MEMORY SAFETY
ATTACKS & DEFENSES

CMSC 414

FEB 06 2018

```
void safe()
{
    char buf[80];
    fgets(buf, 80, stdin);
}
```

```
void safer()
{
    char buf[80];
    fgets(buf, sizeof(buf), stdin);
}
```

```
void safe()
{
    char buf[80];
    fgets(buf, 80, stdin);
}
```

```
void safer()
{
    char buf[80];
    fgets(buf, sizeof(buf), stdin);
}
```

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

```
void safe()
{
    char buf[80];
    fgets(buf, 80, stdin);
}
```

```
void safer()
{
    char buf[80];
    fgets(buf, sizeof(buf), stdin);
}
```

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

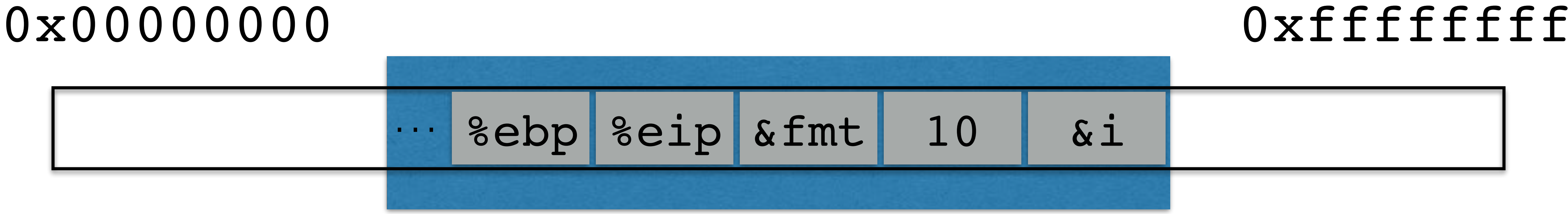
FORMAT STRING VULNERABILITIES

PRINTF FORMAT STRINGS

```
int i = 10;
printf(“%d %p\n”, i, &i);
```

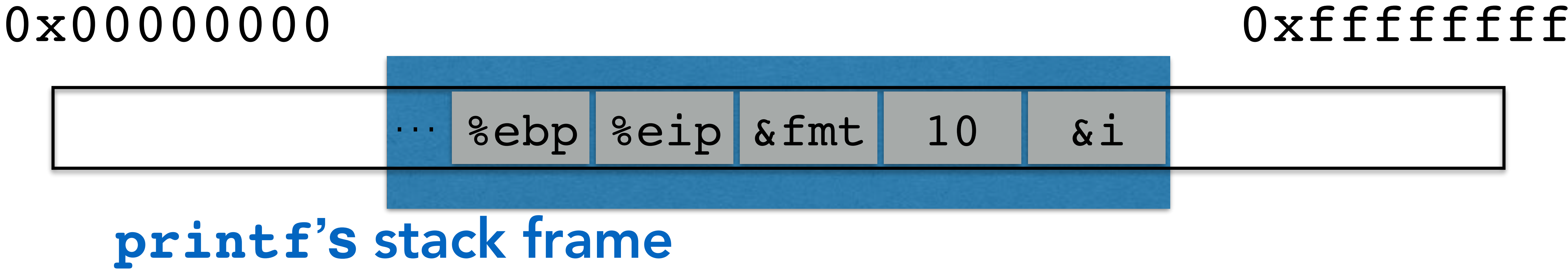
PRINTF FORMAT STRINGS

```
int i = 10;
printf(“%d %p\n”, i, &i);
```



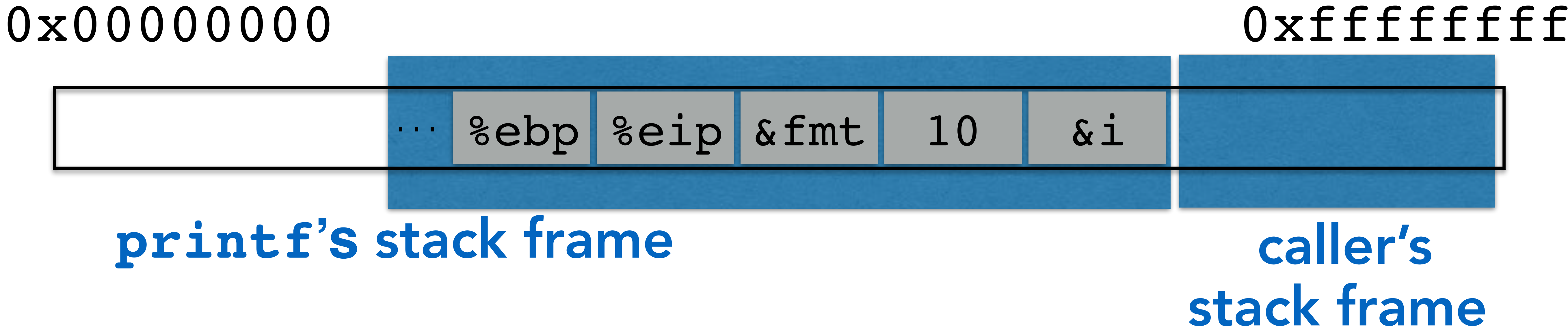
PRINTF FORMAT STRINGS

```
int i = 10;
printf("%d %p\n", i, &i);
```



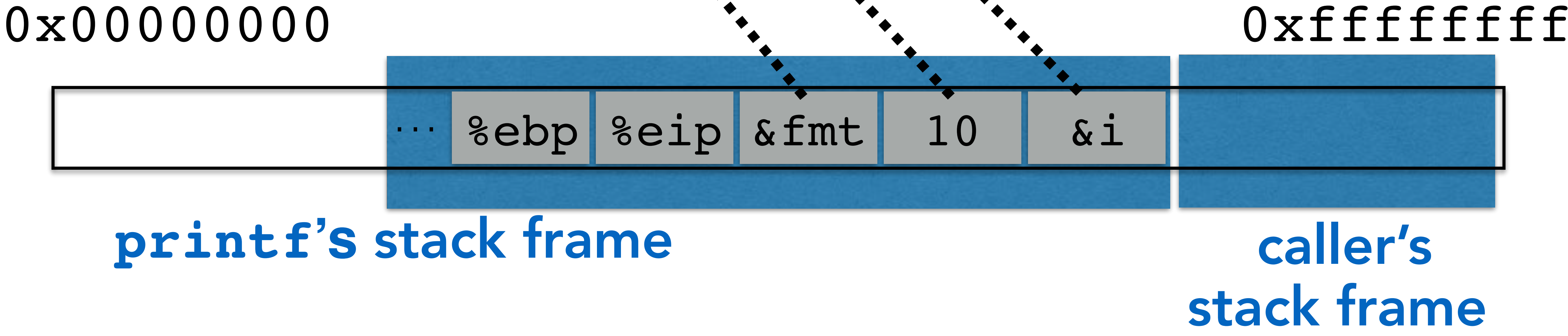
PRINTF FORMAT STRINGS

```
int i = 10;
printf(“%d %p\n”, i, &i);
```



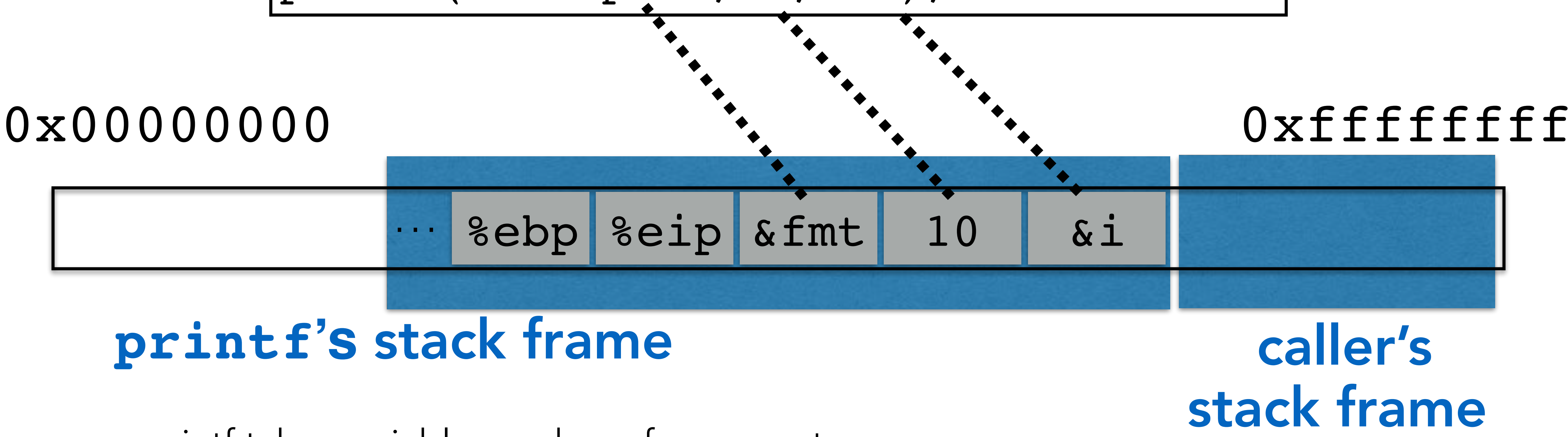
PRINTF FORMAT STRINGS

```
int i = 10;
printf("%d %p\n", i, &i);
```



PRINTF FORMAT STRINGS

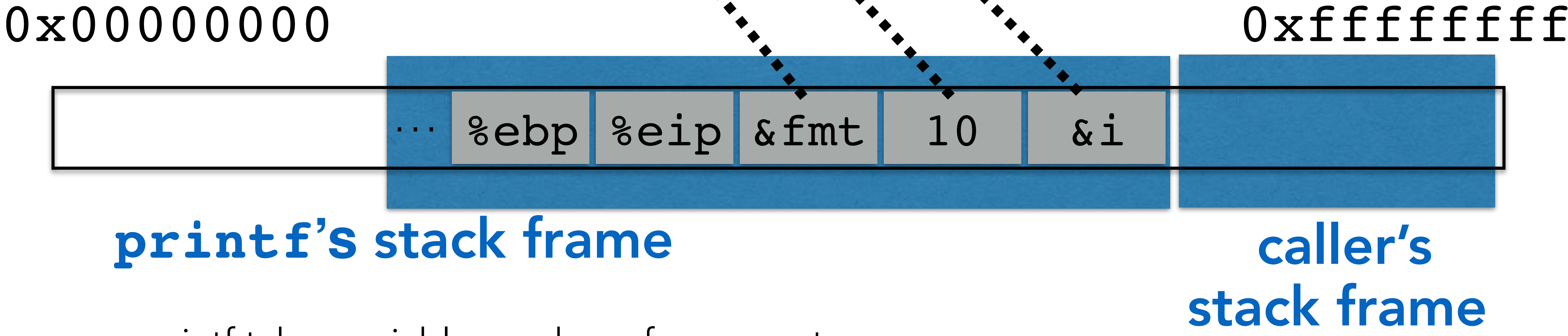
```
int i = 10;
printf("%d %p\n", i, &i);
```



- 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

PRINTF FORMAT STRINGS

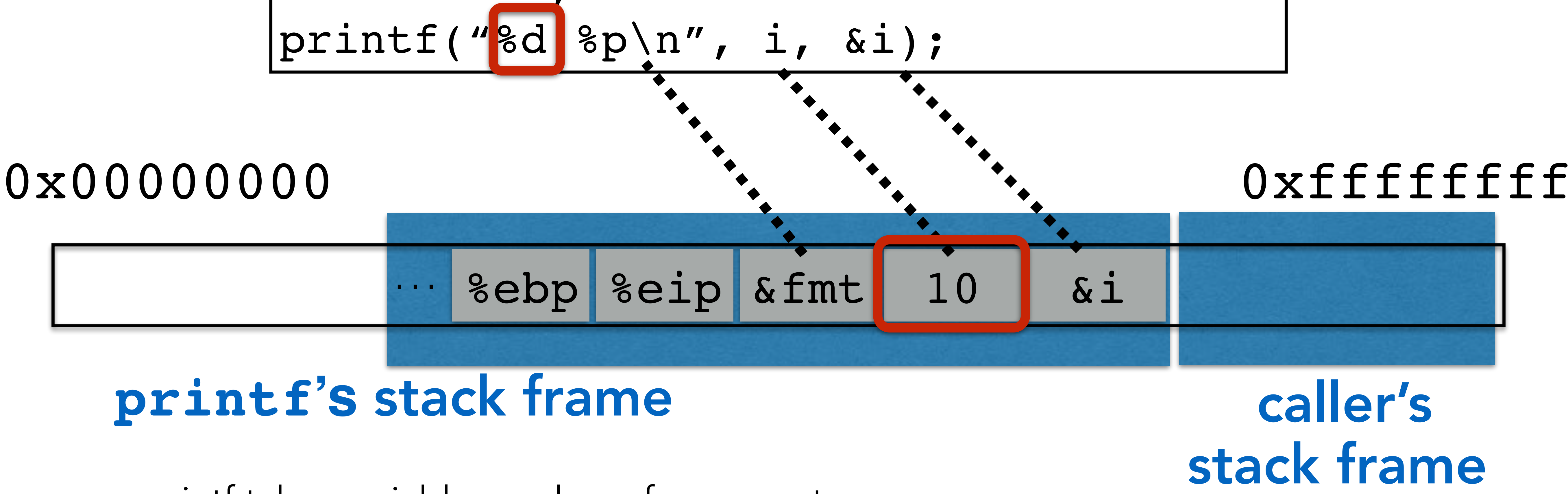
```
int i = 10;
printf("%d %p\n", i, &i);
```



- 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

PRINTF FORMAT STRINGS

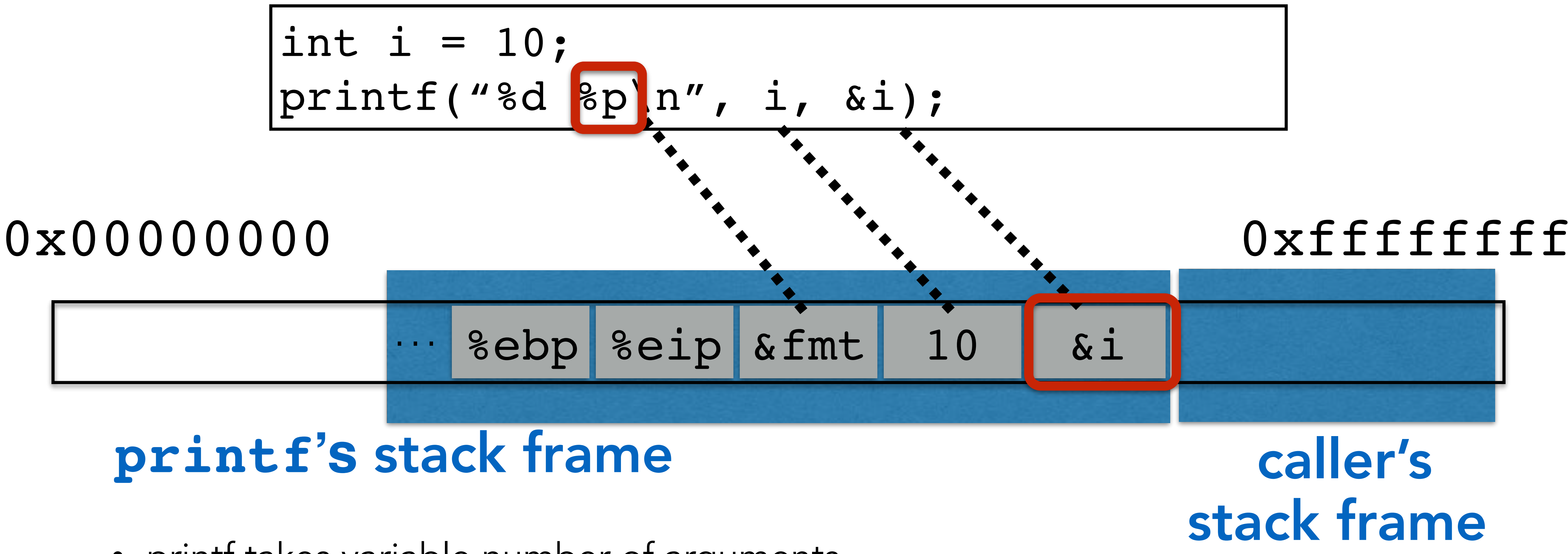
```
int i = 10;
printf("%d %p\n", i, &i);
```



- 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

PRINTF FORMAT STRINGS

```
int i = 10;
printf("%d %p\n", i, &i);
```



- 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

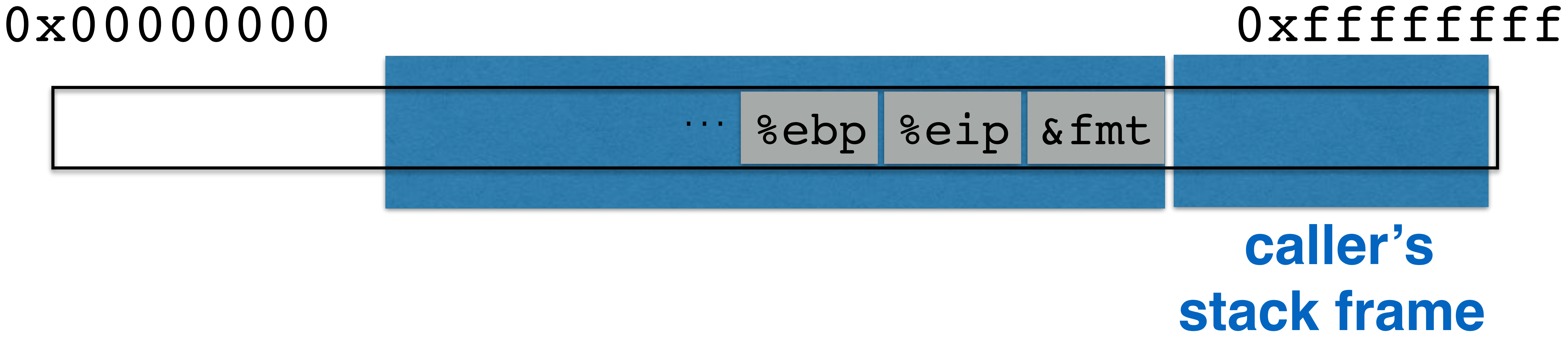
```
void vulnerable()
{
    char buf[80];
    if(fgets(buf, sizeof(buf), stdin)==NULL)
        return;
    printf(buf);
}
```

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

“%d %x"

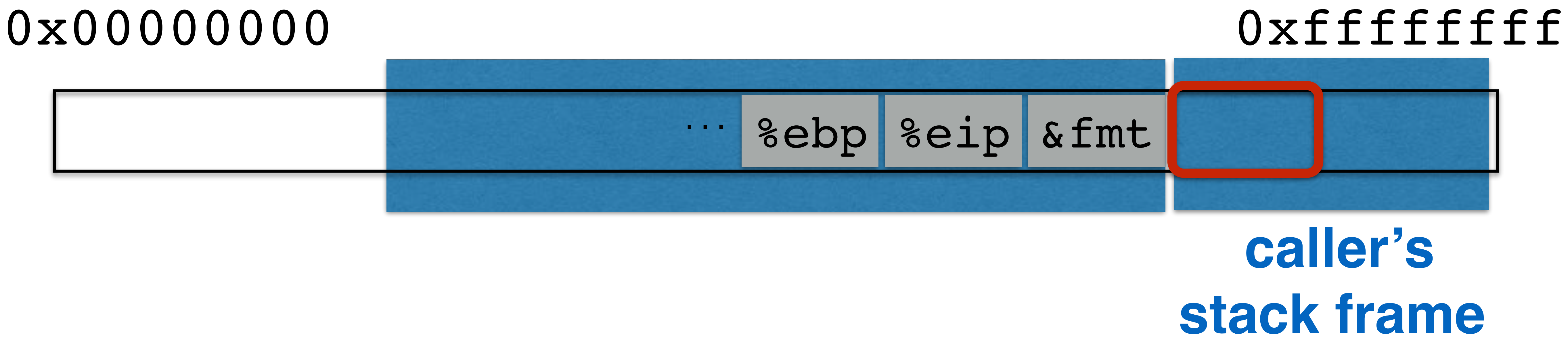
```
void vulnerable()
{
    char buf[80];
    if(fgets(buf, sizeof(buf), stdin)==NULL)
        return;
    printf(buf);
}
```

“%d %x"



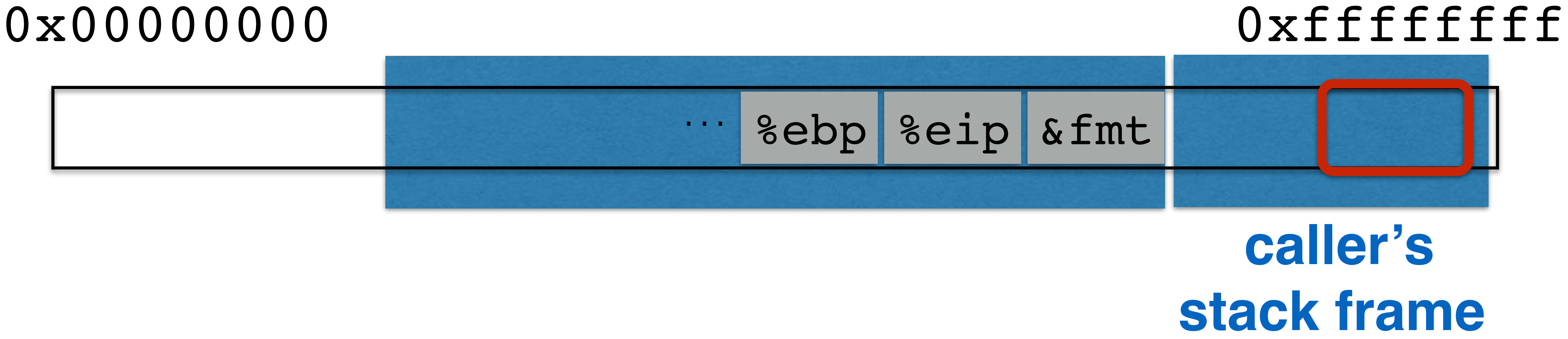
```
void vulnerable()
{
    char buf[80];
    if(fgets(buf, sizeof(buf), stdin)==NULL)
        return;
    printf(buf);
}
```

“%d %x"



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

“%d %x"



FORMAT STRING VULNERABILITIES

FORMAT STRING VULNERABILITIES

- `printf(“100% dml”);`

FORMAT STRING VULNERABILITIES

- `printf(“100% dml”);`
    - Prints stack entry 4 byes above saved %eip

FORMAT STRING VULNERABILITIES

- `printf(“100% dml”);`
  - Prints stack entry 4 byes above saved %eip
- `printf(“%s”);`

FORMAT STRING VULNERABILITIES

- `printf(“100% dml”);`
    - Prints stack entry 4 byes above saved %eip
- `printf(“%s”);`
    - Prints bytes *pointed to* by that stack entry

FORMAT STRING VULNERABILITIES

- `printf(“100% dml”);`
  - Prints stack entry 4 byes above saved %eip
- `printf(“%s”);`
  - Prints bytes *pointed to* by that stack entry
- `printf(“%d %d %d %d …”);`

FORMAT STRING VULNERABILITIES

- `printf(“100% dml”);`
  - 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

FORMAT STRING VULNERABILITIES

- `printf("100% dml");`
  - 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 …");`

FORMAT STRING VULNERABILITIES

- `printf(“100% dml”);`
  - 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

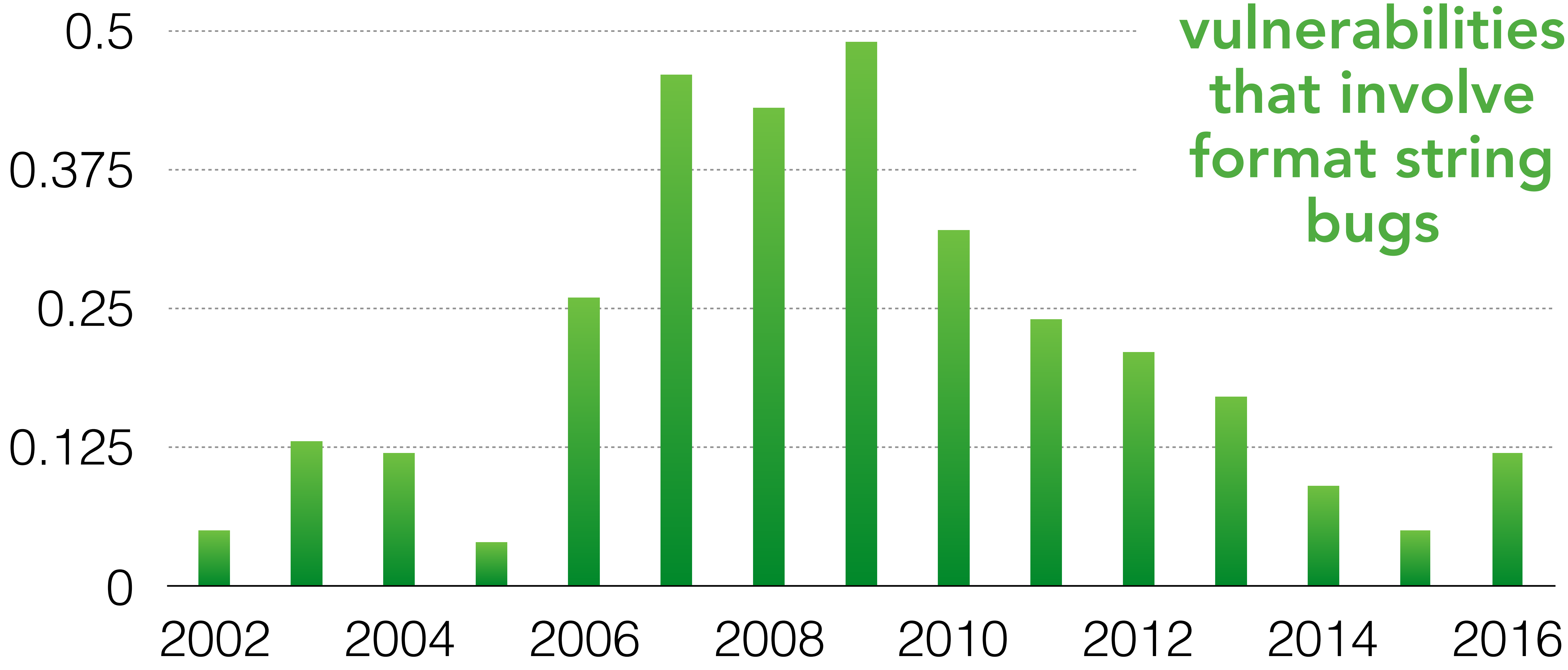
FORMAT STRING VULNERABILITIES

- `printf(“100% dml”);`
  - 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!”)`

FORMAT STRING VULNERABILITIES

- `printf(“100% dml”);`
  - 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

FORMAT STRING PREVALENCE



% of vulnerabilities that involve format string bugs

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

WHAT'S WRONG WITH THIS CODE?

```
#define BUF_SIZE 16
char buf[BUF_SIZE];
void vulnerable()
{
    int len = read_int_from_network();
    char *p = read_string_from_network();
    if(len > BUF_SIZE) {
        printf("Too large\n");
        return;
    }
    memcpy(buf, p, len);
}
```

WHAT'S WRONG WITH THIS CODE?

```
#define BUF_SIZE 16
char buf[BUF_SIZE];
void vulnerable()
{
    int len = read_int_from_network();
    char *p = read_string_from_network();
    if(len > BUF_SIZE) {
        printf(“Too large\n”);
        return;
    }
    memcpy(buf, p, len);
}
```

```
void *memcpy(void *dest, const void *src, size_t n);
```

WHAT'S WRONG WITH THIS CODE?

```
#define BUF_SIZE 16
char buf[BUF_SIZE];
void vulnerable()
{
    int len = read_int_from_network();
    char *p = read_string_from_network();
    if(len > BUF_SIZE) {
        printf(“Too large\n”);
        return;
    }
    memcpy(buf, p, len);
}
```

```
void *memcpy(void *dest, const void *src, size_t n);
typedef unsigned int size_t;
```

WHAT'S WRONG WITH THIS CODE?

```
#define BUF_SIZE 16
char buf[BUF_SIZE];
void vulnerable()
{   Negative
    int len = read_int_from_network();
    char *p = read_string_from_network();
    if(len > BUF_SIZE) {
        printf("Too large\n");
        return;
    }
    memcpy(buf, p, len);
}
```

```
void *memcpy(void *dest, const void *src, size_t n);
typedef unsigned int size_t;
```

WHAT'S WRONG WITH THIS CODE?

```
#define BUF_SIZE 16
char buf[BUF_SIZE];
void vulnerable()
{   Negative
    int len = read_int_from_network();
    char *p = read_string_from_network();
 Ok if(len > BUF_SIZE) {
        printf("Too large\n");
        return;
    }
    memcpy(buf, p, len);
}
```

```
void *memcpy(void *dest, const void *src, size_t n);
typedef unsigned int size_t;
```

WHAT'S WRONG WITH THIS CODE?

```
#define BUF_SIZE 16
char buf[BUF_SIZE];
void vulnerable()
{   Negative
    int len = read_int_from_network();
    char *p = read_string_from_network();
 Ok if(len > BUF_SIZE) {
        printf("Too large\n");
        return;
    }
    memcpy(buf, p, len);
}                Implicit cast to unsigned
```

```
void *memcpy(void *dest, const void *src, size_t n);
typedef unsigned int size_t;
```

INTEGER OVERFLOW
VULNERABILITIES

WHAT'S WRONG WITH THIS CODE?

```
void vulnerable()
{
    size_t len;
    char *buf;

    len = read_int_from_network();
    buf = malloc(len + 5);
    read(fd, buf, len);
    ...
}
```

WHAT'S WRONG WITH THIS CODE?

```
void vulnerable()
{
    size_t len;
    char *buf;
  HUGE
    len = read_int_from_network();
    buf = malloc(len + 5);
    read(fd, buf, len);
    ...
}
```

WHAT'S WRONG WITH THIS CODE?

```
void vulnerable()
{
    size_t len;
    char *buf;

    len = read_int_from_network();
    buf = malloc(len + 5);
    read(fd, buf, len);
    ...
}
```

HUGE

Wrap-around

WHAT'S WRONG WITH THIS CODE?

HUGE

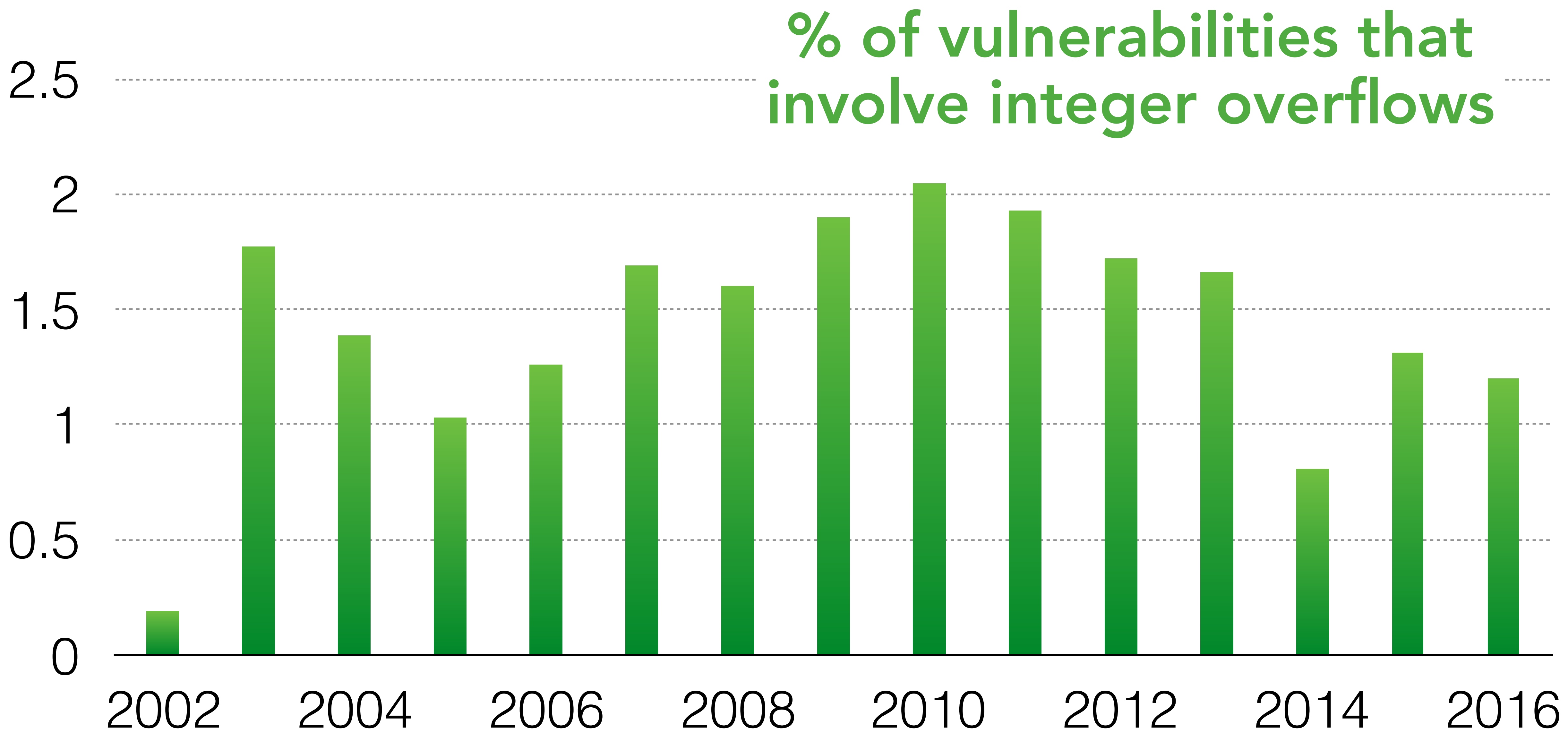
Wrap-around

```
void vulnerable()
{
    size_t len;
    char *buf;

    len = read_int_from_network();
    buf = malloc(len + 5);
    read(fd, buf, len);
    ...
}
```

Takeaway: You have to know the semantics of your programming language to avoid these errors

INTEGER OVERFLOW PREVALENCE



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