
# CMPSC 447
# Other Memory Vulnerabilities

*Trent Jaeger*
*Systems and Internet Infrastructure Security (SIIS) Lab*
*Computer Science and Engineering Department*
*Pennsylvania State University*

# Format String Vulnerabilities

- Who uses printf in their programs?

```
printf ("This class is %s\n", string);
```

  - In some cases, printf can be exploited

# Format String Vulnerabilities

- Who uses printf in their programs?

```
printf ("This class is %s\n", string);
```

  - In some cases, printf can be exploited
- Printf takes a format string and an arbitrary number of subsequent arguments
  - Format string determines what to print
    - Including a set of format parameters
  - Arguments supply input for format parameters
    - Which may be values (e.g., %d) or references (e.g., %s)
- An argument for each format parameter

# Format String Vulnerabilities

- Who uses printf in their programs?
  - In some cases, printf can be exploited
- As usual, arguments are retrieved from the stack
  - What happens when the following is done?

```
printf("%s%s%s%s");
```

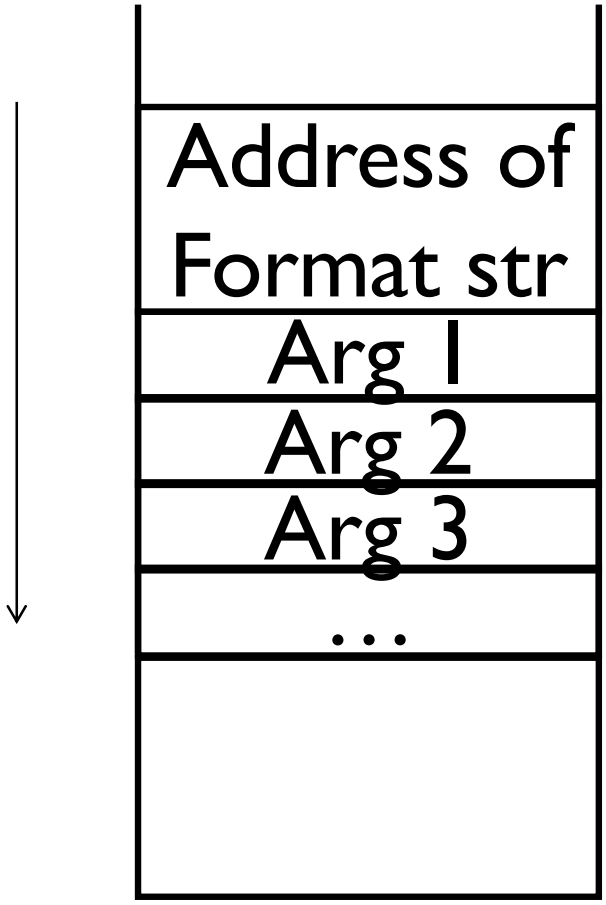
# Format String Vulnerabilities

- Who uses printf in their programs?
  - In some cases, printf can be exploited
- As usual, arguments are retrieved from the stack
  - What happens when the following is done?

```
printf("%s%s%s%s");
```

- Traditionally, compilers do not check for a match between arguments and format string – do now…
  - So, printf would print "strings" using next four values on stack as string addresses – whatever they are

# Printf and the Stack




- Remember these are parameters to a function call
- So, the function expects them on the stack
- Printf will just start reading whatever is above the format string address

# Format String Vulnerabilities

- Who uses printf in their programs?
  - In some cases, printf can be exploited
- As usual, arguments are retrieved from the stack
  - What happens when the following is done?

```
printf(arg);
```

  - Anyone use this? Some people do.

# Format String Vulnerabilities

- Who uses printf in their programs?
  - In some cases, printf can be exploited
- As usual, arguments are retrieved from the stack
  - What happens when the following is done?

```
printf(arg);
```

- Printf can take a variable as an argument – treated as a format string
  - If an adversary can control this argument and put values on the stack, they can direct printf to access that memory – "%s%s%s…"

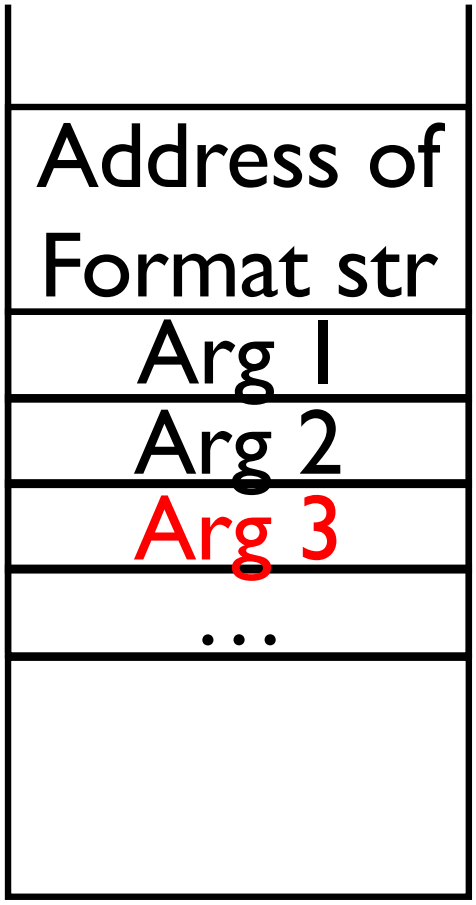
# Format String Vulnerabilities

- Who uses printf in their programs?
  - In some cases, printf can be exploited
- As usual, arguments are retrieved from the stack
  - What happens when the following is done?

```
printf(arg);
```

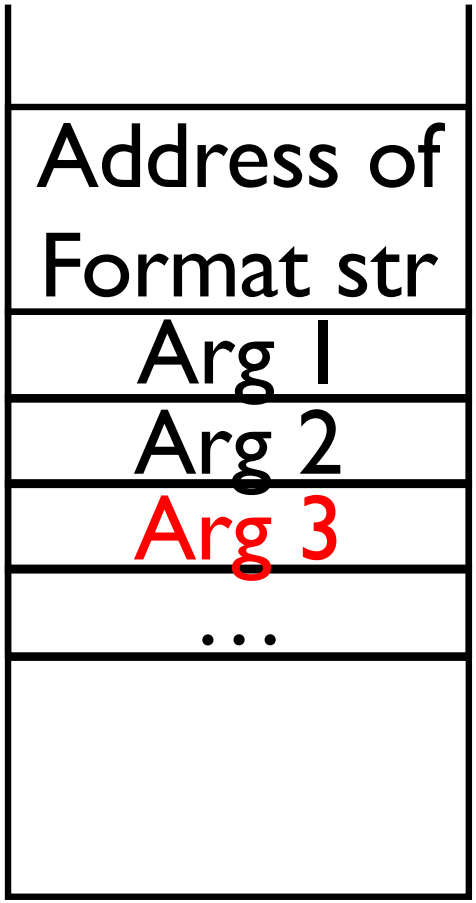
- An "interesting" format parameter type – %n
  - "%n" in a format string tells the printf to write the number of bytes written via the format string processing up to that point to an address specified by the argument

# Printf and the Stack




- Suppose format string generates an adversary-controlled number of bytes
- Suppose adversary controls Arg1-Arg3 on stack
- Adversary can control number of bytes generated by format string with Arg1 and Arg2
- Adversary can direct where to write that number (of bytes) using %n with address at Arg3

# Printf-oriented Programming




- If the program has a loop that calls printf under adversary control
- An adversary can supply inputs to write to any memory address
- Over and over
- To control the execution of the program arbitrarily (Turing complete)

# Prevent Vulnerabilities

- Preventing format string vulnerabilities means limiting the ability of adversaries to control the format string
  - Hard-coded strings w/ no arguments – when you can
  - Hard-coded format strings at least – no printf(arg)
  - Do not use %n
    - Be careful with other references - %s and sprintf can be used to created disclosure attacks
  - Compiler support to match printf arguments with format string

# Take Away

- There are other ways to implement powerful attacks besides overflow vulnerabilities
- We examined a few of the common ones
  - Use-after-free
  - Type confusion
  - Format string vulnerabilities
- Each are capable of implementing arbitrary write primitives that give an adversary arbitrary control of memory
  - We will want to prevent these vulnerabilities