

# *Relazione homework 2*

Luca Mastrobattista

Matricola: 0292461

## Indice

<b>1</b>	<b>Traccia dell'homework</b>	<b>3</b>
1.1	Testo . . . . .	3
1.2	Scadenza . . . . .	3
1.3	Consegna . . . . .	3
<b>2</b>	<b>Ambiente di lavoro</b>	<b>4</b>
<b>3</b>	<b>Metodologia</b>	<b>5</b>
3.1	Informazioni note a priori . . . . .	5
3.2	Finalizzazione dell'obiettivo . . . . .	5
3.3	Ottenimento del codice macchina . . . . .	5
3.4	Osservazione del funzionamento . . . . .	5
3.5	Disassemblaggio del codice macchina . . . . .	6
3.5.1	Riepilogo risultati dell'import . . . . .	6
3.5.2	Informazioni aggiuntive . . . . .	6
3.6	Ricerca del WinMain . . . . .	8
<b>4</b>	<b>Analisi</b>	<b>9</b>
4.1	FUN_00401904_WinMain . . . . .	9
4.2	FUN_004012e0_window_proc . . . . .	9
4.2.1	WM_CREATE . . . . .	9
4.2.1.1	FUN_00401aab_init_ds . . . . .	10
4.2.1.2	FUN_00401b74_show_time . . . . .	11
4.2.1.3	FUN_00401b20_create_timer . . . . .	11
4.2.1.4	FUN_00401c7b_timer_proc . . . . .	12
4.2.1.5	FUN_00403000_check_on_exit . . . . .	13
4.2.2	WM_DESTROY . . . . .	14
4.2.3	WM_COMMAND . . . . .	14
4.2.3.1	FUN_00401dd6_button_clicked . . . . .	14

4.2.3.2	FUN_00401cf4_compute_shutdown_time . .	15
<b>5</b>	<b>Verifica</b>	<b>16</b>

# **1 Traccia dell'homework**

## **1.1 Testo**

Analizzare con Ghidra, utilizzando lo strumento disassemblatore/decompilatore, il programma eseguibile hw2.exe contenuto nell'archivio hw2.zip (password: "AMW21"). Determinare il codice di sblocco che rende funzionale il programma e riassumere in un documento tutte le informazioni acquisite, la metodologia adottata ed i passi logici deduttivi utilizzati nel lavoro di analisi.

## **1.2 Scadenza**

Due settimane dalla data di assegnazione del lavoro: 14/11/2021

## **1.3 Consegna**

Documento in formato PDF inviato come allegato ad un messaggio di posta elettronica all'indirizzo del docente ("`<cognome>@uniroma2.it`"), con subject: "[AMW21] HW2: `<matricola studente>`"

## 2 Ambiente di lavoro

Il file eseguibile è stato caricato su Ghidra istallato su un sistema operativo Linux.

L'ambiente controllato di utilizzo è un sistema operativo Windows 10 virtualizzato con il software *VirtualBox*, in cui sono istallati gli strumenti di monitoraggio.

## 3 Metodologia

### 3.1 Informazioni note a priori

Dalla traccia dell'*homework* si deduce che l'eseguibile risulta *bloccato* in qualche modo, e solo con un particolare codice si riesce a farlo funzionare correttamente.

### 3.2 Finalizzazione dell'obiettivo

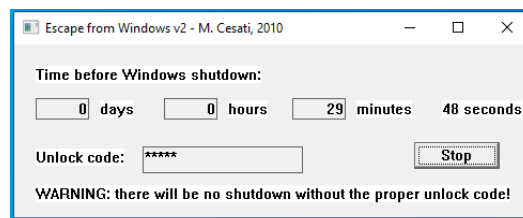
L'analisi dell'applicazione ha come obiettivo quello di individuare il codice di sblocco.

### 3.3 Ottenimento del codice macchina

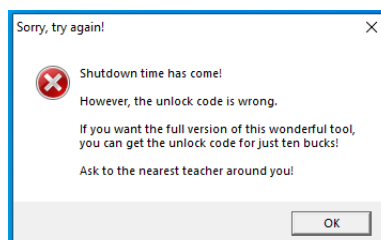
Codice macchina fornito dal professore.

### 3.4 Osservazione del funzionamento

L'applicazione, una volta avviata, crea una finestra con un *countdown* impostato a 30 minuti per default, ma questo tempo è modificabile. Premendo sul pulsante *Go*, il conto alla rovescia inizia. È presente una casella di testo in cui inserire il codice da trovare.



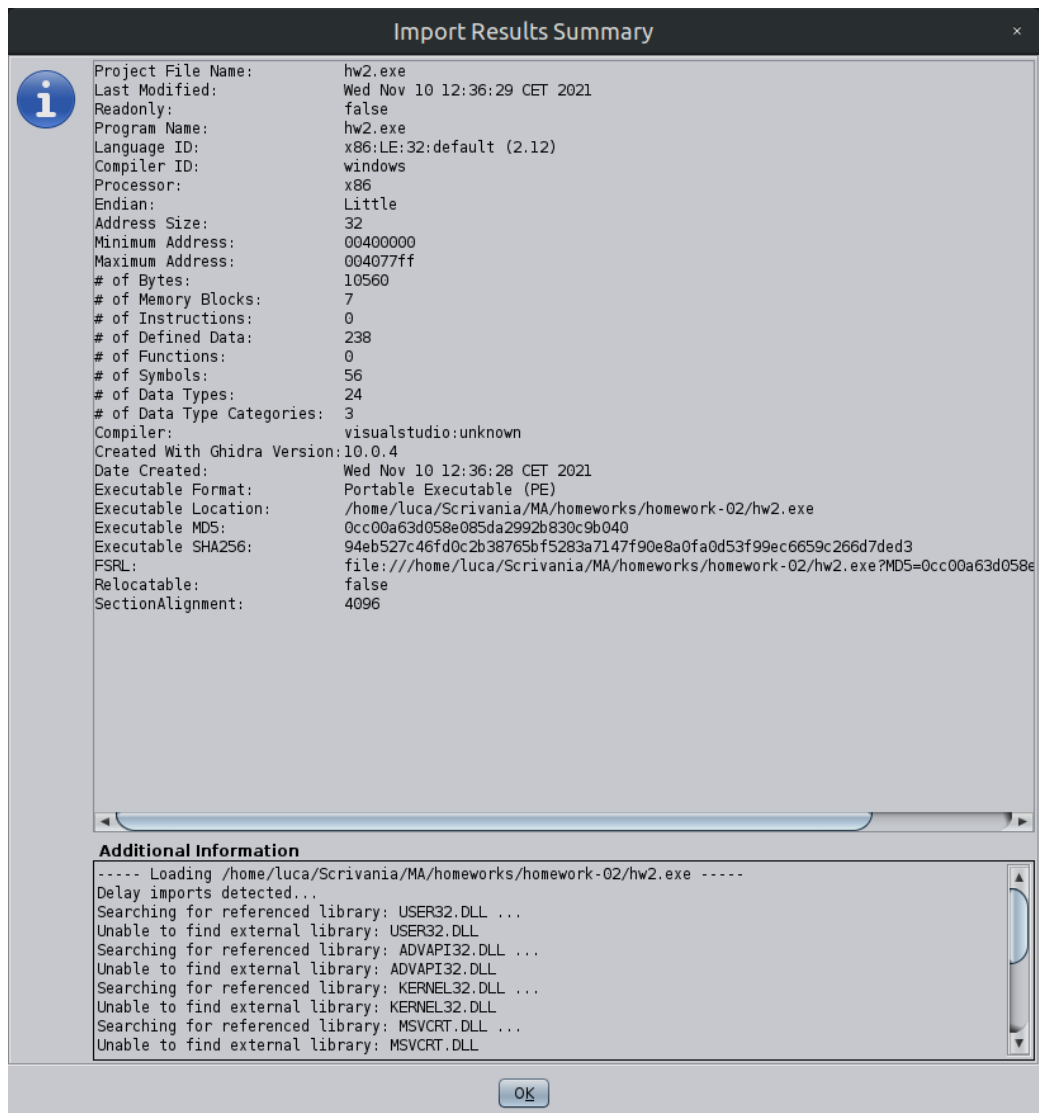
Al termine del *countdown*, se il codice inserito non è corretto, compare una finestra di avviso di codice errato:



## 3.5 Disassemblaggio del codice macchina

Lo strumento che si è utilizzato è il software *Ghidra*.

### 3.5.1 Riepilogo risultati dell'import



### 3.5.2 Informazioni aggiuntive

```
----- Loading /home/luca/Scrivania/MA/homeworks/homework-02/hw2.exe
-----
Delay imports detected...
```

```
Searching for referenced library: USER32.DLL ...
Unable to find external library: USER32.DLL
Searching for referenced library: ADVAPI32.DLL ...
Unable to find external library: ADVAPI32.DLL
Searching for referenced library: KERNEL32.DLL ...
Unable to find external library: KERNEL32.DLL
Searching for referenced library: MSVCRT.DLL ...
Unable to find external library: MSVCRT.DLL
Searching for referenced library: GDI32.DLL ...
Unable to find external library: GDI32.DLL
Finished importing referenced libraries for: hw2b.exe
[ADVAPI32.DLL] -> not found
[GDI32.DLL] -> not found
[KERNEL32.DLL] -> not found
[MSVCRT.DLL] -> not found
[USER32.DLL] -> not found
```

### 3.6 Ricerca del WinMain

Cercando tra le varie librerie importate, in USER32.DLL ci sono le classiche `GetMessageA`, `TranslateMessage`, `DispatchMessageA`. Queste funzioni sappiamo essere le funzioni di gestione del `MainLoop`: cerchiamo allora le referenze a `GetMessage`. Questa funzione viene invocata solo una volta, nella funzione `FUN_00401904_WinMain`, che quindi è il nostro `WinMain`.



## 4 Analisi

### 4.1 FUN\_00401904\_WinMain

Nella funzione, è interessante leggere come viene inizializzato il parametro di tipo `WNDCLASSEX` e, in particolare, il campo `lpfnWndProc`. Infatti, questo campo sarà il puntatore alla *window procedure* che eseguirà quando la finestra riceverà messaggi dal sistema operativo. Questo campo è settato all'indirizzo di `LAB_004012e0`, che quindi deve essere convertito in funzione. Questa sarà la funzione `FUN_004012e0_window_proc`.

### 4.2 FUN\_004012e0\_window\_proc

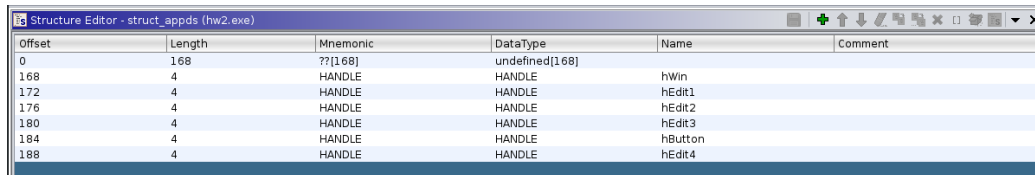
Questa è la funzione che si occupa della gestione dei vari messaggi che arrivano all'applicazione. Ricordiamo che questa funzione non è invocata direttamente ma esegue all'occorrenza dei messaggi e, per questo motivo, il valore dei parametri dipende dal messaggio stesso che arriva.

#### 4.2.1 WM\_CREATE

La gestione di questo messaggio è interessante. Per prima cosa viene recuperato l'*handle* alla istanza dell'applicazione invocando `GetWindowLongA` con il secondo parametro impostato a `GWL_HINSTANCE`. In seguito, viene invocata una `SetWindowLongA`, con cui si impostano gli *user data* associati con la finestra. Il valore passato è il valore puntato dal parametro di input `lParam`. Per messaggi di tipo `WM_CREATE`, il parametro `lParam` è un puntatore a struttura `CREATESTRUCT`. In questo caso si sta recuperando il primo valore della struttura, cioè `LPVOID lpCreateParams`. Se la finestra è stata creata con `CreateWindowEx`, questo campo contiene il valore del parametro `lpParam` passato alla funzione `CreateWindowEx`. La funzione `CreateWindowEx` viene invocata in `WinMain` e, come parametro `lpParam`, prende il risultato di `FUN_00401aab_init_ds`, cioè l'indirizzo della struttura dati di tipo `struct_appds_00406010` lì inizializzata. Capiamo che questa sia una struttura dati per come vengono settati alcuni parametri: prima si inizializza il valore dei 4 byte distanti 168 dall'indirizzo base all'*handler* della finestra recuperata con la chiamata iniziale a `GetWindowLongA`, poi all'interno di un ciclo, partendo dall'indirizzo base, si aggiunge un offset che incrementa di un valore costante per assegnare dei valori. In particolare, si inizializzano i bytes 172, 176, 180, 188, e 184 ai risultati di invocazioni di `CreateWindowExA`. Questo blocco di codice termina invocando

`FUN_00401b74_show_time` e `FUN_00401b20_create_timer`.

La struttura dati ha quindi il seguente aspetto:



Offset	Length	Mnemonic	DataType	Name	Comment
0	168	??[168]	undefined[168]		
168	4	HANDLE	HANDLE	hWin	
172	4	HANDLE	HANDLE	hEdit1	
176	4	HANDLE	HANDLE	hEdit2	
180	4	HANDLE	HANDLE	hEdit3	
184	4	HANDLE	HANDLE	hButton	
188	4	HANDLE	HANDLE	hEdit4	

**4.2.1.1 FUN\_00401aab\_init\_ds** Questa funzione riceve in input un puntatore a funzione e inizializza una struttura dati definita alla variabile globale `struct_appds_00406010`. Qui vengono inizializzati altri campi della struttura: i primi quattro byte sono impostati a 0, i quattro byte in posizione 4 sono impostati a 1000, e quelli in posizione 12 sono impostati a 1800. Quest'ultimo ci ricorda il numero di secondi presenti in mezz'ora, che coincide col valore di default dell'applicazione mostrato quando viene avviata e, per questo, è lecito pensare che sia il tempo in secondi dopo il quale l'applicazione dovrebbe venire spenta. In seguito ci sono due invocazioni della funzione `FUN_004023c0_invoke_snprintf`, funzione che non fa altro che invocare la funzione `_vsnprintf`. Quest'ultima funzione prende in input i seguenti parametri:

- un buffer in cui salvare la stringa formattata passata successivamente;
- la dimensione massima, in byte, che possono essere salvati sul buffer;
- una serie di parametri che servono a formattare la stringa da memorizzare.

La prima invocazione memorizza al massimo 128 caratteri sul buffer identificato nella struttura dati all'offset 24, impostandola alla stringa `"WARNING: there will be no shutdown without the proper unlock code!"`; la seconda, invece, memorizza al massimo 16 caratteri sul buffer identificato nella struttura all'offset 152, inizializzandolo alla stringa `" 0 seconds"`. Infine, si inizializzano i quattro byte in posizione 16 impostandoli a 0 e quelli in posizione 20 impostandoli al parametro ricevuto in input, di cui non conosciamo ancora il tipo.

La struttura dati ha adesso la seguente forma:

Offset	Length	Mnemonic	Data Type	Name	Comment
0	4	ddw	dword	init_0	
4	4	ddw	dword	init_1000	
8	1	??	undefined		
9	1	??	undefined		
10	1	??	undefined		
11	1	??	undefined		
12	4	ddw	dword	shutdown_time	
16	4	ddw	dword	init_0_bis	
20	4	undefined4	undefined4	init_ds_input_par	
24	128	char[128]	char[128]	warning_mess	
152	16	char[16]	char[16]	0_seconds_string	
168	4	HANDLE	HANDLE	hWin	
172	4	HANDLE	HANDLE	hEdit1	
176	4	HANDLE	HANDLE	hEdit2	
180	4	HANDLE	HANDLE	hEdit3	
184	4	HANDLE	HANDLE	hButton	
188	4	HANDLE	HANDLE	hEdit4	

**4.2.1.2 FUN\_00401b74\_show\_time** Questa funzione aggiorna il valore della stringa `0_seconds_string` della struttura dati, impostandola a:

$$s = (\text{shutdown\_time} - \text{init\_0}) - m \cdot 60$$

dove  $m$  è calcolato come:

$$m = \frac{(\text{shutdown\_time} - \text{init\_0}) \cdot 1000}{\text{init\_1000} \cdot 60}$$

Il valore  $m$  rappresenta il numero di minuti rimanenti, perché è risultato della divisione intera. Il valore  $s$ , invece, è il numero di secondi ancora rimanenti. Tutto ciò ha senso, però, solo se si assume che il campo `init_0` memorizzi in realtà il numero di secondi passati da quando è stato premuto il pulsante *Go*; continuando su questa ipotesi, quel campo viene rinominato in `elapsed_seconds`.

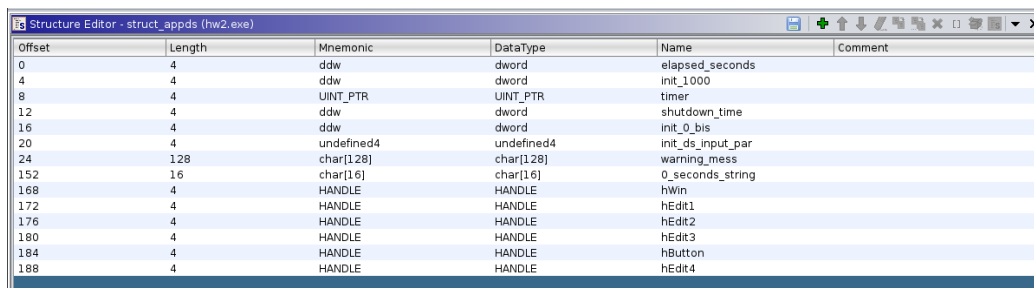
Successivamente, all'interno di un ciclo, viene calcolato il numero di giorni rimanenti, sottraendo a  $m$  il valore 1440, cioè il numero di minuti in un giorno. Ad ogni iterazione, un contatore viene incrementato; il valore del contatore sarà impostato come valore della casella di testo relativa ai giorni. In seguito, utilizza lo stesso meccanismo per calcolare il numero di ore rimanenti, sottraendo questa volta il valore 60. Ciò che rimane, sarà il numero di minuti rimanenti. Vengono così aggiornati i valori delle caselle dei giorni, ore e minuti.

**4.2.1.3 FUN\_00401b20\_create\_timer** Questa funzione invoca semplicemente la funzione di libreria `SetTimer`, ma ci da molte informazioni riguardo ai parametri:

- il primo parametro è l'*handle* alla finestra che deve essere associata con il timer;
- il valore 0 come secondo parametro, per specificare la creazione del timer;

- un valore intero, che specifica ogni quanto tempo eseguire un'operazione, in millisecondi. Il valore passato è il campo `init_1000`: ogni volta che trascorre un secondo verrà invocata una funzione;
- un puntatore a funzione che viene invocata ogni volta che il tempo specificato nel parametro precedente passa. La funzione che viene impostata è `FUN_00401c7b_timer_proc`.

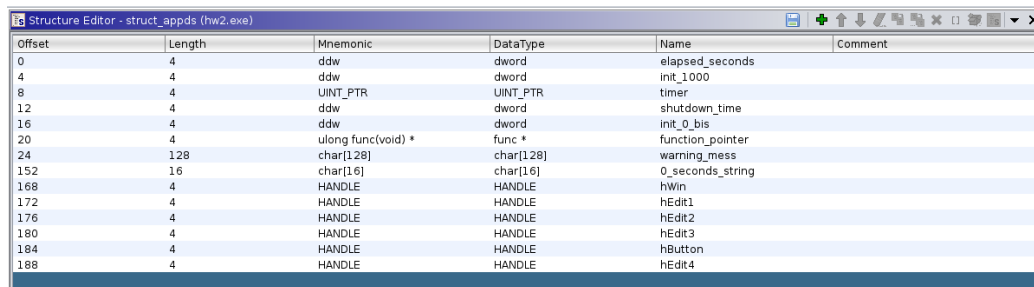
Questa funzione restituisce un `UINT_PTR` che identifica il timer creato. Questo valore è inserito nella struttura dati, in corrispondenza del byte 8:



Offset	Length	Mnemonic	DataType	Name	Comment
0	4	ddw	dword	elapsed_seconds	
4	4	ddw	dword	init_1000	
8	4	UINT_PTR	UINT_PTR	timer	
12	4	ddw	dword	shutdown_time	
16	4	ddw	dword	init_0_bis	
20	4	undefined4	undefined4	init_ds_input_par	
24	128	char[128]	char[128]	warning_mess	
152	16	char[16]	char[16]	0_seconds_string	
168	4	HANDLE	HANDLE	hWin	
172	4	HANDLE	HANDLE	hEdit1	
176	4	HANDLE	HANDLE	hEdit2	
180	4	HANDLE	HANDLE	hEdit3	
184	4	HANDLE	HANDLE	hButton	
188	4	HANDLE	HANDLE	hEdit4	

Questa funzione, quindi, specifica le operazioni da fare ad ogni secondo trascorso. Potremmo quindi essere sulla strada giusta per la ricerca del codice.

**4.2.1.4 FUN\_00401c7b\_timer\_proc** Questa funzione, per prima cosa, incrementa il campo `elapsed_seconds` della struttura dati `struct_appds`. Se è diverso da 0, aggiorna con la funzione `FUN_00401b74_show_time` i valori mostrati. In seguito ridisegna la finestra. Con un controllo, si verifica se il campo `init_0_bis` è diverso da 0 e se **il tempo di shutdown è minore del valore di elapsed\_seconds incrementato**: con quest'ultimo controllo, si sta verificando, in pratica, se il timer è scaduto. Se queste condizione sono entrambe vere, c'è un'istruzione di `CALL` che invoca il contenuto del campo in posizione 20 della struttura dati. Capiamo quindi che quel campo della struttura è un puntatore a funzione:



Offset	Length	Mnemonic	DataType	Name	Comment
0	4	ddw	dword	elapsed_seconds	
4	4	ddw	dword	init_1000	
8	4	UINT_PTR	UINT_PTR	timer	
12	4	ddw	dword	shutdown_time	
16	4	ddw	dword	init_0_bis	
20	4	ulong func(void) *	func *	function_pointer	
24	128	char[128]	char[128]	warning_mess	
152	16	char[16]	char[16]	0_seconds_string	
168	4	HANDLE	HANDLE	hWin	
172	4	HANDLE	HANDLE	hEdit1	
176	4	HANDLE	HANDLE	hEdit2	
180	4	HANDLE	HANDLE	hEdit3	
184	4	HANDLE	HANDLE	hButton	
188	4	HANDLE	HANDLE	hEdit4	

È, con buone probabilità, la funzione che cerchiamo. Ritornando a `FUN_00401aab_init_ds`, vediamo che il campo della struttura dati `function_pointer` viene inizializzato impostandolo al parametro di input. Questo parametro è l'indirizzo di `DAT_00403000`. Questa avrebbe quindi dovuto essere una funzione, tuttavia non è possibile convertirla. Disassemblando con Ghidra, cioè cliccando con il tasto destro sulle istruzioni e selezionando *Disassemble*, si ottengono delle istruzioni macchina, anche se il flusso sembra essere corrotto. Questo può essere riparato cliccando col tasto destro sull'istruzione e premendo *Clear Flow and Repair* come mostrato in [questo video youtube](#).

Una volta disassemblato e riparato il codice, si può analizzare la funzione invocata prima di terminare: `FUN_00403000_check_on_exit`.

**4.2.1.5 `FUN_00403000_check_on_exit`** Questa funzione viene invocata passandogli come input la struttura dati `struct_appds_00406010`. Inizializza il campo `init_0_bis` a 0 e poi invoca la funzione `GetDlgItemTextA`. I parametri passati a questa funzione sono:

- il campo `hWin` della struttura dati, che memorizza l'*handle* alla finestra;
- il valore 5: questo rappresenta l'identificativo del *control* da cui recuperare il testo. È l'identificativo della casella di testo che riceve il codice;
- un buffer in cui memorizzare il testo recuperato. Il valore passato è l'area dello stack identificata da `ESP + 0x26`: il valore recuperato viene messo sullo stack;
- il valore 30, che rappresenta la lunghezza massima recuperabile.

La funzione restituisce il numero di caratteri recuperati.

Il valore restituito viene confrontato col il numero 9 e, se è diverso viene invocata la funzione `FUN_00401ff0_show_error_message` per poi ritornare. Se invece sono stati effettivamente letti 9 caratteri, si verifica che questi corrispondano a `3RnESt0!?`; il confronto è fatto byte per byte e, al primo errore, viene invocata la funzione `FUN_00401ff0_show_error_message`. Se invece tutti i controlli passano, ci si prepara all'invocazione di una serie di funzioni:

- `OpenProcessToken`, per aprire l'*access token* associato al processo;

- `LookupPrivilegeValueA`, per recuperare il `locally unique identifier` (LUID) usato sul sistema locale (identificato dalla stringa `NULL` passata come primo parametro);

Se queste invocazioni non falliscono, viene finalmente invocata `ExitWindowsEx`.

Si può quindi concludere l'analisi affermando che il codice che permette lo *shutdown* è **3RnESt0!?**.

L'obiettivo dell'analisi è stato raggiunto in questo punto, e si potrebbe saltare alla fase di *Verifica*; tuttavia non è stato ancora individuato il ruolo del campo `init_0_bis` della struttura dati e perciò, anche se non richiesto, viene riportato in seguito l'analisi delle funzioni che porta all'apprendimento del suo ruolo.

#### 4.2.2 WM\_DESTROY

In questo blocco di codice si invoca la funzione `FUN_00401b54_kill_timer`, che a sua volta invoca la funzione di libreria `KillTimer` passando come parametro l'identificativo memorizzato nella `struct_appds_00406010` al campo `timer`. Il blocco di codice termina invocando `PostQuitMessage`.

#### 4.2.3 WM\_COMMAND

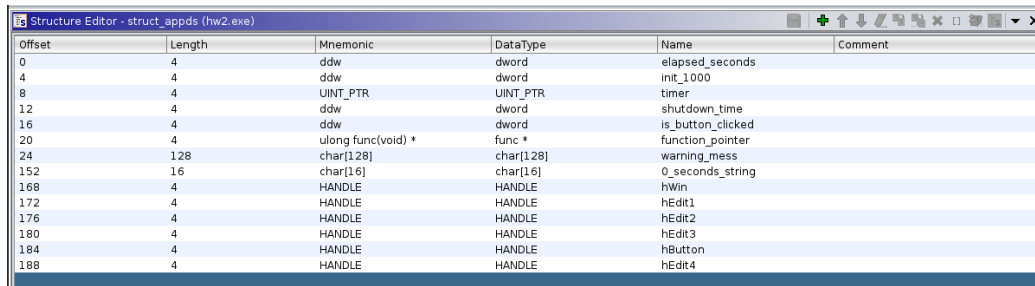
Quando arriva un messaggio di questo tipo, il parametro `wParam` corrisponde all'identificativo del *control* che genera il messaggio mentre `lParam` è l'*handle* alla finestra di quel *control*. In questo blocco di codice si controlla l'*high word*, cioè la parola più significativa, per verificare che sia pari a 0 e si controlla che l'*handle* contenuto in `lParam` passato in input alla funzione sia uguale a quella memorizzata nella struttura dati al campo `hButton`. Se le condizioni sono entrambe vere, si invoca `FUN_00401dd6_button_clicked` prima di terminare.

**4.2.3.1 FUN\_00401dd6\_button\_clicked** Questa funzione controlla il campo `init_0_bis` per verificare se sia uguale a 0. In base al risultato del confronto, esegue 2 blocchi differenti prima di invocare una `RedrawWindow`. Nel caso in cui `init_0_bis` sia 0, si cambia il testo mostrato nella finestra di indice 4, che corrisponde al pulsante: la scritta viene impostata a "Stop". Dopodiché, si aggiorna il valore del campo `init_0_bis` e lo si imposta a 1 prima di invocare 4 volte la funzione `SendDlgItemMessageA`, una per ogni

casella di testo. Il parametro `Msg` di queste invocazioni è 270, che corrisponde a `EM_SETREADONLY`, mentre il parametro `wParam` è un intero che viene utilizzato come `Booleano`: in questo caso, il valore è impostato a `True`. In effetti, una volta premuto sul pulsante *"Go"*, la scritta cambia e le caselle di testo non sono più modificabili. Il blocco di codice termina invocando `FUN_00401cf4_compute_shutdown_time`.

Viceversa, se `init_0_bis` è diverso da 0, il suo valore viene riportato a 0, la scritta del pulsante viene riportata a *"Go"* e alle caselle di testo viene tolto lo stile *read only*. Il blocco termina invocando `FUN_00401b74_show_time` e aggiornando il campo della struttura `0_seconds_string` al valore di default *" 0 seconds"*: infatti, quando si preme il pulsante *Stop*, il valore dei secondi viene riportato a 0, indipendentemente da quale fosse il precedente valore.

Possiamo concludere che il campo `init_0_bis` è in realtà un valore che tiene traccia se il pulsante *Go* è stato premuto e il countdown è attivo. La struttura completa è quindi definita:



Offset	Length	Mnemonic	Data Type	Name	Comment
0	4	ddw	dword	elapsed_seconds	
4	4	ddw	dword	init_1000	
8	4	UINT_PTR	UINT_PTR	timer	
12	4	ddw	dword	shutdown_time	
16	4	ddw	dword	is_button_clicked	
20	4	ulong func(void) *	func *	function_pointer	
24	128	char[128]	char[128]	warning_mess	
152	16	char[16]	char[16]	0_seconds_string	
168	4	HANDLE	HANDLE	hWin	
172	4	HANDLE	HANDLE	hEdit1	
176	4	HANDLE	HANDLE	hEdit2	
180	4	HANDLE	HANDLE	hEdit3	
184	4	HANDLE	HANDLE	hButton	
188	4	HANDLE	HANDLE	hEdit4	

**4.2.3.2 FUN\_00401cf4\_compute\_shutdown\_time** Questa funzione raccoglie i valori nelle caselle di testo e converte il valore dell'utente in minuti; esegue la seguente operazione:

$$m = days \cdot 1440 + hours \cdot 60 + minutes$$

dove *days*, *hours* e *minutes* sono i valori recuperati rispettivamente dalle caselle di giorni, ore e minuti.

Il valore *m* così ottenuto, è utilizzato nel seguente calcolo:

$$\frac{m \cdot init\_1000 \cdot 60}{1000} + elapsed\_seconds$$

Il risultato di questa operazione è il valore in secondi dopo il quale si vuole che il pc venga spento, e infatti viene memorizzato nel campo `shutdown_time` della struttura dati `struct_appds`.

## 5 Verifica

Dopo aver inserito la password **3RnESt0!?** e dopo aver atteso il tempo selezionato, la macchina virtuale viene effettivamente spenta: il codice ottenuto dall'analisi è effettivamente il codice corretto.