FORMAT STRING ATTACKS

Format String Attacks (1)

formatstr1.c

What does this program do?

```c
#include <stdio.h>

int main() {
  char name[256];

  printf("What is your name?\n");
  fgets(name, 255, stdin);
  printf("Hello, ");
  printf(name);
  printf("Nice to meet you.\n");
}
```

Format String Attacks (2)

formatstr2.c

```c
#include <stdio.h>

int main() {
  char name[256];

  printf("What is your name?\n");
  fgets(name, 255, stdin);
  printf("Hello, ");
  printf("%s", name);
  printf("Nice to meet you.\n");

}
```

Hint: How is this program different from the previous slide?

Format String Attacks (2)

formatstr2.c

```c
#include <stdio.h>

int main() {
  char name[256];

  printf("What is your name?\n");
  fgets(name, 255, stdin);
  printf("Hello, ");
  printf("%s", name);
  printf("Nice to meet you.\n");

}
```

Hint: How is this program different from the previous slide?

printf

```c
#include <stdio.h>

int main() {
  int a = 1, b = 2, c = 3;
  printf("a = %d  b = %d  c = %d\n", a, b, c);
}
```

```
[cvwright@ubuntu tmp]$ gcc -o printf printf.c
[cvwright@ubuntu tmp]$
[cvwright@ubuntu tmp]$ ./printf
a = 1  b = 2  c = 3
```

How does printf() actually work?

```
(gdb) disassemble main
Dump of assembler code for function main:
   0x080483e4 <+0>:     push   %ebp
   0x080483e5 <+1>:     mov    %esp,%ebp
   0x080483e7 <+3>:     and    $0xfffffff0,%esp
   0x080483ea <+6>:     sub    $0x20,%esp
   0x080483ed <+9>:     movl   $0x1,0x14(%esp)
   0x080483f5 <+17>:    movl   $0x2,0x18(%esp)
   0x080483fd <+25>:    movl   $0x3,0x1c(%esp)
   0x08048405 <+33>:    mov    $0x8048500,%eax
   0x0804840a <+38>:    mov    0x1c(%esp),%edx
   0x0804840e <+42>:    mov    %edx,0xc(%esp)
   0x08048412 <+46>:    mov    0x18(%esp),%edx
   0x08048416 <+50>:    mov    %edx,0x8(%esp)
   0x0804841a <+54>:    mov    0x14(%esp),%edx
   0x0804841e <+58>:    mov    %edx,0x4(%esp)
   0x08048422 <+62>:    mov    %eax,(%esp)
   0x08048425 <+65>:    call   0x8048300 <printf@plt>
   0x0804842a <+70>:    leave
   0x0804842b <+71>:    ret
End of assembler dump.
```

printf questions

- How many arguments does printf take?
- How does it know how many things to print?
- Where does it get them from?
- How does it know what type they are?

What does this program do?

```c
#include <stdio.h>

int main() {
  printf("%08x  %08x\n");
}
```

Output

```
[cvwright@ubuntu tmp]$ gcc -o printf2 printf2.c
printf2.c: In function ‘main’:
printf2.c:4:3: warning: format ‘%x’ expects a matching ‘unsigned int’ argument [-Wformat]
printf2.c:4:3: warning: format ‘%x’ expects a matching ‘unsigned int’ argument [-Wformat]
[cvwright@ubuntu tmp]$
[cvwright@ubuntu tmp]$ ./printf2
00000000  08048409
```

Output

```
[cvwright@ubuntu tmp]$ gcc -o printf2 printf2.c
printf2.c: In function ‘main’:
printf2.c:4:3: warning: format ‘%x’ expects a matching ‘unsigned int’ argument [-Wformat]
printf2.c:4:3: warning: format ‘%x’ expects a matching ‘unsigned int’ argument [-Wformat]
[cvwright@ubuntu tmp]$
[cvwright@ubuntu tmp]$ ./printf2
00000000  08048409
```

What's this?

Output

```
[cvwright@ubuntu tmp]$ gcc -o printf2 printf2.c
printf2.c: In function 'main':
printf2.c:4:3: warning: format '%x' expects a matching 'unsigned int' argument [-Wformat]
printf2.c:4:3: warning: format '%x' expects a matching 'unsigned int' argument [-Wformat]
[cvwright@ubuntu tmp]$
[cvwright@ubuntu tmp]$ ./printf2
00000000  08048409
```

What's this?

Looks like an address in our code page. Maybe a saved %eip?

Format String Attacks

formatstr1.c

```c
#include <stdio.h>

int main() {
  char name[256];

  printf("What is your name?\n");
  fgets(name, 255, stdin);
  printf("Hello, ");
  printf(name);
  printf("Nice to meet you.\n");
}
```

User controls the format string

Format String Attacks

- Attacker has control of a buffer that's used as a format string
- Carefully-crafted format string allows access to memory
  - %x – Pop the stack and print the value as hex
  - %s – Pop the stack, use that value as a pointer to a string, and print the region of memory there
  - %n – Pop the stack, use that value as a pointer to int, and store the number of bytes printed so far in that int
  - %u – Update the current count of printed bytes

Format String Attacks: The Silver Lining

- On a positive note, format string vulnerabilities are easy to detect
  - printf(string); ← Should be printf("%s", string);
- As a result, their practical impact is relatively low (now)
  - They made a big splash in 1999 / 2000 when first publicized