## Report UNIT 3 WEEK 11.3

Olly

1. All'indirizzo 0040106E il malware effettua una chiamata di funzione alla funzione CreateProcess. Qual è il valore del parametro CommandLine che viene passato sullo stack?

```
PUSH EDX
LEA EAX,DWORD PTR SS:[EBP-58]
PUSH EAX
PUSH 0
PUSH 0
PUSH 0
PUSH 1
PUSH 0
PUSH 1
PUSH 0
PUS
```

Il valore del parametro CommandLine è "cmd", passato sullo stack utilizzando l'indirizzo Malware .00405030.

2. Inserite un breakpoint software all'indirizzo 004015A3. Qual è il valor del registro EDX?

```
Registers (FPU)

EAX 0A280105
ECX 75555666
ECX 75555666
ECX 75555666
ECX 0A280105
E
```

Il valore del registro EDX è 00000A28, in binario base2 0000 0000 0000 0000 1010 0010 1000.

3. Eseguite a questo punto uno step-into. Indicate qual è ora il valore del registro EDX motivando la risposta.

Che istruzione è stata eseguita?

```
Registers (FPU)

EAX 0A280105
ECX 75505000
ECX 7550500
ECX 7550500
ECX 7550500
ECX 75505000
ECX 75505000
ECX
```

Il valore del registro EDX è 00000000, che corrisponde a 0 in decimale. È stata eseguita l'istruzione XOR (Exclusive OR) tra il registro EDX e sé stesso, impostandolo a 0.

```
EDX 0000 0000 0000 1010 0010 1000

XOR EDX 0000 0000 0000 1010 0010 1000

0000 0000 0000 0000 0000 0000 0000 0000
```

4. Inserite un secondo breakpoint all'indirizzo di memoria 004015AF. Qual è il valore del registro ECX? Eseguite uno step-into. Qual è ora il valore di ECX? Spiegate quale istruzione è stata eseguita.

```
| Registers (MMX)
| Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers (MMX) | Registers
```

Il valore del registro ECX è 0A280105, in binario base2 0000 1010 0010 1000 0000 0001 0000 0101.

Dopo aver eseguito lo step-into il valore di ECX è 00000005, che corrisponde a 5 in decimale (0000 0000 0010 1000 0000 0001 0000 0101 binario base2). L'istruzione che è stata eseguita è AND confrontando i bit dei due operandi. Se entrambi i bit corrispondenti sono impostati a 1, il risultato sarà 1. Altrimenti, il risultato sarà 0.

```
ECX 0000 1010 0010 1000 0000 0001 0000 0101

AND 0FF 0000 0000 1111 1111

0000 0000 0010 1000 0000 0001 0000 0101
```

## 5. Spiegare a grandi linee il funzionamento del malware.

Da CFF le librerie importate dinamicamente sono KERNEL32 e WS2 32.

| szAnsi       | (n        | Functions) | Dword    | Dword          |                     |     |  |
|--------------|-----------|------------|----------|----------------|---------------------|-----|--|
| KERNEL32.dll | 38        | ı          | 00004460 | 0000000        |                     |     |  |
| W52_32.dll   | 7         |            | 000044FC | 0000000        |                     |     |  |
| OFTs         | FTs (IAT) | Hint       | Na       | me             |                     |     |  |
| Dword        | Dword     | Word       |          | Ansi           | ₩ 0040409C          | 115 |  |
| 80000073     | 80000073  | N/A        |          | dinal: 0000007 | 00404040            |     |  |
| 00004570     | 00004570  | 003D       | WS       | SASocketA      | 004040A4            |     |  |
| 80000034     | 80000034  | N/A        | Ord      | dinal: 0000003 | <u>I</u> 📆 004040A8 | 3   |  |
| 80000003     | 80000003  | N/A        |          | dinal: 0000000 | 004040              | 116 |  |
| 80000074     | 80000074  | N/A        |          | dinal: 0000007 | 004040B0            |     |  |
| 80000009     | 80000009  | N/A        |          | dinal: 0000000 | 004040B4            |     |  |
| 80000004     | 80000004  | N/A        | Ord      | dinal: 0000000 | TE 004040B4         | 4   |  |

Anche il codice sembra avere meccanismi di offuscamento/compressione per nascondere il suo comportamento e rendere più difficile l'analisi statica.

```
[000000E0 BYTES: COLLAPSED FUNCTION _strcat. PRESS KEYPAD "+" TO EXPAND]
[00000012 BYTES: COLLAPSED FUNCTION _malloc. PRESS KEYPAD "+" TO EXPAND]
[0000002C BYTES: COLLAPSED FUNCTION __nh_malloc. PRESS KEYPAD "+" TO EXPAND]
[00000199 BYTES: COLLAPSED FUNCTION __setmbcp. PRESS KEYPAD "+" TO EXPAND]
```

Nella main vengono richiamate le funzioni di **WS2\_32.dll** per la programmazione del socket e le comunicazioni di rete. Le funzioni richiamate possono far ipotizzare l'implementazione di una backdoor utilizzando **WSASocketA** per creare un socket di rete e **connect** per connettersi al server remoto e inviare o ricevere comandi e dati tramite questa connessione.

```
00 00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00
ดด
  00 00 00 00 00 00 00
                        00 00 00 00 00 00 00
ดด
  00 00 00 00 00 00 00
                        00 00 00 00 00 00 00
  00
     00
        00 00 00 00 00
                        00 00 00 00 00 00 00 00
        00 00 00 00 00
                        00 00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00
  00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00
00
  00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00
00 00 00 00 00 00 00
                        00 00 00 00 00 00 00
00 00 00 00 00 00 00
                        00 00 00 00 00 00 00
  00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00
                        00 00 00 00 00 00 00 00
```

Una grande parte del codice nell'hex-view risulta in byte 00, il contenuto del codice è stato nascosto per rendere più difficile l'analisi. Quando un file viene compresso o criptato, il suo contenuto viene trasformato in una forma che può apparire come sequenze di byte casuali o vuoti nell'hex view.

Considerando l'analisi statica, si può concludere che il funzionamento del malware implementa funzioni che recuperano informazioni sul sistema, ovvero una backdoor.

```
; DWORD __stdcall WaitEorSingleObject(MANDLE hHar
; char aProgramNameUnk[]
aProgramNameUnk db '<program name unknown>',0
                                                                 extrn WaitForSingleObject dword
                                                 ; BOOL __stdcall CreaterrocessH(Lrcs|K lpHpplicat
                 align 4
 char aGetlastactivep[]
                                                                  extrn CreateProcessA:dword
aGetlastactivep db 'GetLastActivePopup',0 ; DA; void __stdcall Sleep(DWUKD dwMilliseconds)
                                                                  extrn Sleep dword
                                                                                           ; DATA XE
                alion 10h
 char aGetactivewindo[]
                                                                                             -main+2
aGetactivewindo db 'GetActiveWindow',0 ; DATA; DWORD __stdcall GetModuleFileNameA(HMODULE hMoc
; char ProcName[]
                                                                 extrn GetModuleFileNameA:dword ;
                db 'MessageBoxA',0
ProcName
                                          ; DATA
 char LibFileName[]
hFileName db 'user32.dll',0
                                                 ; BOOL __stdcall GetStringTypeA(LCID Locale,DWORK
                                                                 extrn GetStringTypeA:dword
LihFileName
; int __stdcall LCMapS<u>tringW(LCID L</u>ocale,DWORD ; LPSTR GetCommandLineA(void)
                extrn LCMapStringW dword ; DATA
                                                                    extrn GetCommandLineA dword ;
                                              _cr ; DWORD GetVersion(voi<mark>d)</mark>
; int __stdcall LCMapStringA(LCID Locale,DWORD
                                                                    extrn GetVersion:dword
                extrn LCMapStringA dword ; DATA
                                                                                               ; Get
                                                                                                 and
; int __stdcall MultiButeToWideChar(UINT_CodePa
                extrn MultiByteToWideChar dword; void __stdcall ExitProcess(UINT uExitCode)
                                                                    extrn ExitProcess:dword ; DAT
; HMODULE __stdcall LoadLibraryA(LPCSTR ipLibFi ; BOOL __stdcall TerminateProcess(HANDLE hPro
extrn coadlibraryA:dword; DATA extrn reminateP; FARPROC stdcall GetProcAddress(HMODULE hMod; HANDLE GetCurrentProcess(void)
                                                                    extrn TerminateProcess:dword
                extrn GetProcAddress dword
                                                                    extrn GetCurrentProcess:dword
```