Computer Security HW1 Write-Up

Account: soyccan
Name: 陳正康
Student ID: B07902143

Part 1

使用 ghidra 進行靜態分析

```
0040166d a1 6c d1        MOV       EAX,[->KERNEL32.DLL::GetModuleHandleA]          = 0000d2f2
         40 00
00401672 ff d0           CALL      EAX=>KERNEL32.DLL::GetModuleHandleA
00401674 83 ec 04        SUB       ESP,0x4
00401677 89 45 e8        MOV       dword ptr [EBP + Stack[-0x20]],EAX              moduleHandle
                                                                                   == imageBaseAddr
                                                                                   == 0x400000
0040167a 8b 45 e8        MOV       EAX,dword ptr [EBP + Stack[-0x20]]
0040167d 8b 40 3c        MOV       EAX,dword ptr [EAX + 0x3c]                      *(imageBaseAddr + 0x3c)
                                                                                   == newHeaderAddr (IMAGE_NT_HEADER)
                                                                                   == 0x400080
00401680 89 c2           MOV       EDX,EAX
00401682 8b 45 e8        MOV       EAX,dword ptr [EBP + Stack[-0x20]]
00401685 01 d0           ADD       EAX,EDX
00401687 89 45 e4        MOV       dword ptr [EBP + Stack[-0x24]],EAX              imageNTHeader
0040168a c7 45 dc        MOV       dword ptr [EBP + Stack[-0x2c]],0x30
         30 00 00 00
```

```
00401666  MOV    dword ptr [ESP]=>param0,0x0

0040166d  MOV    EAX,[->KERNEL32.DLL::GetModuleHandleA]  = 0000d2f2

00401672  CALL   EAX=>KERNEL32.DLL::GetModuleHandleA
00401674  SUB    ESP,0x4
00401677  MOV    dword ptr [EBP + Stack[-0x20]],EAX      moduleHandle
                                                        == imageBaseAddr
                                                        == 0x400000
```

從 `GetModuleHandleA` 的回傳值看出[ebp-0x20]存放著IMAGE BASE ADDRESS

```
0040167a  MOV    EAX,dword ptr [EBP + Stack[-0x20]]
0040167d  MOV    EAX,dword ptr [EAX + 0x3c]              *(imageBaseAddr + 0x3c)
                                                         == newHeaderAddr
                                                         == imageNTHeader
                                                         == 0x400080
00401680  MOV    EDX,EAX
00401682  MOV    EAX,dword ptr [EBP + Stack[-0x20]]
00401685  ADD    EAX,EDX
00401687  MOV    dword ptr [EBP + Stack[-0x24]],EAX      imageNTHeader
```

看到 `imageBaseAddr + 0x3c` 的地方
ghidra的header有寫:

```
0040003c 80 00 00 00     ddw       80h                    e_lfanew      File ad
```

或是從IMAGE_DOS_HEADER的定義:

```
typedef struct _IMAGE_DOS_HEADER
{
     WORD e_magic;
     WORD e_cblp;
     WORD e_cp;
     WORD e_crlc;
     WORD e_cparhdr;
     WORD e_minalloc;
     WORD e_maxalloc;
     WORD e_ss;
     WORD e_sp;
     WORD e_csum;
     WORD e_ip;
     WORD e_cs;
     WORD e_lfarlc;
     WORD e_ovno;
     WORD e_res[4];
     WORD e_oemid;
     WORD e_oeminfo;
     WORD e_res2[10];
     LONG e_lfanew;
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```

都可看出 `*(imageBaseAddr + 0x3c)` 是指向PE header的位置
型態是 `IMAGE_NT_HEADER`
用x64dbg看是0x400080
開頭是 `"PE"` 沒錯

conclusion

- `[ebp - 0x24]: imageNTHeader`

Part 2

```
0040168a  MOV    dword ptr [EBP + Stack[-0x2c]],0x30
00401691  MOV    EAX,dword ptr [EBP + Stack[-0x2c]]
00401694  MOV    EAX,dword ptr FS:[EAX]                fs:[0x30]
                                                       == PEB
00401697  MOV    dword ptr [EBP + Stack[-0x30]],EAX
0040169a  MOV    EAX,dword ptr [EBP + Stack[-0x30]]
0040169d  MOV    dword ptr [EBP + Stack[-0x28]],EAX
```

以上這段把fs:[0x30]的位置存進[ebp-0x30]和[ebp-0x28]
查了一下: https://en.wikipedia.org/wiki/Win32_Thread_Information_Block
(https://en.wikipedia.org/wiki/Win32_Thread_Information_Block)
fs:[0x30]在32-bit Windows裡是指向Process Environment Block (PEB)

conclusion

- `[ebp-0x30] and [ebp-0x28]: PEBAddr`

Part 3

```
004016d5   MOV        EAX,dword ptr [EBP + Stack[-0x24]]
004016d8   MOV        EAX,dword ptr [EAX + 0x8]
004016db   MOV        dword ptr [ESP]=>param0,EAX
004016de   CALL       FUN_00401600

004016e3   MOV        [DAT_0040c040],EAX                                = ??

004016e8   MOV        EAX,[DAT_0040c040]                                = ??

004016ed   MOV        dword ptr [ESP + param1],EAX
004016f1   MOV        dword ptr [ESP]=>param0,s_[+]_It's_a_time_mach   = "[+] It

004016f8   CALL       printf                                            int print

004016fd   MOV        EAX,[DAT_0040c040]                                = ??

00401702   CMP        EAX,1985

00401707   JZ         LAB_00401715
00401709   MOV        dword ptr [ESP]=>param0,s_[!]_WARNING:_it_migh   = "[!] WA

00401710   CALL       puts                                              int puts(

00401715   MOV        EAX,dword ptr [EBP + PEB_]
00401718   MOVZX      EAX,byte ptr [EAX + 0x2]
0040171c   TEST       AL,AL
0040171e   JZ         LAB_00401727
00401720   MOV        EAX,s_[HARMFUL!]_00409190                         = "[HARMF

00401725   JMP        LAB_0040172c
```

以上這段呼叫 `FUN_00401600` ，參數是 `*(imageNTHeader + 0x8)` ，回傳值放
在 `DAT_0040c040` 這個全域變數
如果 `DAT_0040c040` =1985: 輸出"[SAFE]"
否則: 輸出"[HARMFUL]"

conclusion

`FUN_00401600` 應該是個吐出年分的函數
`DAT_0040c040` 存年分

Part 4: FUN_00401600

ghidra直接decompile

```
int FUN_00401600(undefined1 param_1)
{
  tm *ptVar1;

  ptVar1 = localtime((time_t *)&param_1);
  return ptVar1->tm_year + 1900;
}
```

它把參數視為 time_t 型態並取出年分

觀察Part 3說的參數:

`*(imageNTHeader + 0x8)`

查閱:

```
typedef struct _IMAGE_NT_HEADERS {
  DWORD                   Signature;
  IMAGE_FILE_HEADER       FileHeader;
  IMAGE_OPTIONAL_HEADER32 OptionalHeader;
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;
typedef struct _IMAGE_FILE_HEADER {
  WORD  Machine;
  WORD  NumberOfSections;
  DWORD TimeDateStamp;
  DWORD PointerToSymbolTable;
  DWORD NumberOfSymbols;
  WORD  SizeOfOptionalHeader;
  WORD  Characteristics;
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

得到0x8的地方是 TimeDateStamp ，型態是 time_t (UNIX Epoch time)
故等下把這裡patch成1985年

ghidra:

```
00400088 89 64 8c 5d     ddw       5D8C6489h              TimeDateStamp
0040008c 00 00 00 00     ddw       0h                     PointerToSym...
00400090 00 00 00 00     ddw       0h                     NumberOfSymb...
```

Part 5

```
printf("[!] Time Machine Guarder: %s\n");
printf("[+] input password to launch time machine: ");
gets(&password);
i = 0;
while (sVar3 = strlen(&password), i < sVar3) {
  (&password)[i] = (&password)[i] | 0x20;
  i = i + 1;
}
printf("[!] reading ... the.... passw0r..d.....\n");
j = 0;
while (j < 19) {
  (&password)[j] = (&password)[j] ^ *(char *)(iVar1 + 2) + ((char)DAT_0040c040 +
  ;
  if ((&password)[j] != (&gblPass)[j]) {
    puts("[!] oops... time machine g0t some trouble in the 0ld tim3... ");
    break;
  }
  j = j + 1;
}
k = 0;
while (k < 19) {
  (&password)[k] = (&password)[k] ^ (&gblXorMask)[k];
  k = k + 1;
}
printf("[+] a flag found by time machine at %i:\n\t%s\n");
```

接下來是讀入密碼的部分

第一步先OR 0x20

第二步:

```
password[j] ^= *(iVar1+2) + ((char)DAT_0040c040 + 63) * 2 + 127U
```

第三步:

```
password[k] ^= gblXorMask[k]
```

由Part 3知 DAT_0040c040 存年分，且將被patch成1985

接下來看iVar1，組語相關的部分有:

```
004017bf 8b 45 e0        MOV        EAX,dword ptr [EBP + Stack[-0x28]]
004017c2 0f b6 40 02     MOVZX      EAX,byte ptr [EAX + 0x2]
```

由Part 2知[ebp-0x28]是 PEBAddr

iVar1 = *(PEBAddr + 0x2)

查閱:

```
typedef struct _PEB32 {
UCHAR InheritedAddressSpace;
UCHAR ReadImageFileExecOptions;
UCHAR BeingDebugged;
UCHAR BitField;
ULONG Mutant;
ULONG ImageBaseAddress;
PPEB_LDR_DATA Ldr;
ULONG ProcessParameters;
ULONG SubSystemData;
ULONG ProcessHeap;
ULONG FastPebLock;
ULONG AtlThunkSListPtr;
ULONG IFEOKey;
ULONG CrossProcessFlags;
ULONG UserSharedInfoPtr;
ULONG SystemReserved;
ULONG AtlThunkSListPtr32;
ULONG ApiSetMap;
} PEB32, *PPEB32;
```

知道此欄位是 `BeingDebugged` ，在沒用debugger時iVar1應為0

conclusion

after:

```
password[j] |= 0x20
password[j] ^= 0 + (1985 + 63) * 2 + 127
password[j] ^= gblXorMask[j]
=> password == flag
```

Solution

Time Travel

Python:

```
>>> hex(int(datetime.datetime(year=1985, month=1, day=1).timestamp()))
'0x1c36da00'
```

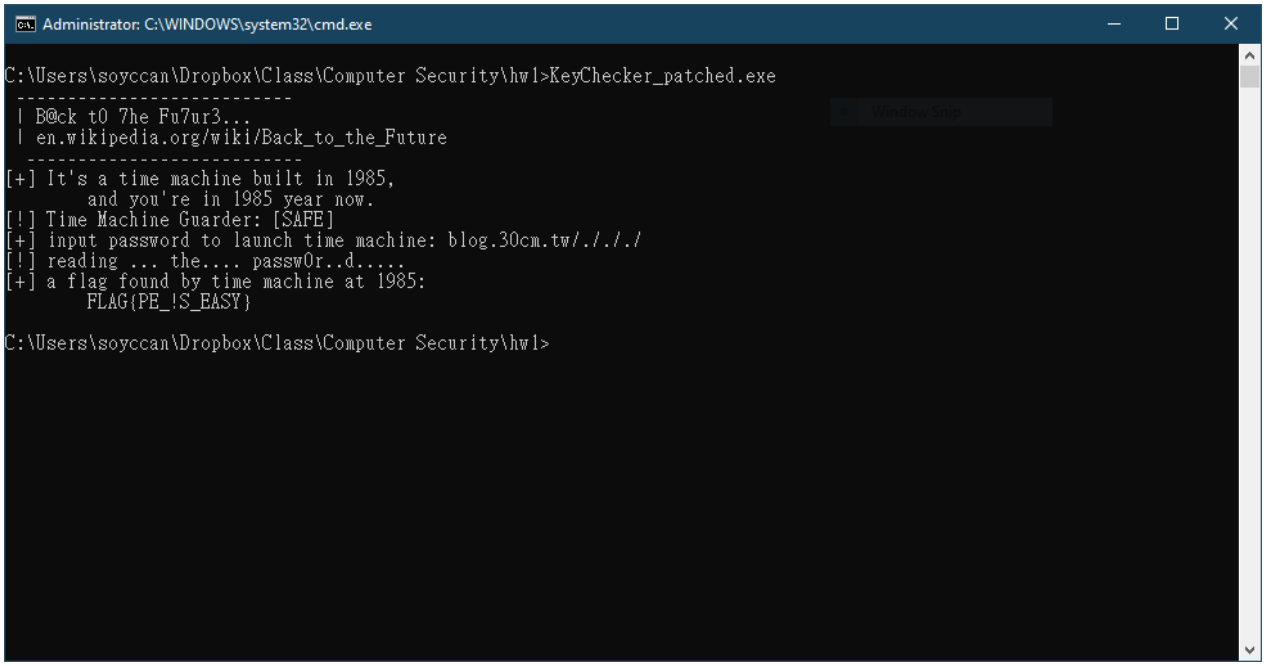
把exe檔的0x88位址patch成0x1c36da00

Recover Password

把gblPass: 0x408008和gblXorMask: 0x40801c
給dump出來，即可逆推密碼
詳見 `sol.py`

- 密碼:
  blog.30cm.tw/./../ (http://blog.30cm.tw/./../)

- flag:
  FLAG{PE_!S_EASY}



Further Study

- 為何Patch過的程式要用admin權限才能跑
  - 是否是Windows某種保護機制?