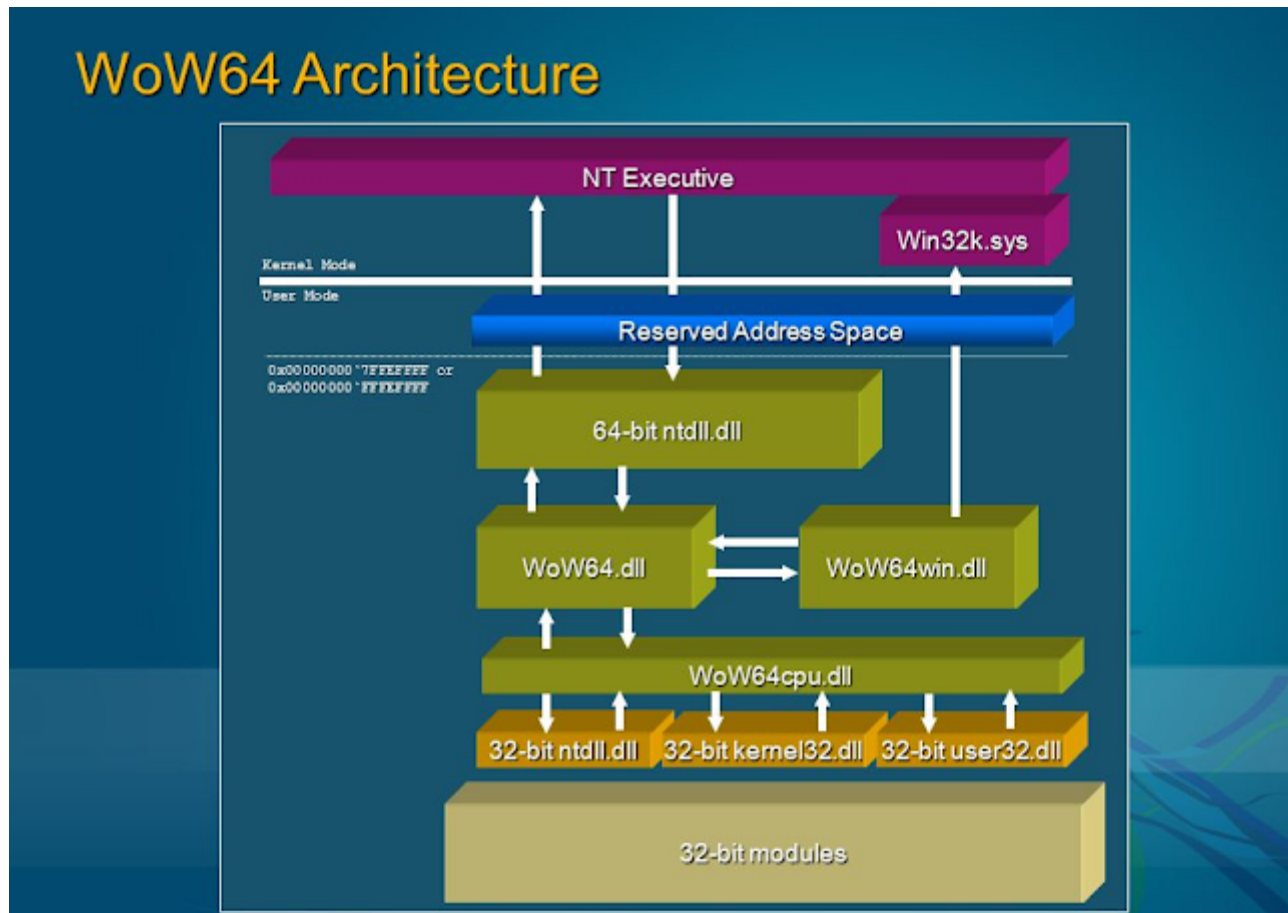


[RE015] “Heaven’s Gate” Một kĩ thuật cũ nhưng hiệu quả

blog.vincss.net/vi/re015-heavens-gate-mot-ki-thuat-cu-nhung-hieu-qua-2/

04/06/2020

“Heaven’s Gate” là tên thường gọi của một kỹ thuật cho phép binary 32-bit thực thi các lệnh 64-bit mà không cần tuân theo luồng xử lý chuẩn trên môi trường (*Windows 32-bit on Windows 64-bit*) WoW64. Có thể hiểu đơn giản WoW64 như là một sandbox nhằm hỗ trợ các ứng dụng 32-bit chạy liền mạch trên Windows 64-bit. Các tiến trình WoW64 thường gồm hai phiên bản của ntdll.dll. Đầu tiên là phiên bản 32-bit (được tải từ thư mục SysWow64), nó giúp chuyển tiếp các lời gọi hệ thống (system calls) tới môi trường WoW64. Thứ hai là phiên bản 64-bit (được tải từ thư mục System32), tiếp nhận thông tin từ môi trường WoW64 và chịu trách nhiệm chuyển đổi user-mode sang kernel-mode. Bên cạnh đó là một số các DLLs đặc biệt khác, bao gồm: wow64.dll, wow64cpu.dll, wow64win.dll.



Hình 1: Mô hình kiến trúc của WoW64
Nguồn: <https://slideplayer.com/slide/4482200/>

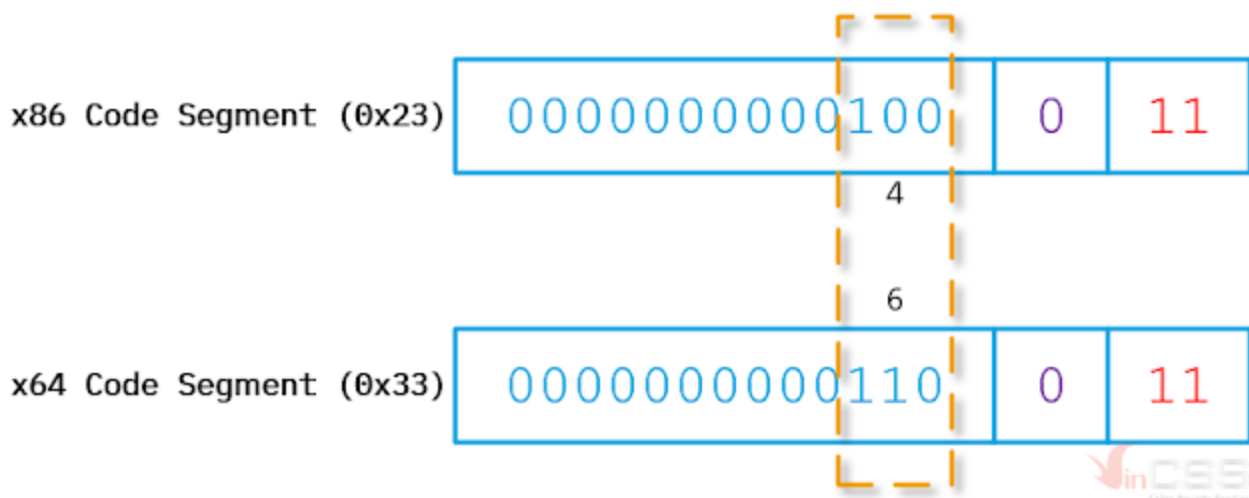
Với kiến trúc trên, có thể thấy bản thân một tiến trình 32-bit không thể nhìn phần 64-bit và bị giới hạn trong việc sử dụng các DLLs 32-bit. Do đó, nếu một mã độc 32-bit muốn thực hiện inject vào tiến trình 64-bit, bắt buộc phải sử dụng các hàm API 64-bit. *Làm thế nào để các mã độc giải quyết được bài toán này?* Khá đơn giản, ở chế độ real mode, CPU có một thanh ghi đoạn (segment register) 16-bit, thanh ghi này được nhân với 16 (bằng cách dịch trái 4 bit) sau đó cộng thêm với địa chỉ đầu vào để cung cấp một địa chỉ 20 bit, cho phép khả năng truy cập toàn bộ 1 MB bộ nhớ. Tuy nhiên, ở chế độ protected mode, việc phân chia có sự khác biệt đáng kể. Thanh ghi đoạn lúc này không chiếm cả 16-bit, nó được chia thành 3 phần: Bộ chọn segment (Segment Selector), TI (Chỉ định bảng mô tả – Table Indicator), và RPL (Mức đặc quyền yêu cầu – Request Privilege Level):

- Segment Selector (13 bits), quy định lúc nào GDT (Global Descriptor Table)/LDT (Local Descriptor Table) được sử dụng.
- TI (1 bit), quy định bảng mô tả nào sẽ được sử dụng (0 cho GDT, 1 cho LDT).
- RPL (2 bit), quy định mức bảo vệ hiện đang sử dụng bởi CPU. Đây là cách CPU theo dõi mức đặc quyền của các thao tác đang thực thi.

Các process trên môi trường Windows 64-bit đều có 2 đoạn mã (code segment): segment 0x23 tương ứng với chế độ x86 và segment 0x33 tương ứng với chế độ x64. Để chuyển sang chế độ 64-bit, không cần thay đổi code segment thành 0x33, chỉ đơn giản bằng cách thay đổi bit trong segment selector. Việc thay đổi segment selector sẽ không ảnh hưởng tới các trường TI hay RPL nói trên.

- 0x23 biểu diễn ở nhị phân: 0010 0011
- 0x33 biểu diễn ở nhị phân: 0011 0011

Như vậy, chỉ cần thay đổi phần segment selector từ 4 thành 6:



Hình 2: Biểu diễn code segment

Do đó, thông qua việc thay đổi lựa chọn segment, hoàn toàn có thể chuyển sang chế độ 64-bit thông qua gọi lệnh “CALL 0x33:Address” hoặc “JMP 0x33:Address”. Kỹ thuật này đã được giới thiệu khá lâu và hiện tại vẫn đang được các mã độc như GuLoader, Sodinokibi sử dụng rất nhiều, bởi nó cho phép thực hiện inject code vào cả hai tiến trình 64-bit và 32-bit từ một tiến trình 32-bit duy nhất trên môi trường WoW64.



Hình 3: Sơ đồ luồng thực thi của mã độc

Để kiểm tra môi trường thực thi xem có phải 64-bit hay không, mã độc thường gọi hàm API IsWow64Process. Nếu kết quả trả về là TRUE, mã độc sẽ biết được tiến trình 32-bit của nó đang thực thi trên Windows 64 bit.

```
Wow64Process = 0;
hKernel32 = GetModuleHandleW(L"kernel32");
IsWow64Process = GetProcAddress(hKernel32, "IsWow64Process");
if ( !IsWow64Process || (hProcess = GetCurrentProcess(), v8 = IsWow64Process(hProcess, &Wow64Process))
    v8 = Wow64Process;
Flag_IsWow64Process = v8;
```

Hình 4: Đoạn mã kiểm tra môi trường thực thi

Dựa vào kết quả trả về, mã độc sẽ đưa ra lựa chọn rẽ nhánh thực hiện tương ứng:

```

proc_id = 0;
if ( flag_IsWow64Process )
{
    do
    {
        v2 = f_inject_64bit_proc(0, lpszCommandLine, &decoded_pe_for_64bit, &proc_id);
        Sleep(0x2BCu);
    }
    while ( !v2 );
    result = proc_id;
}
else
{
    do
    {
        v4 = f_inject_32bit_proc(0, lpszCommandLine, &decoded_pe_for_32bit, &proc_id);
        Sleep(0x2BCu);
    }
    while ( !v4 );
    result = proc_id;
}

```

Hình 5: Mã độc lựa chọn thực hiện inject code dựa vào kết quả kiểm tra môi trường thực thi

Như đã đề cập ở trên, để có thể inject được payload vào tiến trình 64-bit, mã độc cần phải sử dụng các hàm 64-bit tương ứng. Để làm được điều này, mã độc sẽ thực hiện lấy handle của 64-bit ntdll:

```

push    offset str_ntdll_1 ; "ntdll.dll"
movlpd  qword ptr [ebp-20h], xmm0
mov     dword ptr [ebp-520h], 10007h
mov     dword ptr [eax], 0
call    near ptr f_get_ntdll ; heavens gate technique

```

Hình 6: Gọi hàm lấy handle của 64-bit ntdll

Đi sâu vào hàm `f_get_ntdll` để tìm hiểu cách mã độc chuyển sang thực hiện lệnh ở chế độ 64-bit, tại đây mã độc gọi hàm `f_get_PEB64` để lấy địa chỉ của PEB (Process Environment Block):

```

; int __cdecl f_get_ntdll(LPSTR szntdll)
f_get_ntdll proc far                                     ; CODE XREF: sub_401900+18↓p
                                                         ; f_inject_64bit_proc+51↓p

var_40C = dword ptr -40Ch
var_30  = qword ptr -30h
var_10  = qword ptr -10h
var_8   = qword ptr -8
szntdll = dword ptr 0Ch

    push    ebp
    mov     ebp, esp
    sub     esp, 3F4h
    push    ebx
    push    esi
    push    edi
    call   near ptr f_get_PEB64

```

Hình 7: Gọi hàm lấy địa chỉ của PEB

PEB là một cấu trúc dữ liệu đặc biệt chứa thông tin mô tả của process. Thông qua PEB, có thể lấy được danh sách các DLLs được nạp cùng process, các tham số khởi động của process, ImageBaseAddress, địa chỉ heap, kiểm tra xem có đang bị debug hay không, tìm địa chỉ base của bất kỳ DLLs nào được nạp, ... Chi tiết mã lệnh tại hàm `f_get_PEB64` áp dụng kĩ thuật Heaven's Gate như sau:

```

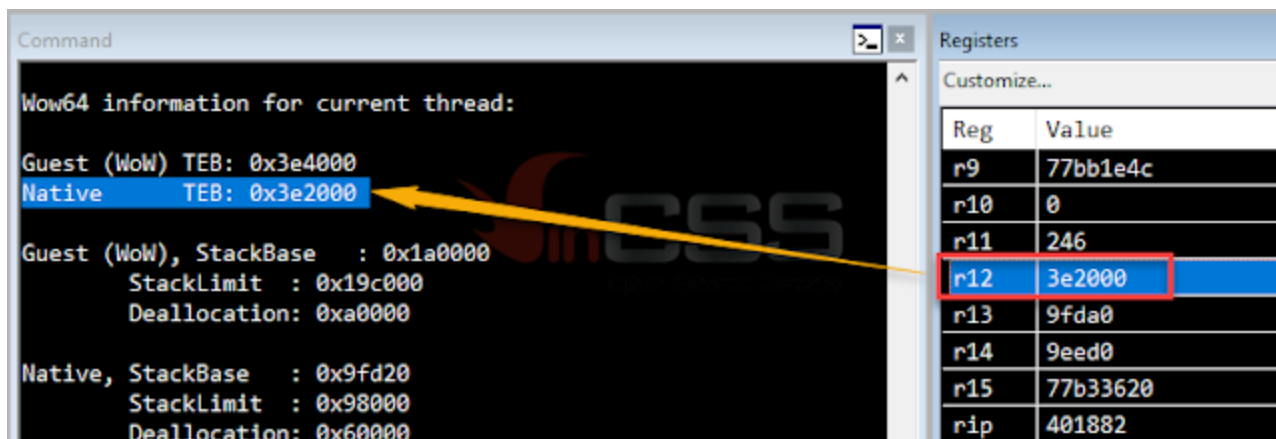
.text:00401860      push     ebp
.text:00401861      mov     ebp, esp
.text:00401863      sub     esp, 18h
.text:00401866      push     ebx
.text:00401867      xorps  xmm0, xmm0
.text:0040186A      push     esi
.text:0040186B      push     edi
.text:0040186C      movlpd [ebp+var_10], xmm0
.text:00401871      movlpd [ebp+var_8], xmm0
.text:00401876      push   33h ; '3'
.text:00401878      call   $+5 ; segment selector 0x33
              ; call 0x0040187D
.text:0040187D      add     dword ptr [esp], 5 ; esp = 0x00401882
.text:00401881      retf   ; switch to 64-bit mode
              ; exec code at 0x00401882 (in 64-bit mode: 0000003300401882)
.text:00401881      f_get_PEB64 endp ; sp-analysis failed
.text:00401881
.text:00401882      ; ===== SUBROUTINE =====
.text:00401882      f_get_TEB64 proc far
.text:00401882      var_4 = dword ptr -4
.text:00401882      push   r12 ; push r12 (pointer to TEB64)
.text:00401883
.text:00401884      pop    qword ptr ss:[rbp-0x8] ; *(rbp - 8) = TEB64
.text:00401887      call   0x40188C ; call 0x40188c
.text:00401887
.text:0040188C      mov    dword ptr ss:[rsp+0x4], 0x23 ; segment selector 0x23
.text:00401894      add    dword ptr ss:[rsp], 0x0 ; rsp = 0x401899
.text:00401898      retf   ; switch back to 32-bit mode
              ; exec code at 0x401899 (in 64-bit mode: 0000002300401899)
.text:00401898      f_get_TEB64 endp ; sp-analysis failed
.text:00401898
.text:00401899

```

Hình 8: Mã độc chuyển sang thực hiện lệnh ở chế độ 64-bit

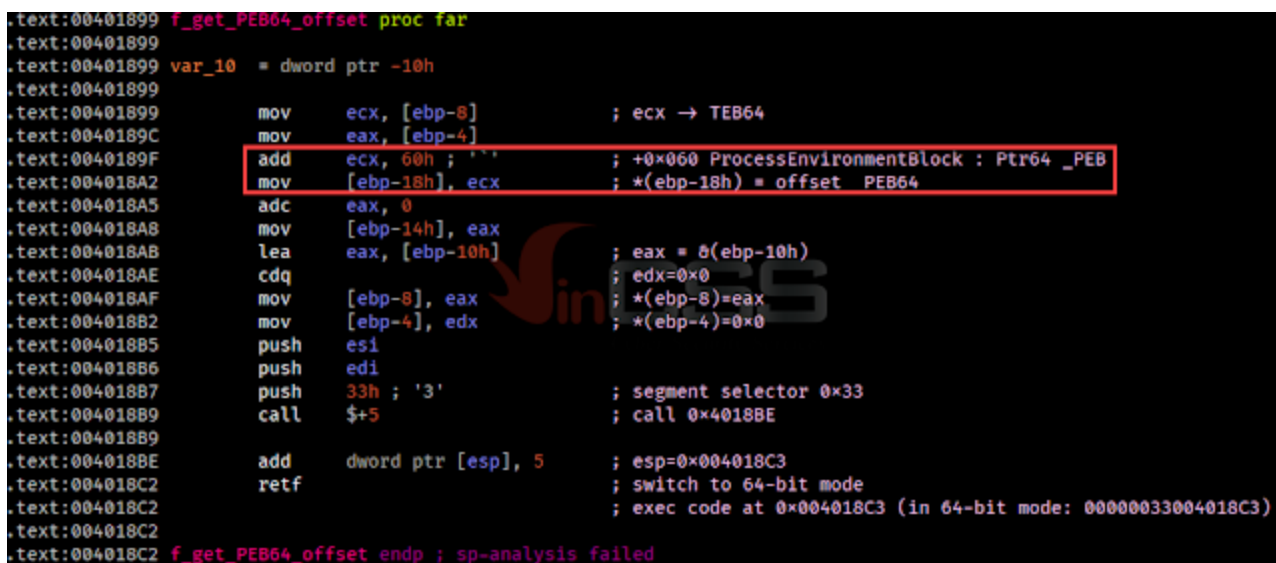
Như trên hình, segment 0x33 được đẩy lên Stack, sau đó mã độc gọi lệnh `call $+5`, thực chất là một lệnh `call` tới địa chỉ phía dưới (trong trường hợp này là 0x0040187D). Bằng lệnh `call` này, địa chỉ trở về là 0x0040187D sẽ nằm ở đỉnh Stack, sau đó được +5 thành 0x00401882. Cuối cùng, gọi lệnh `retf`, là một “far return”, nó nhận hai tham số DWORD từ Stack là segment:address. Vì vậy, khi `retf` thực thi, nó sẽ tương ứng với lệnh `jmp` với địa chỉ trở về thực tế là 0x33:0x00401882. Thông qua việc thay đổi code segment này, mã lệnh tại địa chỉ được chỉ định là 0x00401882 sẽ được hiểu ở chế độ 64-bit như trên hình.

Các lệnh từ 0x00401882 chịu trách nhiệm gán nội dung của thanh ghi r12 vào một biến trên Stack, sau đó chuyển lại về chế độ 32-bit. Thanh ghi r12 lúc này chính là con trỏ tới cấu trúc TEB64.

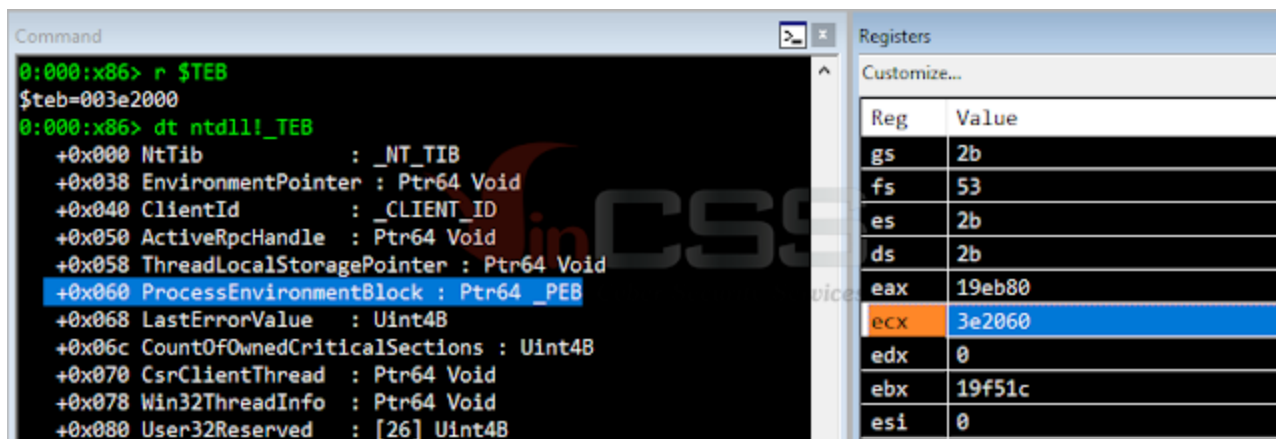


Hình 9: Thanh ghi r12 chứa con trỏ tới cấu trúc PEB64

Lệnh `retf` tại `0x00401898` sẽ chuyển hướng thực thi tới địa chỉ tiếp theo tại `0x00401899` ở chế độ 32-bit, có nhiệm vụ lấy offset tới `PEB64` và chuyển sang thực hiện lệnh ở chế độ 64-bit tại địa chỉ tiếp theo là `0x004018C3`:



Hình 10: Lấy offset tới PEB64



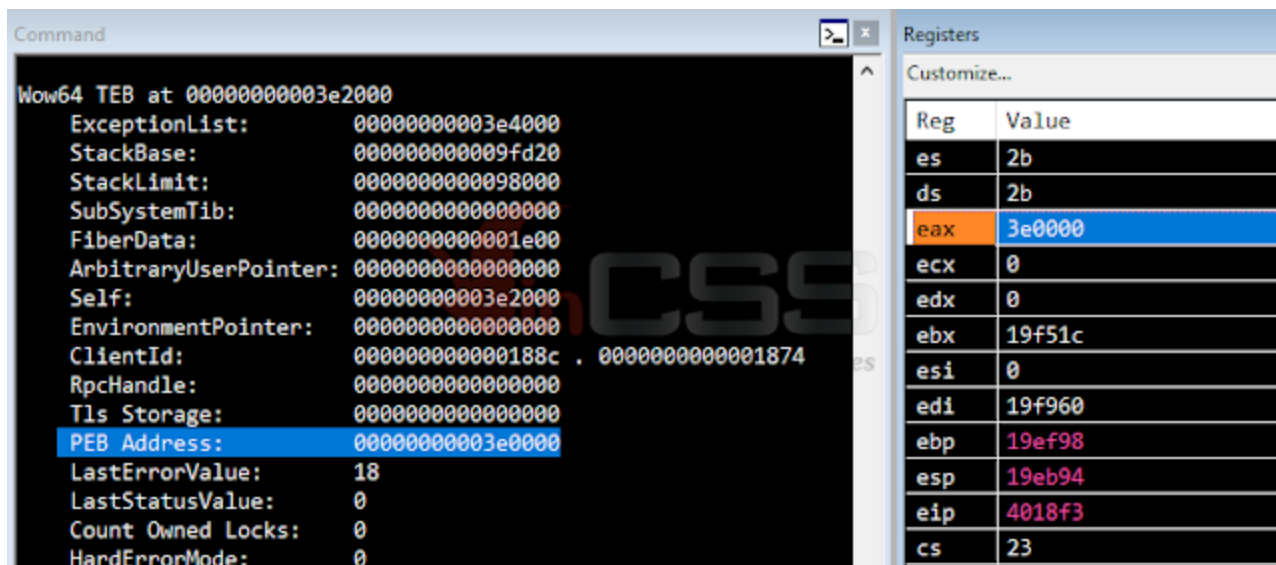
Hình 11: Giá trị của thanh ghi ecx sau khi thực hiện tới lệnh retf

Đoạn code tiếp theo thực hiện từ địa chỉ 0x004018C3, nhiệm vụ cuối cùng là lấy ra địa chỉ của PEB64 và gán cho cặp thanh ghi edx:eax.



Hình 12: Lấy địa chỉ của PEB64

Kết quả, hai thanh ghi edx:eax sẽ trở tới cấu trúc PEB64.



Hình 13: Cập thanh ghi edx:eax trở tới cấu trúc PEB64

Với việc có được địa chỉ của PEB, mã độc sẽ thực hiện các bước tiếp theo:

- Lấy offset của Ldr (+0x018 Ldr).
- Thông qua Ldr có được cấu trúc `_PEB_LDR_DATA`.
- Từ `_PEB_LDR_DATA` lấy offset của `InLoadOrderModuleList` (+0x010 `InLoadOrderModuleList`)
- `InLoadOrderModuleList` là danh sách chứa tất cả các DLLs được nạp vào bộ nhớ cùng process. Dựa vào nó, mã độc lấy ra tên của DLL được nạp, so sánh với chuỗi "ntdll.dll", nếu tìm thấy sẽ trả về địa chỉ base của thư viện ntdll.dll.

Như vậy, bằng cách áp dụng kĩ thuật Heaven's Gate, mã độc sau khi thực hiện xong hàm `f_get_ntdll` sẽ có được `edx:eax` trở tới địa chỉ base của ntdll.dll như hình dưới đây:

```
Command
Wow64 PEB at 00000000003e0000
InheritedAddressSpace: No
ReadImageFileExecOptions: No
BeingDebugged: Yes
ImageBaseAddress: 0000000000400000
Ldr: 00007ffae0b453e0
Ldr.Initialized: Yes
Ldr.InInitializationOrderModuleList: 00000000006a3450 . 00000000006a49f0
Ldr.InLoadOrderModuleList: 00000000006a3600 . 00000000006a49d0
Ldr.InMemoryOrderModuleList: 00000000006a3610 . 00000000006a49e0
Base TimeStamp Module
400000 5a10b7e Nov 19 06:04:30 2017 C:\Users\Administrator\Desktop\process_
7ffae09e0000 99ca0526 Oct 06 04:20:38 2051 C:\WINDOWS\SYSTEM32\ntdll.dll
7ffadfc80000 529d0489 Dec 03 05:07:05 2013 C:\WINDOWS\System32\wow64.dll
7ffadfec0000 1cf7e3a9 May 27 09:08:41 1985 C:\WINDOWS\System32\wow64win.dll
77b30000 c0b23215 Jun 12 03:48:21 2072 C:\WINDOWS\System32\wow64cpu.dll
SubSystemData: 0000000000000000
ProcessHeap: 00000000006a0000
```

eax	e09e0000
ecx	0
edx	7ffa

Hình 14: Cặp thanh ghi edx:eax trở tới địa chỉ base của ntdll.dll

Sau khi có được địa chỉ của ntdll.dll, mã độc sẽ sử dụng các hàm native từ thư viện này để thực hiện công việc còn lại là inject payload vào tiến trình 64-bit. Danh sách các hàm thường sử dụng trong kỹ thuật inject bao gồm:

- NtGetContextThread
- NtReadVirtualMemory
- NtUnmapViewOfSection
- NtAllocateVirtualMemory
- NtWriteVirtualMemory
- NtSetContextThread
- NtResumeThread



Hình 15: Luồng code thực hiện inject payload

Kết quả sau khi thực thi toàn bộ đoạn code trên, mã độc đã inject payload thành công vào tiến trình **notepad.exe** 64-bit:

dumped_fix.exe	0.22	1,932 K	5,452 K	2716	32-bit
notepad.exe	0.01	9,348 K	7,192 K	808	64-bit Notepad

Hình 16: Mã độc inject payload thành công vào tiến trình 64-bit

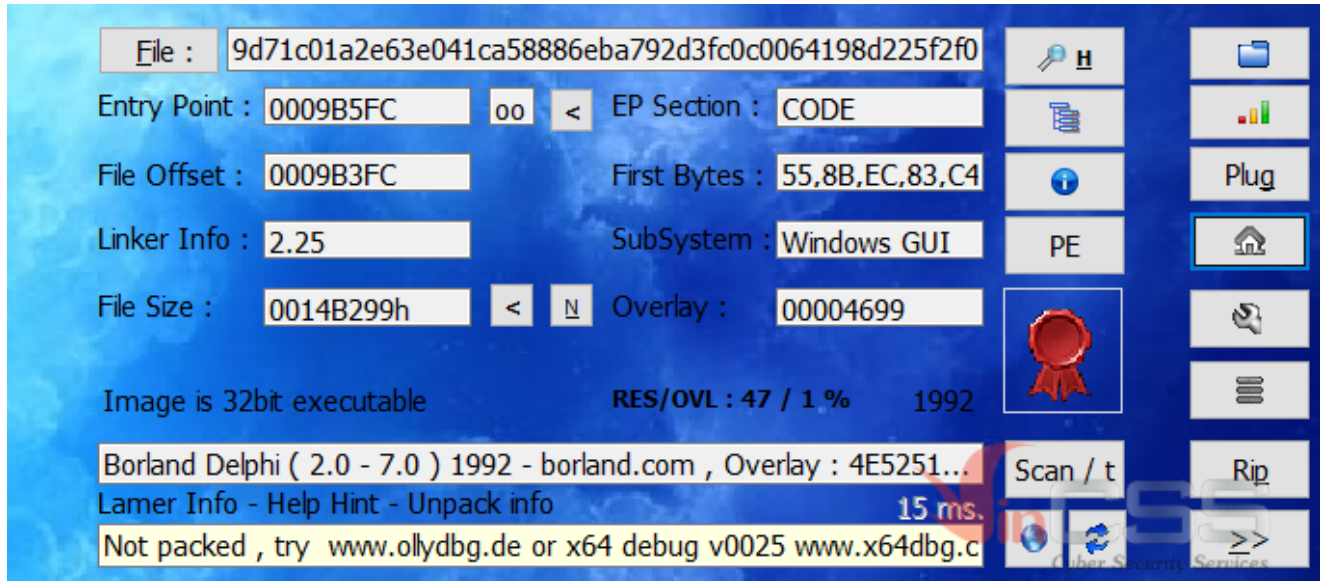
Tài liệu tham khảo:

Tran Trung Kien (aka m4n0w4r)

R&D Center – VinCSS (a member of Vingroup)

[↗ Trở lại](#)

[Bài viết liên quan](#)



📅 17/12/2023

[RE016] Malware Analysis: ModiLoader

1. Giới thiệu Gần đây, tôi có tìm hiểu một dòng loader có tên là ModiLoader. Loader này được phát tán thông qua các dịch vụ Malspam để lừa người dùng thực thi mã độc. Tương tự như các dòng loader khác, ModiLoader cũng thông qua nhiều bước (stage) để tải về payload cuối cùng có nhiệm vụ đánh [...]



📅 12/12/2023

[RE027] Nhóm APT Mustang Panda có thể vẫn đang tiếp tục hoạt động tấn công vào các tổ chức tại Việt Nam

Tại VinCSS, chúng tôi liên tục chủ động theo dõi tình hình an ninh mạng, săn tìm các mẫu mã độc và đánh giá mức độ nguy hiểm của chúng, đặc biệt là các mẫu mã độc nhắm tới Việt Nam. Gần đây, trong quá trình thực hiện hunting trên nền tảng của VirusTotal, thực hiện tìm kiếm các mẫu byte đặc trưng liên quan tới nhóm Mustang Panda (PlugX), chúng tôi đã phát hiện một loạt mẫu mã độc mà chúng tôi nghi ngờ là của nhóm này được tải lên từ Việt Nam.



📅 24/04/2022

[RE026] A Deep Dive into Zloader – the Silent Night

Zloader, một banking trojan còn biết đến với những tên gọi khác như Terdot hay Zbot. Dòng trojan này được phát hiện lần đầu tiên vào năm 2016, và theo thời gian số lượng phát tán của nó liên tục gia tăng. Code của Zloader được cho là xây dựng dựa trên mã nguồn bị rò rỉ của mã độc Zeus nổi tiếng. Vào năm 2011, khi mã nguồn của Zeus được công khai thì từ đó tới nay nó được sử dụng trong nhiều mẫu mã độc khác nhau.



📅 27/09/2021

[RE025] TrickBot ... many tricks

Được phát hiện lần đầu vào năm 2016, tới thời điểm hiện tại TrickBot (còn được biết đến với những tên gọi khác như TrickLoader hay Trickster) đã trở thành một trong những mã độc nguy hiểm và phổ biến nhất hiện nay. Những kẻ đứng đằng sau TrickBot liên tục phát triển để thêm các tính năng và thủ thuật mới. Mã độc này được phát triển dưới dạng mô-đun, theo đó payload chính sẽ chịu trách nhiệm tải các plugin khác có khả năng thực hiện các tác vụ cụ thể, bao gồm đánh cắp tài khoản và thông tin nhạy cảm, cung cấp khả năng truy cập từ xa, lây lan qua mạng cục bộ, và tải xuống phần mềm độc hại khác.