



EPICODE

Anatoliy Prsyazhnyuk
Antonio De Cesare
Alessandro Bossi
Rossella Amore

Claudio La Torre
Riccardo Lupieri
Riccardo Di Pasquale
Pietro Laera
Davide Bassolino



Giorno 1:

Con riferimento al file eseguibile Malware_Build_Week_U3, rispondere ai seguenti quesiti utilizzando i tool e le tecniche apprese nelle lezioni teoriche:

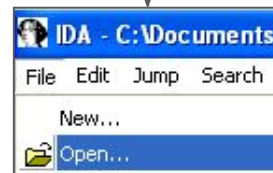
Nel presente report, verrà fornita un'analisi dettagliata del codice assembly del file malware dell'Unit 3. L'obiettivo principale è condurre un'analisi statica utilizzando l'ambiente di reverse engineering IDA Pro. Durante l'analisi, verranno esaminati passo dopo passo i vari passaggi eseguiti per comprendere il funzionamento del malware.

```
; int __cdecl main(int argc,const char **argv,const char *envp)
_main proc near

hModule= dword ptr -11Ch
Data= byte ptr -118h
var_8= dword ptr -8
var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h
```



Malware_Build_Week_U3



Functions



Function name

sub_401000

sub_401080

_main



- Quanti parametri sono passati alla funzione Main()?

```
.text:004011D0 ; !!!!!!!!!!!!!!! S U B R O U T I N E !!!!!!!!!!!!!!!
.text:004011D0
.text:004011D0 ; Attributes: bp-based frame
.text:004011D0
.text:004011D0 ; int __cdecl main(int argc,const char **argv,const char *envp)
.text:004011D0 _main      proc near          ; CODE XREF: start+AF↓p
.text:004011D0
.text:004011D0 hModule      = dword ptr -11Ch
.text:004011D0 Data        = byte ptr -118h
.text:004011D0 var_8        = dword ptr -8
.text:004011D0 var_4        = dword ptr -4
.text:004011D0 argc         = dword ptr  8
.text:004011D0 argv         = dword ptr  0Ch
.text:004011D0 envp         = dword ptr  10h
```

La riga di codice definisce la funzione **main** come punto di ingresso del programma assembly. La funzione main accetta tre parametri: **argc**, **argv**, e **envp**, che verranno utilizzati per gestire gli argomenti passati al programma e le variabili di ambiente durante l'esecuzione.

- Quante variabili sono dichiarate all'interno della funzione Main()?

```
.text:004011D0 ; !!!!!!!!!!!!!!! S U B R O U T I N E !!!!!!!!!!!!!!!
.text:004011D0
.text:004011D0 ; Attributes: bp-based frame
.text:004011D0
.text:004011D0 ; int __cdecl main(int argc,const char **argv,const char *envp)
.text:004011D0 _main      proc near          ; CODE XREF: start+AF↓p
.text:004011D0
.text:004011D0 hModule      = dword ptr -11Ch
.text:004011D0 Data        = byte ptr -118h
.text:004011D0 var_8        = dword ptr -8
.text:004011D0 var_4        = dword ptr -4
.text:004011D0 argc         = dword ptr  8
.text:004011D0 argv         = dword ptr  0Ch
.text:004011D0 envp         = dword ptr  10h
```

- Nella funzione "**main**" vengono dichiarate **quattro variabili locali**.
- hModule**: Variabile locale di tipo dword (32 bit) che viene dichiarata all'indirizzo di memoria -11Ch rispetto all'EBP (Extended Base Pointer).
 - Data**: Variabile locale di tipo byte (8 bit) che viene dichiarata a indirizzo di memoria -118h rispetto all'EBP.
 - var_8**: Variabile locale di tipo dword (32 bit) che viene dichiarata a indirizzo di memoria -8 rispetto all'EBP.
 - var_4**: Variabile locale di tipo dword (32 bit) che viene dichiarata a indirizzo di memoria -4 rispetto all'EBP.

- Quali sezioni sono presenti all'interno del file eseguibile? Descrivete brevemente almeno 2 di quelle identificate



Malware_Build_Week_U3.exe									
Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations ...	Linenumber...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	00005646	00001000	00006000	00001000	00000000	00000000	0000	0000	60000020
.rdata	000009AE	00007000	00001000	00007000	00000000	00000000	0000	0000	40000040
.data	00003EA8	00008000	00003000	00008000	00000000	00000000	0000	0000	C0000040
.rsrc	00001A70	0000C000	00002000	0000B000	00000000	00000000	0000	0000	40000040

.text



Contiene il codice eseguibile del malware, comprese le istruzioni e le costanti utilizzate. Qui si trova la logica principale del malware.

.data



Contiene i dati inizializzati durante l'esecuzione del malware, come **variabili globali** o costanti utilizzate per conservare informazioni durante l'esecuzione.

.rdata



Contiene **dati di sola lettura (read-only)**, come costanti o stringhe di testo che non possono essere modificati durante l'esecuzione del malware.

.rsrc



Contiene le **risorse del malware**, come immagini, icone o suoni, che vengono utilizzate per scopi grafici o comunicativi all'interno del malware.



- Quali librerie importa il Malware? Per ognuna delle librerie importate, fate delle ipotesi sulla base della sola analisi statica delle funzionalità che il Malware potrebbe implementare. Utilizzate le funzioni che sono richiamate all'interno delle librerie per supportare le vostre ipotesi.

Malware_Build_Week_U3.exe						
Module Name	Imports	OFTs	TimeStamp	ForwarderChain	Name RVA	FTs (IAT)
	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	51	00007534	00000000	00000000	0000769E	0000700C
ADVAPI32.dll	2	00007528	00000000	00000000	000076D0	00007000

Alcune funzioni possibilmente utilizzabili a scopo malevolo:

LoadResource: Carica una risorsa da un modulo. Potrebbe essere utilizzata per eseguire un attacco di tipo DLL, iniettando codice malevolo all'interno di un processo esistente.

VirtualAlloc: Alloca memoria virtuale all'interno di un processo. Potrebbe essere utilizzata per eseguire un attacco di tipo buffer overflow, sovrascrivendo aree di memoria adiacenti con codice malevolo.

GetCommandLineA: Restituisce la riga di comando del processo corrente. Potrebbe essere utilizzata per ottenere informazioni sensibili sui parametri passati al processo, come password o altre informazioni riservate.

WriteFile: Scrive dei dati in un file. Potrebbe essere utilizzata per sovrascrivere file importanti con dati dannosi o indesiderati, compromettendo l'integrità dei file o il funzionamento del sistema.

FindResourceA: Cerca risorse in un modulo. Potrebbe essere utilizzata per individuare risorse malevole all'interno di un modulo, ad esempio un file eseguibile o una libreria, e caricarle o modificarle a fini dannosi.

LockResource: Blocca una risorsa in memoria. Potrebbe essere utilizzata per bloccare una risorsa in memoria e manipolarla in modo malevolo, come sovrascrivere i dati di una risorsa con codice dannoso o alterarne il contenuto per scopi dannosi.

SizeofResource: Restituisce la dimensione di una risorsa. Potrebbe essere utilizzata per determinare la dimensione di una risorsa malevola, ad esempio un file o un oggetto, al fine di sfruttare limiti di buffer o violazioni di sicurezza nel sistema.

RegSetValueExA: Imposta il valore di una voce nel registro. Potrebbe essere utilizzata per modificare o creare voci nel registro per scopi malevoli, come l'installazione di un malware, la modifica delle impostazioni di sicurezza o il disabilitamento delle funzionalità di sicurezza.

RegCreateKeyExA: Crea o apre una chiave nel registro. Potrebbe essere utilizzata per creare nuove chiavi nel registro o aprire chiavi esistenti per scopi malevoli, come l'installazione di un malware o l'inserimento di impostazioni malevole all'interno del registro. Potrebbe anche essere utilizzata per bypassare le restrizioni di sicurezza e ottenere privilegi elevati nel sistema.

Librerie:

Kernel32.dll: Libreria che fornisce funzioni per interagire con il sistema operativo. Il malware potrebbe utilizzarla per creare o modificare file, creare o terminare processi, gestire la memoria, modificare le chiavi di registro e chiamare altre API di Windows.

Advapi32.dll: Libreria che fornisce funzioni relative alla sicurezza, gestione di servizi e manipolazione delle chiavi di registro. Il malware potrebbe usarla per creare o modificare chiavi di registro, criptare dati, modificare autorizzazioni ai file o processi e manipolare i servizi di sistema.



- Quali librerie importa il Malware? Per ognuna delle librerie importate, fate delle ipotesi sulla base della sola analisi statica delle funzionalità che il Malware potrebbe implementare. Utilizzate le funzioni che sono richiamate all'interno delle librerie per supportare le vostre ipotesi.

Malware_Build_Week_U3.exe						
Module Name	Imports	OFTs	TimeStamp	ForwarderChain	Name RVA	FTs (IAT)
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	51	00007534	00000000	00000000	0000769E	0000700C
ADVAPI32.dll	2	00007528	00000000	00000000	000076D0	00007000

Date le librerie importate e le funzioni, è molto probabile che il malware sia di tipo **DROPPER**, date funzioni come **FindResource**, **LoadResource**, **LockResource** e la mancanza di funzioni che creino una connessione a internet. Funzioni come **CreateFile**, **WriteFile** e **ReadFile** lasciano ipotizzare che il malware non esegua immediatamente il malware droppato, ma che lo salvi sul disco tramite creazione di un file.

Il malware importa anche **LoadLibrary** e **GetProcAddress**, il che suggerisce che utilizzi altre librerie importate in runtime, non visualizzabili con analisi statica basica.

Le funzioni importate da **Advapi32**, inoltre, lasciano ipotizzare che il malware tenti di ottenere permanenza sul sistema aggiungendo e modificando chiavi di registro.



Address	Ordinal	Name	Library
00407000		RegSetValueExA	ADVAPI32
00407004		RegCreateKeyExA	ADVAPI32

Librerie:

Kernel32.dll: Libreria che fornisce funzioni per interagire con il sistema operativo. Il malware potrebbe utilizzarla per creare o modificare file, creare o terminare processi, gestire la memoria, modificare le chiavi di registro e chiamare altre API di Windows.

Advapi32.dll: Libreria che fornisce funzioni relative alla sicurezza, gestione di servizi e manipolazione delle chiavi di registro. Il malware potrebbe usarla per creare o modificare chiavi di registro, criptare dati, modificare autorizzazioni ai file o processi e manipolare i servizi di sistema.

□ Lo scopo della funzione chiamata alla locazione di memoria 00401021

```
.text:00401021      call    ds:RegCreateKeyExA
.text:00401027      test    eax, eax
.text:00401029      jz      short loc_401032
.text:0040102B      mov     eax, 1
.text:00401030      jmp     short loc_40107B
```

La riga di codice evidenziata esegue una chiamata alla funzione “RegCreateKeyExA” della libreria Windows ADVAPI32.dll, utilizzando l’istruzione “call”. Questa funzione è impiegata per creare o aprire una chiave nel registro di sistema di Windows, consentendo l’accesso e la modifica delle informazioni di configurazione e impostazioni del sistema e delle applicazioni.

□ Come vengono passati i parametri alla funzione alla locazione 00401021;



Per passare i parametri alla funzione, viene utilizzato lo **stack**, che è una struttura dati **LIFO (last-in, first-out)**. Nello stack, vengono riservati spazi per i parametri della funzione. La funzione chiamata può quindi accedere ai parametri utilizzando gli offset relativi rispetto al puntatore dello stack, utilizzando i registri di base come l’ESP (Stack Pointer) o l’EBP (Extended Base Pointer) per calcolare gli indirizzi di memoria dei parametri.



```
.text:00401009      push    eax                ; phkResult
.text:0040100A      push    0                 ; lpSecurityAttributes
.text:0040100C      push    0F003Fh           ; samDesired
.text:00401011      push    0                 ; dwOptions
.text:00401013      push    0                 ; lpClass
.text:00401015      push    0                 ; Reserved
.text:00401017      push    offset SubKey      ; "SOFTWARE\\Microsoft\\Windows NT\\CurrentVe"...
.text:0040101C      push    80000002h          ; hKey
.text:00401021      call    ds:RegCreateKeyExA
```




□ Che oggetto rappresenta il parametro alla locazione **00401017**

```
.text:00401015      push     0                ; Reserved
.text:00401017      push     offset SubKey    ; "SOFTWARE\\Microsoft\\Windows NT\\CurrentVe"...
.text:0040101C      push     80000002h        ; hKey
.text:00401021      call    ds:RegCreateKey; char SubKey[]
.text:00401027      test     eax, eax         SubKey      db 'SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Winlogon',0
.text:00401029      jz       short loc_401032 ; DATA XREF: sub_401000+17↑o
```

L'oggetto alla locazione di memoria **00401017** è la chiave di registro, utilizzata per ottenere la persistenza.


Questa determinata chiave ha come percorso "Software\\Microsoft\\Windows NT\\CurrentVersion\\WinLogon".

□ Il significato delle istruzioni comprese tra gli indirizzi **00401027** e **00401029**.

```
.text:00401027      test     eax, eax
.text:00401029      jz       short loc_401032
.text:0040102B      mov     eax, 1
.text:00401030      jmp     short loc_40107B
```

Nella prima stringa il codice **confronta il registro eax con se stesso utilizzando un'operazione di AND** L'obiettivo di questa istruzione è controllare se il valore di eax è zero o meno.

La seconda stringa è **un'istruzione di salto condizionale**. Se l'istruzione precedente ha impostato il registro eax a zero, allora il salto viene eseguito, portando l'esecuzione a **loc_401032**



```
test    eax, eax
jz      short loc_401032
```

La prima parte di codice verifica se il valore di `eax` è uguale a zero. Se la condizione è soddisfatta, viene eseguito il codice specificato.

Nella seconda parte il codice controlla il valore di `eax` e utilizza le istruzioni `if` per dirigere il flusso di esecuzione del programma in base alla condizione specificata.

Dall'indirizzo 27 a 29:

```
#include <stdio.h>
```

```
int main() {
    int eax = 0;

    if (eax == 0) {
        goto loc_401032;
    }
}
```

```
loc_401032:
    // Codice da eseguire se eax è uguale a zero

    return 0;
}
```

```
test    eax, eax
jz      short loc_401032
mov     eax, 1
jmp     short loc_40107B
```

Dall'indirizzo 27 a 30:

```
int main() {
    int eax = 0;

    // Controllo se eax è uguale a zero
    if (eax == 0) {
        // Salto a loc_401032 se la condizione è verificata
        goto loc_401032;
    }
}
```

```
// Assegno il valore 1 a eax
eax = 1;
```

```
// Salto a loc_40107B
goto loc_40107B;
```

```
loc_401032:
    // Codice da eseguire se eax è uguale a zero
```

```
loc_40107B:
    // Codice da eseguire dopo il salto a loc_40107B

    return 0;
}
```




□ Valutate ora la chiamata alla locazione **00401047**, qual è il valore del parametro «ValueName»?

```
.text:0040103E
.text:00401043
.text:00401046
.text:00401047
```

```
push offset ValueName ; "GinaDLL"
mov  eax, [ebp+hObject]
push eax                ; hKey
call ds:RegSetValueExA
```

Nella chiamata alla locazione **00401047**, il valore del parametro **ValueName** corrisponde al valore contenuto all'indirizzo di memoria specificato da [offset ValueName], che in questo caso è “GinaDLL”.

Nel complesso delle due funzionalità appena viste, spiegate quale funzionalità sta implementando il Malware in questa sezione.

```
push offset SubKey ; "SOFTWARE\\Microsoft\\Windows NT\\CurrentVe"...
push 80000002h      ; hKey
call ds:RegCreateKeyExA
test  eax, eax
jz    short loc_401032
mov   eax, 1
jmp   short loc_40107B
```

```
mov  ecx, [ebp+cbData] ; CODE XREF: sub_401000+29↑j
push ecx               ; cbData
mov  edx, [ebp+lpData]
push edx               ; lpData
push 1                 ; dwType
push 0                 ; Reserved
push offset ValueName ; "GinaDLL"
mov  eax, [ebp+hObject]
push eax               ; hKey
call ds:RegSetValueExA
```

Nella prima funzionalità implementata, il malware crea una sottochiave di registro al path “Software\Microsoft\Windows NT\CurrentVersion\Winlogon” con la funzione **RegCreateKeyExA**, tramite l'istruzione **push** che inserisce l'offset SubKey nello stack.

Nella seconda funzionalità, chiamata **RegSetValueExA**, il malware rinomina la sottochiave in “GinaDLL”, probabilmente per camuffarsi da normale file di sistema nel registro e ottenere la persistenza, pushando l'offset **ValueName** nello stack.

In entrambi i casi, gli handle di registro (**hKey**) sono pushati sullo stack per identificare e accedere al registro di sistema.