

# Homework 1

Pierpaolo Spaziani - 0243651 (Matricola della triennale)

## Identificazione del main

Per l'analisi del programma il primo obiettivo posto è stato quello di **individuare il main**.

Cercando in "Imports → MSVCRT.DLL", è stato osservato che la `printf()` viene utilizzata solo nella funzione `FUN_00401790`, indizio che suggerisce che questa parte dell'applicazione è stata scritta dal programmatore e non dal sistema. Utilizza inoltre anche `puts()`, e l'unica altra funzione che ne fa uso è la `FUN_00401530`, quindi probabilmente entrambe sono state scritte dal programmatore.

Sono stati quindi esaminati i loro Function Call Graph: entrambe le funzioni vengono invocate da `FUN_00402a30`, questo porta a pensare che nessuna delle due sia il main ma che siano due funzioni invocate dal main.

Inoltre la funzione `FUN_00401790` ritorna un void e `FUN_00401530` ritorna un `LPSTR*`, punti a sfavore per essere riconosciute come main.

Guardando invece `FUN_00402a30` risulta interessante osservare che:

1. Viene chiamata direttamente da entry;
2. Nonostante non ne venga riconosciuto il valore di ritorno, la variabile che a fine funzione viene ritornata può valere 0 o ad 1;
3. Prende 2 parametri e nella funzione il secondo parametro viene utilizzato con offset multipli di 4, potrebbe quindi essere l'argv;
4. Viene utilizzata `atoi` sul secondo parametro.

Tutte queste osservazioni portano a pensare che questa funzione sia il main.

Si procede quindi provando a cambiare l'intestazione della funzione:

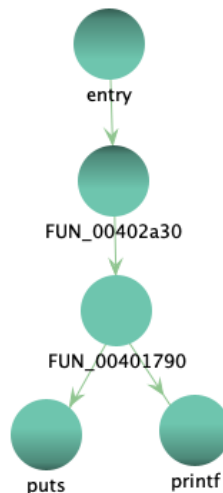
```
da: undefined4 FUN_00402a30 (int param_1, int param_2)
in : int main (int argc, char **argv)
```

Come ipotizzato, il secondo parametro che viene utilizzato con offset è l'argv che viene utilizzato nelle varie posizioni.

## Analisi del main

Una cosa inusuale che viene fuori è la presenza di un if/else che ha come condizione un controllo su `argv[argc]`, cosa apparentemente senza senso poiché l'ultimo parametro passato da riga di comando si trova in posizione `argc-1`. Tuttavia, sapendo che il programmatore non vorrebbe far eseguire il programma, si deduce che è un controllo che risulta sempre vero e che permette di rendere invisibile all'utente il vero funzionamento dell'applicazione, ovvero ciò che si trova nell'else.

Function Call Graph - FUN\_00401790



Function Call Graph - FUN\_00401530



Prima di analizzare il funzionamento del programma va detto che prima dell'if viene chiamata la funzione `FUN_004019a0` che è stata ritenuta una funzione di sistema principalmente per 2 motivi:

1. Viene invocata sia dal `main` che dalla `entry`, una funzione scritta dal programmatore dovrebbe essere invocata solo dal `main` o dalle funzioni che lui invoca;
2. Chiama la funzione `_onexit` che, come riporta la documentazione Microsoft, "registra una routine da chiamare in fase di uscita".

A questo punto il codice `main` che il Decompilatore suggerisce è il seguente:

```

C:\Decompile: main - (hw1.exe)
1
2 int __cdecl main(int argc, char **argv)
3
4 {
5     HKEY pHVar1;
6     LPSTR *ppCVar2;
7     char *pcVar3;
8     int iVar4;
9
10    FUN_004019a0();
11    if (argv[argc] == (char *)0x0) {
12        iVar4 = 0;
13    }
14    else {
15        if (argc < 3) {
16            argv[1] = (char *)0x0;
17            argv[2] = (char *)0x0;
18        }
19        pHVar1 = (HKEY)atoi(argv[1]);
20        if (pHVar1 == (HKEY)0x0) {
21            pHVar1 = (HKEY)0x80000002;
22        }
23        pcVar3 = argv[2];
24        if (pcVar3 == (LPCSTR)0x0) {
25            pcVar3 = "SYSTEM\\ControlSet001\\Control";
26        }
27        ppCVar2 = FUN_004018a0(pHVar1, pcVar3);
28        iVar4 = 1;
29        if (ppCVar2 != (LPSTR *)0x0) {
30            FUN_00401790(ppCVar2);
31            iVar4 = 0;
32        }
33    }
34    return iVar4;
35 }

```

Procedendo con l'analisi dell'applicazione risulta conveniente dare dei nomi più chiari alle funzioni ed alle variabili:

- `FUN_004019a0` → `system_func`
- `FUN_004018a0` → `func_1`
- `FUN_00401790` → `func_2`
- `iVar4` → `int_return`
- `pHVar1` → `hKey`
- `pcVar2` → `var_string`

```

C:\Decompile: main - (hw1.exe)
1
2 int __cdecl main(int argc, char **argv)
3
4 {
5     HKEY hKey;
6     LPSTR *ppCVar1;
7     char *var_string;
8     int int_return;
9
10    system_func();
11    if (argv[argc] == (char *)0x0) {
12        int_return = 0;
13    }
14    else {
15        if (argc < 3) {
16            argv[1] = (char *)0x0;
17            argv[2] = (char *)0x0;
18        }
19        hKey = (HKEY)atoi(argv[1]);
20        if (hKey == (HKEY)0x0) {
21            hKey = (HKEY)0x80000002;
22        }
23        var_string = argv[2];
24        if (var_string == (LPCSTR)0x0) {
25            var_string = "SYSTEM\\ControlSet001\\Contro
26        }
27        ppCVar1 = func_1(hKey, var_string);
28        int_return = 1;
29        if (ppCVar1 != (LPSTR *)0x0) {
30            func_2(ppCVar1);
31            int_return = 0;
32        }
33    }
34    return int_return;
35 }

```

Da questo codice si inizia ad individuare il filo logico che il programma eseguirebbe in assenza dell'if iniziale: c'è un controllo sul numero degli argomenti inseriti, due assegnazione particolari in determinati casi e poi l'invocazione delle funzioni `func_1` e `func_2`.

Si osserva che la variabile rinominata `hKey`, nel caso fosse 0, gli viene assegnato un numero che non sembra essere un decimale, non sembra essere una quantità che ha valore numerico. Vedendo l'assegnazione di tipo `HKEY` e provando a vedere quel valore a cosa possa essere associato con "Set Equate", si osserva che fa riferimento a `HKEY_LOCAL_MACHINE`, un registro di sistema che, come riporta la documentazione Microsoft, "contiene informazioni di configurazione specifiche per il computer". Inoltre si osserva anche che, se la variabile `var_string` è vuota gli viene assegnata la stringa "`SYSTEM\\ControlSet001\\Control`", una classe specifica di quel registro.

Si prosegue quindi analizzando la funzione `func_1`.

Per prima cosa si osserva il valore di ritorno, un `LPSTR*`, ci si aspetta quindi che restituisca un puntatore a puntatori di stringhe, una lista annidata.

I parametri che prende sono la `hKey` e la `var_string` di cui si è già discusso precedentemente.

Utilizza 3 funzioni:

- `RegOpenKeyExA` e `RegCloseKey` definite nella libreria di sistema Windows;
- `FUN_00401530` che è invece da analizzare, per comodità ridenominata `func_3`.

**RegOpenKeyExA:** apre la chiave del registro di sistema specificato.

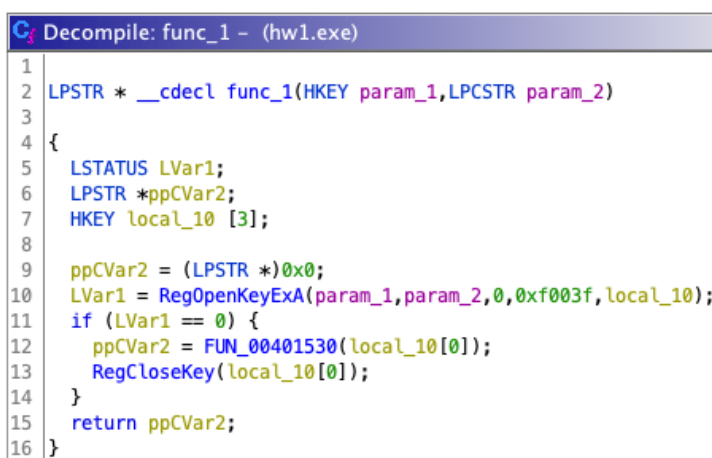
- I primi due parametri `param_1` e `param_2`, ovvero `hKey` e la `var_string`, sono rispettivamente l'handle per la chiave di registro e il nome della sottochiave del registro da aprire;
- il terzo parametro specifica le opzioni di apertura;
- il quarto parametro con "Set Equate" può essere associato a `KEY_ALL_ACCESS`, come descritto nella documentazione Microsoft;
- il quinto è il puntatore alla variabile che riceve l'handle per la chiave aperta.

**RegCloseKey:** chiude un handle per la chiave di registro di sistema specificata.

**func\_3:** prende come parametro l'handle per la chiave aperta da `RegOpenKeyExA` e utilizza principalmente 3 funzioni definite nella libreria di sistema Windows:

- `RegQueryInfoKeyA` che recupera le informazioni sulla chiave di registro di sistema;
- `RegEnumKeyExA` che enumera le sottochiavi della chiave di registro di sistema;
- `RegEnumValueA` che enumera i valori per la chiave di registro di sistema.

È proprio la variabile di ritorno di `func_3` che viene ritornata anche da `func_1` e, come si ipotizzava, è proprio una lista annidata con tutte le informazioni della chiave di registro di sistema. Infatti, come si può vedere sia dalle funzioni utilizzate che dal codice, `func_3` spacchetta tutte le informazioni della chiave di registro e le inserisce nella variabile di



```

C: Decompiler: func_1 - (hw1.exe)
1
2 LPSTR * __cdecl func_1(HKEY param_1,LPCSTR param_2)
3
4 {
5     LSTATUS LVar1;
6     LPSTR *ppCVar2;
7     HKEY local_10 [3];
8
9     ppCVar2 = (LPSTR *)0x0;
10    LVar1 = RegOpenKeyExA(param_1,param_2,0,0xf003f,local_10);
11    if (LVar1 == 0) {
12        ppCVar2 = FUN_00401530(local_10[0]);
13        RegCloseKey(local_10[0]);
14    }
15    return ppCVar2;
16 }
  
```

uscita, ovvero quella su cui vengono allocati 52 spazi di memoria per puntatori a stringhe da una malloc. Si potrebbe pensare ad una struttura, tuttavia le funzioni che vengono utilizzate non prendono in ingresso quelle definite dal sistema Windows. Sapendo che la variabile è stata popolata da funzioni note, anche se sono state create e utilizzate strutture dal programmatore, si può evitare di definirle al momento dell'analisi di questa applicazione, poiché basta far riferimento alla documentazione delle funzioni utilizzate per sapere le informazioni ottenute a cosa si riferiscono.

Si torna quindi nel `main` e si osserva che, se il puntatore ritornato dalla `func_1` non è nullo, si entra nella `func_2`.

Questa funzione è la prima che era stata presa in considerazione nell'analisi per la presenza dei `printf`. Adesso, sapendo come è fatto il parametro che prende in ingresso, è facile capire che quello che fa è stampare su standard output tutte le informazioni della chiave di registro analizzata precedentemente.

Si vede infatti come nelle `printf` vengono specificate tutte le informazioni della chiave:

- "Class ..."
- "Security descriptor..."
- "Time ..."
- "Sub-keys ..."
- "Values ..."

In particolare, il `printf` associato alla stringa "Security descriptor" ha come parametro l'ottava posizione del puntatore. Quando `func_3` precedentemente invoca `RegQueryInfoKeyA`, come 11esimo parametro usa proprio 8 come offset; andando a vedere la documentazione della funzione, si può vedere che l'11esimo parametro è

"lpcbSecurityDescriptor", ovvero il puntatore a una variabile che riceve la dimensione del descrittore di sicurezza della chiave.

Stesso discorso può essere fatto per il 12esimo parametro. Viene utilizzato con offset 9 e, nella documentazione, corrisponde a "lpftLastWriteTime", il puntatore a una struttura `FILETIME` che riceve l'ultima ora di scrittura. Infatti nella `func_2` viene stampato con la stringa "Time" con posizione 9 e 10 (essendo un valore a 64 bit occupa 2 posizioni). Allo stesso modo, la posizione 11 nella variabile viene popolata da `RegEnumKeyExA` con i puntatori alle informazioni delle sottochiavi, come nome, dimensione ecc., e la posizione 12 da `RegEnumValueA` con i puntatori ai valori delle sottochiavi.

```

C: Decompile: func_2 - (hw1.exe)
1
2 void __cdecl func_2(LPSTR *param_1)
3
4 {
5     int iVar1;
6     LPSTR pCVar2;
7     uint uVar3;
8     LPSTR pCVar4;
9     undefined4 uVar5;
10
11     printf("Class: %s\n", *param_1);
12     printf("Security descriptor: 0x%lx\n", param_1[8]);
13     pCVar4 = param_1[10];
14     printf("Time: %08lx%08lx\n", param_1[9], pCVar4);
15     pCVar2 = param_1[11];
16     if (pCVar2 != (LPSTR)0x0) {
17         puts("Sub-keys:");
18         do {
19             printf("\t%s\n", *(undefined4 *) (pCVar2 + 8), pCVar4);
20             pCVar2 = *(LPSTR *) (pCVar2 + 4);
21         } while (pCVar2 != (LPSTR)0x0);
22     }
23     pCVar4 = param_1[12];
24     if (pCVar4 != (LPSTR)0x0) {
25         puts("Values:");
26         do {
27             uVar5 = *(undefined4 *) (pCVar4 + 0x400c);
28             printf("\t%s: [%lu] ", pCVar4 + 8, uVar5);
29             if (*(int *) (pCVar4 + 0x4110) != 0) {
30                 uVar3 = 0;
31                 do {
32                     iVar1 = uVar3 + 0x4010;
33                     uVar3 = uVar3 + 1;
34                     printf(" %02x", (uint)(byte)pCVar4[iVar1], uVar5);
35                 } while (uVar3 <= *(uint *) (pCVar4 + 0x4110) && *(u
36             )
37             printf(" {%s}\n", pCVar4 + 0x4010, uVar5);
38             pCVar4 = *(LPSTR *) (pCVar4 + 4);
39         } while (pCVar4 != (LPSTR)0x0);
40     }
41     return;
42 }
43

```

```

RegQueryInfoKeyA(param_1, pCVar2, (LPDWORD)(ppCVar1 + 1), (LPDWORD)0x0,
(LPDWORD)(ppCVar1 + 2), (LPDWORD)(ppCVar1 + 3),
(LPDWORD)(ppCVar1 + 4), (LPDWORD)(ppCVar1 + 5),
(LPDWORD)(ppCVar1 + 6), (LPDWORD)(ppCVar1 + 7),
(LPDWORD)(ppCVar1 + 8), (PFILETIME)(ppCVar1 + 9));

```

Il printf associato alla stringa "Class" invece, stampa la classe della chiave che è stata analizzata. Corrisponde, se inserita dall'utente, a quella specificata in argv[2], altrimenti viene utilizzata "SYSTEM\\ControlSet001\\Control".

La funzione poi ritorna al main, il quale imposta la variabile di ritorno a 0 e termina l'esecuzione del programma.

## Conclusione

Ciò che è stato appreso dell'applicazione, tralasciano la parte che ne fa terminare subito l'esecuzione, è che, lanciandola senza argomenti, accede al registro di sistema "HKEY\_LOCAL\_MACHINE\\SYSTEM\\ControlSet001\\Control" e stampa su standard output tutte le relative informazioni di chiave, sottochiavi e valori, ritornando 0. Invece, in caso di avvio con l'inserimento da parte dell'utente di una coppia di argomenti validi, ovvero una HKEY e una classe della chiave, l'applicazione procede con la stessa esecuzione ma relativamente al registro di sistema specificato. In caso non venga trovato un registro di sistema valido l'esecuzione termina ritornando il valore 1.