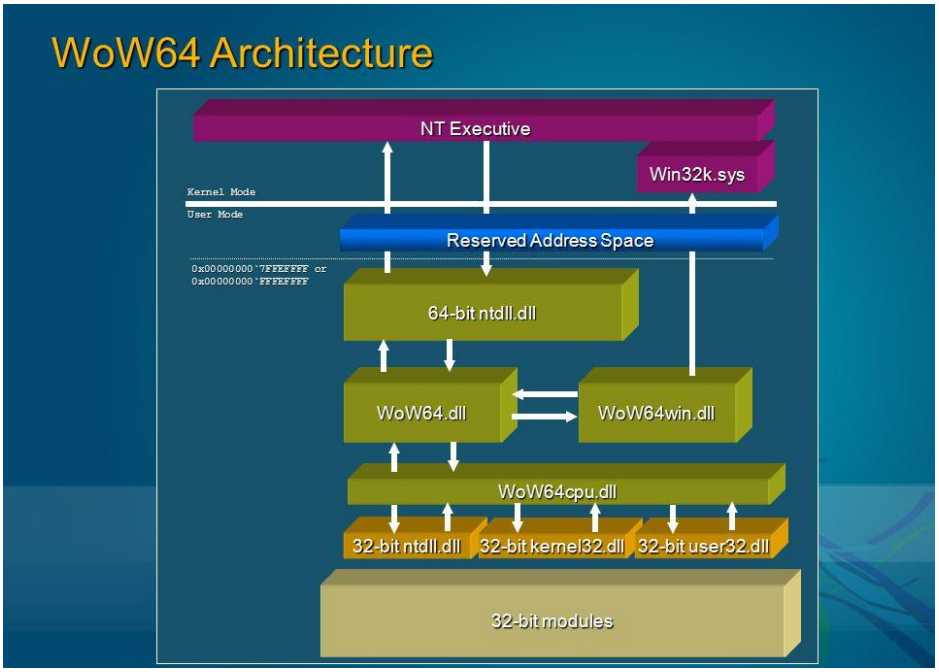
"**Heaven's Gate**" là tên thường gọi của một kỹ thuật cho phép binary 32-bit thực thi các lệnh 64-bit mà không cần tuân theo luồng xử lý chuẩn trên môi trường WoW64 *(Windows 32-bit on Windows 64-bit)*. WoW64. Hay có thể hiểu đơn giản, Heaven's Gate như là một sandbox nhằm hỗ trợ các ứng dụng 32-bit 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 64-bit. Do đó, nếu một mã độc 32-bit muốn thực hiện inject vào tiến trình 64-bit, bắt buộc phải sử dụng các 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).

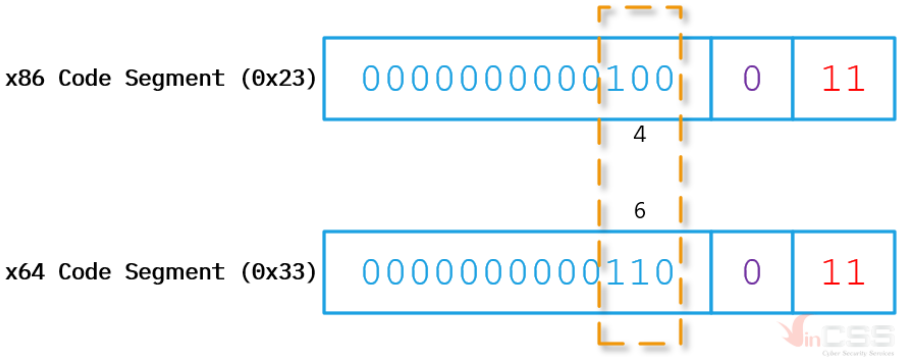
♦ RPL (2 bit), quy định mức bảo vệ hiện đang sử dụng bởi CPU. Đây là cách CPU theo dõi mức đặc quyền của các thao tác đang thực thi.

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`". Kĩ 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`:

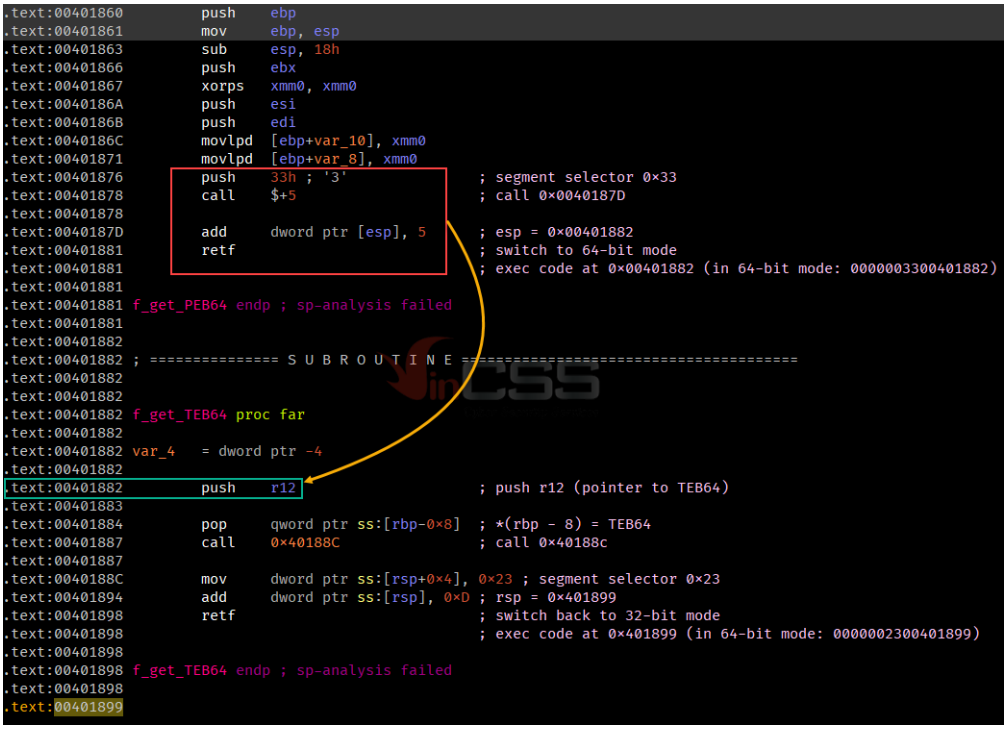
```
push    offset str_ntdlldll_1 ; "ntdll.dll"
movlpd  qword ptr [ebp-20h], xmm0
mov     dword ptr [ebp-520h], 10007h
mov     dword ptr [eax], 0
call    near ptr f_get_ntdll ; heavens gate technique
```

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):

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 processs, `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:

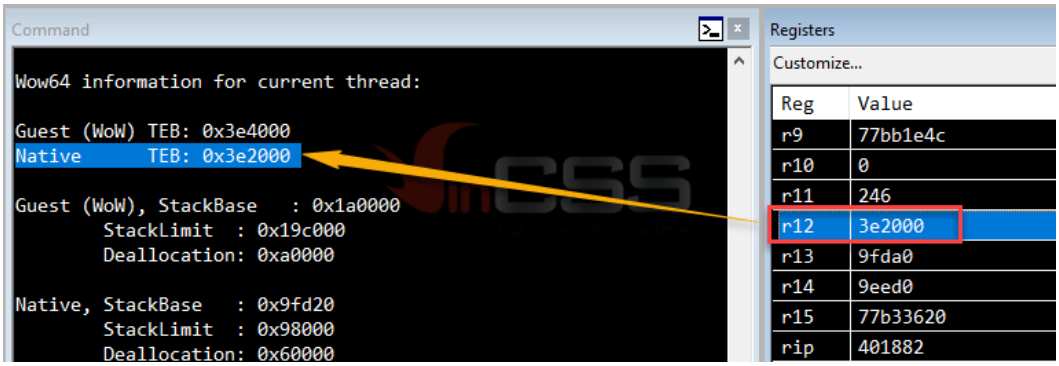


Hình 8: Mã độc chuyển sang thực hiện lệnh ở chế độ 64-bit

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

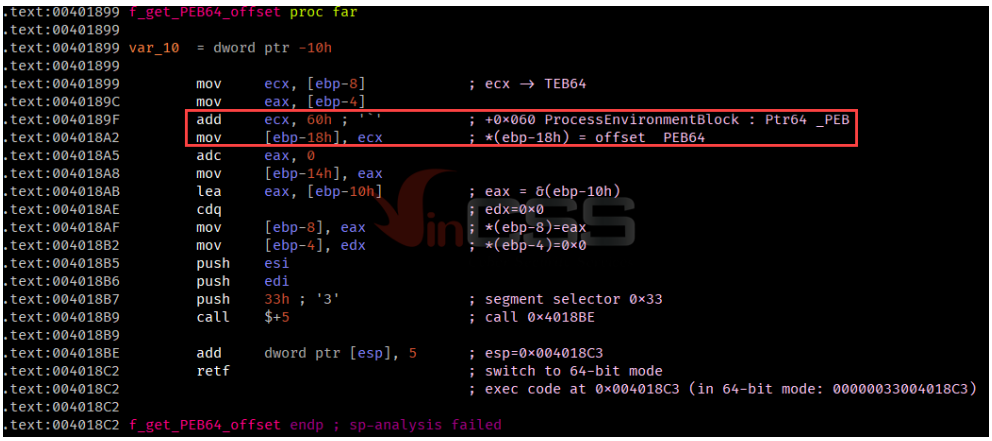
việc thay đổi code segment này, mã lệnh tại địa chỉ được chỉ định là `0x00401882` sẽ được hiểu ở chế độ 64-bit như trên hình.

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`.

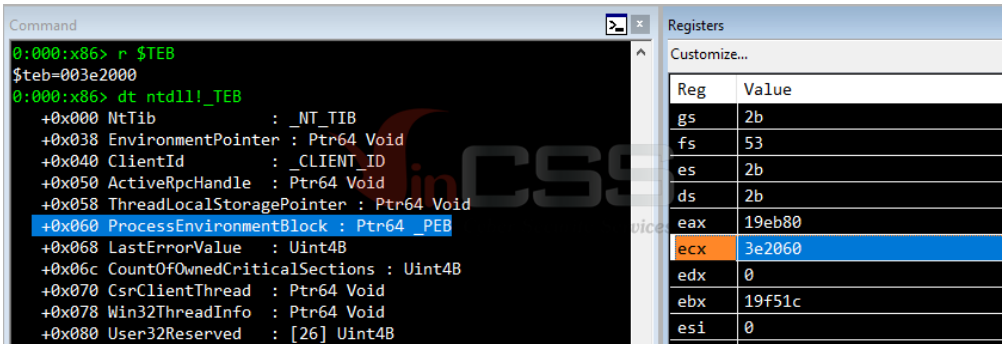


Hình 9: Thanh ghi r12 chứa con trỏ tới cấu trúc PEB64

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`



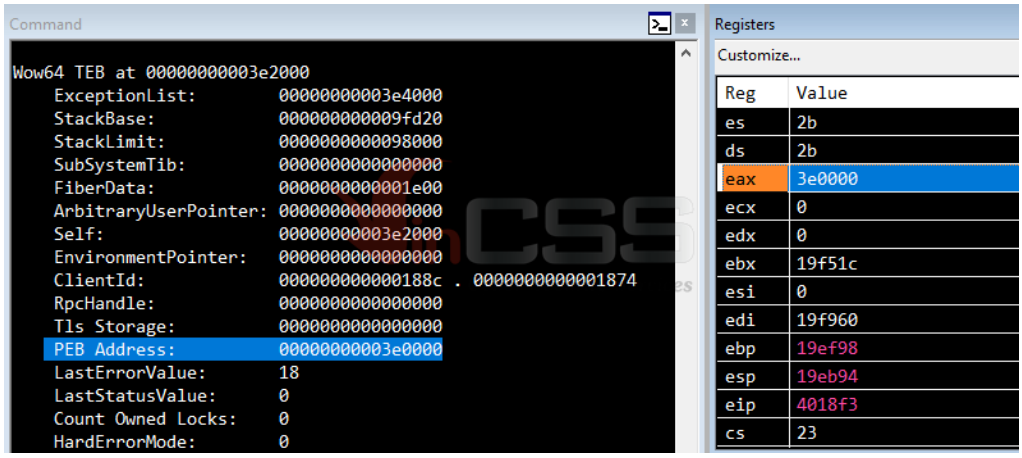
Hình 11: Giá trị của thanh ghi `ecx` sau khi thực hiện tới lệnh `retf`.

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`.

```
.text:004018C3 sub_4018C3 proc far
.text:004018C3
.text:004018C3 var_4   = dword ptr -4
.text:004018C3
.text:004018C3         push    qword ptr ss:[rbp-0x8]
.text:004018C6         pop     rdi                     ; rdi = &(rbp-0x10)
.text:004018C7
.text:004018C8         push    qword ptr ss:[rbp-0x18]
.text:004018CB         pop     rsi                     ; rsi offset _PEB64
.text:004018CC
.text:004018CD         push    0x8
.text:004018CF         pop     rcx                     ; rcx = 0x8
.text:004018D0
.text:004018D1         rep movsbbyte ptr ds:[rdi], byte ptr ds:[rsi] ; copies 8 bytes from rsi to rdi
.text:004018D1                                         ; *(rdi) → _PEB64
.text:004018D3         call    0x4018D8
.text:004018D3
.text:004018D8         mov     dword ptr ss:[rsp+0x4], 0x23 ; segment selector 0x23
.text:004018E0         add     dword ptr ss:[rsp], 0xD
.text:004018E4         retf                            ; exec code at 0x004018E5 (int 64-bit mdoe: 00000023004018e5)
.text:004018E4
.text:004018E4 sub_4018C3 endp ; sp-analysis failed
.text:004018E4
.text:004018E5
.text:004018E5 ; =============== S U B R O U T I N E =======================================
.text:004018E5
.text:004018E5
.text:004018E5 sub_4018E5 proc near
.text:004018E5         pop     edi
.text:004018E6         pop     esi
.text:004018E7         mov     eax, [ebp-10h]
.text:004018EA         mov     edx, [ebp-0Ch]          ; edx:eax → _PEB64
.text:004018ED         pop     edi
.text:004018EE         pop     esi
.text:004018EF         pop     ebx
.text:004018F0         mov     esp, ebp
.text:004018F2         pop     ebp
.text:004018F3         retn
.text:004018F3
.text:004018F3 sub_4018E5 endp ; sp-analysis failed
```

Hình 12: Lấy địa chỉ của PEB64

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

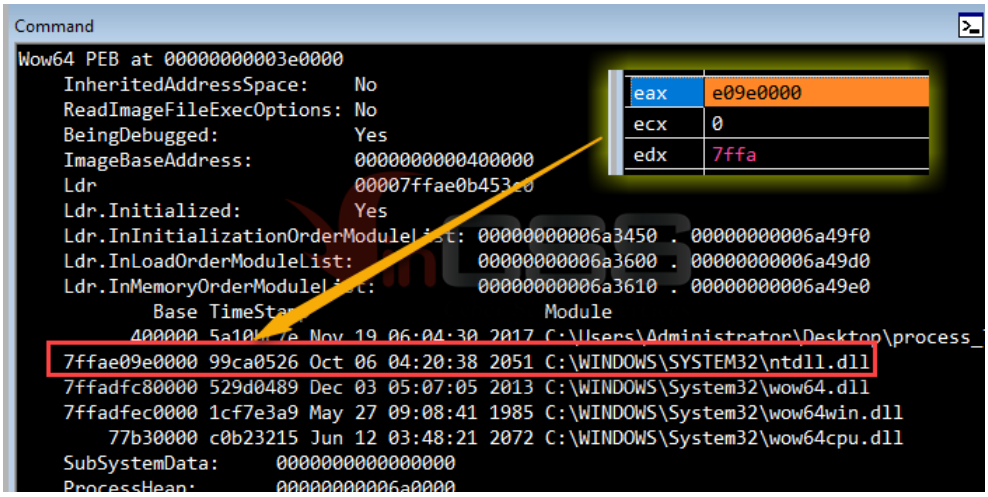


Hình 13: Cặp thanh ghi `edx:eax` trỏ tới cấu trúc `PEB64`

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 này 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`.

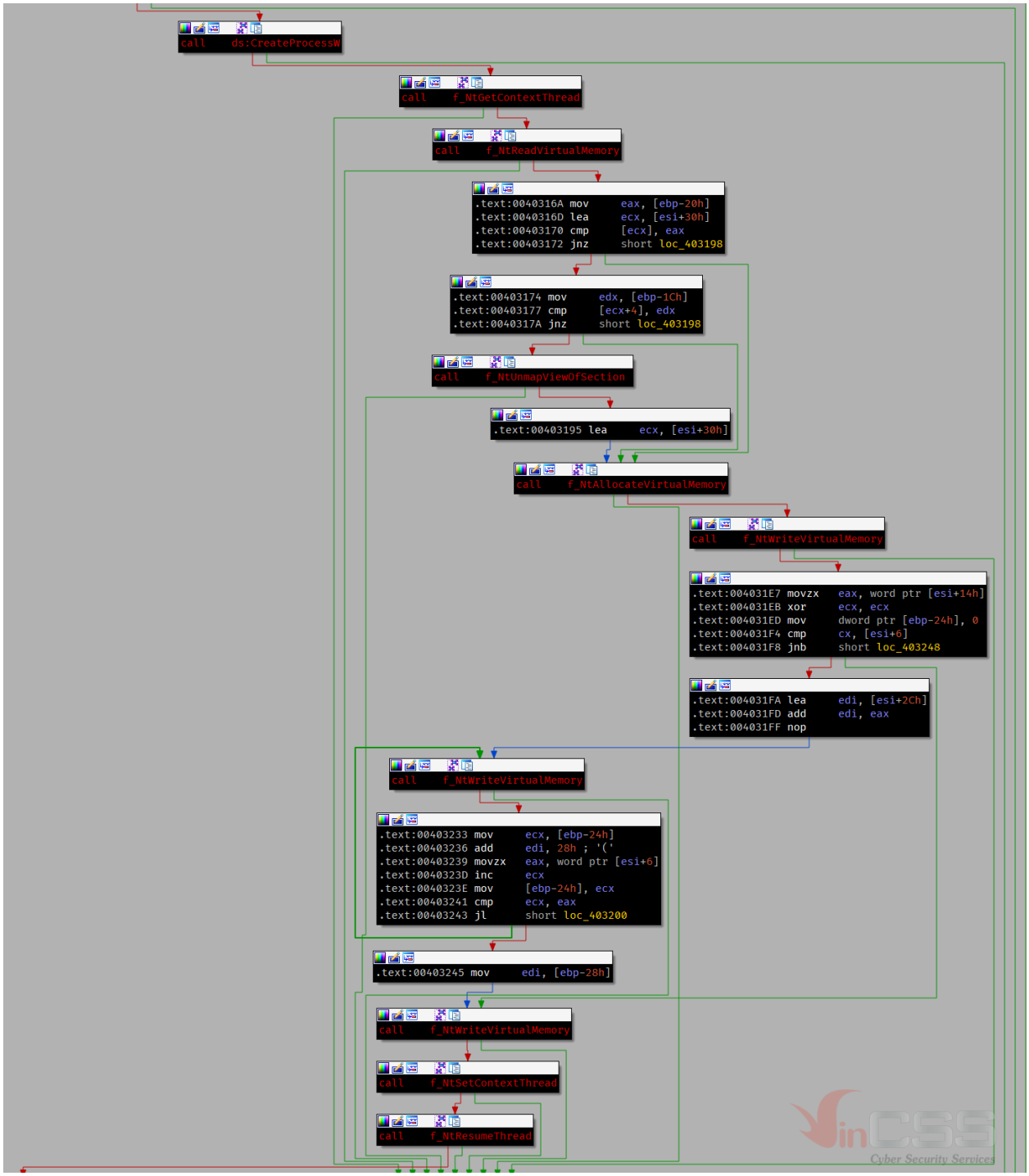
Như vậy, bằng cách áp dụng kĩ thuật Heaven's Gate, mã độc sau khi thực hiện xong hàm `f_get_ntdll` sẽ có được `edx:eax` trỏ tới địa chỉ base của `ntdll.dll` như hình dưới đây:



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

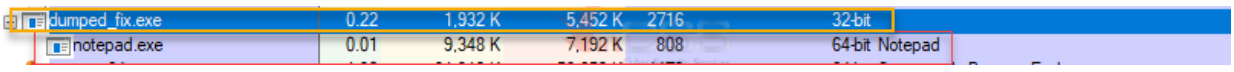
Sau khi có được địa chỉ của `ntdll.dll`, mã độc sẽ sử dụng các hàm native từ thư viện này để thực hiện công việc còn lại là inject payload vào tiến trình 64-bit. Danh sách các hàm thường sử dụng trong kĩ thuật inject bao gồm:

- `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:



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

Refs:

https://docs.microsoft.com/en-us/windows/win32/winprog64/running-32-bit-applications

https://www.malwaretech.com/2014/02/the-0x33-segment-selector-heavens-gate.html

https://www.geoffchappell.com/studies/windows/win32/ntdll/structs/teb/index.htm

https://ntopcode.wordpress.com/2018/02/26/anatomy-of-the-process-environment-block-peb-windows-internals/