05:12:25.4082…	104.17.253.239	192.168.233.128	HTTP	2…		Continuation
12807	05:12:25.4082…	192.168.233.128	104.17.253.239	TCP	56		49816 → 80 [ACK] Seq=13577 Ack=23409549 Win=65535 Le
12808	05:12:25.4105…	104.17.253.239	192.168.233.128	HTTP	1…		Continuation
12809	05:12:25.4111…	104.17.253.239	192.168.233.128	HTTP	1…		Continuation
12810	05:12:25.4111…	192.168.233.128	104.17.253.239	TCP	56		49816 → 80 [ACK] Seq=13577 Ack=23412373 Win=65535 Le
12811	05:12:25.4123…	104.17.253.239	192.168.233.128	HTTP	1…		Continuation
12812	05:12:25.4128…	104.17.253.239	192.168.233.128	HTTP	1…		Continuation
12813	05:12:25.4128…	192.168.233.128	104.17.253.239	TCP	56		49816 → 80 [ACK] Seq=13577 Ack=23415197 Win=65535 Le
12814	05:12:25.4146…	104.17.253.239	192.168.233.128	HTTP	1…		Continuation
12815	05:12:25.4151…	104.17.253.239	192.168.233.128	HTTP	1…		Continuation
12816	05:12:25.4151…	192.168.233.128	104.17.253.239	TCP	56		49816 → 80 [ACK] Seq=13577 Ack=23418021 Win=65535 Le
12817	05:12:25.4168…	104.17.253.239	192.168.233.128	HTTP	1…		Continuation
12818	05:12:25.4174…	104.17.253.239	192.168.233.128	HTTP	1…		Continuation
12819	05:12:25.4174…	192.168.233.128	104.17.253.239	TCP	56		49816 → 80 [ACK] Seq=13577 Ack=23420845 Win=65535 Le
12820	05:12:25.4203…	104.17.253.239	192.168.233.128	HTTP	1…		Continuation
12821	05:12:25.4208…	104.17.253.239	192.168.233.128	HTTP	1…		Continuation
12822	05:12:25.4208…	192.168.233.128	104.17.253.239	TCP	56		49816 → 80 [ACK] Seq=13577 Ack=23423669 Win=65535 Le
12823	05:12:25.4232…	104.17.253.239	192.168.233.128	HTTP	2…		Continuation
12824	05:12:25.4233…	192.168.233.128	104.17.253.239	TCP	56		49816 → 80 [ACK] Seq=13577 Ack=23426493 Win=65535 Le
12825	05:12:25.4250…	104.17.253.239	192.168.233.128	HTTP	2…		Continuation
12826	05:12:25.4251…	192.168.233.128	104.17.253.239	TCP	56		49816 → 80 [ACK] Seq=13577 Ack=23429317 Win=65535 Le
12827	05:12:25.4272…	104.17.253.239	192.168.233.128	HTTP	1…		Continuation
12828	05:12:25.4279…	104.17.253.239	192.168.233.128	HTTP	1…		Continuation
12829	05:12:25.4279…	192.168.233.128	104.17.253.239	TCP	56		49816 → 80 [ACK] Seq=13577 Ack=23432141 Win=65535 Le
12830	05:12:25.4307…	104.17.253.239	192.168.233.128	HTTP	1…		Continuation
12831	05:12:25.4568…	104.17.253.239	192.168.233.128	HTTP	8…		Continuation
12832	05:12:25.4569…	192.168.233.128	104.17.253.239	TCP	56		49816 → 80 [ACK] Seq=13577 Ack=23442025 Win=65535 Le
12833	05:12:25.4603…	104.17.253.239	192.168.233.128	HTTP	1…		Continuation
12834	05:12:25.4603…	192.168.233.128	104.17.253.239	TCP	56		49816 → 80 [ACK] Seq=13577 Ack=23457557 Win=65535 Le
12835	05:12:25.4710…	104.17.253.239	192.168.233.128	HTTP	1…		Continuation
12836	05:12:25.4974…	104.17.253.239	192.168.233.128	HTTP	3…		Continuation
12837	05:12:25.4974…	192.168.233.128	104.17.253.239	TCP	56		49816 → 80 [ACK] Seq=13577 Ack=23495681 Win=65535 Le
12838	05:12:25.5153…	104.17.253.239	192.168.233.128	HTTP	8…		Continuation
12839	05:12:25.5154…	192.168.233.128	104.17.253.239	TCP	56		49816 → 80 [ACK] Seq=13577 Ack=23504153 Win=65535 Le
12840	05:12:25.5235…	104.17.253.239	192.168.233.128	HTTP	7…		Continuation
12841	05:12:25.5235…	192.168.233.128	104.17.253.239	TCP	56		49816 → 80 [ACK] Seq=13577 Ack=23511213 Win=65535 Le
12842	05:12:25.5299…	104.17.253.239	192.168.233.128	HTTP	1…		Continuation
12843	05:12:25.5299…	192.168.233.128	104.17.253.239	TCP	56		49816 → 80 [ACK] Seq=13577 Ack=23523921 Win=65535 Le

This ias all the capture, as we notice here, the app uses HTTP, DNS. My local ip is 192.168.233.128.

2.4 Comparing Benign and Malicious Traffic

2.4.1 Objective

Compare the network traffic of a benign application with that of a known malware sample to highlight key differences.

2.4.2 Steps

1. **Analyze Both Traffic Samples:**

   - Load both the malware and benign pcap files into Wireshark.

- Use filters to isolate relevant traffic (e.g., DNS, HTTP, or custom protocols).
- Compare the two traffic samples, noting differences in IP addresses, ports, frequency of connections, and data payloads.

2. **Identify Malicious Indicators:**

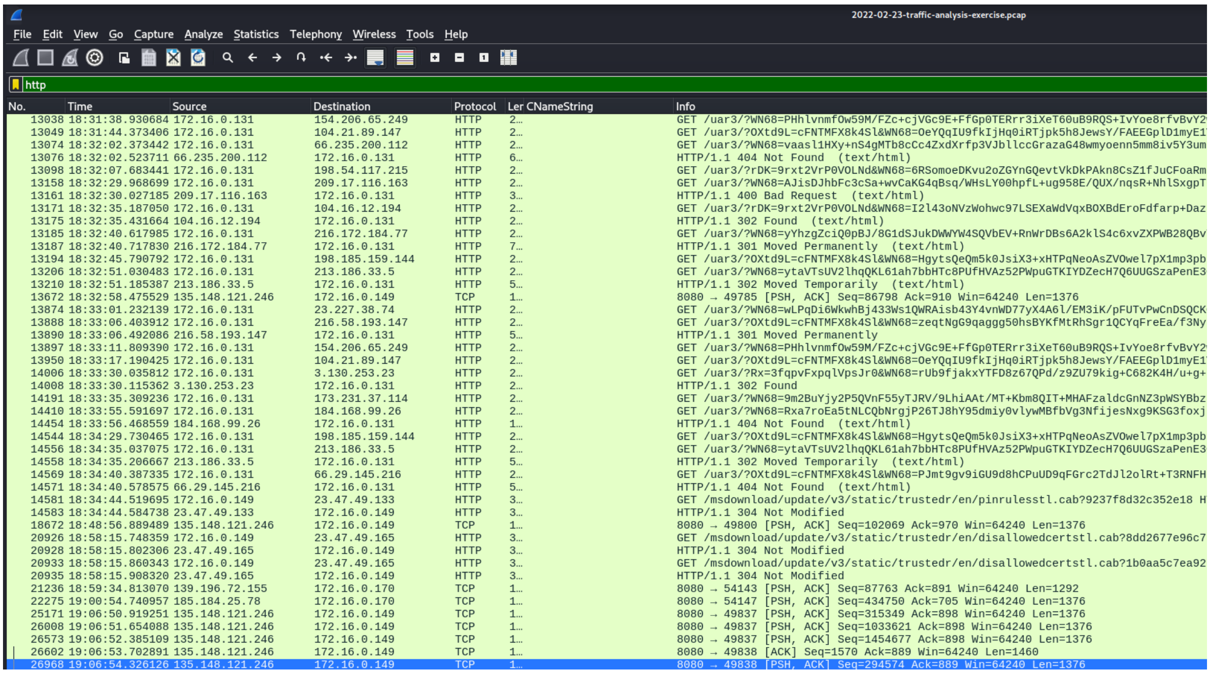
   - What hosts/user account names are active on this network?
   - What type of malware are they infected with?

3. Report

   - A comparative analysis report that clearly identifies the differences between benign and malicious traffic, with practical examples from the Wireshark captures.

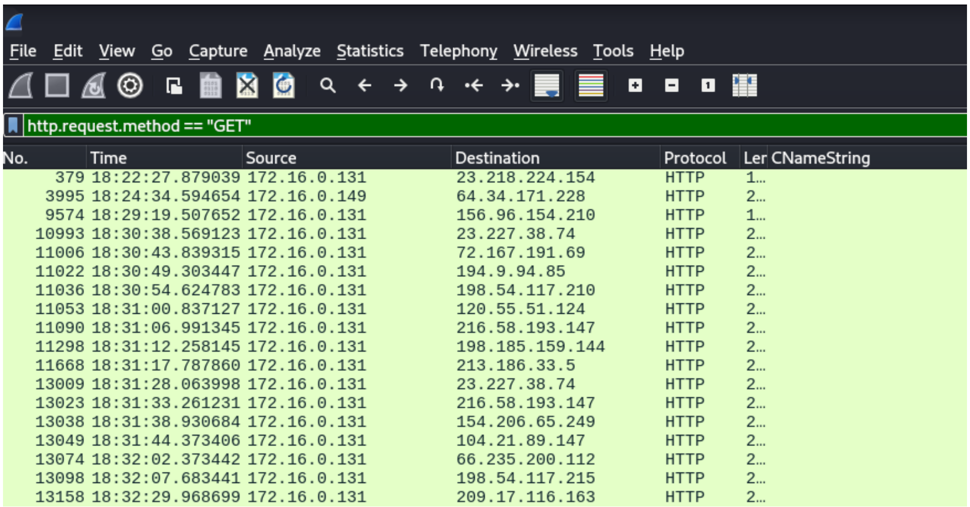
2.4.3 Results

Well, first we are going to check the protocols such as HTTP or DNS in the infected PCAP:



```
http
No.   Time              Source          Destination      Protocol Ler CNameString  Info
13038 18:31:38.930684 172.16.0.131    154.206.65.249   HTTP     2…               GET /uar3/?WN68=PHhlvnmfOw59M/FZc+cjVGc9E+FfGp0TERrr3iXeT60uB9RQS+IvYoe8rfvBvY2
13049 18:31:44.373406 172.16.0.131    104.21.89.147    HTTP     2…               GET /uar3/?OXtd9L=cFNTMFX8k4Sl&WN68=OeYQqIU9fkIjHq0iRTjpk5h8JewsY/FAEEGplD1myE1
13074 18:32:02.373442 172.16.0.131    66.235.200.112   HTTP     2…               GET /uar3/?WN68=vaasl1HXy+nS4gMTb8cCc4ZxdXrfp3VJbllccGrazaG48wmyoenn5mm8iv5Y3um
13076 18:32:02.523711 66.235.200.112  172.16.0.131     HTTP     6…               HTTP/1.1 404 Not Found  (text/html)
13098 18:32:07.683441 172.16.0.131    198.54.117.215   HTTP     2…               GET /uar3/?rDK=9rxt2VrP0VOLNd&WN68=6RSomoeDKvu2oZGYnGQevtVkDkPAkn8CsZ1fJuCFoaRm
13158 18:32:29.968699 172.16.0.131    209.17.116.163   HTTP     2…               GET /uar3/?WN68=AJisDJhbFc3cSa+wvCaKG4qBsq/WHsLY00hpfL+ug958E/QUX/nqsR+NhlSxgpT
13161 18:32:30.027185 209.17.116.163  172.16.0.131     HTTP     3…               HTTP/1.1 400 Bad Request  (text/html)
13171 18:32:35.187050 172.16.0.131    104.16.12.194    HTTP     2…               GET /uar3/?rDK=9rxt2VrP0VOLNd&WN68=I2l43oNVzWohwc97LSEXaWdVqxBOXBdEroFdfarp+Daz
13175 18:32:35.431664 104.16.12.194   172.16.0.131     HTTP     2…               HTTP/1.1 302 Found  (text/html)
13185 18:32:40.617985 172.16.0.131    216.172.184.77   HTTP     2…               GET /uar3/?WN68=yYhzgZciQ0pBJ/8G1dSJukDWWYW4SQVbEV+RnWrDBs6A2klS4c6xvZXPWB28QBv
13187 18:32:40.717830 216.172.184.77  172.16.0.131     HTTP     7…               HTTP/1.1 301 Moved Permanently  (text/html)
13194 18:32:45.790792 172.16.0.131    198.185.159.144  HTTP     2…               GET /uar3/?OXtd9L=cFNTMFX8k4Sl&WN68=HgytsQeQm5k0JsiX3+xHTPqNeoAsZVOwel7pX1mp3pb
13206 18:32:51.030483 172.16.0.131    213.186.33.5     HTTP     2…               GET /uar3/?WN68=ytaVTsUV2lhqQKL61ah7bbHTc8PUfHVAz52PWpuGTKIYDZecH7Q6UUGSzaPenE3
13210 18:32:51.185387 213.186.33.5    172.16.0.131     HTTP     5…               HTTP/1.1 302 Moved Temporarily  (text/html)
13672 18:32:58.475529 135.148.121.246 172.16.0.149     TCP      1…               8080 → 49785 [PSH, ACK] Seq=86798 Ack=910 Win=64240 Len=1376
13874 18:33:01.232139 172.16.0.131    23.227.38.74     HTTP     2…               GET /uar3/?WN68=wLPqDi6WkwhBj433Ws1QWRAisb43Y4vnWD77yX4A6l/EM3iK/pFUTvPwCnDSQCK
13888 18:33:06.403912 172.16.0.131    216.58.193.147   HTTP     2…               GET /uar3/?OXtd9L=cFNTMFX8k4Sl&WN68=zeqtNgG9qaggg50hsBYKfMtRhSgr1QCYqFreEa/f3Ny
13890 18:33:06.492086 216.58.193.147  172.16.0.131     HTTP     5…               HTTP/1.1 301 Moved Permanently
13897 18:33:11.809390 172.16.0.131    154.206.65.249   HTTP     2…               GET /uar3/?WN68=PHhlvnmfOw59M/FZc+cjVGc9E+FfGp0TERrr3iXeT60uB9RQS+IvYoe8rfvBvY2
13950 18:33:17.190425 172.16.0.131    104.21.89.147    HTTP     2…               GET /uar3/?OXtd9L=cFNTMFX8k4Sl&WN68=OeYQqIU9fkIjHq0iRTjpk5h8JewsY/FAEEGplD1myE1
14006 18:33:30.035812 172.16.0.131    3.130.253.23     HTTP     2…               GET /uar3/?Rx=3fqpvFxpqlVpsJr0&WN68=rUb9fjakxYTFD8z67QPd/z9ZU79kig+C682K4H/u+g+
14008 18:33:30.115362 3.130.253.23    172.16.0.131     HTTP     2…               HTTP/1.1 302 Found
14191 18:33:35.309236 172.16.0.131    173.231.37.114   HTTP     2…               GET /uar3/?WN68=9m2BuYjy2P5QVnF55yTJRV/9LhiAAt/MT+Kbm8QIT+MHAFzaldcGnNZ3pWSYBbz
14410 18:33:55.591697 172.16.0.131    184.168.99.26    HTTP     2…               GET /uar3/?WN68=Rxa7roEa5tNLCQbNrgjP26TJ8hY95dmiy0vlywMBfbVg3NfijesNxg9KSG3foxj
14454 18:33:56.468559 184.168.99.26   172.16.0.131     HTTP     1…               HTTP/1.1 404 Not Found  (text/html)
14544 18:34:29.730465 172.16.0.131    198.185.159.144  HTTP     2…               GET /uar3/?OXtd9L=cFNTMFX8k4Sl&WN68=HgytsQeQm5k0JsiX3+xHTPqNeoAsZVOwel7pX1mp3pb
14556 18:34:35.037075 172.16.0.131    213.186.33.5     HTTP     2…               GET /uar3/?WN68=ytaVTsUV2lhqQKL61ah7bbHTc8PUfHVAz52PWpuGTKIYDZecH7Q6UUGSzaPenE3
14558 18:34:35.206667 213.186.33.5    172.16.0.131     HTTP     5…               HTTP/1.1 302 Moved Temporarily  (text/html)
14569 18:34:40.387335 172.16.0.131    66.29.145.216    HTTP     2…               GET /uar3/?OXtd9L=cFNTMFX8k4Sl&WN68=PJmt9gv9iGU9d8hCPuUD9qFGrc2TdJl2olRt+T3RNFH
14571 18:34:40.578575 66.29.145.216   172.16.0.131     HTTP     5…               HTTP/1.1 404 Not Found  (text/html)
14581 18:34:44.519695 172.16.0.149    23.47.49.133     HTTP     3…               GET /msdownload/update/v3/static/trustedr/en/pinrulesstl.cab?9237f8d32c352e18 H
14583 18:34:44.584738 23.47.49.133    172.16.0.149     HTTP     3…               HTTP/1.1 304 Not Modified
18672 18:48:56.889489 135.148.121.246 172.16.0.149     TCP      1…               8080 → 49800 [PSH, ACK] Seq=102069 Ack=970 Win=64240 Len=1376
20926 18:58:15.748359 172.16.0.149    23.47.49.165     HTTP     3…               GET /msdownload/update/v3/static/trustedr/en/disallowedcertstl.cab?8dd2677e96c7
20928 18:58:15.802306 23.47.49.165    172.16.0.149     HTTP     3…               HTTP/1.1 304 Not Modified
20933 18:58:15.860343 172.16.0.149    23.47.49.165     HTTP     3…               GET /msdownload/update/v3/static/trustedr/en/disallowedcertstl.cab?1b0aa5c7ea92
20935 18:58:15.908320 23.47.49.165    172.16.0.149     HTTP     3…               HTTP/1.1 304 Not Modified
21236 18:59:34.813070 139.196.72.155  172.16.0.170     TCP      1…               8080 → 54143 [PSH, ACK] Seq=87763 Ack=891 Win=64240 Len=1292
22275 19:00:54.740957 185.184.25.78   172.16.0.170     TCP      1…               8080 → 54147 [PSH, ACK] Seq=434750 Ack=705 Win=64240 Len=1376
25171 19:06:50.919251 135.148.121.246 172.16.0.149     TCP      1…               8080 → 49837 [PSH, ACK] Seq=315349 Ack=898 Win=64240 Len=1376
26008 19:06:51.654088 135.148.121.246 172.16.0.149     TCP      1…               8080 → 49837 [PSH, ACK] Seq=1033621 Ack=898 Win=64240 Len=1376
26573 19:06:52.385109 135.148.121.246 172.16.0.149     TCP      1…               8080 → 49837 [PSH, ACK] Seq=1454677 Ack=898 Win=64240 Len=1376
26602 19:06:53.702891 135.148.121.246 172.16.0.149     TCP      1…               8080 → 49838 [ACK] Seq=1570 Ack=889 Win=64240 Len=1460
26968 19:06:54.326126 135.148.121.246 172.16.0.149     TCP      1…               8080 → 49838 [PSH, ACK] Seq=294574 Ack=889 Win=64240 Len=1376
```

Well, it is surprising that whenever we filter for http, we get a lot of GET Methods. Now we are going to filter for the GET:



```
http.request.method == "GET"
No.   Time              Source        Destination      Protocol Ler CNameString
  379 18:22:27.879039 172.16.0.131  23.218.224.154   HTTP     1…
 3995 18:24:34.594654 172.16.0.149  64.34.171.228    HTTP     2…
 9574 18:29:19.507652 172.16.0.131  156.96.154.210   HTTP     1…
10993 18:30:38.569123 172.16.0.131  23.227.38.74     HTTP     2…
11006 18:30:43.839315 172.16.0.131  72.167.191.69    HTTP     2…
11022 18:30:49.303447 172.16.0.131  194.9.94.85      HTTP     2…
11036 18:30:54.624783 172.16.0.131  198.54.117.210   HTTP     2…
11053 18:31:00.837127 172.16.0.131  120.55.51.124    HTTP     2…
11090 18:31:06.991345 172.16.0.131  216.58.193.147   HTTP     2…
11298 18:31:12.258145 172.16.0.131  198.185.159.144  HTTP     2…
11668 18:31:17.787860 172.16.0.131  213.186.33.5     HTTP     2…
13009 18:31:28.063998 172.16.0.131  23.227.38.74     HTTP     2…
13023 18:31:33.261231 172.16.0.131  216.58.193.147   HTTP     2…
13038 18:31:38.930684 172.16.0.131  154.206.65.249   HTTP     2…
13049 18:31:44.373406 172.16.0.131  104.21.89.147    HTTP     2…
13074 18:32:02.373442 172.16.0.131  66.235.200.112   HTTP     2…
13098 18:32:07.683441 172.16.0.131  198.54.117.215   HTTP     2…
13158 18:32:29.968699 172.16.0.131  209.17.116.163   HTTP     2…
```

We notice there is a particular one on number 3995. If we download them and check the file type, we found that is an executable.

Now lets compare, the benign traffic is pretty normal, normal DNS interactions and everything. nothing much to see, sometimes firefox block the connections. whereas in the malicious we see more weird things and connections.

Chapter 3

YARA

3.1 Basic YARA Rule Creation and Execution

3.1.1 Objective

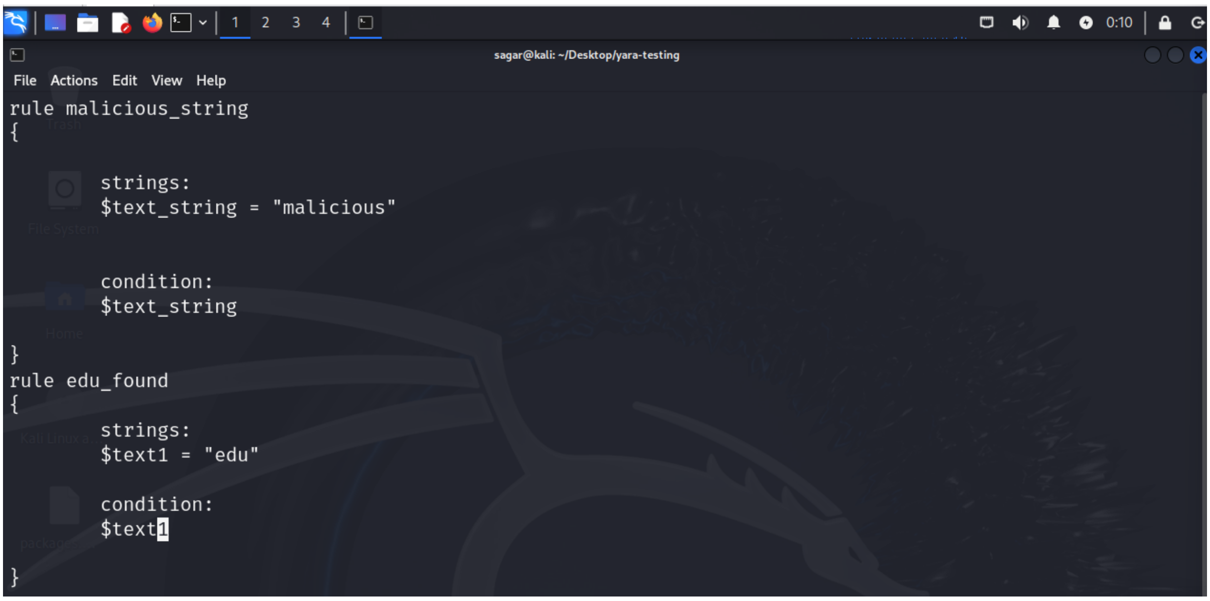
Learn how to create and execute a basic YARA rule to identify specific strings in a file.

3.1.2 Steps

1. **Create a Simple YARA Rule:**
   - Open a text editor and write a YARA rule to match a specific string.
   - Save the file as SimpleRule.yar.
2. **Prepare a Test File:**
   - Create a text file containing the string ”malicious string” along with other random text.
   - Save this file as test file.txt.
3. **Run the YARA Rule:**
   - Open a command-line terminal and navigate to the directory containing your rule and test file.
   - Execute the YARA rule using the command: yara SimpleRule.yar test file.txt

3.1.3 Results

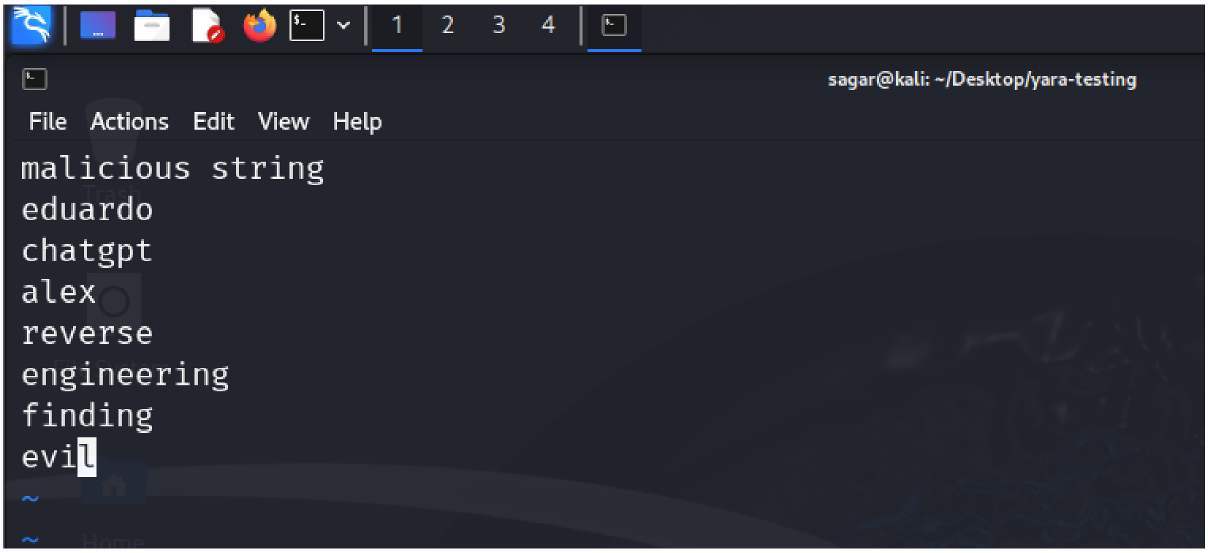
For the purposes of this exercise, I created 2 rules, which are going to be prompted if we encounter them in the example ***file.txt***. This is the file **testing.yar** that contains the two rules:



```
rule malicious_string
{

        strings:
        $text_string = "malicious"


        condition:
        $text_string
}
rule edu_found
{
        strings:
        $text1 = "edu"

        condition:
        $text1
}
```

testing.yar

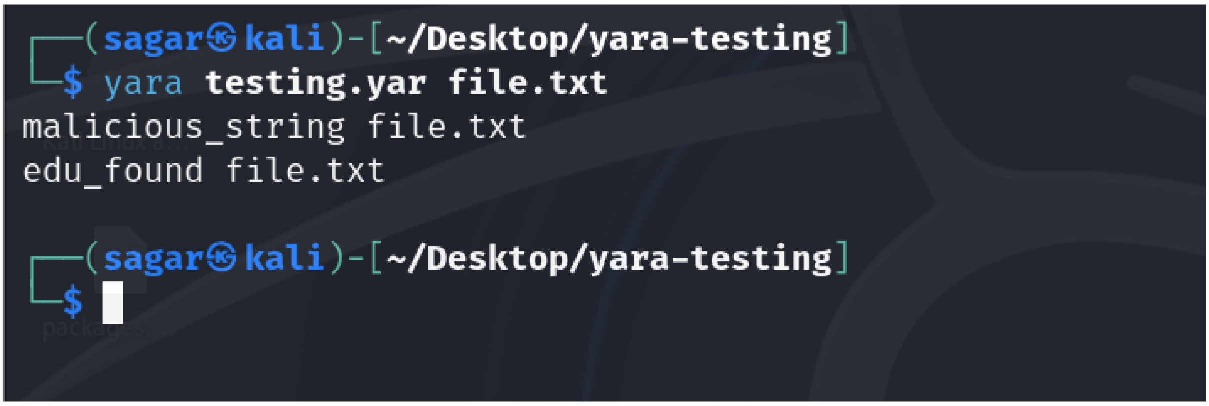
This is **file.txt**:



```
malicious string
eduardo
chatgpt
alex
reverse
engineering
finding
evil
~
~
```

file.txt

This is the the output of the program:



```
┌──(sagar㉿kali)-[~/Desktop/yara-testing]
└─$ yara testing.yar file.txt
malicious_string file.txt
edu_found file.txt

┌──(sagar㉿kali)-[~/Desktop/yara-testing]
└─$ 
```

The explanation behind is that if it encounters the string that we declared on the rules, it will tell tell you the rules and the file that is triggered on.

3.2 Complex YARA Rules w/ Boolean Logic

3.2.1 Objective

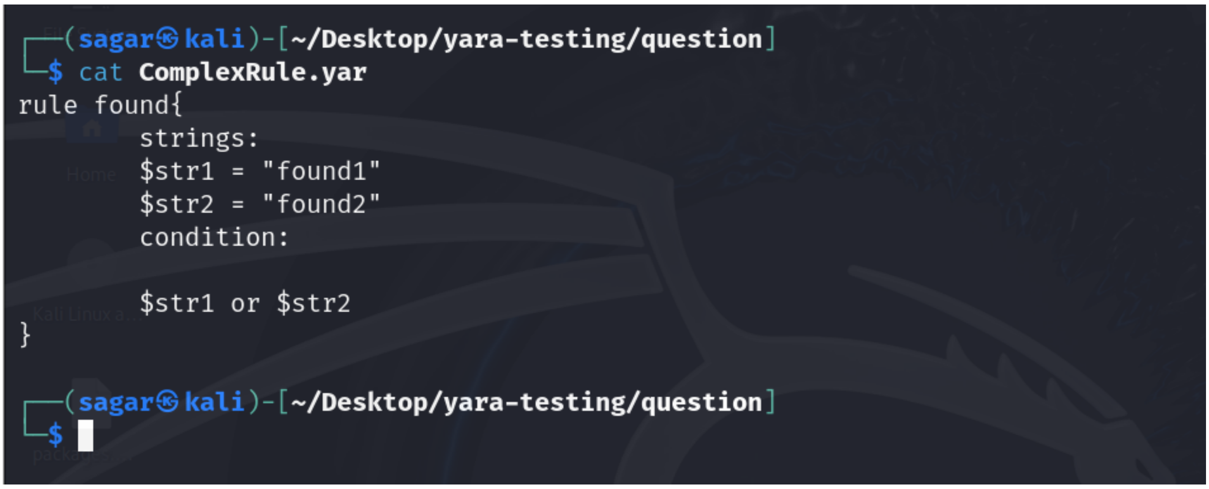
Create a more complex YARA rule using boolean logic to identify files containing multiple specific strings.

3.2.2 Steps

1. **Define the Rule:**
   - Create a YARA rule that looks for two strings using boolean logic.
   - Save the rule as ComplexRule.yar.
2. **Create Test Files:**
   - Prepare two text files: one containing both strings and another containing only one of these strings.
   - Name them test file1.txt and test file2.txt.
3. **Execute the Rule:**
   - Run the YARA rule against both files using the command: yara ComplexRule.yar test file1.txt.
   - Observe the output for each file.

3.2.3 Results

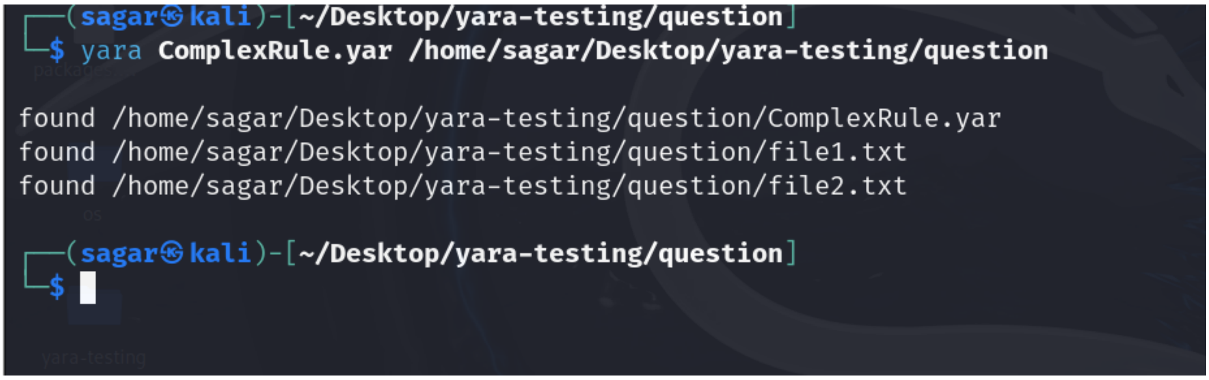
For this one I created the rule found with two strings that will prompt found if in one of the files or the other one has the strings ”found1” or ”found2”.



```
┌──(sagar㉿kali)-[~/Desktop/yara-testing/question]
└─$ cat ComplexRule.yar
rule found{
        strings:
        $str1 = "found1"
        $str2 = "found2"
        condition:

        $str1 or $str2
}

┌──(sagar㉿kali)-[~/Desktop/yara-testing/question]
└─$ 
```

ComplexRule.yar

This is the output of the rule, so it can read all the files we created, we have to pass all the directory so that it can apply the rules to the files.



```
┌──(sagar㉿kali)-[~/Desktop/yara-testing/question]
└─$ yara ComplexRule.yar /home/sagar/Desktop/yara-testing/question

found /home/sagar/Desktop/yara-testing/question/ComplexRule.yar
found /home/sagar/Desktop/yara-testing/question/file1.txt
found /home/sagar/Desktop/yara-testing/question/file2.txt

┌──(sagar㉿kali)-[~/Desktop/yara-testing/question]
└─$ ▌
```

3.3 Scanning a Directory for Malicious Files

3.3.1 Objective

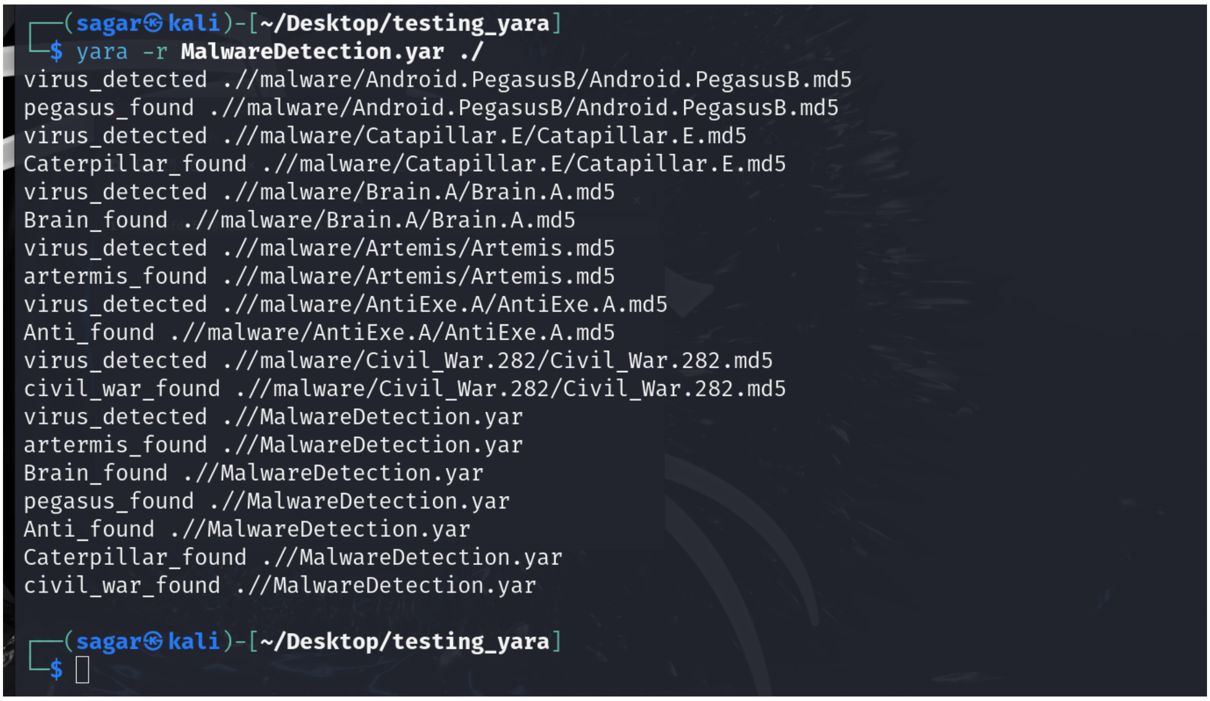
Use YARA to scan a directory containing multiple files and identify those that match specific patterns.

3.3.2 Steps

1. Prepare a Set of Files:
   - Collect a set of files from TheZoo GitHub Repository and place them in a directory named any preferred name.
   - Alternative: VirusShare (registration required).
2. **Write a YARA Rule for Scanning:**
   - Create at least a YARA rule that looks for common malware signatures.
   - Save this as MalwareDetection.yar.
3. **Scan the Directory:**
   - Use YARA to scan all files in the directory with the command: yara -r MalwareDetection.yar any preferred name/.
   - Review the output to see which files match the rule.

3.3.3 Results

The results for this one are quite interesting. So using the md5 file I created a rule that will detect in the case that is present inside the directory:



```
┌──(sagar㉿kali)-[~/Desktop/testing_yara]
└─$ yara -r MalwareDetection.yar ./
virus_detected .//malware/Android.PegasusB/Android.PegasusB.md5
pegasus_found .//malware/Android.PegasusB/Android.PegasusB.md5
virus_detected .//malware/Catapillar.E/Catapillar.E.md5
Caterpillar_found .//malware/Catapillar.E/Catapillar.E.md5
virus_detected .//malware/Brain.A/Brain.A.md5
Brain_found .//malware/Brain.A/Brain.A.md5
virus_detected .//malware/Artemis/Artemis.md5
artermis_found .//malware/Artemis/Artemis.md5
virus_detected .//malware/AntiExe.A/AntiExe.A.md5
Anti_found .//malware/AntiExe.A/AntiExe.A.md5
virus_detected .//malware/Civil_War.282/Civil_War.282.md5
civil_war_found .//malware/Civil_War.282/Civil_War.282.md5
virus_detected .//MalwareDetection.yar
artermis_found .//MalwareDetection.yar
Brain_found .//MalwareDetection.yar
pegasus_found .//MalwareDetection.yar
Anti_found .//MalwareDetection.yar
Caterpillar_found .//MalwareDetection.yar
civil_war_found .//MalwareDetection.yar

┌──(sagar㉿kali)-[~/Desktop/testing_yara]
└─$ 
```

if we run it in the whole zoo, it wont have much effect, only for those files that are inside the rules. now I created a couple rules to check if it contains that.

Chapter 4

Ghidra

4.1 Import and Basic Analysis of a PE File

4.1.1 Objective

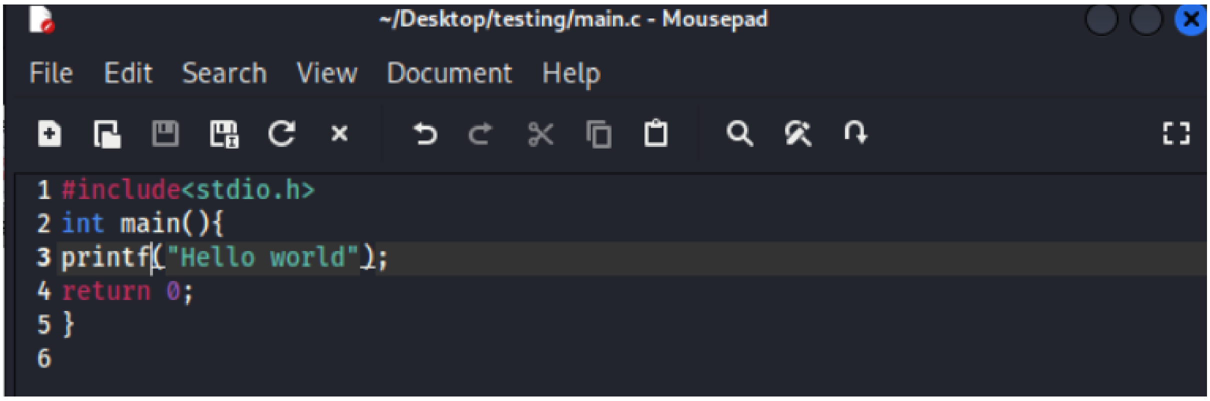
Familiarize yourself with Ghidra’s interface by importing a Portable Executable (PE) file and performing basic disassembly and decompilation analysis.

4.1.2 Steps

- **Obtain a PE file:** Download a benign PE file from TheZoo Project on GitHub or Contagio Malware Dump or create a simple ”Hello World” program in C and compile it.
- **Start Ghidra and Create a New Project.**
- **Import the PE File:** Click ‘File Import File‘ and select your PE file.
  - Ghidra will analyze the file upon importing.
- **Explore the Disassembly View:** Double-click the imported file in the Project Manager to open it in the CodeBrowser.
- **Use the Decompiler:** Click on a function in the disassembly view and switch to the decompiler window to see the C-like pseudocode.
- **Identify Functions and Variables:** Explore the functions and variables identified by Ghidra. You can rename the functions and variables.

4.1.3 Results

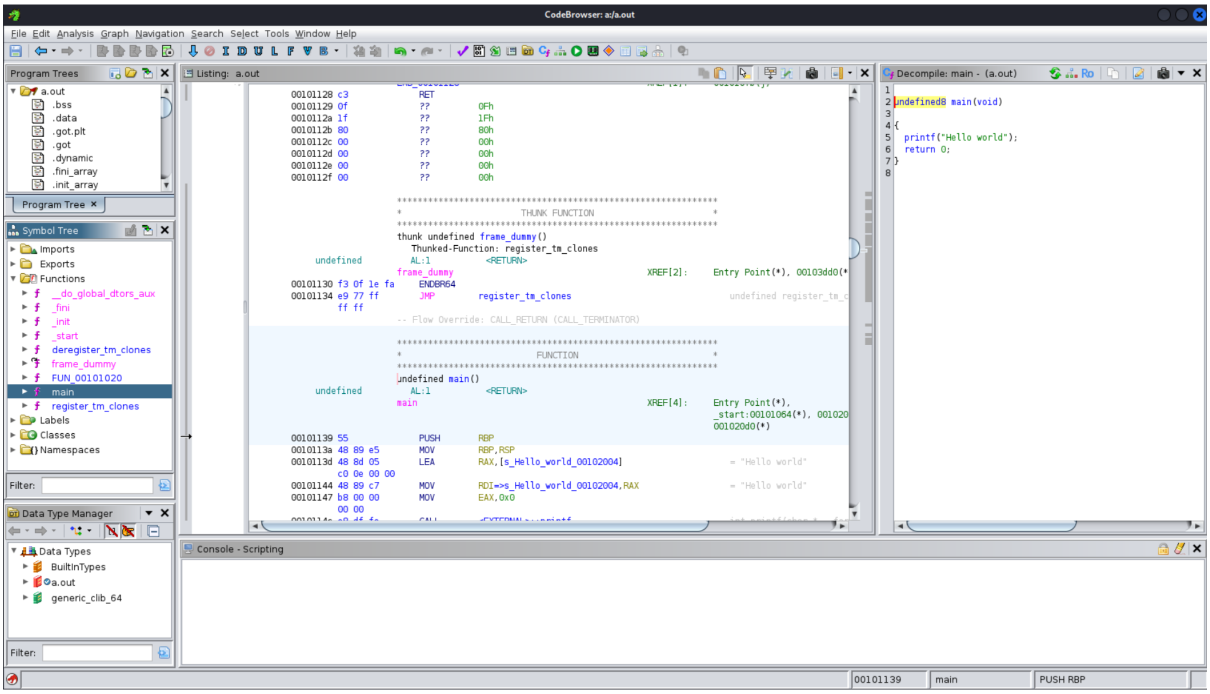
For the testing purposes, I created a **main.c** file. This file is going to be oppened in Guidra.


~/Desktop/testing/main.c - Mousepad
File Edit Search View Document Help

```
#include<stdio.h>
int main(){
printf("Hello world");
return 0;
}
```

main.c

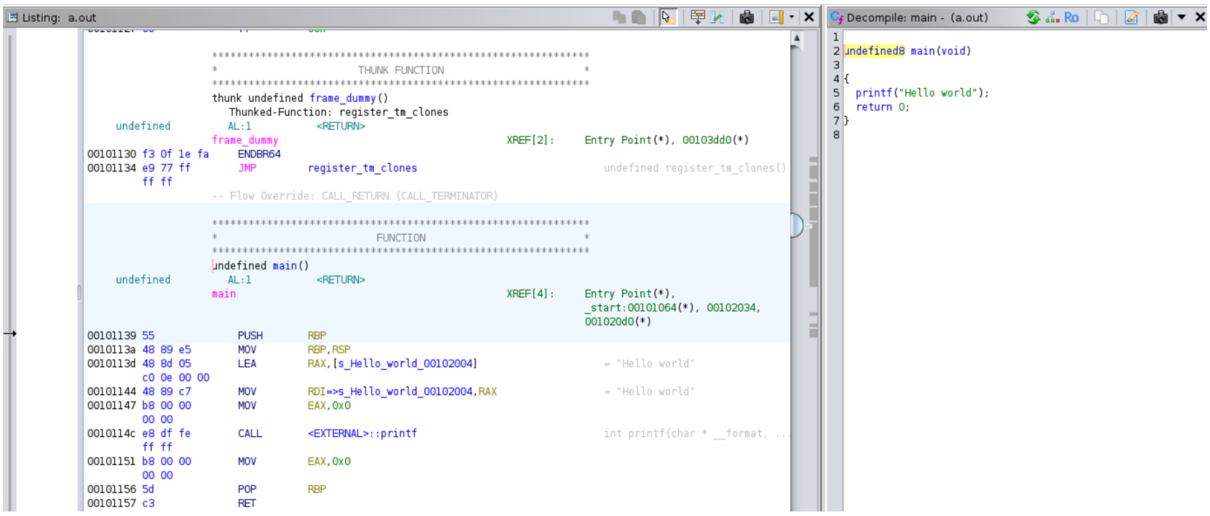
If we open the file with ghidra, which by the way, we have to open the executable. we are going to get something like this.



```
00101128 c3             RET
00101129 0f             ??          0Fh
0010112a 1f             ??          1Fh
0010112b 80             ??          80h
0010112c 00             ??          00h
0010112d 00             ??          00h
0010112e 00             ??          00h
0010112f 00             ??          00h

                     **************************************************************
                     *                       THUNK FUNCTION                       *
                     **************************************************************
                     thunk undefined frame_dummy()
                       Thunked-Function: register_tm_clones
     undefined         AL:1          <RETURN>
                     frame_dummy                                 XREF[2]:     Entry Point(*), 00103dd0(*
00101130 f3 0f 1e fa    ENDBR64
00101134 e9 77 ff       JMP         register_tm_clones                        undefined register_tm_c
         ff ff
                     -- Flow Override: CALL_RETURN (CALL_TERMINATOR)

                     **************************************************************
                     *                          FUNCTION                          *
                     **************************************************************
                     undefined main()
     undefined         AL:1          <RETURN>
                     main                                        XREF[4]:     Entry Point(*),
                                                                              _start:00101064(*), 001020
                                                                              001020d0(*)
00101139 55             PUSH        RBP
0010113a 48 89 e5       MOV         RBP,RSP
0010113d 48 8d 05       LEA         RAX,[s_Hello_world_00102004]              = "Hello world"
         c0 0e 00 00
00101144 48 89 c7       MOV         RDI=>s_Hello_world_00102004,RAX           = "Hello world"
00101147 b8 00 00       MOV         EAX,0x0
         00 00
```

```

undefined8 main(void)

{
  printf("Hello world");
  return 0;
}

```

If we focus on the right side of the previous picture, we can notice that on the left is the the disassembly version of it and the right is going to be the decompiled version of it in c.



```
                     **************************************************************
                     *                       THUNK FUNCTION                       *
                     **************************************************************
                     thunk undefined frame_dummy()
                       Thunked-Function: register_tm_clones
     undefined         AL:1          <RETURN>
                     frame_dummy                                 XREF[2]:     Entry Point(*), 00103dd0(*)
00101130 f3 0f 1e fa    ENDBR64
00101134 e9 77 ff       JMP         register_tm_clones                        undefined register_tm_clones()
         ff ff
                     -- Flow Override: CALL_RETURN (CALL_TERMINATOR)

                     **************************************************************
                     *                          FUNCTION                          *
                     **************************************************************
                     undefined main()
     undefined         AL:1          <RETURN>
                     main                                        XREF[4]:     Entry Point(*),
                                                                              _start:00101064(*), 00102034,
                                                                              001020d0(*)
00101139 55             PUSH        RBP
0010113a 48 89 e5       MOV         RBP,RSP
0010113d 48 8d 05       LEA         RAX,[s_Hello_world_00102004]              = "Hello world"
         c0 0e 00 00
00101144 48 89 c7       MOV         RDI=>s_Hello_world_00102004,RAX           = "Hello world"
00101147 b8 00 00       MOV         EAX,0x0
         00 00
0010114c e8 df fe       CALL        <EXTERNAL>::printf                        int printf(char * __format, ..
         ff ff
00101151 b8 00 00       MOV         EAX,0x0
         00 00
00101156 5d             POP         RBP
00101157 c3             RET
```

```

undefined8 main(void)

{
  printf("Hello world");
  return 0;
}

```

4.2 Function Identification and Renaming

4.2.1 Objective

Improve readability and understanding of a binary by identifying and renaming key functions and variables.

4.2.2 Steps

- **Continue from Exercise 1 or import a New Binary**
- **Identify Key Function:**

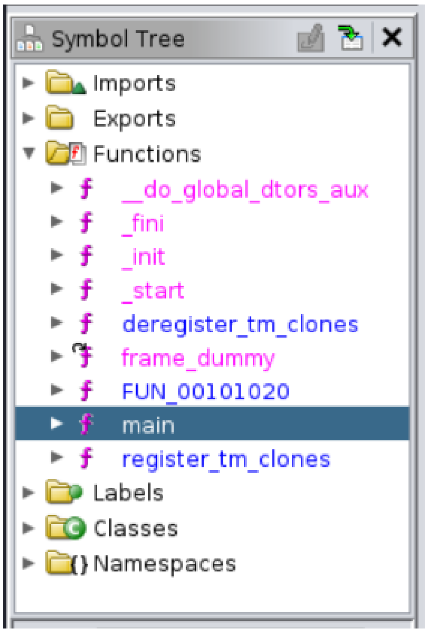
  - Use the symbol tree to explore all functions identified by Ghidra.
  - Double-click on important functions like ‘main‘, ‘WinMain‘, or others that handle significant tasks.

- **Rename Functions and Variables:**
  - Click on a function or variable, press ‘L‘ or right-click, and choose ‘Rename‘.
  - Provide meaningful names based on their role in the program.

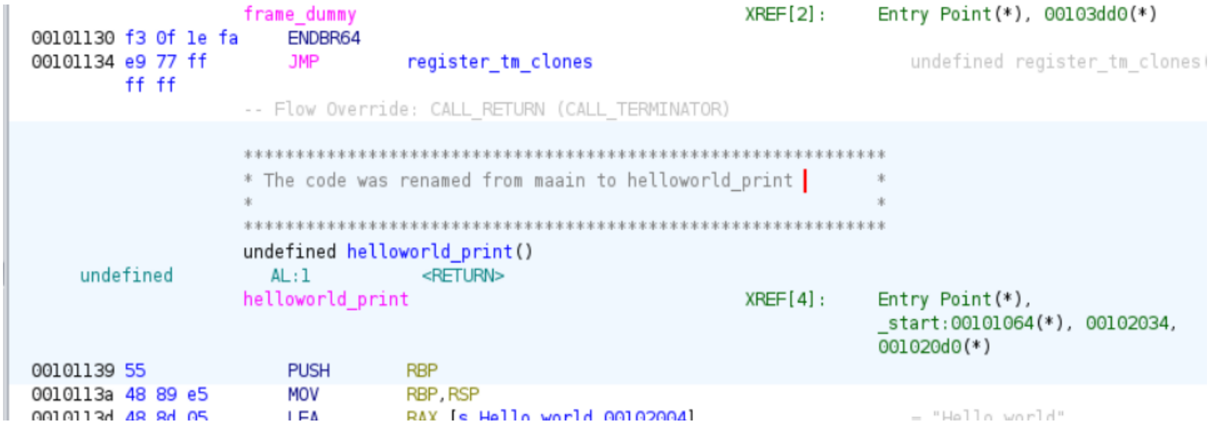
- **Document Your Changes:**
  - Use Ghidra’s ”Comments” feature to annotate functions and variables with descriptions.
  - Maintain a simple document or use Ghidra’s built-in features to record your changes.

4.2.3 Results

This is the Symbol tree and it displays all the functions that the executable had:



I used plate comment to leave that meaningful comment for anyone that uses the executable again.



```
                         frame_dummy                                         XREF[2]:     Entry Point(*), 00103dd0(*)
 00101130 f3 0f 1e fa        ENDBR64
 00101134 e9 77 ff           JMP          register_tm_clones                              undefined register_tm_clones
          ff ff
                         -- Flow Override: CALL_RETURN (CALL_TERMINATOR)

                         ****************************************************************
                         * The code was renamed from maain to helloworld_print          *
                         *                                                              *
                         ****************************************************************
                         undefined helloworld_print()
     undefined              AL:1           <RETURN>
                         helloworld_print                                    XREF[4]:     Entry Point(*),
                                                                                          _start:00101064(*), 00102034,
                                                                                          001020d0(*)
 00101139 55                 PUSH         RBP
 0010113a 48 89 e5           MOV          RBP,RSP
 0010113d 48 8d 05           LEA          RAX,[s_Hello_world_00102004]                    = "Hello world"
```

4.3 Patching a Binary

4.3.1 Objective

Learn how to modify a binary's behavior by creating a simple patch in Ghidra.

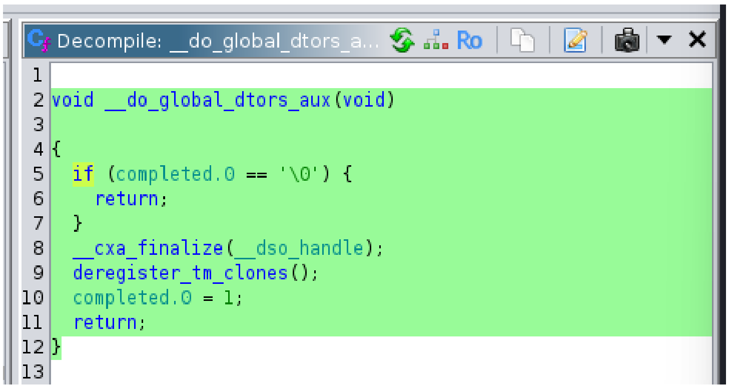
4.3.2 Steps

- **Open the Binary in Ghidra:** Continue using the PE file from previous exercises.
- **Identify a Conditional Check:**
  - Look for a simple conditional statement in the decompiled code (e.g., an 'if' statement).
  - For example, find a condition that checks if a user is an admin ('if (isAdmin)').
- **Patch the Binary:**
  - Modify the condition so that it always evaluates to 'true' or 'false'.
  - In the disassembly view, right-click on the relevant instruction and select 'Patch Instruction'.
  - Change the instruction to alter the program's flow (e.g., force the condition to always be true).
- **Save and Test the Patch:**
  - Save the patched binary and re-run the analysis in Ghidra.
  - Optionally, export the patched binary and run it in a controlled environment to see the effect.

4.3.3 Results

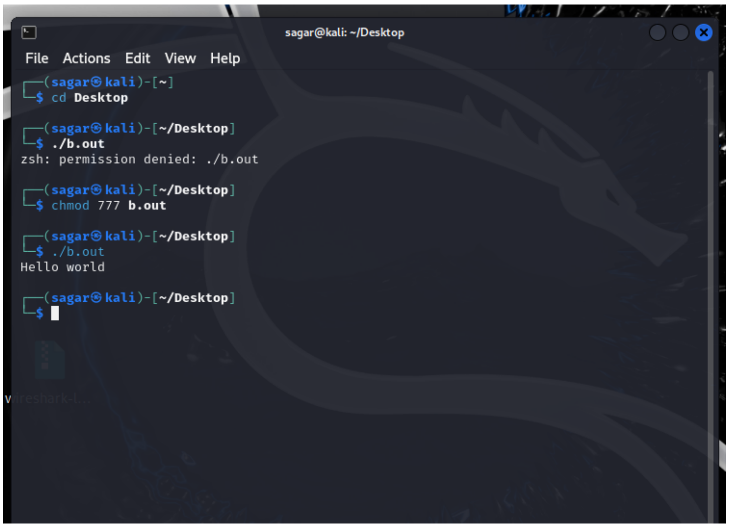
First i started creating another program that had a simple if-statement, the problem it had is that whenever you decompile it, if its unreachable code it will ignore it. So on the whole section of functions I started searching for an if-statement.

the file depicted below had it and it was just a matter of changing the jump-if-not zero (JNZ) to jump if zero (JZ) and the result is the following:



```

void __do_global_dtors_aux(void)

{
  if (completed.0 == '\0') {
    return;
  }
  __cxa_finalize(__dso_handle);
  deregister_tm_clones();
  completed.0 = 1;
  return;
}

```

Of course if we run it again, it will work, with no problems:



```
┌──(sagar㉿kali)-[~]
└─$ cd Desktop

┌──(sagar㉿kali)-[~/Desktop]
└─$ ./b.out
zsh: permission denied: ./b.out

┌──(sagar㉿kali)-[~/Desktop]
└─$ chmod 777 b.out

┌──(sagar㉿kali)-[~/Desktop]
└─$ ./b.out
Hello world

┌──(sagar㉿kali)-[~/Desktop]
└─$ █
```

There are benefits on this, because we can change the requirement of being administrator in an application and we could possibly gain access to it.

4.4 Scripting with Ghidra

4.4.1 Objective

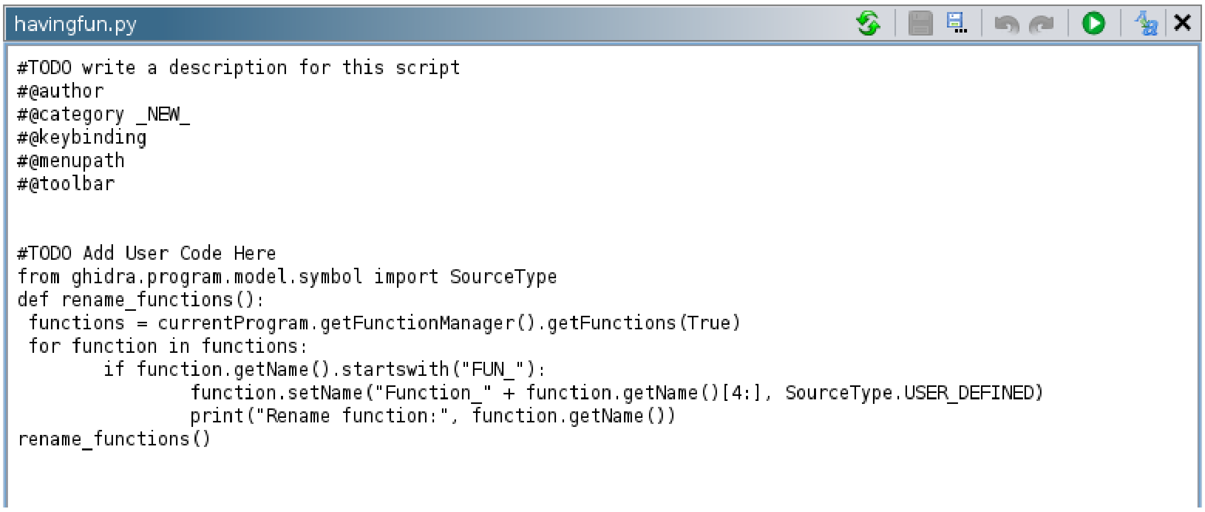
Automate repetitive tasks or create custom analysis tools using Ghidra's scripting capabilities.

4.4.2 Steps

- **Run the Script:**
  - Execute the script in Ghidra by selecting it in the Script Manager and clicking ‘Run‘.
  - Observe how the functions are renamed according to your script.
- **Verify and Document Results:**
  - Check the function names in the CodeBrowser to ensure the script worked as expected.
  - Document the script and its effects for future reference.

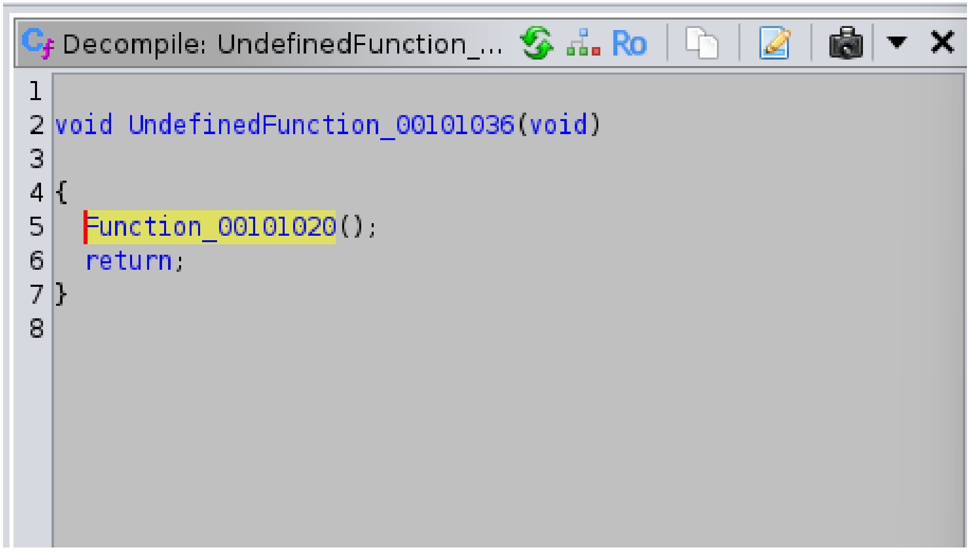
4.4.3 Results

This is the Script on python:



```
#TODO write a description for this script
#@author
#@category _NEW_
#@keybinding
#@menupath
#@toolbar


#TODO Add User Code Here
from ghidra.program.model.symbol import SourceType
def rename_functions():
 functions = currentProgram.getFunctionManager().getFunctions(True)
 for function in functions:
        if function.getName().startswith("FUN_"):
                function.setName("Function_" + function.getName()[4:], SourceType.USER_DEFINED)
                print("Rename function:", function.getName())
rename_functions()
```

This is the result of the script:



```

void UndefinedFunction_00101036(void)

{
  Function_00101020();
  return;
}

```

4.5 Cross-Referencing and Call Graph Analysis

4.5.1 Objective

Understand the relationships between functions using Ghidra's cross-reference and call graph tools.

4.5.2 Steps

- **Open the Binary:** Use the same PE file from previous exercises or import a new one.
- **Analyze Function Cross-References:**
  - Select a function in the CodeBrowser and right-click to choose 'References Show References to'.
  - This should show you all the places in the code where this function is called.
- **Generate a Call Graph:**

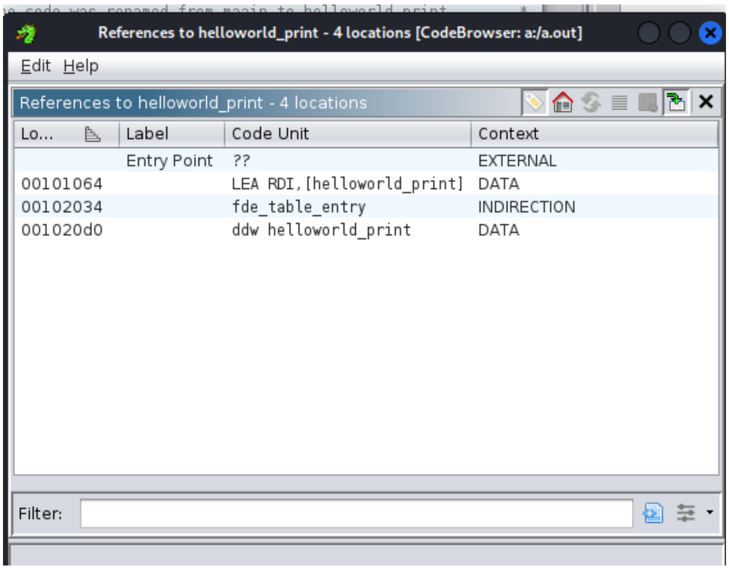
  - Right-click on a function and select ‘Graph Function Call Graph’. This generates a visual representation of how functions interact with each other, showing which functions call which.

- **Interpret the Call Graph:**
  - Analyze the call graph to understand the program’s structure. Identify key functions that are central to the program’s operation and those that might be entry points for vulnerabilities

- **Document the Findings:**
  - Create a report or presentation summarizing the call graph and cross-referencing results.
  - Highlight important functions and their relationships, noting any potential security concerns.

4.5.3 Results

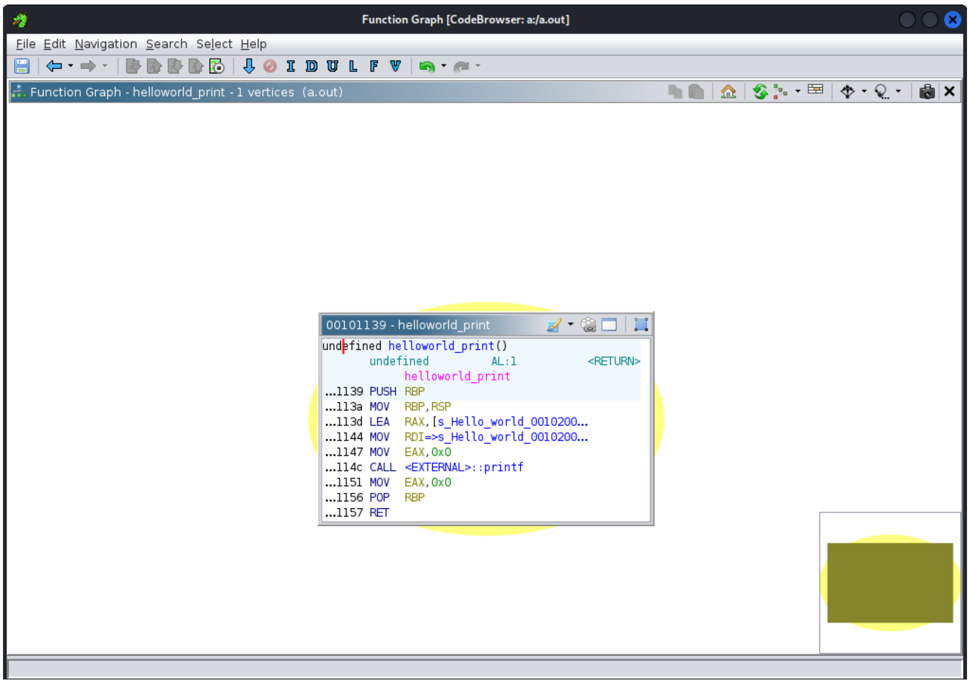
These are the cross-references to address:


References to helloworld_print - 4 locations [CodeBrowser: a:/a.out]
Edit Help
References to helloworld_print - 4 locations
Filter:

Lo...	Label	Code Unit	Context
	Entry Point	??	EXTERNAL
00101064		LEA RDI,[helloworld_print]	DATA
00102034		fde_table_entry	INDIRECTION
001020d0		ddw helloworld_print	DATA

To get to the graph view, we only have to look up in the navbar and we can see this buttons, is the one that says graph view.

This is the graph:


Function Graph [CodeBrowser: a:/a.out]
File Edit Navigation Search Select Help
Function Graph - helloworld_print - 1 vertices (a.out)
00101139 - helloworld_print

```
undefined helloworld_print()
        undefined         AL:1           <RETURN>
              helloworld_print
...1139 PUSH  RBP
...113a MOV   RBP,RSP
...113d LEA   RAX,[s_Hello_world_0010200...
...1144 MOV   RDI=>s_Hello_world_0010200...
...1147 MOV   EAX,0x0
...114c CALL  <EXTERNAL>::printf
...1151 MOV   EAX,0x0
...1156 POP   RBP
...1157 RET
```

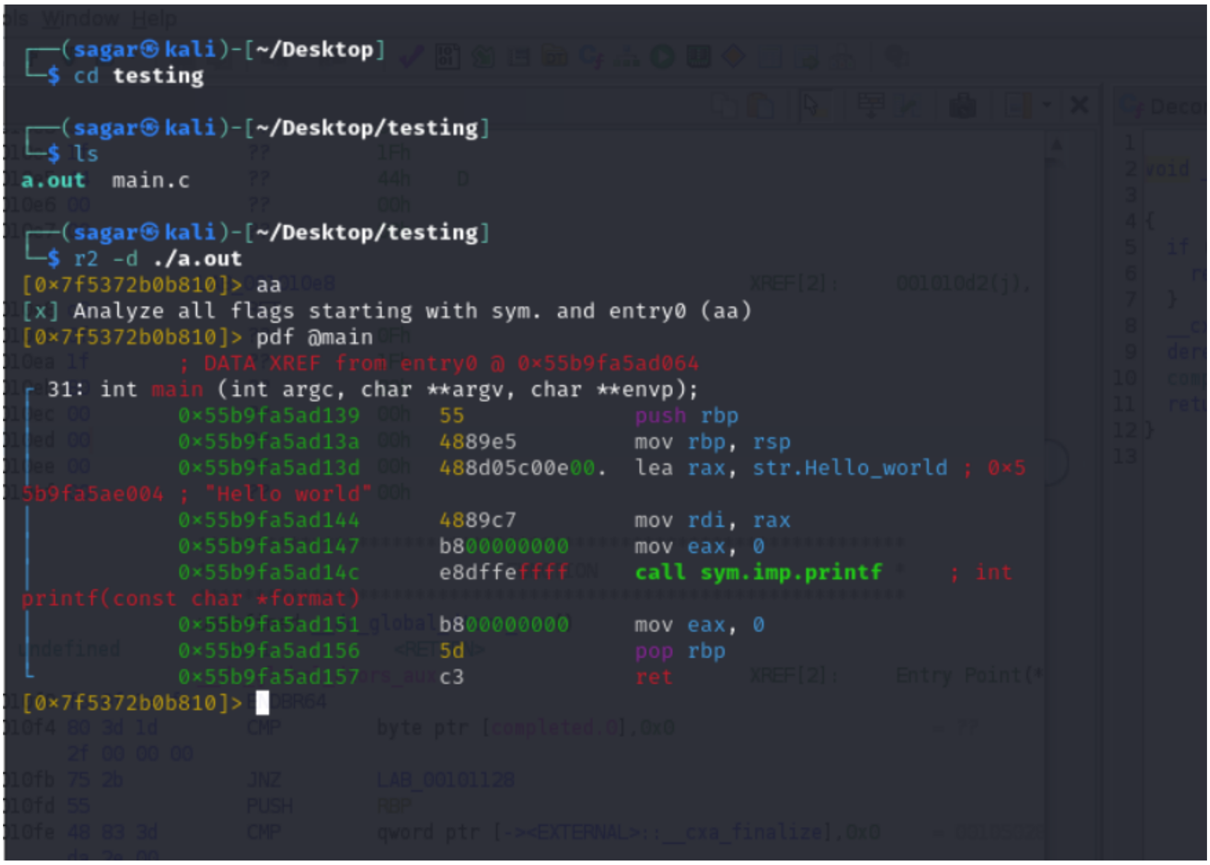
As we can observe in the images, there is only one function on the graph, since its a little helloworld in c.

Chapter 5

Radare2

5.1 Basic

For this portion of the report we did not get any challenge or task but here is an explanation of the **Radare2**. We are going to still be using the Helloworld that is on c.



```
┌──(sagar㉿kali)-[~/Desktop]
└─$ cd testing

┌──(sagar㉿kali)-[~/Desktop/testing]
└─$ ls
a.out  main.c

┌──(sagar㉿kali)-[~/Desktop/testing]
└─$ r2 -d ./a.out
[0x7f5372b0b810]> aa
[x] Analyze all flags starting with sym. and entry0 (aa)
[0x7f5372b0b810]> pdf @main
            ; DATA XREF from entry0 @ 0x55b9fa5ad064
┌ 31: int main (int argc, char **argv, char **envp);
│           0x55b9fa5ad139      55             push rbp
│           0x55b9fa5ad13a      4889e5         mov rbp, rsp
│           0x55b9fa5ad13d      488d05c00e00.  lea rax, str.Hello_world ; 0x5
5b9fa5ae004 ; "Hello world"
│           0x55b9fa5ad144      4889c7         mov rdi, rax
│           0x55b9fa5ad147      b800000000     mov eax, 0
│           0x55b9fa5ad14c      e8dffeffff     call sym.imp.printf       ; int
printf(const char *format)
│           0x55b9fa5ad151      b800000000     mov eax, 0
│           0x55b9fa5ad156      5d             pop rbp
└           0x55b9fa5ad157      c3             ret
[0x7f5372b0b810]> 
```

As we can see in the image, we use **aa** to analyze and then we are going to use pdf @main to see all the main function and we notice that there are the instructions in assembly.

I used this video to understand Radare2:

Video Helpful