Write-up cho Chal6 của Flare-on4.

1. Thu thập thông tin

- Target nhận được từ website là một file dll: `payload.dll`

- Kiểm tra thông tin sơ bộ:
  - Định dạng file là **PE+(64)**
  - Resource của file không có thông tin gì đặc biệt.
  - Các hàm API được Import chủ yếu từ `kernel32.dll` và `user32.dll`

Offset	Name	Func. Count	Bound?	OriginalFirstThun	TimeDateStamp	Forwarder	NameRVA	FirstThunk
1704C	KERNEL32.dll	69	FALSE	18088	0	0	18312	10000
17060	USER32.dll	1	FALSE	182B8	0	0	1832E	10230

  - Thông tin liên quan tới Export function:

    `Ordinal: 1, Function Name: EntryPoint, RVA: 0x00005C00`

Offset	Ordinal	Function RVA	Name RVA	Name	Forwarder
17028	1	5C00	1803E	EntryPoint	

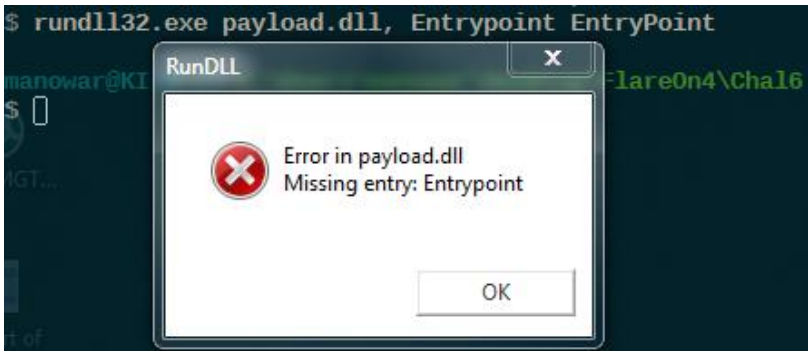
  - Về string, có một số string quan trọng như sau:

    `“Insert clever error message here!”`

    `“rundll32 payload.dll, EntryPoint EntryPoint”` <-- String này gợi ý cách thức gọi hàm mà dll này đã export ở trên.

2. Chạy thử payload.dll

Thử chạy `payload.dll` với thông tin gợi ý ở trên:

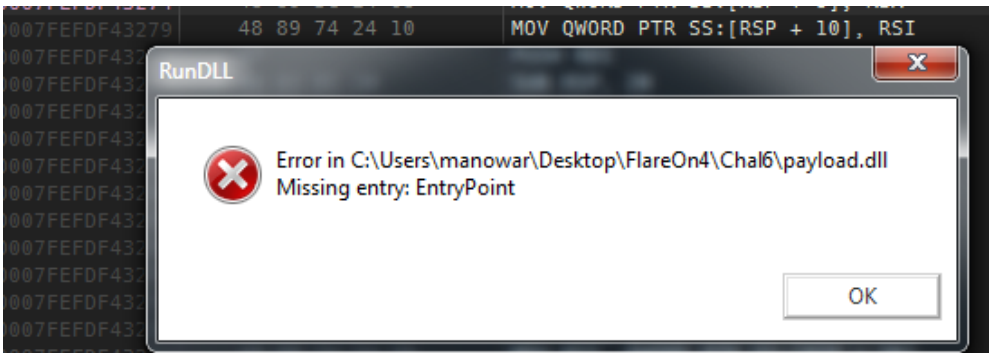


Nhận được thông báo "**Missing entry: Entrypoint**". Như vậy, nghi ngờ khả năng là hàm `Entrypoint` mà dll này export đã bị thay đổi khi chạy `payload.dll`.

Để kiểm tra xem có đúng là export function bị thay đổi hay không, sử dụng **x64dbg** để debug:

- Load `C:\Windows\System32\rundll32.exe` vào x64dbg.
- Chỉnh lại tham số truyền vào tại `Debug > Change Command Line`. Sửa tương tự như sau: "`"C:\Windows\System32\rundll32.exe" C:\Users\manowar\Desktop\FlareOn4\Chal6\payload.dll, EntryPoint EntryPoint`"
- Chỉnh lại `Options > Preferences, Settings > Envents`, chọn DLL Load & DLL Entry

Sau đó nhấn **F9** để run và quan sát kĩ khi `payload.dll` được load lên, stop tại `DLLMain` và thực thi hoàn toàn:

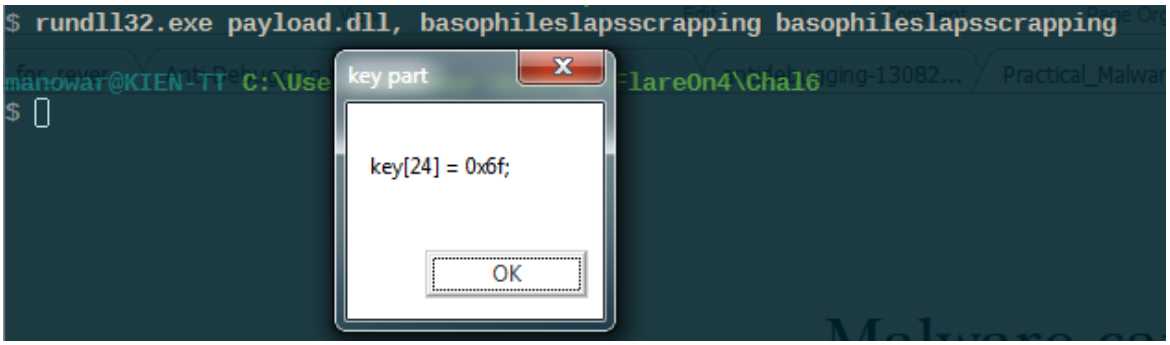


Giữ nguyên thông báo trên, sử dụng plugin là **OllyDumpEx** để thực hiện dump toàn bộ `payload.dll`, lưu lại với tên mới là `payload_dump_64.dll`. Kiểm tra lại bằng **CFF Explorer**, thấy thông tin về Export function bị thay đổi như sau:

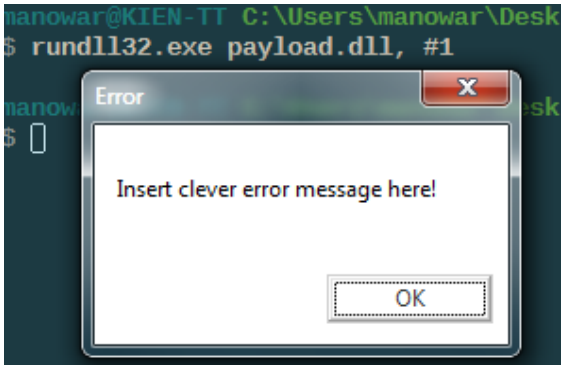
Ordinal	Function RVA	Name Ordinal	Name RVA	Name
(nFunctions)	Dword	Word	Dword	szAnsi
00000001	00005A50	0000	0000403D	basophileslapsscrapping

*Lưu ý: Ordinal vẫn là **1**, như vậy kết luận hàm `Entrypoint` đã bị đổi tên thành **`basophileslapsscrapping`**.*

Với thông tin có được như trên, thử chạy lại với lệnh như sau: **rundll32.exe payload.dll, basophileslapsscrapping basophileslapsscrapping**. Kết quả, nhận được một thông báo kí tự thứ 24 của flag là (**0x6f -> ASCII: 'o'**):



Ngoài ra, trong quá trình chạy thử, thử run `payload.dll` theo kiểu gọi ordinal: `rundll32.exe payload.dll, #1` thì nhận được thông báo sau:



3. Phân tích và debug

Load `payload.dll` vào IDA, đầu tiên chuyển tới export function: `EntryPoint` tại **0x0000000180005C00**. Đoạn code tại đây thực hiện việc hiện thị thông báo hướng dẫn cách sử dụng:

```
// STR: "rundll32 payload.dll, EntryPoint EntryPoint", "Usage"
signed __int64 __stdcall EntryPoint()
{
  MessageBoxA(0i64, "rundll32 payload.dll, EntryPoint EntryPoint", "Usage", 0);
  return 1i64;
}
```

Tiếp tục đi theo hướng string **"Insert clever error message here!"**, tới code tại **sub_180005A50**. Song song với quá trình phân tích code bằng IDA, tôi cũng sử dụng x64dbg để load `payload.dll` với tham số truyền vào ("C:\Windows\System32\rundll32.exe" C:\Users\manowar\Desktop\FlareOn4\Chal6\payload.dll, basophileslapsscrapping basophileslapsscrapping), đặt breakpoint tại lệnh:

```
0000000180005A50 mov dword ptr ss:[rsp+0x20], r9d
```

Sau quá trình try hard phân tích và debug code, có được thông tin như sau:

```
int __fastcall sub_180005A50(__int64 a1, __int64 a2, __int64 a3)
{
  _DWORD *export_dir; // rax@1
  __int64 img_export_dir; // ST48_8@1
```

```
unsigned int export_name_VA; // ST38_4@1
char *lpexport_name; // rax@1
__int64 k; // rcx@1
unsigned __int8 chr; // dl@2
int flag; // eax@4
int result; // eax@9
int index_val; // [sp+20h] [bp-168h]@1
signed __int64 len_key; // [sp+28h] [bp-160h]@7
LPVOID lpAddress; // [sp+30h] [bp-158h]@7
__int64 export_func_name; // [sp+40h] [bp-148h]@1
signed __int64 size; // [sp+50h] [bp-138h]@1
DWORD flOldProtect; // [sp+58h] [bp-130h]@7
_DWORD *export_dir_copy; // [sp+60h] [bp-128h]@1
__int64 v18; // [sp+68h] [bp-120h]@7
Rc4Context *context; // [sp+70h] [bp-118h]@9
__int64 v20; // [sp+190h] [bp+8h]@1
__int64 v21; // [sp+198h] [bp+10h]@1
__int64 lpKey; // [sp+1A0h] [bp+18h]@1

lpKey = a3;
v21 = a2;
v20 = a1;
export_dir = (_DWORD *)Get_ExportDirectory(); // Get Export Directory's addr
export_dir_copy = export_dir;
img_export_dir = *export_dir + Dos_Header;    // Get Export Directory Table data
(Export_Dir's RVA + DOS Header)
export_name_VA = *(_DWORD *)(Dos_Header + *(_DWORD *)(img_export_dir + 0x20));//
Get RVA that points to export function name
export_func_name = export_name_VA + Dos_Header;// function name
index_val = *(_DWORD *)(img_export_dir + 4) & 0xFF;// index_val = TimeDateStamp &
0xFF
size = (_BYTE *)*(&off_1800198E0 + (unsigned int)(index_val + 1))
     - (_BYTE *)*(&off_1800198E0 + (unsigned int)index_val);// off_1800198E0 in
.data section
lpexport_name = (char *)(export_name_VA + Dos_Header);
k = lpKey - (_QWORD)lpexport_name;
// Cause command is rundll32 payload.dll, EntryPoint EntryPoint
// This loop checks if the two strings are equal, Export function equals Key (eg.
EntryPoint = EntryPoint)
while ( 1 )
{
  chr = *lpexport_name;
  if ( *lpexport_name != lpexport_name[k] )
  {
    break;
  }
```

```
        ++lpexport_name;
        if ( !chr )
        {
          flag = 0;
          goto LABEL_6;
        }
      }
      flag = -(chr < (unsigned __int8)lpexport_name[k]) | 1;
    LABEL_6:
      if ( flag )
      {
        MessageBoxA(0i64, "Insert clever error message here!", "Error", 0);
        result = 0;
      }
      else
      {
        lpAddress = *(&off_1800198E0 + (unsigned int)index_val);
        VirtualProtect(*(&off_1800198E0   +   (unsigned   int)index_val),   0x1000ui64,
PAGE_EXECUTE_READWRITE, &flOldProtect);// Changes the protection on a region in .text
section to ERW; size: 0x1000
        v18 = export_func_name;
        len_key = 0xFFFFFFFFFFFFFFFFi64;
        do
        {
          ++len_key;
        }
        while ( *(_BYTE *)(v18 + len_key) );

        // Initialize an RC4 context using the supplied key
        // @param[in] context Pointer to the RC4 context to initialize
        // @param[in] key Pointer to the key
        // @param[in] length Length of the key
        // @return Error code
        rc4Init((Rc4Context *)&context, export_func_name, len_key);

        // Encrypt/decrypt data with the RC4 algorithm
        // @param[in] context Pointer to the RC4 context
        // @param[in] input Pointer to the data to encrypt/decrypt
        // @param[in] output Pointer to the resulting data
        // @param[in] length Length of the input data
        rc4Cipher((Rc4Context *)&context, (__int64)lpAddress, (__int64)lpAddress, size);
        result = ((int (__fastcall *)(__int64, __int64, __int64, _QWORD))lpAddress)(v20,
v21, lpKey, (unsigned int)size);// Call to decrypted payload
      }
      return result;
    }
```

Tổng kết lại, **sub_180005A50** này thực hiện công việc như sau:

- B1: Lấy tên của hàm được export bởi `payload.dll`.
- B2: Tính toán một giá trị **index** dựa trên thông tin **TimeDateStamp** (*Giá trị **TimeDateStamp** này sẽ khác nhau và ứng với từng tên hàm được export*). **Index = TimeDateStamp & 0xFF**
- B3: Tính ra một giá tri **size**, giá trị này sẽ được sử dụng như là kích thước cho vùng dữ liệu sẽ được giải mã bởi thuật toán RC4.
- B4: Thực hiện vòng lặp để kiểm tra xem tên của hàm được export có giống với key truyền vào hay không. Vì theo hướng dẫn ban đầu, phải chạy `payload.dll` theo cú pháp lệnh (**rundll32 payload.dll, EntryPoint EntryPoint**), do đó nếu không giống nhau thì sẽ hiện thị thông báo **"Insert clever error message here!"**.
- B5: Dựa trên giá trị **index** có được ở B2, sẽ cộng với **off_1800198E0** để lấy ra vị trí địa chỉ trong vùng **.text** để làm nơi decrypt dữ liệu. Thông qua API là **VirtualProtect**, vùng nhớ này được thay đổi chế độ bảo vệ thành **PAGE_EXECUTE_READWRITE** với kích thước là **0x1000**.
- B6: Khởi tạo một RC4 context sử dụng key được cung cấp (key này trùng với tên của hàm được export).
- B7: Sử dụng **context** đã khởi tạo để giải mã vùng dữ liệu có được ở B5 với kích thước **size** đã tính toán ở B3.
- B8: Cuối cùng là nhảy tới vùng code đã decrypt và thực thi payload tại đây để hiển thị thông báo về kí tự thứ bao nhiêu trong flag có mã hexa là gì.

Tuy nhiên, suy nghĩ một chút thì thấy **sub_180005A50** đã phân tích ở trên là hàm xử lý sau khi hàm `EntryPoint` ban đầu bị thay thế bằng một tên hàm khác. Như vậy, đoạn code thực hiện việc thay thế một tên hàm mới đã được thực hiện trước đó rồi. Làm sao để tìm được đoạn code này?

Lại tiếp tục suy nghĩ, do liên quan đến việc lấy thông tin về export function, tại **sub_180005A50** đã phân tích ở trên có **sub_180004760** (đã được đổi tên thành **Get_ExportDirectory**):

```
export_dir = (_DWORD *)Get_ExportDirectory(); // Get Export Directory's addr
```

```
__int64 __stdcall Get_ExportDirectory()
{
  return NTHeader_signature + 0x88;           // return Export Directory's addr
}
```

Đặt mình vào vị trí của người code thì thường sẽ viết một hàm chung để thực hiện một task nhất định. Do đó, khả năng hàm **Get_ExportDirectory()** này sẽ được gọi ở đâu đó nữa. Sử dụng chức năng **xrefs** của IDA, tìm được nơi gọi tới hàm này tại **sub_180005D30**.

Đặt bp tại:

```
0000000180005D30    mov qword ptr ss:[rsp+0x8], rcx
```

Tiếp tục quá trình try hard phân tích code và debug tại **sub_180005D30**, có được thông tin như sau:

```
__int64 __fastcall sub_180005D30(__int64 a1)
{
```

```
__int64 addr_180005D4D; // rax@1
HANDLE hProcess; // rax@5
__int64 exp_directory; // ST40_8@5
unsigned int index; // ST38_4@5
_WORD *lpAddress; // [sp+30h] [bp-48h]@1
DWORD flOldProtect; // [sp+58h] [bp-20h]@5
__int64 v8; // [sp+80h] [bp+8h]@1

v8 = a1;
LODWORD(addr_180005D4D) = sub_180005E70();    // get addr 0x180005D4D
// Retrieve ImageBase & PE headear address of payload.dll when mapping to memory
for ( lpAddress = (_WORD *)(addr_180005D4D & 0xFFFFFFFFFFFFF000ui64);
      !(unsigned int)GetDOSHeader(lpAddress) || !(unsigned int)GetPEHeader((__int64)lpAddress);
      lpAddress += 0xFFFFF800 )
{
  ;
}
hProcess = GetCurrentProcess();
// lpAddress -> ImageBase (0x180000000) of payload.dll
// Change protection to Execute_Read_Write mode
VirtualProtectEx(hProcess, lpAddress, 0x1000ui64, PAGE_EXECUTE_READWRITE, &flOldProtect);
exp_directory = Get_ExportDirectory();
index = GetIndexFromTime();
DecryptExportFunction(index);           // Decrypt data base on index value, set new TimeDateStamp and new export function
*(_DWORD *)exp_directory = (index << 9) + (unsigned __int64)byte_180001000 - (_DWORD)lpAddress;// set new RVA of IMAGE_EXPORT_DIRECTORY
*(_DWORD *)(exp_directory + 4) = 0x200;   // set new size of IMAGE_EXPORT_DIRECTORY
return v8;
}
```

Tổng kết, **sub_180005D30** thực hiện nhiệm vụ sau:

- B1: Thực hiện vòng lặp, để lấy thông tin liên quan tới ImageBase và PE header của `payload.dll` khi đã mapping vào memory.
- B2: Gọi API **VirtualProtectEx** để thay đổi protection của vùng nhớ tại ImageBase thành **PAGE_EXECUTE_READWRITE**.
- B3: Gọi **Get_ExportDirectory()** để lấy thông tin về Export Directory.
- B4: Tính toán một giá trị **index** dựa trên SystemTime, **index = (SystemTime.wYear + SystemTime.wMonth) % 26**. Qua đó, có thể kết luận giá trị **index** sẽ nằm trong khoảng từ **0 -> 25**.
- B5: Căn cứ trên **index** tính toán được, gọi hàm **DecryptExportFunction(index)** để giải mã dữ liệu, và thiết lập lại giá trị **TimeDateStamp** và tên của export function mới.
- B6: Cuối cùng điều chỉnh lại giá trị RVA và size mới của **IMAGE_EXPORT_DIRECTORY**.

Sau khi có được các thông tin mới về **TimeDateStamp** và tên export function mới, sẽ sử dụng tại **sub_180005D30** để giải mã ra tương ứng với vị trí **index** trong flag.

Như vậy, nếu thay thủ công từng giá trị **index** trả về của hàm **GetIndexFromTime()**, trong khoảng từ **0 – 25**, sẽ có được toàn bộ các export function như sau:

```
index[0] = filingmeteorsgeminately
index[1] = leggykickedflutters
index[2] = incalculabilitycombustionsolvency
index[3] = crappingrewardsanctity
index[4] = evolvablepollutantgavial
index[5] = ammoniatesignifiesshampoo
index[6] = majesticallyunmarredcoagulate
index[7] = roommatedecapitateavoider
index[8] = fiendishlylicentiouslycolouristic
index[9] = sororityfoxyboatbill
index[10] = dissimilitudeaggregativewracks
index[11] = allophoneobservesbashfulness
index[12] = incuriousfatherlinessmisanthropically
index[13] = screensassonantprofessionalisms
index[14] = religionistmightplaythings
index[15] = airglowexactlyviscount
index[16] = thonggeotropicermines
index[17] = gladdingcocottekilotons
index[18] = diagrammaticallyhotfootsid
index[19] = corkerlettermenheraldically
index[20] = ulnacontemptuouscaps
index[21] = impureinternationalisedlaureates
index[22] = anarchisticbuttonedexhibitionistic
index[23] = tantalitemimicryslatted
index[24] = basophileslapsscrapping
index[25] = orphanedirreproducibleconfidences
```

Theo cấu trúc flag của FLARE-ON thì sẽ có dạng **[chars]@flare-on.com**, với thông tin index có được thì kết luận flag có độ dài là **26 (0 .. 25)** kí tự (đã biết được kí tự thứ **24** là chữ **'o'**). Lần lượt thử từng export function có được cho tới khi nhận được thông báo cho biết kí tự tương ứng **index = 13** có mã hex là **0x40** (tương ứng với mã ASCII là **'@'**) thì dừng lại.

Cuối cùng có được flag để submit là: **wuuut-exp0rts@flare-on.com**

Hết.

m4n0w4r