Format string vulnerabilities

Formatted I/O

- C's printf family supports formatted I/O

```
void print_record(int age, char *name)
{
  printf("Name: %s\tAge: %d\n",name,age);
}
```

- Format specifiers
  - Position in string indicates stack argument to print
  - Kind of specifier indicates type of the argument
    - %s = string
    - %d = integer
    - etc.

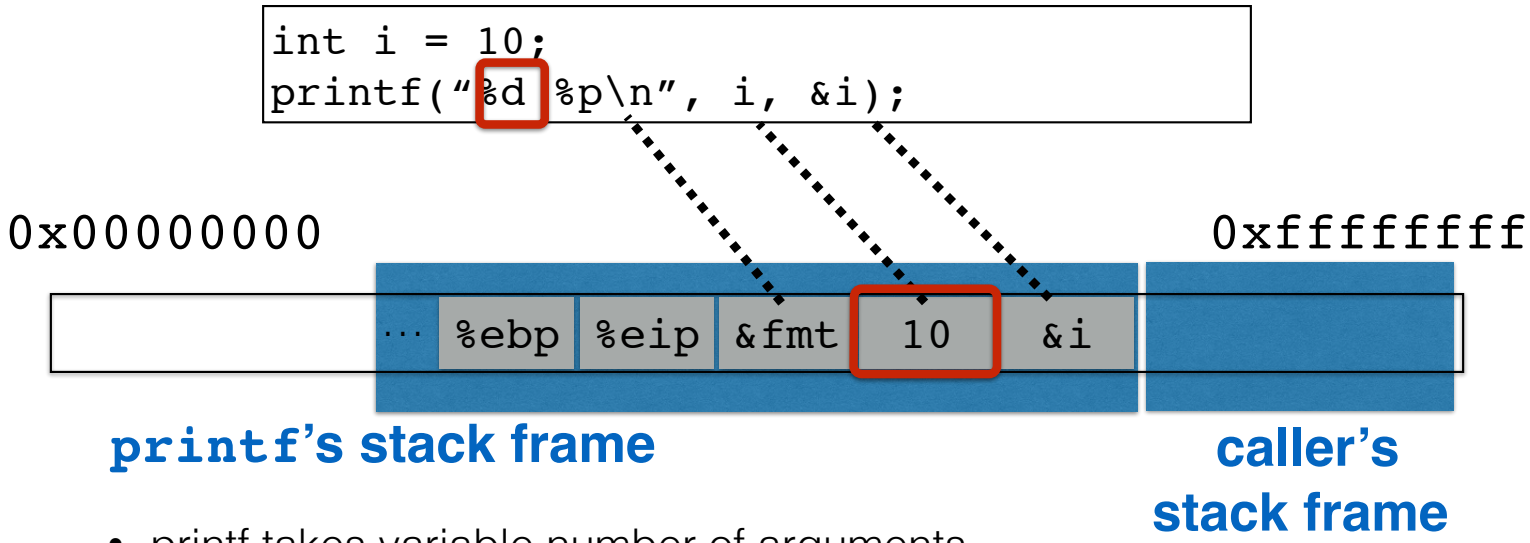
What's the difference?

```
void safe()
{
    char buf[80];
    if(fgets(buf, sizeof(buf), stdin)==NULL)
        return;
    printf("%s",buf);
}
```

```
void vulnerable()
{
    char buf[80];
    if(fgets(buf, sizeof(buf), stdin)==NULL)
        return;
    printf(buf);
}
```

Attacker controls the format string

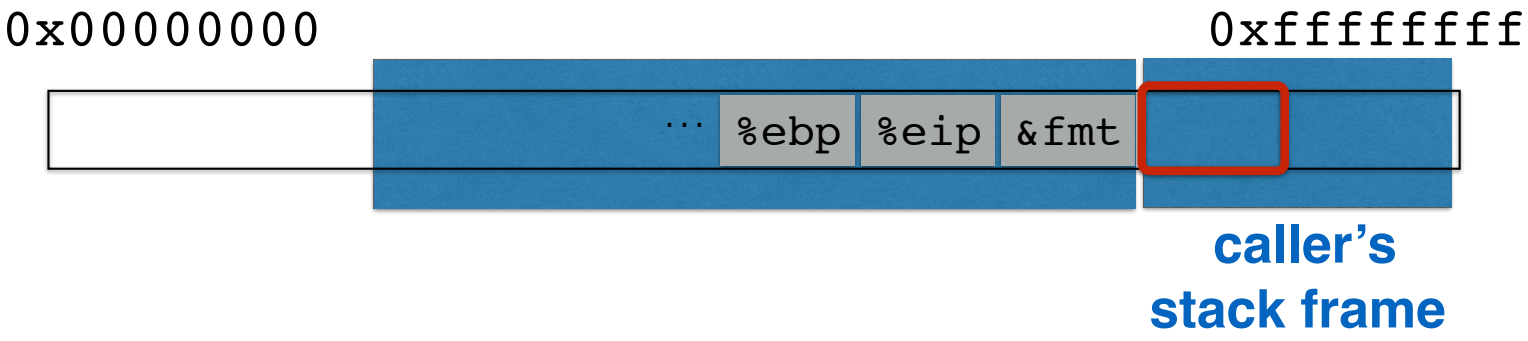
`printf` implementation



- printf takes variable number of arguments
- printf pays no mind to where the stack frame "ends"
- It presumes that you called it with (at least) as many arguments as specified in the format string

```
void vulnerable()
{
    char buf[80];
    if(fgets(buf, sizeof(buf), stdin)==NULL)
        return;
    printf(buf);
}
```

“%d %x"



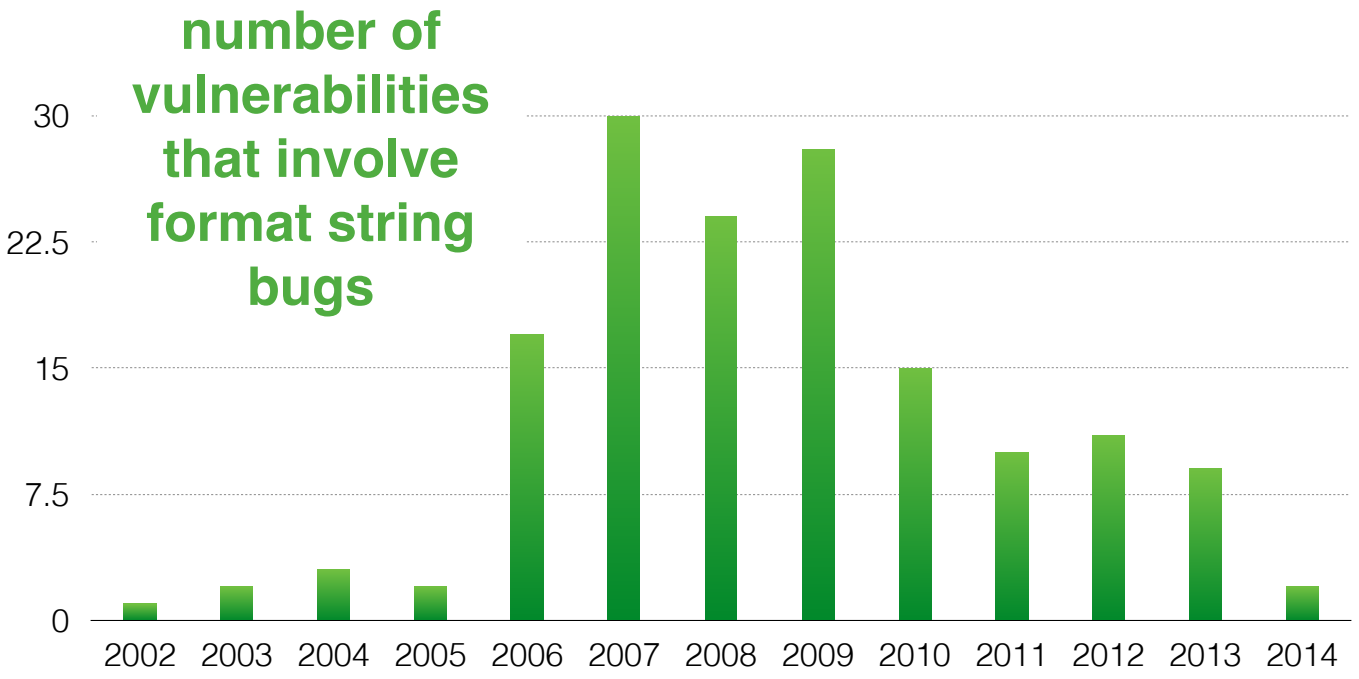
Format string vulnerabilities

- `printf("100% dave");`
  - Prints stack entry 4 byes above saved %eip
- `printf("%s");`
  - Prints bytes *pointed to* by that stack entry
- `printf("%d %d %d %d …");`
  - Prints a series of stack entries as integers
- `printf("%08x %08x %08x %08x …");`
  - Same, but nicely formatted hex
- `printf("100% no way!")`
  - ***WRITES*** the number 3 to address pointed to by stack entry

Why is this a buffer overflow?

- We should think of this as a buffer overflow in the sense that
  - The stack itself can be viewed as a kind of buffer
  - The size of that buffer is determined by the number and size of the arguments passed to a function
- Providing a bogus format string thus induces the program to overflow that “buffer”

Vulnerability prevalence



http://web.nvd.nist.gov/view/vuln/statistics

Time to switch hats

We have seen many styles of attack

What can be done to defend against them?