# Homework 2

**Pierpaolo Spaziani - 0316331**

## Introduzione

Lanciando il programma, sicuri del fatto che non sia un malware dovendo trovare un codice di sblocco da inserire, si nota che è una versione modificata di quello analizzato durante il corso. Se corretto, il codice permette di effettuare lo shutdown della macchina dopo il tempo specificato.

## Identificazione del WinMain

Per l'analisi del programma, essendo basato su una finestra, il primo obiettivo posto è stato quello di **individuare il WinMain**.
Aprendo il Funcion Call Graph di *entry*, si osserva che FUN_00401904 probabilmente è la funzione cercata poiché invoca funzioni della libreria Windows relative alle finestre.
A sostegno di questa ipotesi c'è il numero dei parametri che prende in input, il loro tipo, ma soprattuto il **MessageLoop**, che risulta semplice da individuare in fondo alla procedura, grazie alla presenza delle 3 funzioni principali: GetMessageA, TranslateMessage e DispatchMessage.
Si procede quindi cambiando l'intestazione della funzione con quella standard della WinMain definita sulla documentazione ufficiale.
Essendo un programma che non vuole rivelare il codice da inserire, la probabilità di trovarlo tra le stringhe individuate da Ghidra è bassa, tuttavia è bene fare un tentativo.
Dopo aver aperto la finestra con le stringhe definite, come ci si aspettava non se ne evidenzia alcuna che si possa ipotizzare come codice di sblocco. Vengono fuori però delle informazioni che potrebbero risultare utili successivamente. Infatti oltre alle stringhe già viste durante l'esecuzione del programma, c'è anche "SeShutdownPrivilege", che sembra una stringa passata ad una funzione, ma la relativa posizione di utilizzo nel codice non è nota.

## Analisi del WinMain

Ghidra riconosce in autonomia la struttura WNDCLASSEXA della variabile local_54 poiché passata come argomento a RegisterClassExA. Dopo aver rinominato la variabile local_54 in wclsex, è stata definita come funzione la label che fa riferimento al campo *lpfnWndProc*, ed è stata chiamata WindowProcedure.
Continuando ad esaminare il WinMain si trova la funzione FUN_00401aab che, come visto a lezione, inizializza la struttura dati utilizzata dal programma. Viene quindi rinominata InitDS e si può iniziare a creare la nuova struttura chiamata AppDS. Proseguendo ad etichettare con nomi appropriati i vari campi della struttura durante con l'analisi del programma, si arriva ad avere un risultato simile (i dettagli di come sono stati trovati i campi sono stati inseriti successivamente):

| Offset | Length | Mnemonic | DataType | Name |
|---|---|---|---|---|
| 0 | 4 | UINT | UINT | time |
| 4 | 4 | UINT | UINT | tick_length |
| 8 | 4 | UINT_PTR | UINT_PTR | timer_pointer |
| 12 | 4 | UINT | UINT | shutdown_time |
| 16 | 4 | BOOL | BOOL | check_Go_Stop |
| 20 | 4 | UINT_PTR | UINT_PTR | shutdown_function |
| 24 | 128 | char[128] | char[128] | str_warning |
| 152 | 16 | char[16] | char[16] | secons_label |
| 168 | 24 | HWND[6] | HWND[6] | window_handler |

Nella WindowProcedure si evidenzia tutto il processo di gestione della finestra basato sui messaggi, in particolare i blocchi principali divisi dagli if/else:

- WM_SIZE;
- WM_ACTIVATE,
  diviso a sua volta in:
  - WM_CREATE;
  - WM_DESTROY;
- WM_PAINT;
- WM_COMMAND.

Alla fine del blocco relativo a WM_CREATE, ci sono 2 funzioni non definite da Windows: FUN_00401b74 e FUN_00401b20. La prima è stata già analizzata a lezione ed è relativa ai dialog box della finestra ed aiuta ad etichettare i campi della struttura. La seconda chiama SetTimer il cui ultimo parametro è la label di riferimento alla TimerProc. Si procede quindi definendo la label come funzione e ridenominandola TimerProc.

All'interno di questa funzione si nota una cosa strana all'interno di un if: una CALL sull'offset 20 di AppDS. Inoltre, osservando la condizione di entrata, si capisce che fa riferimento al funzionamento del programma allo scadere del timer. Viene infatti controllato che il tasto di avvio del timer sia stato premuto e che il timer abbia raggiunto il tempo prestabilito per lo shutdown.

Decompile: TimerProc - (hw2-2.exe)

```
1
2  void TimerProc(HWND param_1)
3
4  {
5    uint time;
6
7    time = AppDS.time + 1;
8    AppDS.time = time;
9    if (AppDS.check_Go_Stop != 0) {
10     secondsLabel();
11   }
12   RedrawWindow(param_1,(RECT *)0x0,(HRGN)0x0,5);
13   if ((AppDS.check_Go_Stop != 0) && (AppDS.shutdown_time <= time)) {
14     (*(code *)AppDS.shutdown_function)(&AppDS);
15     return;
16   }
17   return;
18 }
```

Andando a vedere i riferimenti all'interno del codice per questo campo della struttura, si osserva che viene chiamato 2 volte: una è quella appena vista, l'altra avviene in scrittura in InitDS. Qui viene posto pari al parametro di input della funzione, ovvero DAT_0040300. Analizzando il dato si vede che Ghidra non è riuscito a risolverlo, provando a disassemblarlo viene fuori una nuova funzione.

Inizialmente questa strada era stata scartata poiché erano presenti molti errori di decompilazione all'interno della nuova funzione, tuttavia è stata notata la presenza di alcune funzioni note, tra cui PostQuitMessage e LookupPrivilegeValueA con la stringa "SeShutdownPrivilege", vista già tra le stringhe trovate da Ghidra ad inizio analisi. Questo suggerisce che il disassemblaggio di questa parte di codice non è sbagliato. Guardando meglio si può vedere un controllo a singoli caratteri con la parola **3RnESt0!?**, che porta a

pensare che sia il codice cercato. Inoltre c'è anche un'altra parte all'interno di questa funzione che Ghidra non ha risolto. Disassemblando anche questo viene fuori proprio la funzione ExitWindowsEx. Lo shutdown del sistema avviene qui.

Provando quindi ad eseguire il programma inserendo la parola **3RnESt0!?** la teoria viene confermata, avviene infatti l'arresto del sistema.

# Dettagli struttura AppDS

Le funzioni riportate sono quelle che aiutano a capire come è fatta la struttura AppDS.

**time** rappresenta il tempo trascorso. Viene inizializzato a 0 in InitDS e poi incrementato di 1 nella TimerProc, funzione chiamata ogni volta che avviene il time-out.

**tick_lenght** viene inizializzato a 1000 da InitDS e rappresenta i millisecondi per il time-out. Ciò si evince dal fatto che viene passato come terzo parametro a SetTimer.

**timer_pointer** è il puntatore al timer essendo il valore di ritorno di SetTimer.

**shutdown_time** sono i millisecondi che devono trascorrere prima dello shutdown del sistema. Si evince dal fatto che: in InitDS viene inizializzato a 1800, ovvero i 3 minuti impostati di default per il timer all'avvio del programma, viene ricalcolato nella timeCalculator quando viene premuto il pulsante Go e poi perchè viene utilizzato come condizione di entrata nell'if che porta all'arresto della macchina.

```
time                XREF[8]:  InitDS:00401ab1(W),
                              InitDS:00401b19(*),
                              InitDS:00401b19(*),
                              secondsLabel:00401b89(R),
                              TimerProc:00401c82(R),
                              TimerProc:00401c8b(W),
                              TimerProc:00401cd9(*),
                              timeCalculator:00401dbd(R)
tick_length         XREF[4]:  InitDS:00401abb(W),
                              startTimer:00401b2e(R),
                              secondsLabel:00401ba5(R),
                              timeCalculator:00401d9f(R)
timer_pointer       XREF[2]:  startTimer:00401b4d(W),
                              timerKiller:00401b5a(R)
shutdown_time       XREF[4]:  InitDS:00401ac5(W),
                              secondsLabel:00401b83(R),
                              TimerProc:00401cca(R),
                              timeCalculator:00401dc3(W)
check_Go_Stop       XREF[6]:  InitDS:00401b07(W),
                              TimerProc:00401c91(R),
                              TimerProc:00401cc1(R),
                              switchGoStop:00401de3(R),
                              switchGoStop:00401df0(W),
                              switchGoStop:00401f25(W)
shutdown_function   XREF[2]:  InitDS:00401b14(W),
                              TimerProc:00401ce0
str_warning         XREF[1]:  InitDS:00401adf(*)
secons_label        XREF[3]:  InitDS:00401afb(*),
                              secondsLabel:00401bdd(*),
                              switchGoStop:00401ed6(*)
window_handler      XREF[3]:  secondsLabel:00401b7d(R),
                              timeCalculator:00401cfd(R),
                              switchGoStop:00401ddd(R)
```

**check_Go_Stop** è un booleano che permette di capire se il programma si trova in stato di Stop o Go, aggiornato in switchGoStop ogni volta che viene premuto il pulsante.

**shutdown_function** è il campo già stato commentato che porta alla funzione di shutdown.

**str_warning** e **seconds_label** sono le stringhe che si trovano nella finestra, facilmente riconoscibili in InitDS con le relative dimensioni.

**window_handler** è l'handler della finestra.

Decompile: InitDS - (hw2-2.exe)

```
1
2  undefined4 * __cdecl InitDS(UINT_PTR param_1)
3
4  {
5    AppDS.time = 0;
6    AppDS.tick_length = 1000;
7    AppDS.shutdown_time = 1800;
8    stringStamp(AppDS.str_warning,128,
9                "WARNING: there will be no shutdown without the proper unlock code!");
10   stringStamp(AppDS.secons_label,16," 0 seconds");
11   AppDS.check_Go_Stop = 0;
12   AppDS.shutdown_function = param_1;
13   return &AppDS.time;
14 }
```

Decompile: switchGoStop - (hw2-2.exe)

```
1
2  void switchGoStop(void)
3
4  {
5    HWND hDlg;
6
7    hDlg = AppDS.window_handler[0];
8    if (AppDS.check_Go_Stop == 0) {
9      SetDlgItemTextA(AppDS.window_handler[0],4,"Stop");
10     AppDS.check_Go_Stop = 1;
11     SendDlgItemMessageA(hDlg,1,0xcf,1,0);
12     SendDlgItemMessageA(hDlg,2,0xcf,1,0);
13     SendDlgItemMessageA(hDlg,3,0xcf,1,0);
14     SendDlgItemMessageA(hDlg,5,0xcf,1,0);
15     timeCalculator();
16   }
17   else {
18     AppDS.check_Go_Stop = 0;
19     SetDlgItemTextA(AppDS.window_handler[0],4,"Go");
20     SendDlgItemMessageA(hDlg,1,0xcf,0,0);
21     SendDlgItemMessageA(hDlg,2,0xcf,0,0);
22     SendDlgItemMessageA(hDlg,3,0xcf,0,0);
23     SendDlgItemMessageA(hDlg,5,0xcf,0,0);
24     secondsLabel();
25     stringStamp(AppDS.secons_label,0x10," 0 seconds");
26   }
27   RedrawWindow(hDlg,(RECT *)0x0,(HRGN)0x0,5);
28   return;
29 }
```

Decompile: timeCalculator - (hw2-2.exe)

```
1
2  void timeCalculator(void)
3
4  {
5    UINT minuti;
6    int time;
7    int local_14;
8    HWND windowHandler;
9
10   windowHandler = AppDS.window_handler[0];
11   minuti = GetDlgItemInt(AppDS.window_handler[0],1,&local_14,0);
12   time = 0;
13   if (local_14 != 0) {
14     time = minuti * 1440;
15   }
16   minuti = GetDlgItemInt(windowHandler,2,&local_14,0);
17   if (local_14 != 0) {
18     time = time + minuti * 60;
19   }
20   minuti = GetDlgItemInt(windowHandler,3,&local_14,0);
21   if (local_14 != 0) {
22     time = time + minuti;
23   }
24   AppDS.shutdown_time = (time * AppDS.tick_length * 60) / 1000 + AppDS.time;
25   secondsLabel();
26   return;
27 }
```

Decompile: secondsLabel - (hw2-2.exe)

```
1
2  void secondsLabel(void)
3
4  {
5    uint minutes;
6    UINT days;
7    int timeLeft;
8    UINT hours;
9    HWND windowHandle;
10
11   windowHandle = AppDS.window_handler[0];
12   timeLeft = AppDS.shutdown_time - AppDS.time;
13   minutes = AppDS.tick_length * 60;
14   stringStamp(AppDS.secons_label,16,"%2ld seconds");
15   days = 0;
16   for (minutes = (uint)(timeLeft * 1000) / minutes; 1439 < minutes; minutes = minutes - 1440) {
17     days = days + 1;
18   }
19   hours = 0;
20   for (; 59 < minutes; minutes = minutes - 60) {
21     hours = hours + 1;
22   }
23   SetDlgItemInt(windowHandle,1,days,0);
24   SetDlgItemInt(windowHandle,2,hours,0);
25   SetDlgItemInt(windowHandle,3,minutes,0);
26   return;
27 }
```

Decompile: startTimer - (hw2-2.exe)

```
1
2  void __cdecl startTimer(HWND param_1)
3
4  {
5    AppDS.timer_pointer = SetTimer(param_1,0,AppDS.tick_length,TimerProc);
6    return;
7  }
```