Homework 3

Pierpaolo Spaziani - 0316331

Introduzione

Lanciando il programma si nota che è una versione modificata di quello analizzato durante il corso e con stesso funzionamento del precedente homework. Se corretto, il codice inserito permette di effettuare lo shutdown della macchina dopo il tempo specificato.

Analisi statica

Essendo la stessa applicazione già vista ed analizzata, procedendo con l'analisi statica con Ghidra si riconosce, con alcune eccezioni, lo stesso schema nel codice.
Si trova infatti la stessa WinMain ma con 2 nuove funzioni in aggiunta:
FUN_004024a0 e FUN_00401560.

La prima, rinominata AntiDebug1, contiene IsDebuggerPresent come condizione per la ShowWindow.

La seconda, rinominata Mappatura, salva la lunghezza della parte alta del file e la sua mapped view; questi dati vengono utilizzati nella funzione FUN_004016b0, presente nel blocco WM_CREATE della WindowProcedure, per fare un controllo sull'integrità del file nel caso venisse modificato. Infatti avviando l'eseguibile modificato, compare la finestra di errore descritta nella FUN_004042a0, rinominata AntiDebug3, contenuta nella TimerProcedure. Questa funzione è stata modificata dal programmatore inserendo delle istruzioni '9a 42' che impediscono il corretto disassemblaggio del codice.

Decompile: AntiDebug1 - (hw3.exe)

```
1
2  void __cdecl AntiDebug1(HWND param_1,int param_2)
3
4  {
5    BOOL BVar1;
6
7    BVar1 = IsDebuggerPresent();
8    if (BVar1 == 0) {
9      ShowWindow(param_1,param_2);
10     return;
11   }
12                     /* WARNING: Subroutine does nc
13   ExitProcess(0);
14 }
```

Decompile: AntiDebug3 - (hw3.exe)

```
1
2  /* WARNING: Globals starting with '_' overlap smal
3
4  void __cdecl AntiDebug3(int param_1)
5
6  {
7    int iVar1;
8    bool bVar2;
9    char local_10c [128];
10   char local_8c [132];
11
12   if (*(int *)(param_1 + 192) != _DAT_004050a4) {
13     iVar1 = DAT_00407100 + 1;
14     bVar2 = DAT_00407100 == 0;
15     DAT_00407100 = iVar1;
16     if (bVar2) {
17       FUN_004026a0(local_10c,0x80,"%s=%lu/0x%lx");
18       FUN_004026a0(local_8c,0x80,"DEBUG %s:%d");
19       MessageBoxA((HWND)0x0,local_10c,local_8c,0);
20     }
21                     /* WARNING: Subroutine does not
22     ExitProcess(0);
23   }
24   return;
25 }
```

Sempre nella WinMain c'è la FUN_00401830, chiamata InitDS durante il corso e nell'homework 2, che differisce dalla precedente per la presenza della FUN_004016f0, rinominata AntiDebug2. In questa funzione è presente in molte posizioni l'istruzione 'eb ff c0 48', inserita dal programmatore per impedire il corretto disassemblaggio del codice. Per poter quindi procedere con la corretta analisi, è necessario modificare il codice sostituendo con dei NOP le istruzioni di disturbo. Una volta terminata la procedura è necessario fare il Clear, ripetere il Disassemblaggio e fare il Re-create della funzione per avere la procedura pulita. Nella funzione in chiaro è possibile individuare il caricamento dinamico, eseguito carattere per

Decompile: AntiDebug2 - (hw3.exe)

```
26     local_2d = 'k';
27     local_29 = 'e';
28     local_2c = 'e';
29     local_2b = 'r';
30     local_2a = 'n';
31     local_28 = 'l';
32     local_27 = '3';
33     local_26 = '2';
34     local_25 = '.';
35     local_24 = 'd';
36     local_22 = 'l';
37     local_23 = 'l';
38     local_21 = '\0';
39     hModule = LoadLibraryA(&local_2d);
40     local_2d = 'O';
41     local_24 = 'u';
42     local_29 = 'u';
43     local_2c = 'u';
44     local_21 = 't';
45     local_28 = 't';
46     local_2b = 't';
47     local_2a = 'p';
48     local_27 = 'D';
49     local_26 = 'e';
50     local_25 = 'b';
```

carattere, della libreria kernel32.dll e della funzione OutputDebugStringA. Il controllo della presenza del debugger con questa funzione, viene effettuato nel blocco WM_SIZE della WindowProcedure con la funzione FUN_00404000.

```
51    local_1d = 'g';
52    local_23 = 'g';
53    local_22 = 'S';
54    local_20 = 'r';
55    local_1f = 'i';
56    local_1e = 'n';
57    local_1c = 'A';
58    local_1b = '\0';
59    GetProcAddress(hModule,&local_2d);
60    return;
61  }
```

Nella WindowProcedure c'è inoltre la funzione FUN_00401dc0, rinominata AntiDebug3, nella quale, sempre per impedire il debugging dell'applicazione, viene cambiato il flag BeingDebugged nel segmento FS:[30h] ad offset 2.

Decompile: AntiDebug4 - (hw3.exe)

```
1
2  void __cdecl AntiDebug4(int *param_1)
3
4  {
5    int in_FS_OFFSET;
6
7    *param_1 = *param_1 + (uint)((*(uint *)(*(int *)(in_FS_OFFSET + 0x30) + 2) & 7) != 0);
8    return;
9  }
```

Come nello scorso homework, la funzione relativa allo shutdown del sistema viene settata in InitDS, chiamata nella TimerProcedure e fa riferimento a DAT_004040e0. Una volta modificato il dato in funzione, disassemblato e rimossi i '9a 42' presenti che confondono il disassemblatore, viene fuori la funzione di gestione una volta scaduto il timer. È possibile osservare infatti la funzione FUN_00404040 nella quale, una volta rimosse le istruzioni '9a 42', è presente l'ExitWindowsEx.
La cosa che risalta nella funzione è un do/while di 52 cicli, nel quale vengono fatti delle manipolazioni sulla funzione FUN_004050c0, la quale risulta impossibile da analizzare essendo incomprensibile.

Avendo individuato tutti i sistemi anti-debugging e non potendo più proseguire con l'analisi statica, è stato deciso di passare all'analisi dinamica dopo aver patchato una copia dell'eseguibile per rimuovere i vari riferimenti ai sistemi anti-debug appena analizzati.

Analisi dinamica

Passando ad OllyDbg sul codice patchato, è possibile procedere mettendo un breakpoint all'indirizzo 004019ed, ovvero il punto nella TimerProc dove si esegue il primo dei due controlli che permettono di entrare nella funzione di gestione allo scadere del timer.

```
004019e3 a1 30 70 40 00   MOV   EAX,[AppDS_check_Go_Stop]
004019e8 83 ec 10         SUB   ESP,0x10
004019eb 85 c0            TEST  EAX,EAX
004019ed 74 08            JZ    LAB_004019f7
004019ef 3b 35 2c 70 40 00 CMP  ESI,dword ptr [AppDS_shutdown_time]
004019f5 73 11            JNC   LAB_00401a08
```

Decompile: TimerProc - (hw3.exe)

```
1
2  void TimerProc(HWND param_1)
3
4  {
5    uint atmTime;
6    uint time;
7
8    atmTime = AppDS_time.time;
9    time = AppDS_time.time + 1;
10   AppDS_time.time = time;
11   if (AppDS_time.check_Go_Stop != 0) {
12     secondsLabel();
13   }
14   RedrawWindow(param_1,(RECT *)0x0,(HRGN)0x0,5);
15   if ((AppDS_time.check_Go_Stop != 0) && (AppDS_time.shutdown_time <= time)) {
16     (*(code *)AppDS_time.shutdown_function)(&AppDS_time);
17   }
18   if ((atmTime & 7) == 0) {
19     AntiDebug3((int)&AppDS_time);
20     return;
21   }
22   return;
23 }
```

Il jump nell'if, che non dovrebbe essere eseguito perchè nell'applicazione non è stato avviato il timer, può essere effettuato modificando manualmente il flag Z da 1 a 0 prima di effettuare l'istruzione all'indirizzo 004019ed e poi modificando il flag C da 1 a 0 per l'istruzione ad indirizzo 004019f5. In questo modo è possibile andare avanti e portare l'esecuzione alla funzione FUN_004040e0, ovvero la ShutdownFunction.
Andando avanti si procede fino ad entrare nel do/while visto precedentemente, nel quale viene effettuato lo shift del codice. Questa tecnica è stata utilizzata per rendere il codice

statico illeggibile. Terminato, si entra nella funzione FUN_004050c0, completamente differente da quella presente su Ghidra. Analizzando il codice si può vedere che vengono caricati man mano in EAX dei valori esadecimali con cui vengono eseguiti degli XOR e dei confronti con dei valori esadecimali prestabiliti.

Seguendo il flusso e prendendo nota dei valori trovati, è possibile eseguire gli XOR e convertire il risultato ottenuto da esadecimale in ASCII e arrivare al seguente risultato:

3F	28	2F	A5	5D	47	3D	4F	3F
XOR	XOR	XOR	XOR	XOR	XOR	XOR	XOR	XOR
0C	5A	61	C0	2E	13	0D	70	1E
=	=	=	=	=	=	=	=	=
33	72	4e	65	73	54	30	3f	21
↓	↓	↓	↓	↓	↓	↓	↓	↓
3	**r**	**N**	**e**	**s**	**T**	**0**	**?**	**!**

Provando ad inserire la stringa trovata nel programma originale è stato accertato che **3rNesT0?!** è effettivamente la stringa cercata.