MEMORY SAFETY
ATTACKS & DEFENSES

CMSC 414

FEB 07 2019

```
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);
}
```

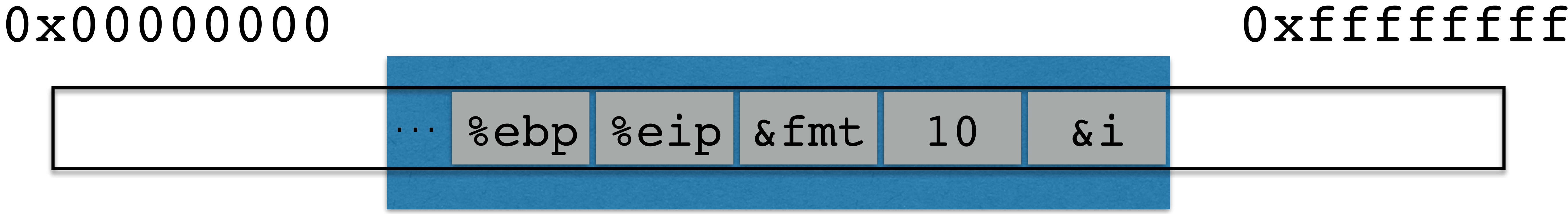
FORMAT STRING
VULNERABILITIES

PRINTF FORMAT STRINGS

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

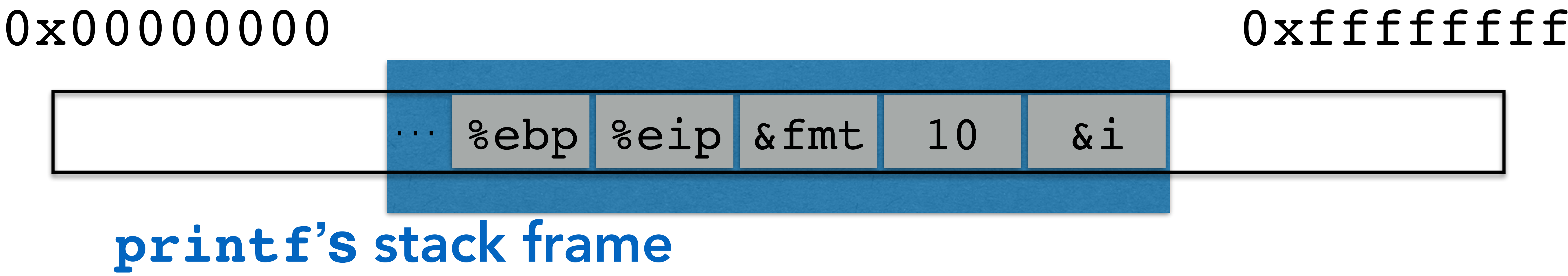
PRINTF FORMAT STRINGS

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



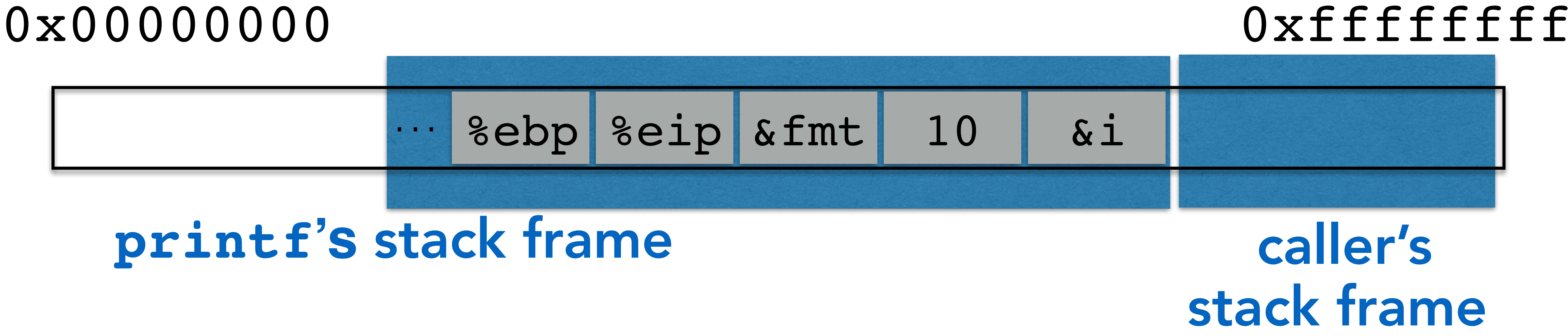
PRINTF FORMAT STRINGS

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

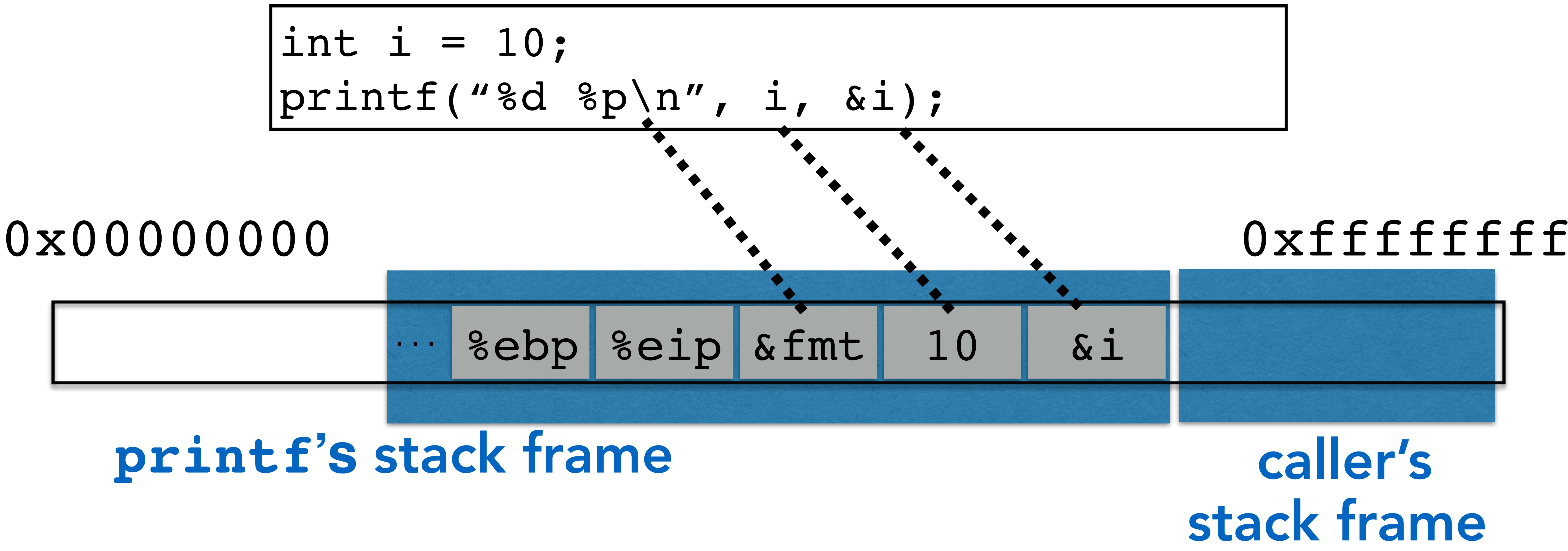


PRINTF FORMAT STRINGS

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



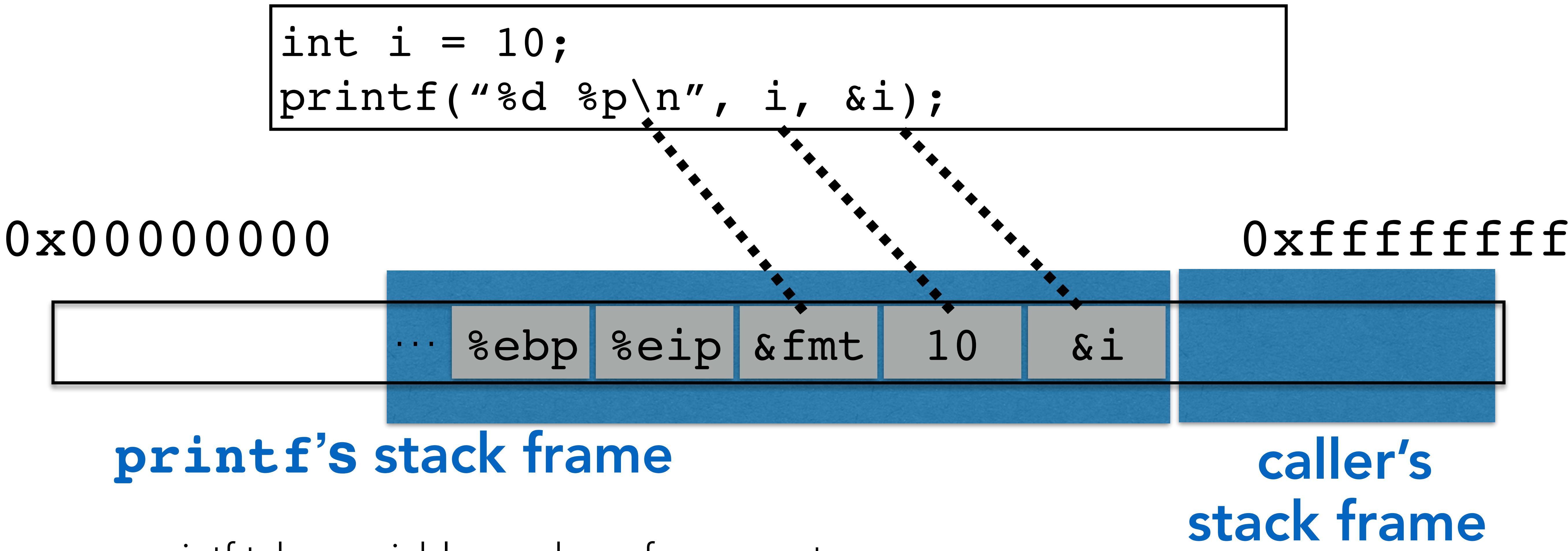
PRINTF FORMAT STRINGS



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

PRINTF FORMAT STRINGS

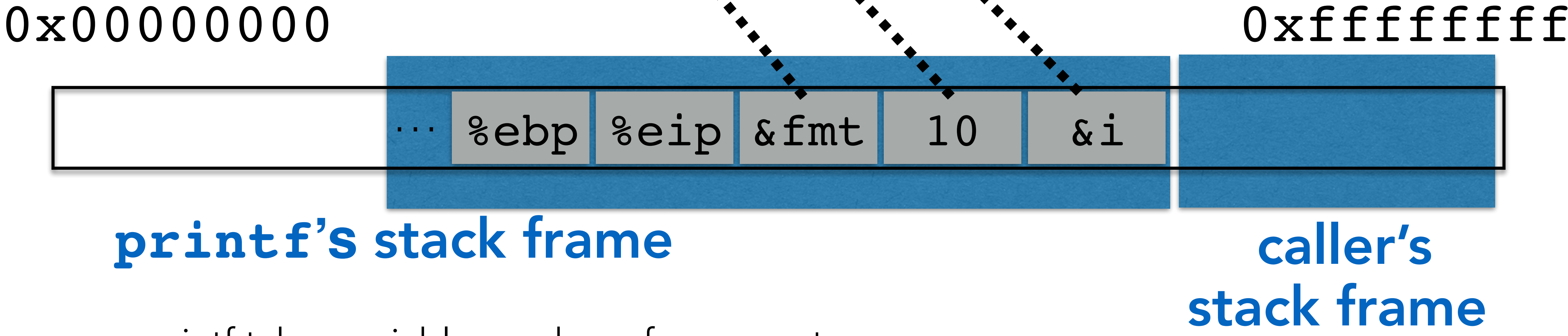
```
int i = 10;
printf(“%d %p\n”, i, &i);
```



- 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

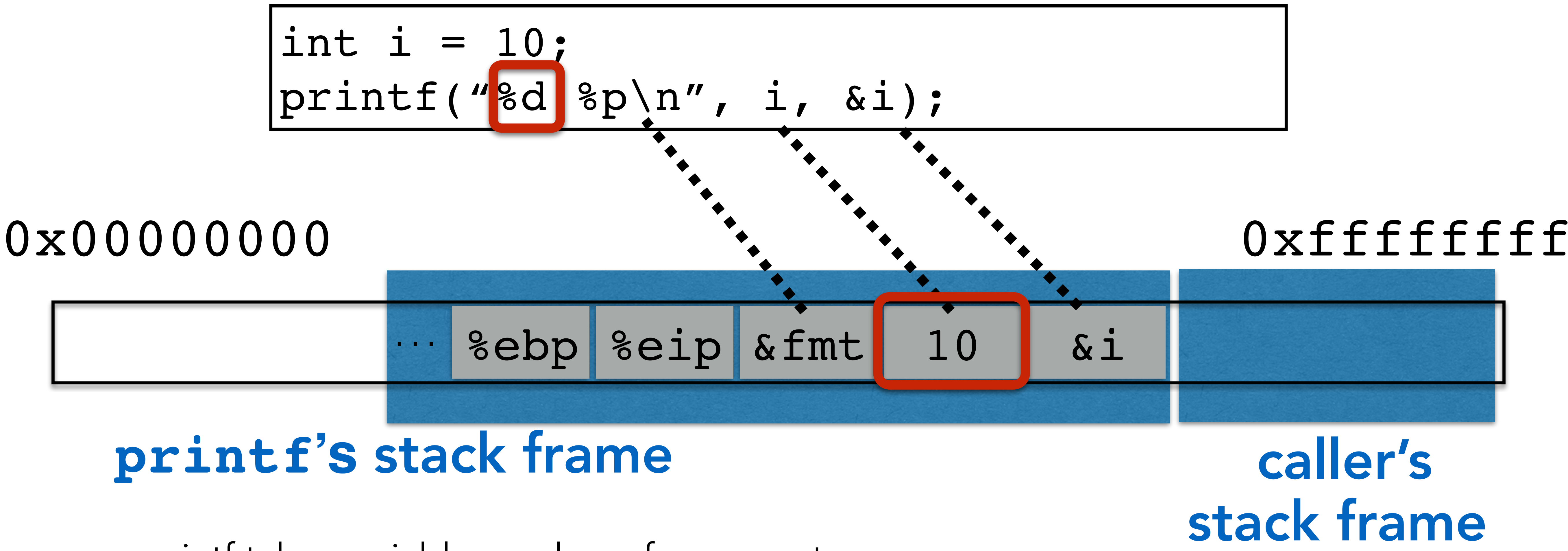
```
int i = 10;
printf("%d %p\n", i, &i);
```



- 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

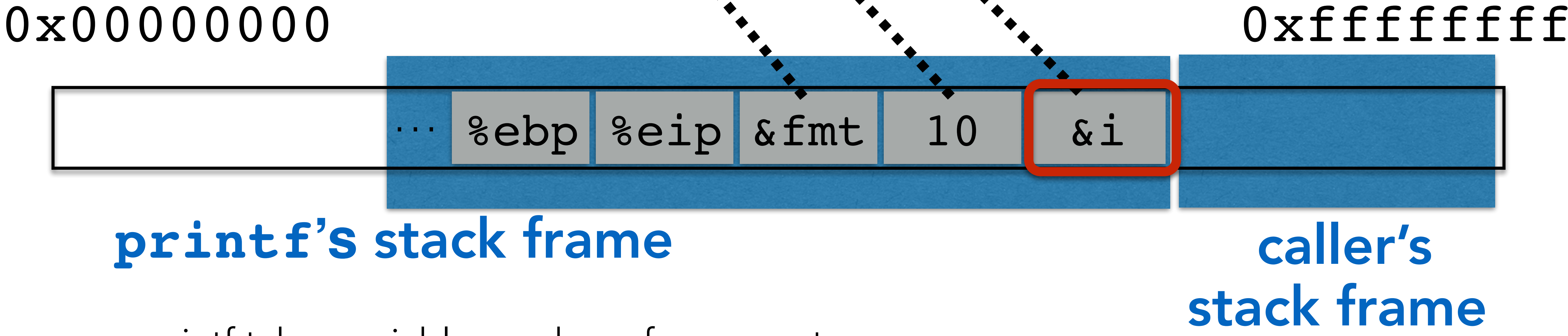
```
int i = 10;
printf("%d %p\n", i, &i);
```



- 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

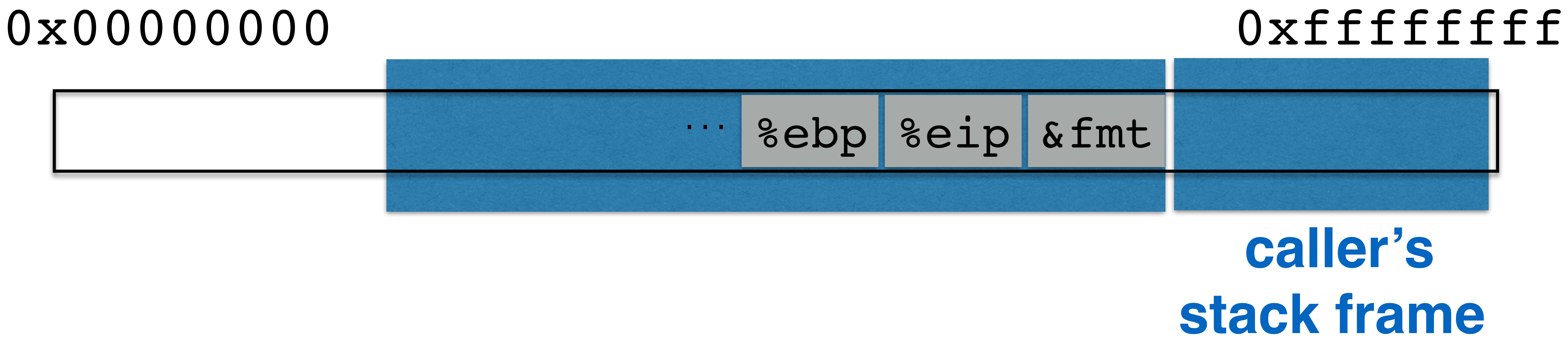
```
void vulnerable()
{
    char buf[80];
    if(fgets(buf, sizeof(buf), stdin)==NULL)
        return;
    printf(buf);
}
```

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

“%d %x"

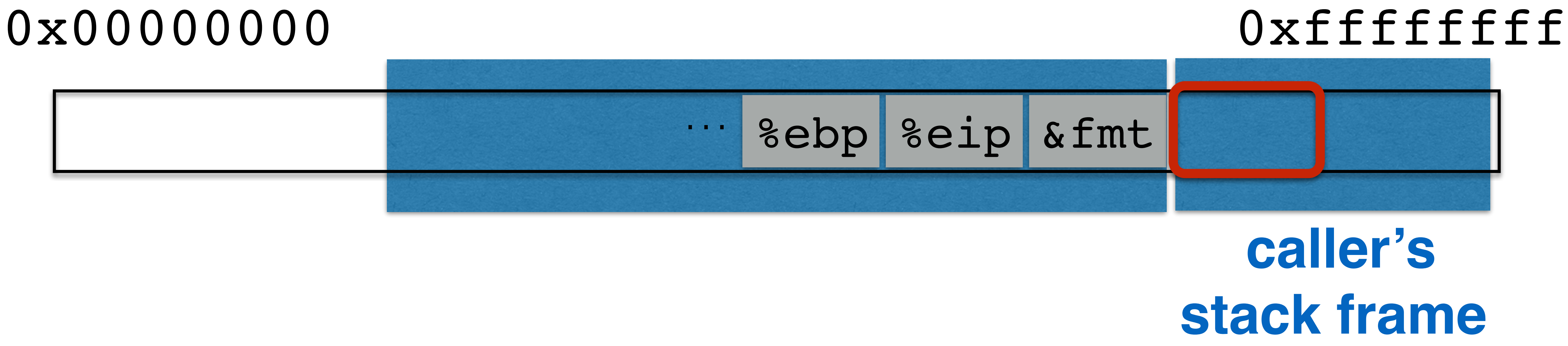
```
void vulnerable()
{
    char buf[80];
    if(fgets(buf, sizeof(buf), stdin)==NULL)
        return;
    printf(buf);
}
```

“%d %x"



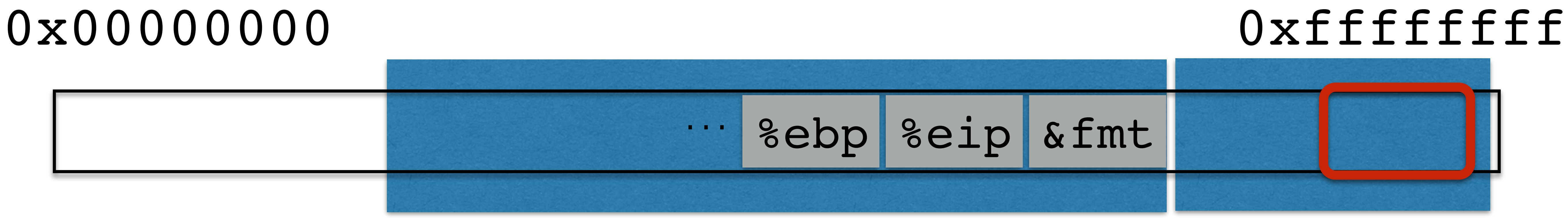
```
void vulnerable()
{
    char buf[80];
    if(fgets(buf, sizeof(buf), stdin)==NULL)
        return;
    printf(buf);
}
```

“%d %x"



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

“%d %x"



caller’s stack frame

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

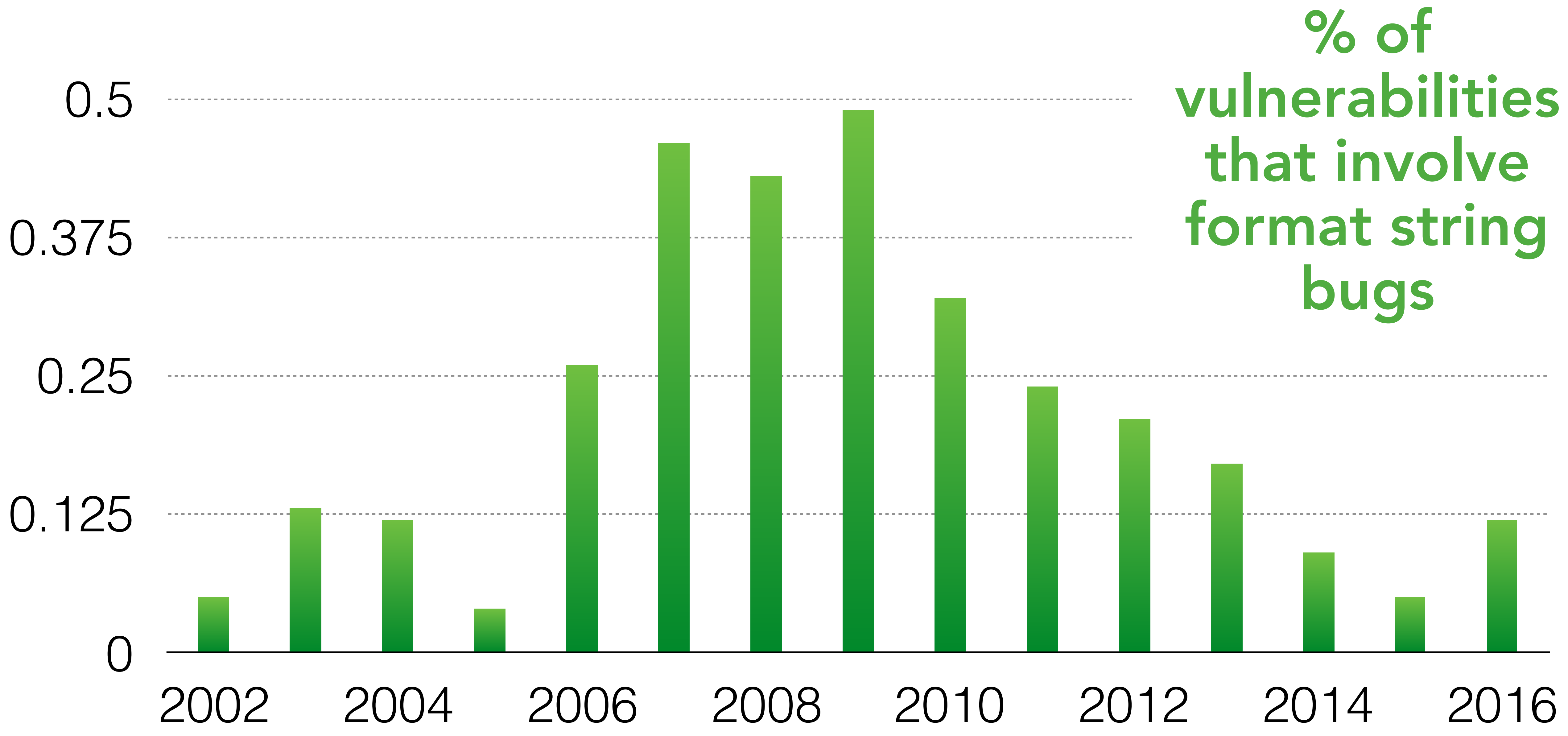
FORMAT STRING VULNERABILITIES

- `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



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;

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

HUGE

WHAT'S WRONG WITH THIS CODE?

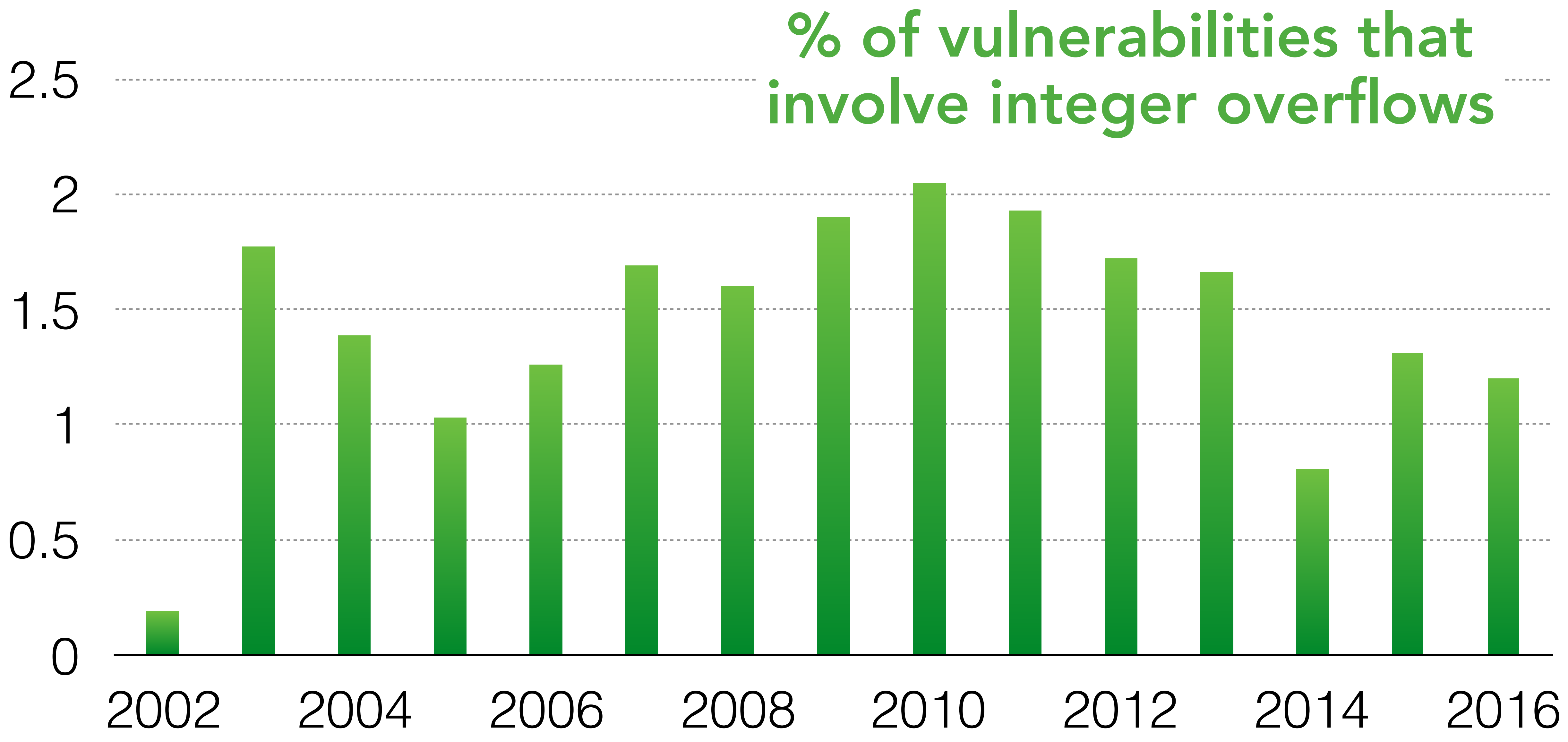
```
void vulnerable()
{
    size_t len;
    char *buf;
  HUGE
    len = read_int_from_network();
    buf = malloc(len + 5); Wrap-around
    read(fd, buf, len);
    ...
}
```

WHAT'S WRONG WITH THIS CODE?

```
void vulnerable()
{
    size_t len;
    char *buf;
  HUGE
    len = read_int_from_network();
    buf = malloc(len + 5); Wrap-around
    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

What’s wrong with this code?

Suppose that it has higher privilege than the user

```
int main() {
    char buf[1024];
    ...
    if(access(argv[1], R_OK) != 0) {
        printf(“cannot access file\n”);
        exit(-1);
    }

    file = open(argv[1], O_RDONLY);
    read(file, buf, 1023);
    close(file);
    printf(“%s\n”, buf);
    return 0;
}
```

What’s wrong with this code?

Suppose that it has higher privilege than the user

uid

euid

```
int main() {
    char buf[1024];
    ...
    if(access(argv[1], R_OK) != 0) {
        printf(“cannot access file\n”);
        exit(-1);
    }

    file = open(argv[1], O_RDONLY);
    read(file, buf, 1023);
    close(file);
    printf(“%s\n”, buf);
    return 0;
}
```

What’s wrong with this code?

Suppose that it has higher privilege than the user

~attacker/mystuff.txt

uid

euid

```
int main() {
    char buf[1024];
    ...
    if(access(argv[1], R_OK) != 0) {
        printf(“cannot access file\n”);
        exit(-1);
    }

    file = open(argv[1], O_RDONLY);
    read(file, buf, 1023);
    close(file);
    printf(“%s\n”, buf);
    return 0;
}
```

What’s wrong with this code?

Suppose that it has higher privilege than the user

~attacker/mystuff.txt

uid

euid

```
int main() {
    char buf[1024];
    ...
    if(access(argv[1], R_OK) != 0) {
        printf(“cannot access file\n”);
        exit(-1);
    }

    file = open(argv[1], O_RDONLY);
    read(file, buf, 1023);
    close(file);
    printf(“%s\n”, buf);
    return 0;
}
```

What's wrong with this code?

Suppose that it has higher privilege than the user

uid

euid

~attacker/mystuff.txt

```
int main() {
    char buf[1024];
    ...
    if(access(argv[1], R_OK) != 0) {
        printf("cannot access file\n");
        exit(-1);
    }
```

ln -s /usr/sensitive ~attacker/mystuff.txt

```
    file = open(argv[1], O_RDONLY);
    read(file, buf, 1023);
    close(file);
    printf("%s\n", buf);
    return 0;
}
```

What’s wrong with this code?

Suppose that it has higher privilege than the user

~attacker/mystuff.txt

uid

```
int main() {
    char buf[1024];
    ...
    if(access(argv[1], R_OK) != 0) {
        printf(“cannot access file\n”);
        exit(-1);
    }
```

```
ln -s /usr/sensitive ~attacker/mystuff.txt
```

euid

```
    file = open(argv[1], O_RDONLY);
    read(file, buf, 1023);
    close(file);
    printf(“%s\n”, buf);
    return 0;
}
```

“Time of Check/Time of Use” Problem (TOCTOU)

Avoiding TOCTOU

```
      int main() {
          char buf[1024];
          ...
uid       if(access(argv[1], R_OK) != 0) {
              printf("cannot access file\n");
              exit(-1);
          }



euid      file = open(argv[1], O_RDONLY);
          read(file, buf, 1023);
          close(file);

          printf(buf);
      }
```

Avoiding TOCTOU

uid

euid

```
int main() {
    char buf[1024];
    ...
    if(access(argv[1], R_OK) != 0) {
        printf("cannot access file\n");
        exit(-1);
    }




    file = open(argv[1], O_RDONLY);
    read(file, buf, 1023);
    close(file);

    printf(buf);
}
```

Avoiding TOCTOU

uid

euid

```
int main() {
    char buf[1024];
    ...
    if(access(argv[1], R_OK) != 0) {
        printf("cannot access file\n");
        exit(-1);
    }
    euid = geteuid();
    uid = getuid();
    seteuid(uid);   // Drop privileges
    file = open(argv[1], O_RDONLY);
    read(file, buf, 1023);
    close(file);

    printf(buf);
}
```

Avoiding TOCTOU

```
int main() {
    char buf[1024];
    ...
    if(access(argv[1], R_OK) != 0) {
        printf("cannot access file\n");
        exit(-1);
    }
    euid = geteuid();
    uid = getuid();
    seteuid(uid);   // Drop privileges
    file = open(argv[1], O_RDONLY);
    read(file, buf, 1023);
    close(file);
    seteuid(euid);  // Restore privileges
    printf(buf);
}
```

uid

euid