Midterm 1 Review

Memory Safety & Web Attacks

What’s on the midterm?

- Lecture material through Feb 9
- Lecture notes ("Notes on ..." on the course site)
- Section material, per "Handout w/ solutions"
- Concepts developed in the homeworks
- Section 8 and Section 10 of the ASLR Smack & Laugh Reference for Project 1

What's *not* on the midterm?

- Bonus readings
- Historical anecdotes
- Krebs on Security .com
- UC Cyber Security Awareness Training

Reasoning about memory safety

- Preconditions: what must hold for function to operate correctly
- Postconditions: what holds after function completes
- Invariants: conditions that always hold at a given point in a function (this particularly matters for loops).
    - For simplicity, you can omit from your invariant any terms that appear in the precondition that will be true throughout the execution of the function.

Precondition

```
/* requires: p != NULL
       (and p a valid pointer) */
int deref(int *p) {
    return *p;
}
```

Precondition: what needs to hold for function to operate correctly.

Needs to be expressed in a way that a person writing code to call the function knows how to evaluate.

Postcondition

```
/* ensures:  retval != NULL (and a  valid  pointer) */
void *mymalloc(size_t  n)  {
    void *p   =  malloc(n);
    if (!p)   {
        perror("malloc");
        exit(1);
    }
    return p;
}
```

Postcondition: what the function promises will hold upon its return.

Memory Safety - Strategy

General correctness proof strategy for memory safety:

(1) Identify each point of memory access
(2) Write down precondition it requires
(3) Propagate requirement up to beginning of function

SP16 - Midterm 1 - Q5

```
/* Requires: ??? */
void shuffle(int a[], int b[], size_t m, size_t n) {
    for (size_t i = 0; i < m; i++) {
        int tmp = a[i];
        a[i] = b[n-i];
        b[n-i] = tmp;
    }
}
```

For each of the candidate preconditions in parts (a)–(d), answer whether that following **precondition** is sufficient to ensure that shuffle() will be memory-safe. If it is not sufficient, also **specify an example** of an input that would satisfy the precondition but could cause memory-unsafe behavior.

SP16 - Midterm 1 - Q5

```
/* Requires: ??? */
void shuffle(int a[], int b[], size_t m, size_t n) {
    for (size_t i = 0; i < m; i++) {
        int tmp = a[i];
        a[i] = b[n-i];
        b[n-i] = tmp;
    }
}
```

Precondition: `a != NULL && b != NULL`

SUFFICIENT **INSUFFICIENT**

SP16 - Midterm 1 - Q5

```
/* Requires: ??? */
void shuffle(int a[], int b[], size_t m, size_t n) {
    for (size_t i = 0; i < m; i++) {
        int tmp = a[i];
        a[i] = b[n-i];
        b[n-i] = tmp;
    }
}
```

Precondition: `a != NULL && b != NULL`

SUFFICIENT

INSUFFICIENT

SP16 - Midterm 1 - Q5

```
/* Requires: ??? */
void shuffle(int a[], int b[], size_t m, size_t n) {
    for (size_t i = 0; i < m; i++) {
        int tmp = a[i];
        a[i] = b[n-i];
        b[n-i] = tmp;
    }
}
```

Precondition: `a != NULL && b != NULL`

Example of how to exploit:
shuffle(?, ?, ?, ?);

SP16 - Midterm 1 - Q5

```
/* Requires: ??? */
void shuffle(int a[], int b[], size_t m, size_t n) {
    for (size_t i = 0; i < m; i++) {
        int tmp = a[i];
        a[i] = b[n-i];
        b[n-i] = tmp;
    }
}
```

Precondition: `a != NULL && b != NULL`

Example of how to exploit:
shuffle({0}, {0}, 2, 1);

SP16 - Midterm 1 - Q5

```
/* Requires: ??? */
void shuffle(int a[], int b[], size_t m, size_t n) {
    for (size_t i = 0; i < m; i++) {
        int tmp = a[i];
        a[i] = b[n-i];
        b[n-i] = tmp;
    }
}
```

Precondition: `a != NULL && b != NULL && m == 0 && n == 0`

SUFFICIENT **INSUFFICIENT**

SP16 - Midterm 1 - Q5

```
/* Requires: ??? */
void shuffle(int a[], int b[], size_t m, size_t n) {
    for (size_t i = 0; i < m; i++) {
        int tmp = a[i];
        a[i] = b[n-i];
        b[n-i] = tmp;
    }
}
```

Precondition: `a != NULL && b != NULL && m == 0 && n == 0`

SUFFICIENT

INSUFFICIENT

SP16 - Midterm 1 - Q5

```
/* Requires: ??? */
void shuffle(int a[], int b[], size_t m, size_t n) {
    for (size_t i = 0; i < m; i++) {
        int tmp = a[i];
        a[i] = b[n-i];
        b[n-i] = tmp;
    }
}
```

Precondition: `a != NULL && b != NULL && m == n`

SUFFICIENT **INSUFFICIENT**

SP16 - Midterm 1 - Q5

```
/* Requires: ??? */
void shuffle(int a[], int b[], size_t m, size_t n) {
    for (size_t i = 0; i < m; i++) {
        int tmp = a[i];
        a[i] = b[n-i];
        b[n-i] = tmp;
    }
}
```

Precondition: `a != NULL && b != NULL && m == n`

SUFFICIENT

INSUFFICIENT

SP16 - Midterm 1 - Q5

```
/* Requires: ??? */
void shuffle(int a[], int b[], size_t m, size_t n) {
    for (size_t i = 0; i < m; i++) {
        int tmp = a[i];
        a[i] = b[n-i];
        b[n-i] = tmp;
    }
}
```

Precondition: `a != NULL && b != NULL && m == n`

Example of how to exploit:
shuffle(?, ?, ?, ?);

SP16 - Midterm 1 - Q5

```
/* Requires: ??? */
void shuffle(int a[], int b[], size_t m, size_t n) {
    for (size_t i = 0; i < m; i++) {
        int tmp = a[i];
        a[i] = b[n-i];
        b[n-i] = tmp;
    }
}
```

Precondition: `a != NULL && b != NULL && m == n`

Example of how to exploit:
shuffle({0}, {0}, 2, 2);

SP16 - Midterm 1 - Q5

```
/* Requires: ??? */
void shuffle(int a[], int b[], size_t m, size_t n) {
    for (size_t i = 0; i < m; i++) {
        int tmp = a[i];
        a[i] = b[n-i];
        b[n-i] = tmp;
    }
}
```

Precondition: `a != NULL && b != NULL && m < size(a) && n < size(b)`

SUFFICIENT **INSUFFICIENT**

SP16 - Midterm 1 - Q5

```
/* Requires: ??? */
void shuffle(int a[], int b[], size_t m, size_t n) {
    for (size_t i = 0; i < m; i++) {
        int tmp = a[i];
        a[i] = b[n-i];
        b[n-i] = tmp;
    }
}
```

Precondition: `a != NULL && b != NULL && m < size(a) && n < size(b)`

SUFFICIENT

INSUFFICIENT

SP16 - Midterm 1 - Q5

```
/* Requires: ??? */
void shuffle(int a[], int b[], size_t m, size_t n) {
    for (size_t i = 0; i < m; i++) {
        int tmp = a[i];
        a[i] = b[n-i];
        b[n-i] = tmp;
    }
}
```

Precondition: `a != NULL && b != NULL && m < size(a) && n < size(b)`

Example of how to exploit:

shuffle(?, ?, ?, ?);

SP16 - Midterm 1 - Q5

```
/* Requires: ??? */
void shuffle(int a[], int b[], size_t m, size_t n) {
    for (size_t i = 0; i < m; i++) {
        int tmp = a[i];
        a[i] = b[n-i];
        b[n-i] = tmp;
    }
}
```

Precondition: `a != NULL && b != NULL && m < size(a) && n < size(b)`

Example of how to exploit:

shuffle({0,1,2}, {0}, 2, 0);

SP16 - Midterm 1 - Q5

```
/* Requires: ??? */
void shuffle(int a[], int b[], size_t m, size_t n) {
    for (size_t i = 0; i < m; i++) {
        int tmp = a[i];
        a[i] = b[n-i];
        b[n-i] = tmp;
    }
}
```

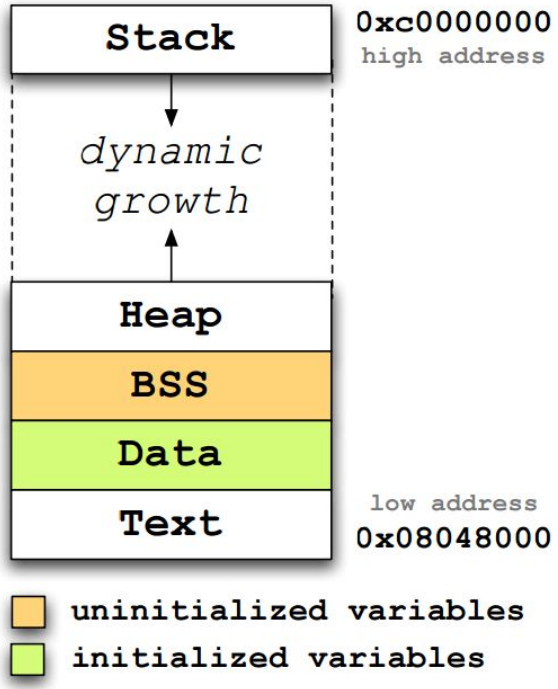
Suggest a better precondition. Your precondition should be sufficient to ensure that shuffle() is memory-safe, and be as general as possible. Don't worry about what shuffle() is trying to accomplish; it just needs to be memory-safe.

SP16 - Midterm 1 - Q5

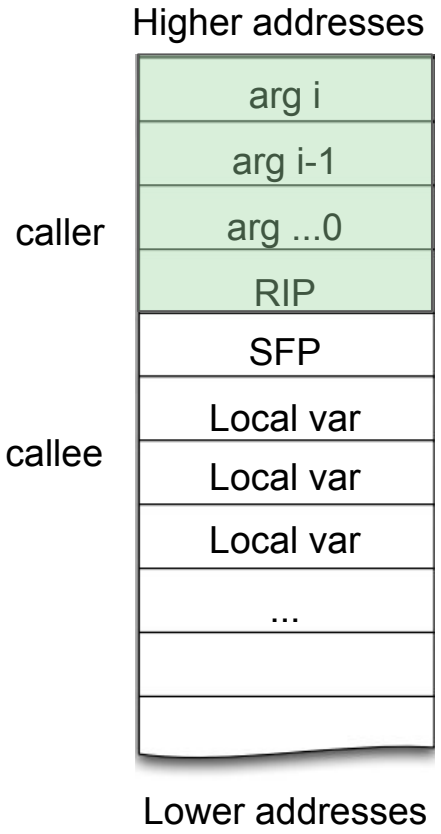
```
/* Requires: a != NULL &&
             b != NULL &&
             m <= size(a) &&
             n < size(b) &&
             m <= n+1
 */

void shuffle(int a[], int b[], size_t m, size_t n) {
    for (size_t i = 0; i < m; i++) {
        int tmp = a[i];
        a[i] = b[n-i];
        b[n-i] = tmp;
    }
}
```

Memory Layout

- **Stack**
  - grows towards *decreasing* addresses.
  - is initialized at *run-time*.
- **Heap**
  - grow towards *increasing* addresses.
  - is initialized at *run-time*.
- **BSS** section
  - size fixed at *compile-time*.
  - is initialized at *run-time*.
  - was grouped into **Data** in CS61C.
- **Data** section
  - is initialized at *compile-time*.
- **Text** section
  - holds the program instructions (read-only).



Stack Layout



x86 Instructions

`push` - decrements `ESP` by 4, then places its operand into the contents of the 32-bit location at address `[ESP]`

`pop` - moves the 4 bytes located at memory location `[ESP]` into the specified register or memory location, and then increments `ESP` by 4

`call` - pushes the current code location onto the stack and then performs an unconditional jump to the code location indicated by the label operand

`leave` - moves `ESP` to `EBP` then pops `EBP`

`ret` - pops a code location off the stack. It then performs an unconditional jump to the retrieved code location.

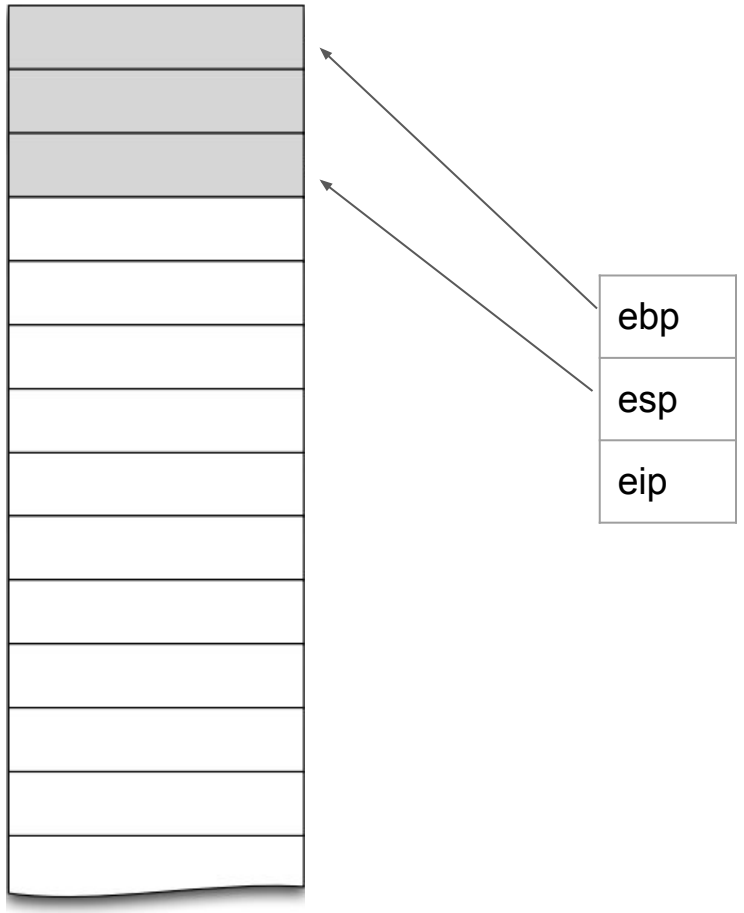
x86 Instructions

```
void foo(int a, int b, int c) {
    int bar[2];
    char qux[3];
    bar[0] = 'A';
    qux[0] = 0x2a;
}

int main(void) {
    int i = 1;
    foo(1, 2, 3);
    return 0;
}
```

```
main:
    pushl %ebp
    movl %esp,%ebp
    subl $4,%esp
    movl $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call foo
    addl $12,%esp
    xorl %eax,%eax
    leave ret
```

x86 Instructions

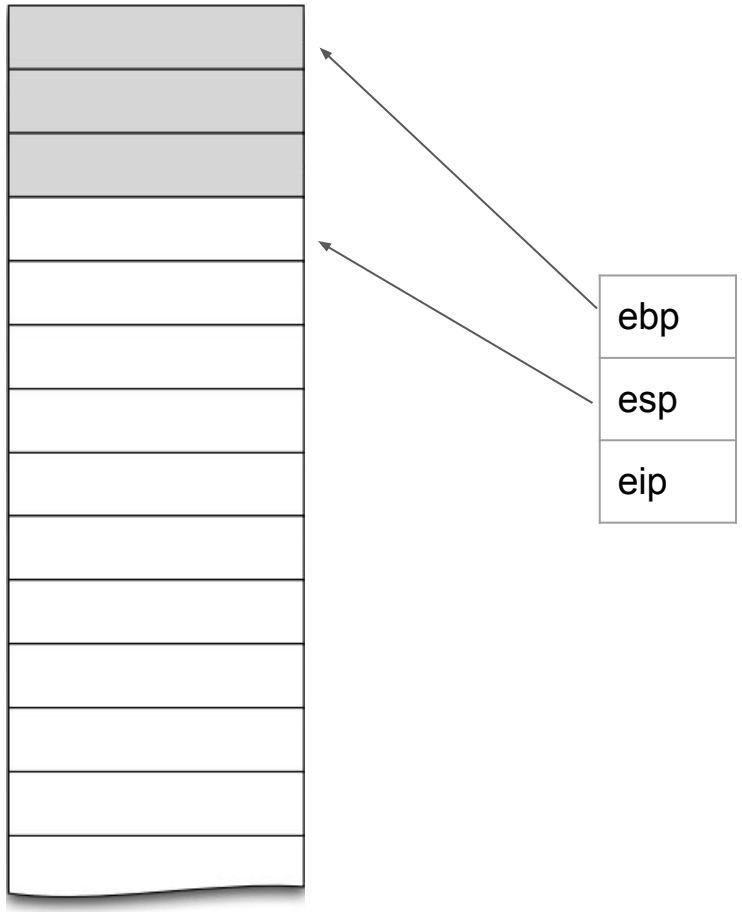
```
main:
     pushl %ebp
     movl %esp,%ebp
     subl $4,%esp
     movl $1,-4(%ebp)
     pushl $3
     pushl $2
     pushl $1
     call foo
     addl $12,%esp
     xorl %eax,%eax
     leave
     ret
```



x86 Instructions

```
main:
     pushl %ebp
     movl %esp,%ebp
     subl $4,%esp
     movl $1,-4(%ebp)
     pushl $3
     pushl $2
     pushl $1
     call foo
     addl $12,%esp
     xorl %eax,%eax
     leave
     ret
```

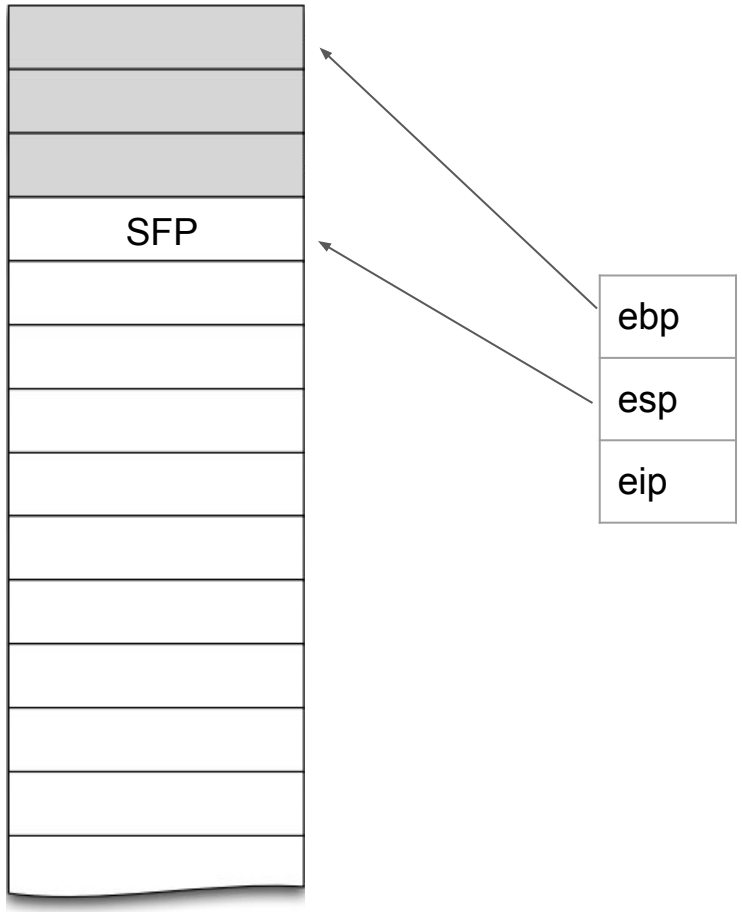
Push:
Subtract 4 from `ESP`
Place value of operand on the stack at the location pointed to by `ESP`



x86 Instructions

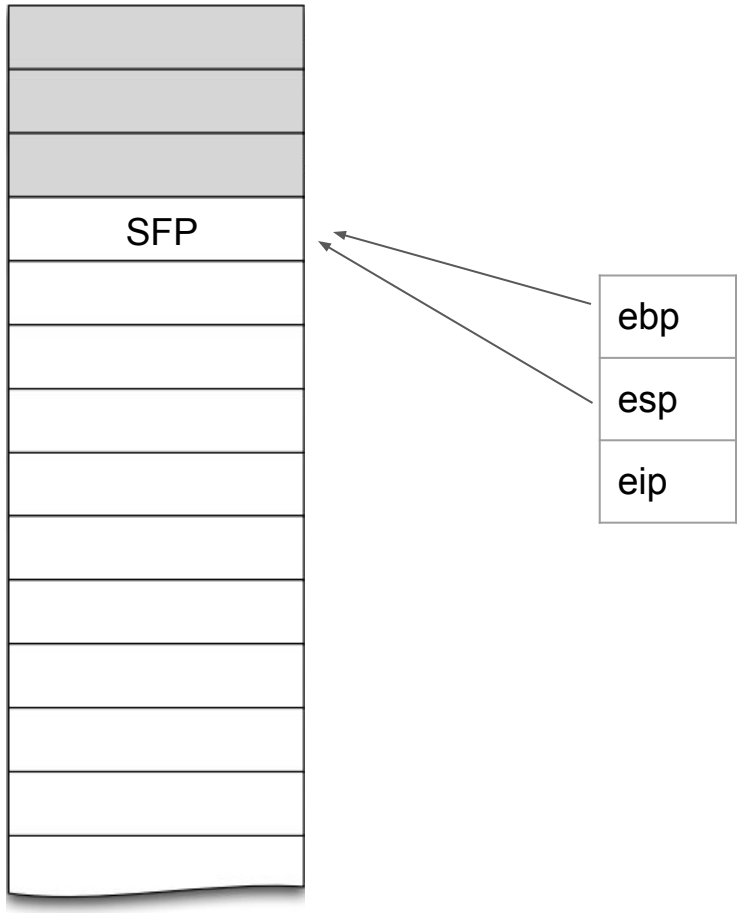
```
main:
     pushl %ebp
     movl %esp,%ebp
     subl $4,%esp
     movl $1,-4(%ebp)
     pushl $3
     pushl $2
     pushl $1
     call foo
     addl $12,%esp
     xorl %eax,%eax
     leave
     ret
```

Push:
Subtract 4 from `ESP`
Place value of operand on the stack at the location pointed to by `ESP`



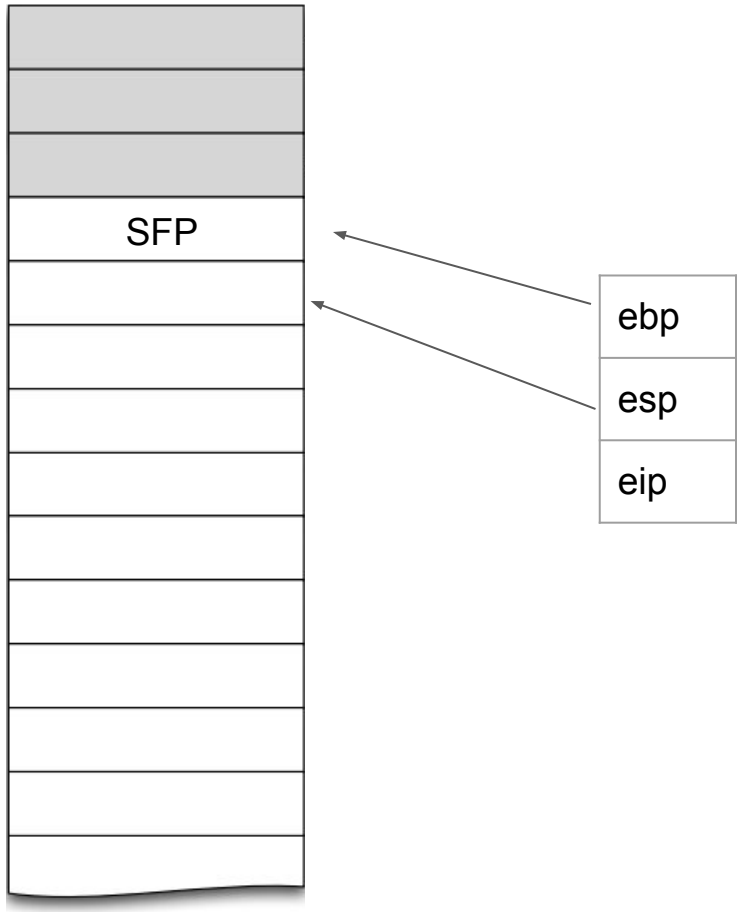
x86 Instructions

```
main:
     pushl %ebp
     movl %esp,%ebp
     subl $4,%esp
     movl $1,-4(%ebp)
     pushl $3
     pushl $2
     pushl $1
     call foo
     addl $12,%esp
     xorl %eax,%eax
     leave
     ret
```



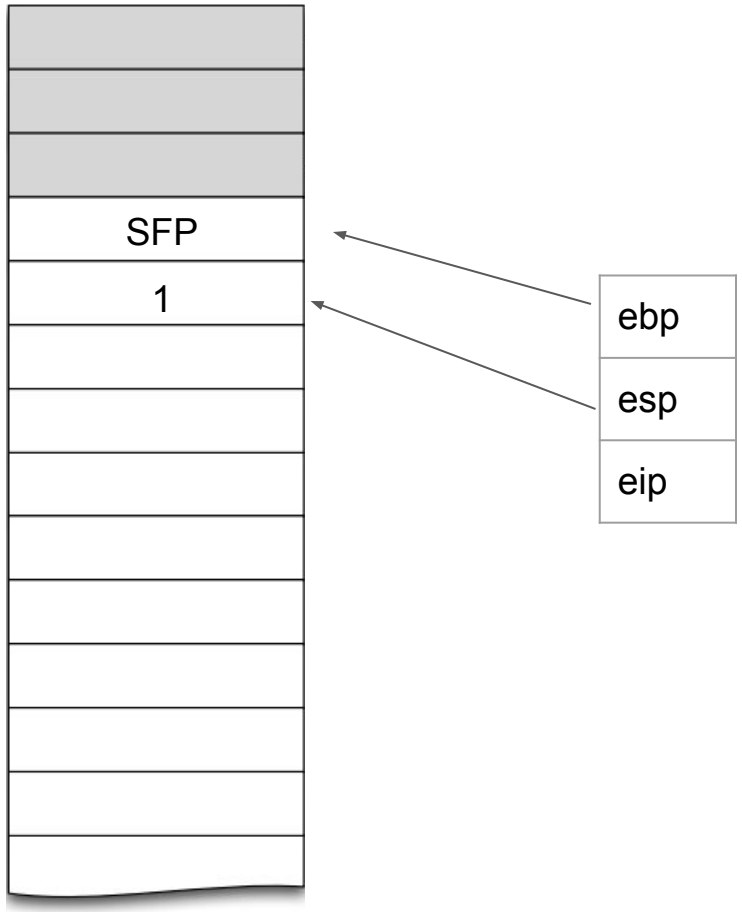
x86 Instructions

```
main:
     pushl %ebp
     movl %esp,%ebp
     subl $4,%esp
     movl $1,-4(%ebp)
     pushl $3
     pushl $2
     pushl $1
     call foo
     addl $12,%esp
     xorl %eax,%eax
     leave
     ret
```



x86 Instructions

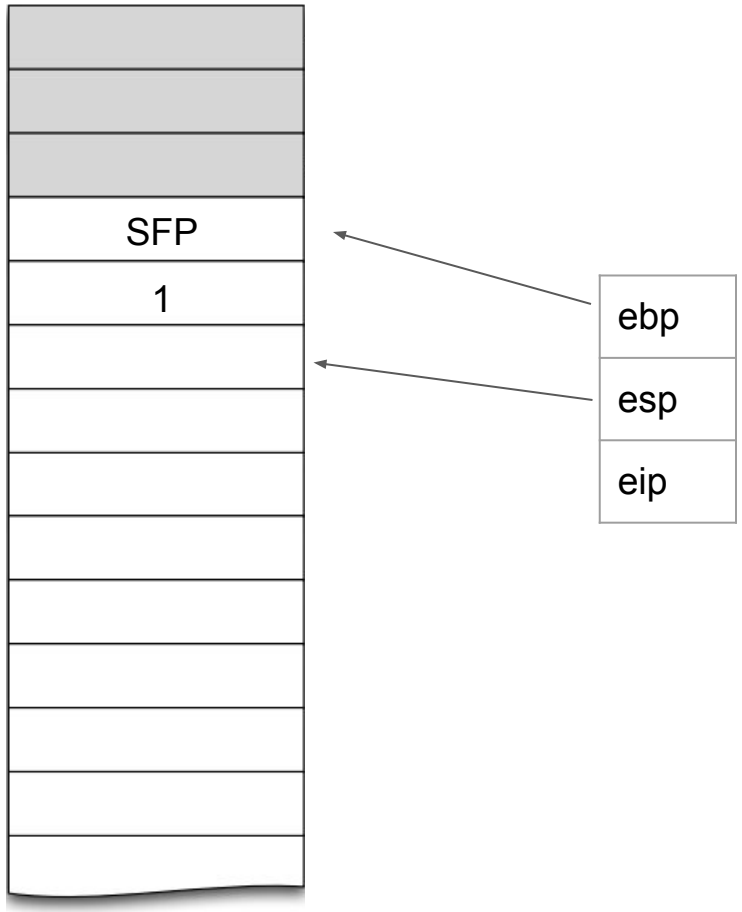
```
main:
     pushl %ebp
     movl %esp,%ebp
     subl $4,%esp
     movl $1,-4(%ebp)
     pushl $3
     pushl $2
     pushl $1
     call foo
     addl $12,%esp
     xorl %eax,%eax
     leave
     ret
```



x86 Instructions

```
main:
     pushl %ebp
     movl %esp,%ebp
     subl $4,%esp
     movl $1,-4(%ebp)
     pushl $3
     pushl $2
     pushl $1
     call foo
     addl $12,%esp
     xorl %eax,%eax
     leave
     ret
```

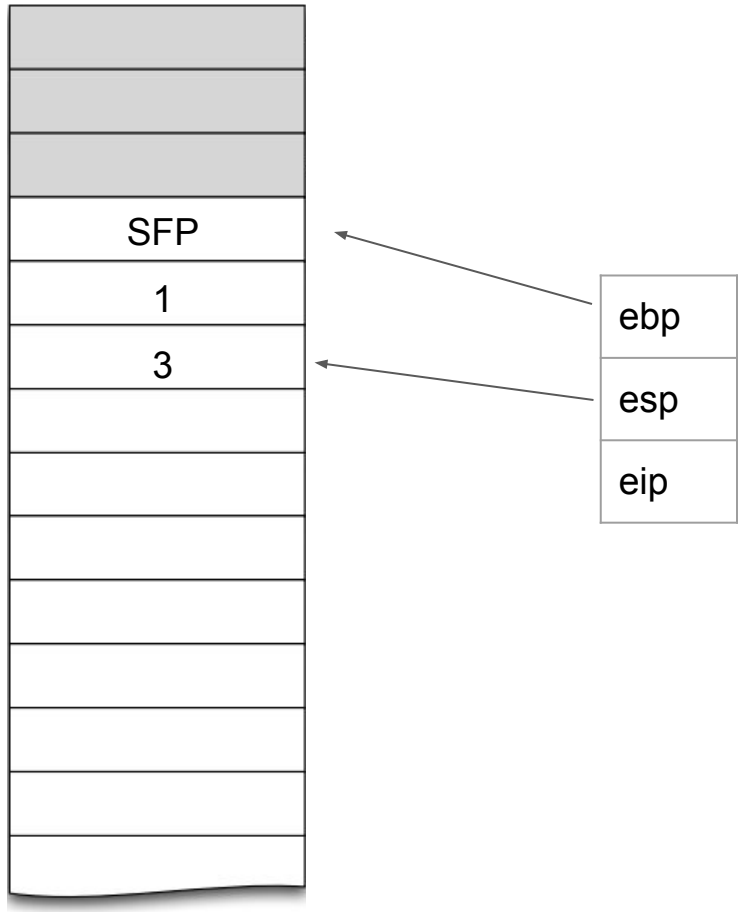
Push:
Subtract 4 from `ESP`
Place value of operand on the stack at the location pointed to by `ESP`



x86 Instructions

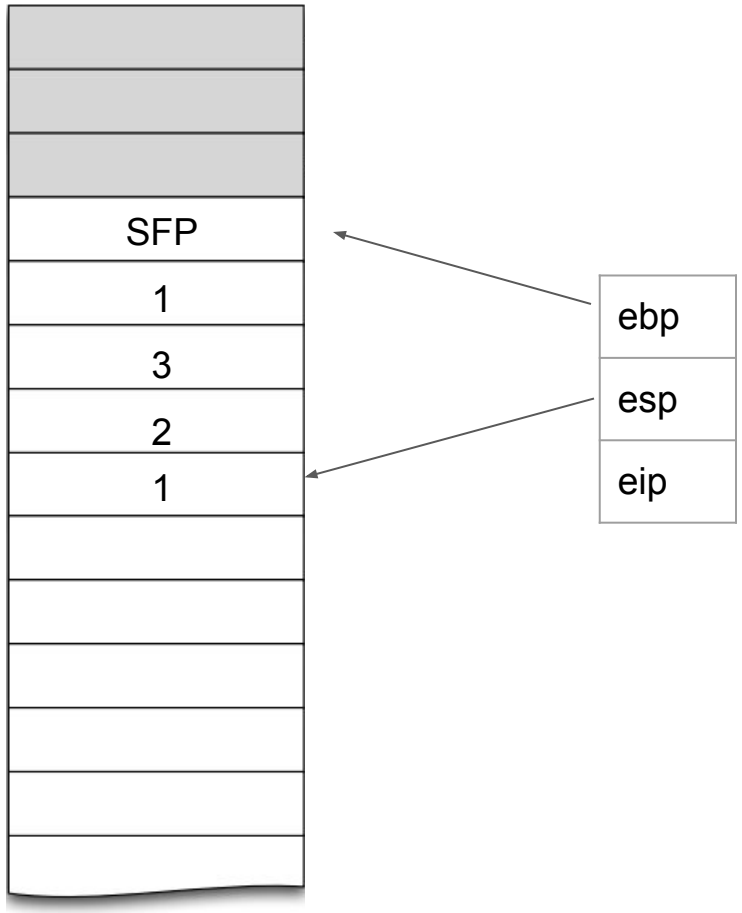
```
main:
     pushl %ebp
     movl %esp,%ebp
     subl $4,%esp
     movl $1,-4(%ebp)
     pushl $3
     pushl $2
     pushl $1
     call foo
     addl $12,%esp
     xorl %eax,%eax
     leave
     ret
```

Push:
Subtract 4 from `ESP`
Place value of operand on the stack at the location pointed to by `ESP`



x86 Instructions

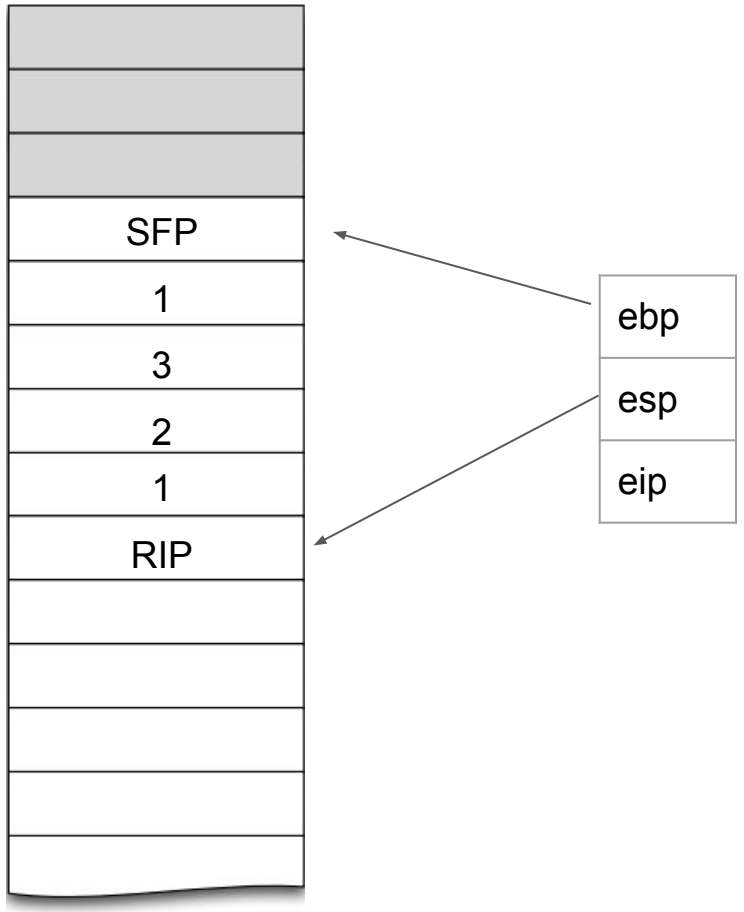
```
main:
     pushl %ebp
     movl %esp,%ebp
     subl $4,%esp
     movl $1,-4(%ebp)
     pushl $3
     pushl $2
     pushl $1
     call foo
     addl $12,%esp
     xorl %eax,%eax
     leave
     ret
```



x86 Instructions

```
main:
     pushl %ebp
     movl %esp,%ebp
     subl $4,%esp
     movl $1,-4(%ebp)
     pushl $3
     pushl $2
     pushl $1
     call foo
     addl $12,%esp
     xorl %eax,%eax
     leave
     ret
```

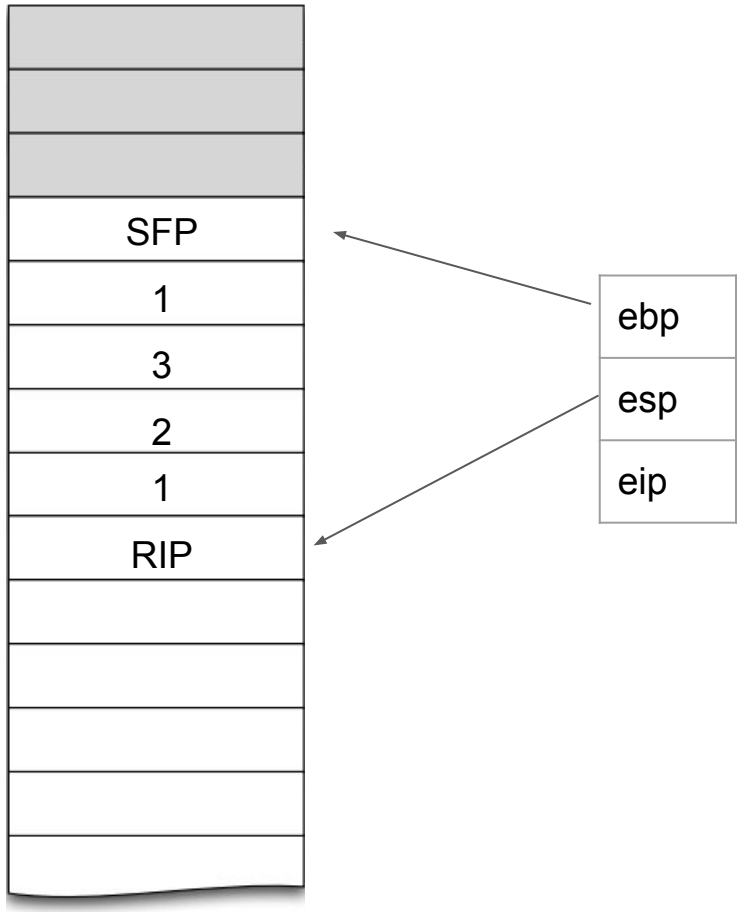
Call:
Pushes the current code location onto the stack.
Performs an unconditional jump to the code location indicated by the label operand



x86 Instructions

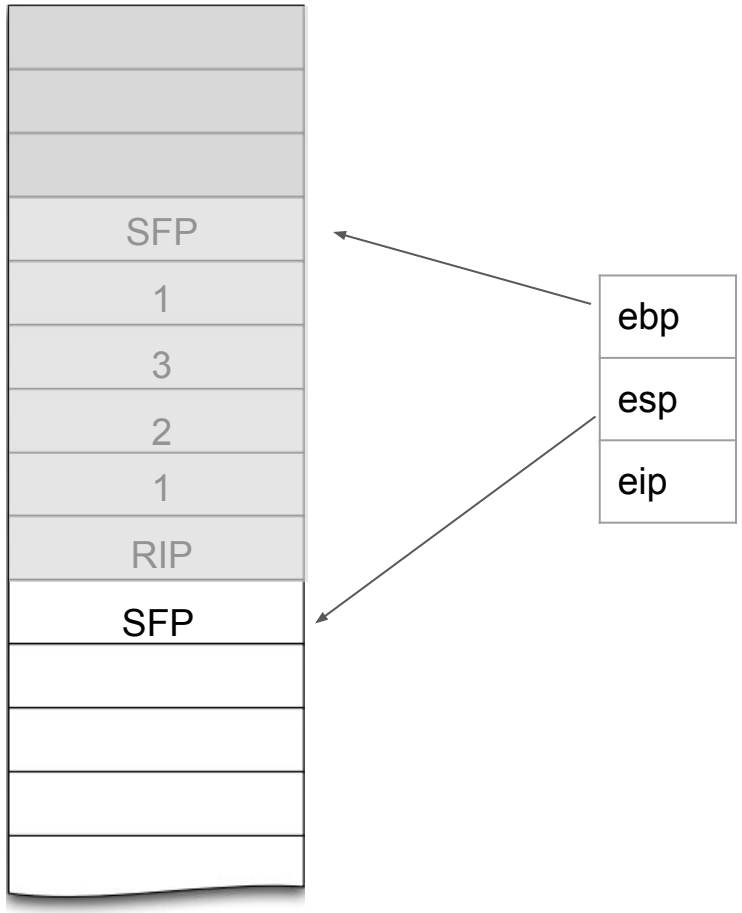
```
foo:
     pushl %ebp
     movl %esp,%ebp
     subl $12,%esp
     movl $65,-8(%ebp)
     movb $66,-12(%ebp)
     leave
     ret
```

Call:
Pushes the current code location onto the stack.
Performs an unconditional jump to the code location indicated by the label operand



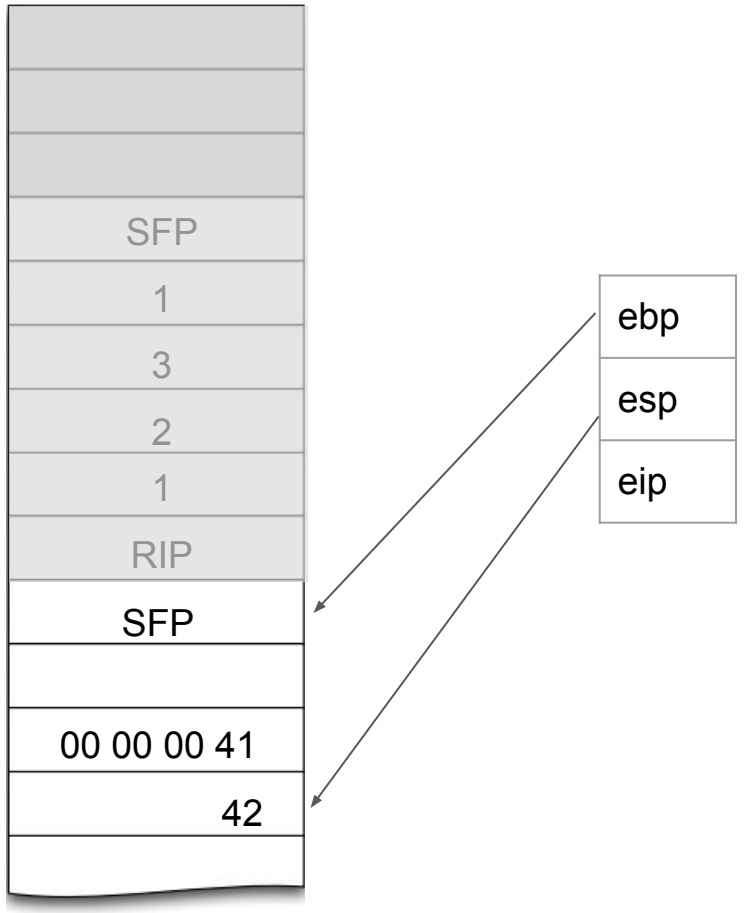
x86 Instructions

```
foo:
     pushl %ebp
     movl %esp,%ebp
     subl $12,%esp
     movl $65,-8(%ebp)
     movb $66,-12(%ebp)
     leave
     ret
```



x86 Instructions

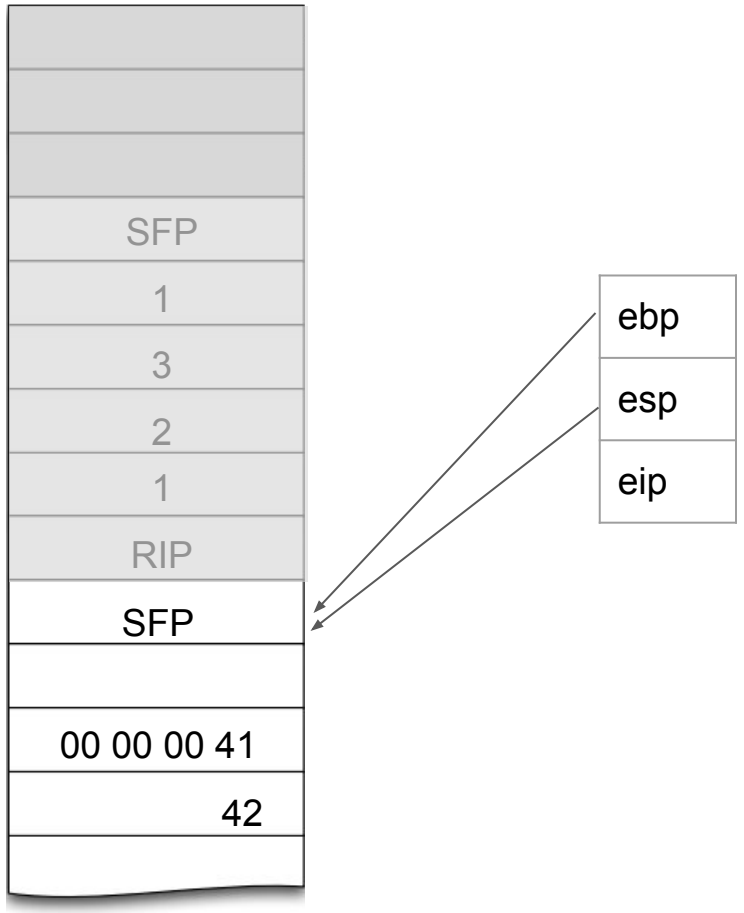
```
foo:
     pushl %ebp
     movl %esp,%ebp
     subl $12,%esp
     movl $65,-8(%ebp)
     movb $66,-12(%ebp)
     leave
     ret
```



x86 Instructions

```
foo:
     pushl %ebp
     movl %esp,%ebp
     subl $12,%esp
     movl $65,-8(%ebp)
     movb $66,-12(%ebp)
     leave
     ret
```

Leave:
Moves `ESP` to `EBP`
Pops `EBP`

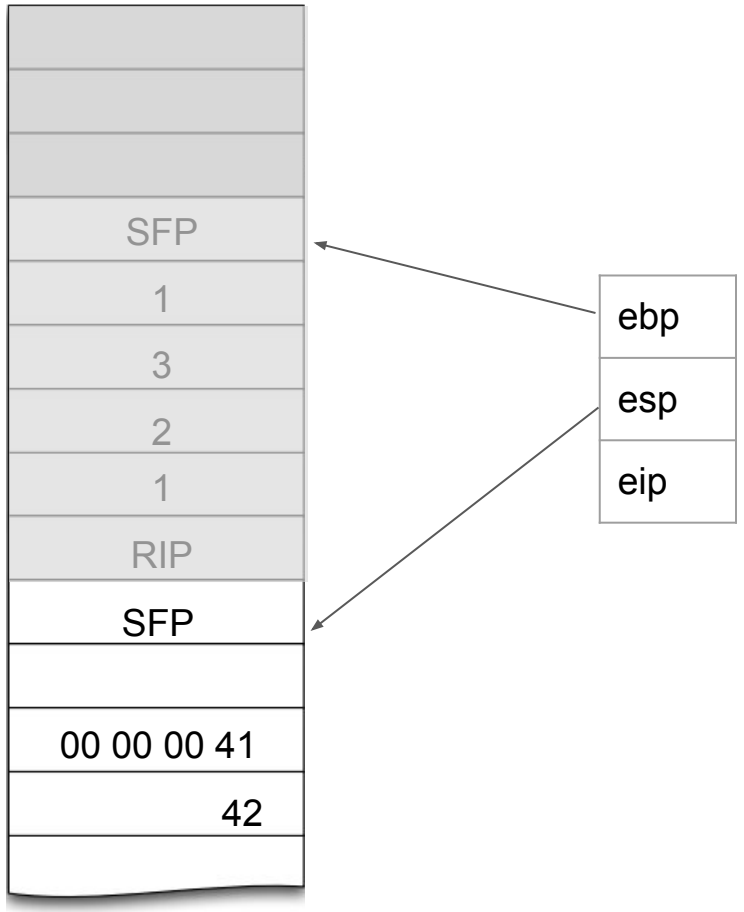


x86 Instructions

```
foo:
     pushl %ebp
     movl %esp,%ebp
     subl $12,%esp
     movl $65,-8(%ebp)
     movb $66,-12(%ebp)
     leave
     ret
```

Leave:
Moves `ESP` to `EBP`
Pops `EBP`

Pop:
Moves the 4 bytes located at memory location `[ESP]` into the specified register
Increments ESP by 4

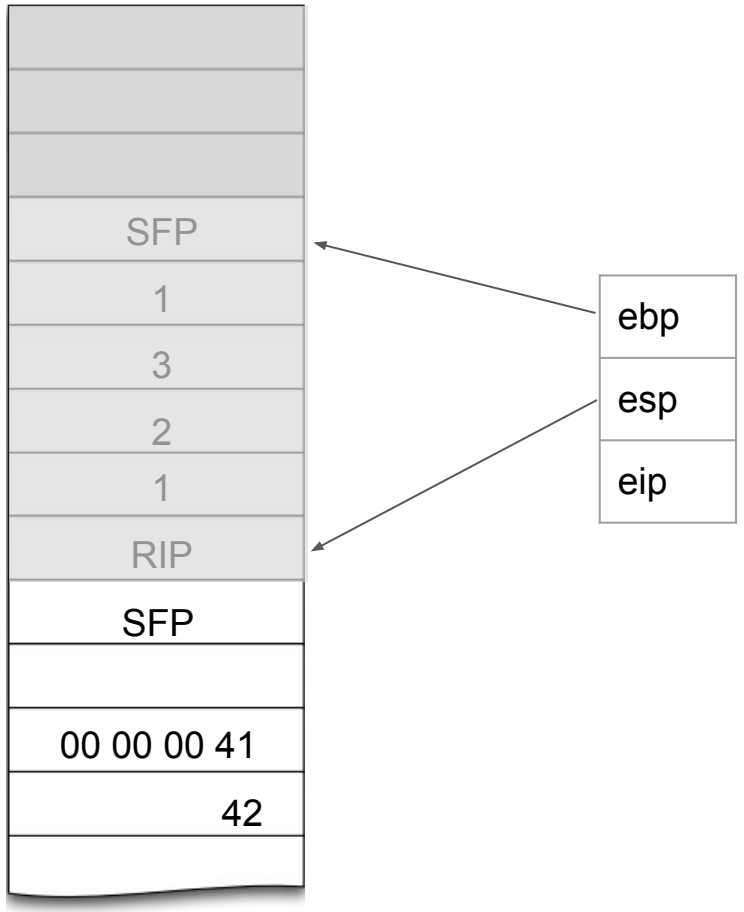


x86 Instructions

```
foo:
     pushl %ebp
     movl %esp,%ebp
     subl $12,%esp
     movl $65,-8(%ebp)
     movb $66,-12(%ebp)
     leave
     ret
```

Leave:
Moves `ESP` to `EBP`
Pops `EBP`

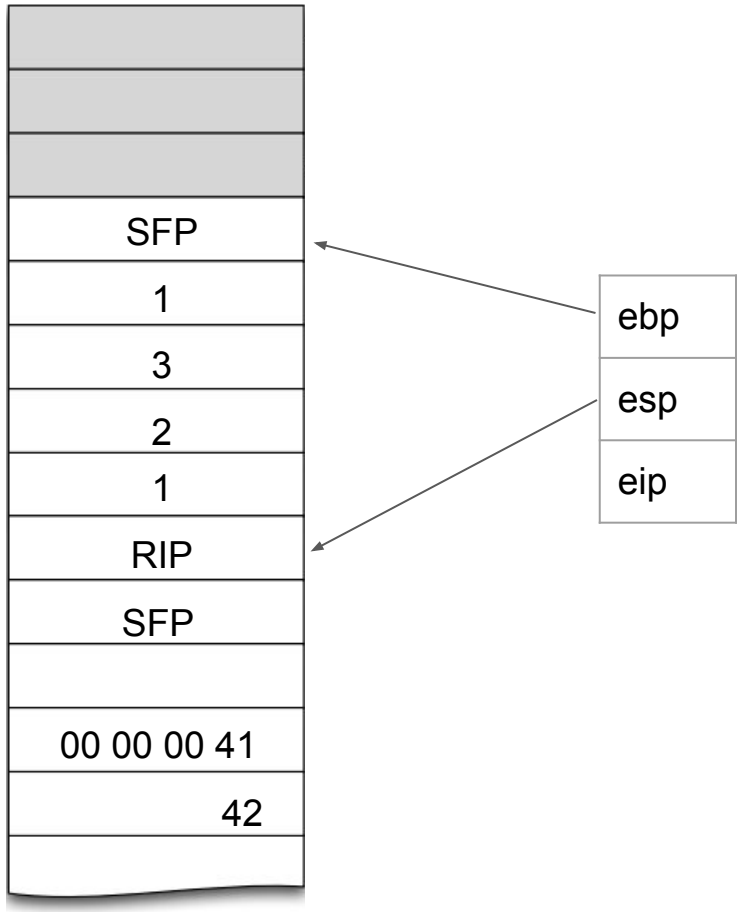
Pop:
Moves the 4 bytes located at memory location `[ESP]` into the specified register
Increments ESP by 4



x86 Instructions

```
foo:
     pushl %ebp
     movl %esp,%ebp
     subl $12,%esp
     movl $65,-8(%ebp)
     movb $66,-12(%ebp)
     leave
     ret
```

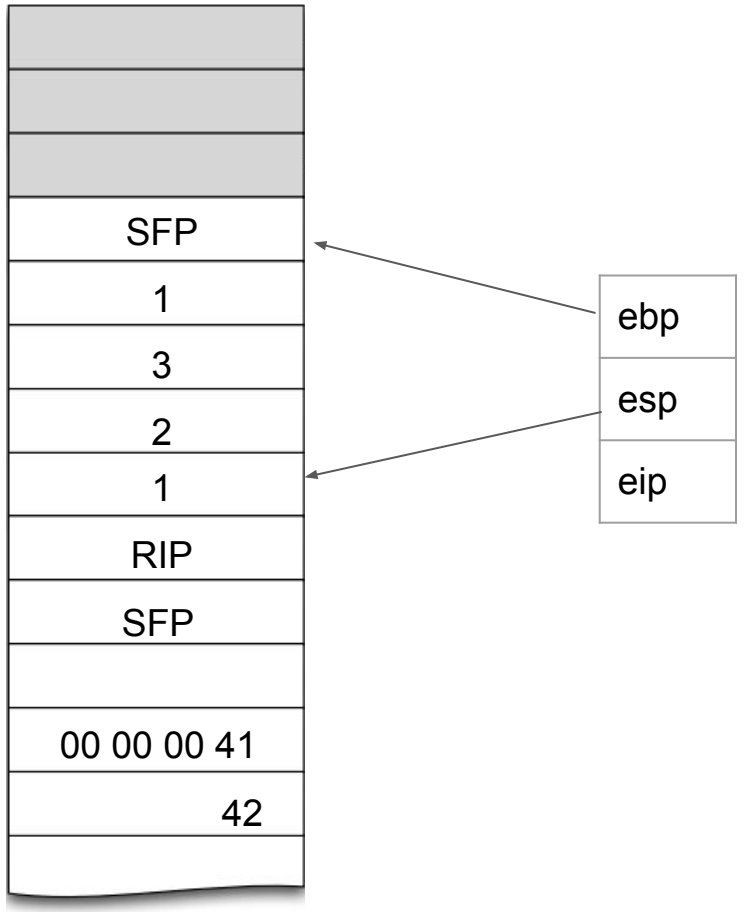
Ret:
Pops a code location off the stack.
Unconditional jump to the retrieved code location



x86 Instructions

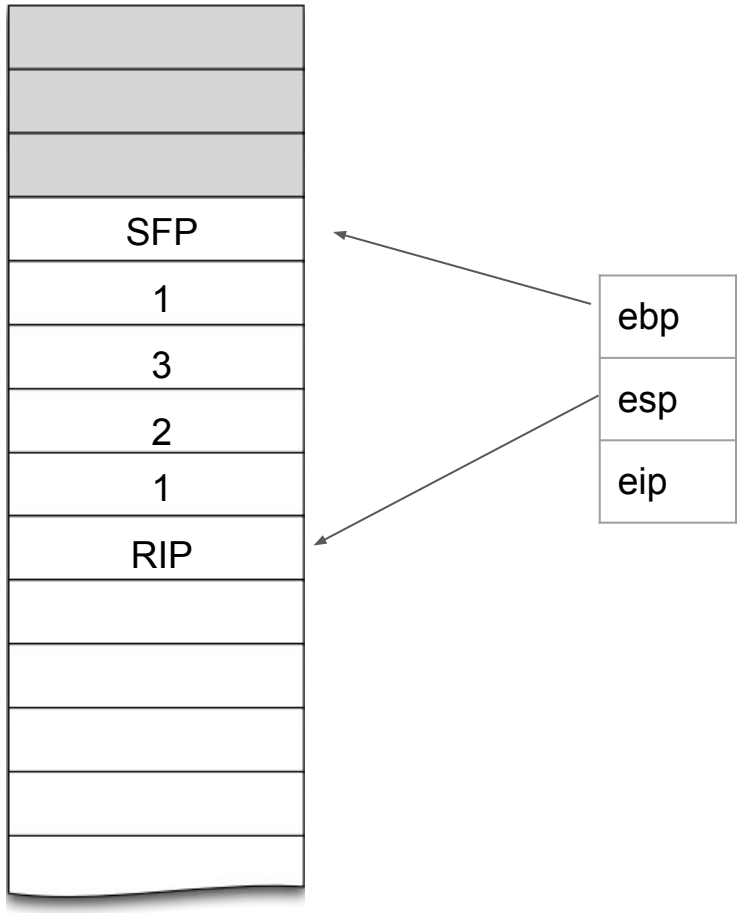
```
foo:
     pushl %ebp
     movl %esp,%ebp
     subl $12,%esp
     movl $65,-8(%ebp)
     movb $66,-12(%ebp)
     leave
     ret
```

Ret:
Pops a code location off the stack.
Unconditional jump to the retrieved code location



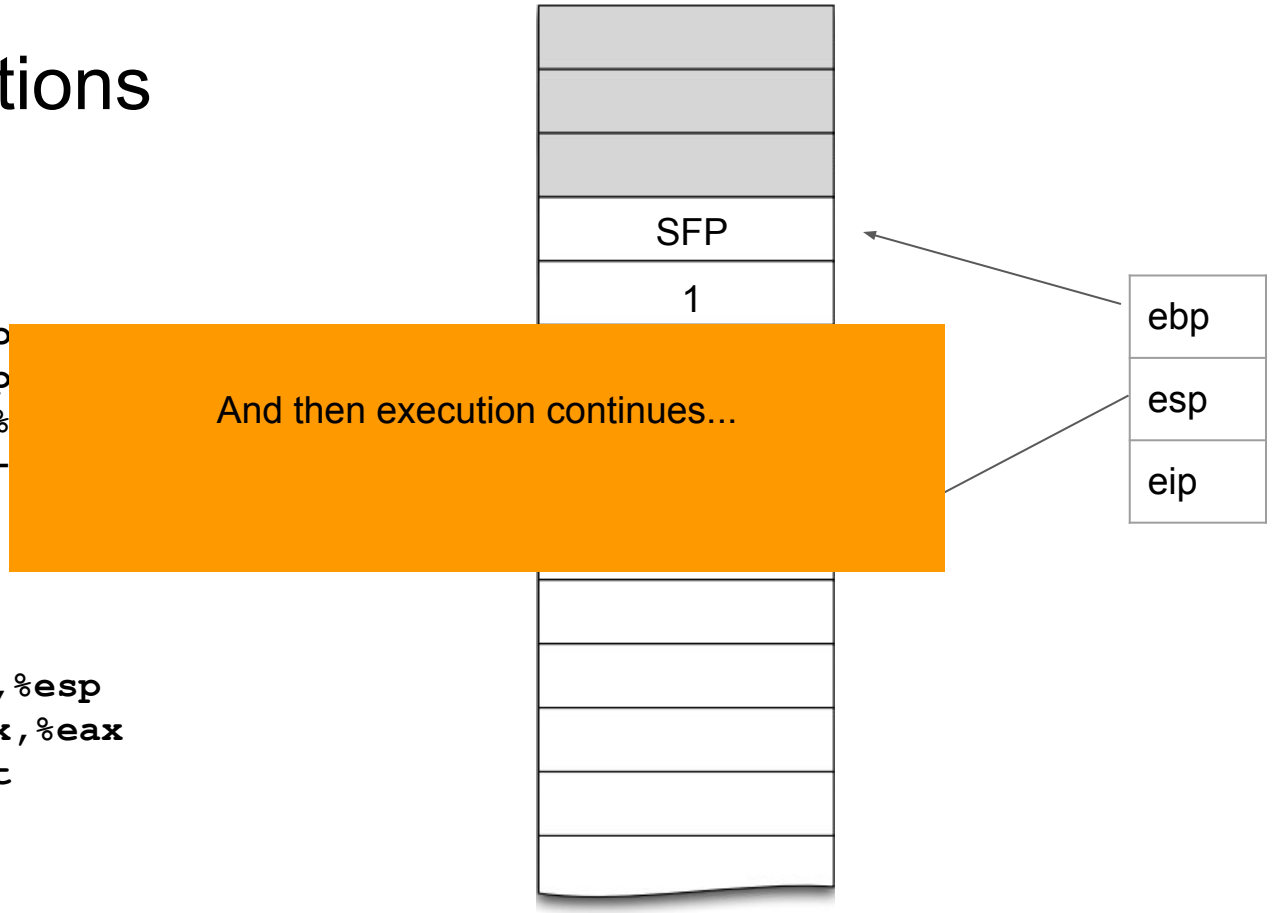
x86 Instructions

```
main:
     pushl %ebp
     movl %esp,%ebp
     subl $4,%esp
     movl $1,-4(%ebp)
     pushl $3
     pushl $2
     pushl $1
     call foo
     addl $12,%esp
     xorl %eax,%eax
     leave ret
```



x86 Instructions

```
main:
     pushl %eb
     movl %esp
     subl $4,%
     movl $1,-
     pushl $3
     pushl $2
     pushl $1
     call foo
     addl $12,%esp
     xorl %eax,%eax
     leave ret
```



FA16 - Midterm1 - Q6

```
typedef struct {
unsigned int salary;
unsigned int age;
} record_t;

# define MAX_DB_SIZE 64
enum field_t { SALARY = 0 , AGE = 1 };

bool updateRecord ( unsigned int id, field_t field, int data ){
    record_t database [ MAX_DB_SIZE ];
    if ( id <= MAX_DB_SIZE ) {
        if ( field == SALARY ) database [ id ].salary = data;
        if ( field == AGE ) database [ id ].age = data;
        return true;
    } else return false; //invalid argument
}
```

FA16 - Midterm1 - Q6

```
typedef struct {
unsigned int salary;
unsigned int age;
} record_t;

# define MAX_DB_SIZE 64
enum field_t { SALARY = 0 , AGE = 1 };

bool updateRecord ( unsigned int id, field_t field, int data ){
    record_t database [ MAX_DB_SIZE ];
    if ( id <= MAX_DB_SIZE ) {
        if ( field == SALARY ) database [ id ].salary = data;
        if ( field == AGE ) database [ id ].age = data;
        return true;
    } else return false; //invalid argument
}
```

Where is the vulnerability?

FA16 - Midterm1 - Q6

```
typedef struct {
unsigned int salary;
unsigned int age;
} record_t;

# define MAX_DB_SIZE 64
enum field_t { SALARY = 0 , AGE = 1 };

bool updateRecord ( unsigned int id, field_t field, int data ){
    record_t database [ MAX_DB_SIZE ];
    if ( id <= MAX_DB_SIZE ) {
        if ( field == SALARY ) database [ id ].salary = data;
        if ( field == AGE ) database [ id ].age = data;
        return true;
    } else return false; //invalid argument
}
```

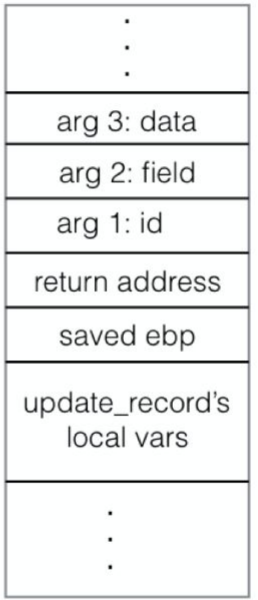
Where is the vulnerability?

FA16 - Midterm1 - Q6

```
typedef struct {
unsigned int salary;
unsigned int age;
} record_t;

# define MAX_DB_SIZE 64
enum field_t { SALARY = 0 , AGE = 1 };

bool updateRecord ( unsigned int id, field_t field, int data ){
    record_t database [ MAX_DB_SIZE ];
    if ( id <= MAX_DB_SIZE ) {
        if ( field == SALARY ) database [ id ].salary = data;
        if ( field == AGE ) database [ id ].age = data;
        return true;
    } else return false; //invalid argument
}
```

We have shellcode at address 0xdeadbeef. How can we invoke **`updateRecord`** to exploit the program and run our shellcode?

(32-bit x86 machine)

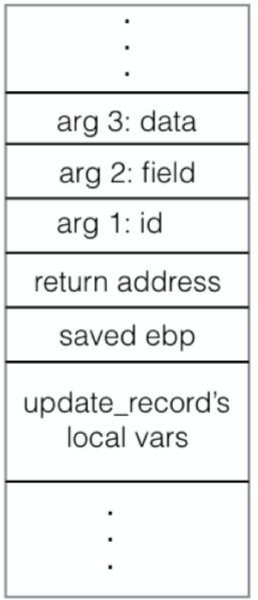


(b) Stack Frame Layout

FA16 - Midterm1 - Q6

```
typedef struct {
unsigned int salary;
unsigned int age;
} record_t;

# define MAX_DB_SIZE 64
enum field_t { SALARY = 0 , AGE = 1 };

bool updateRecord ( unsigned int id, field_t field, int data ){
    record_t database [ MAX_DB_SIZE ];
    if ( id <= MAX_DB_SIZE ) {
        if ( field == SALARY ) database [ id ].salary = data;
        if ( field == AGE ) database [ id ].age = data;
        return true;
    } else return false; //invalid argument
}
```

```
updateRecord(64, AGE, 0xdeadbeef)
```

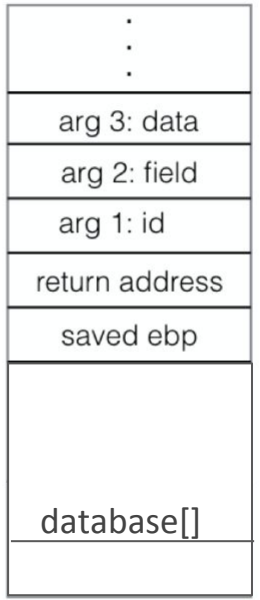


(b) Stack Frame Layout

FA16 - Midterm1 - Q6

```
typedef struct {
unsigned int salary;
unsigned int age;
} record_t;

# define MAX_DB_SIZE 64
enum field_t { SALARY = 0 , AGE = 1 };

bool updateRecord ( unsigned int id, field_t field, int data ){
    record_t database [ MAX_DB_SIZE ];
    if ( id <= MAX_DB_SIZE ) {
        if ( field == SALARY ) database [ id ].salary = data;
        if ( field == AGE ) database [ id ].age = data;
        return true;
    } else return false; //invalid argument
}
```

```
updateRecord(64, AGE, 0xdeadbeef)
```

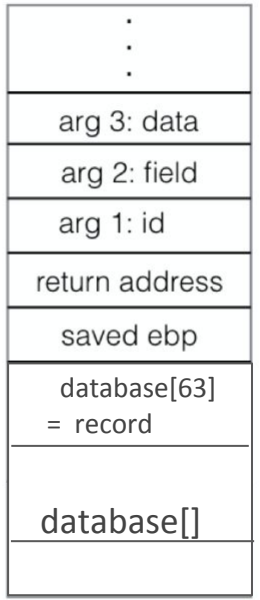


(b) Stack Frame Layout

FA16 - Midterm1 - Q6

```
typedef struct {
unsigned int salary;
unsigned int age;
} record_t;

# define MAX_DB_SIZE 64
enum field_t { SALARY = 0 , AGE = 1 };

bool updateRecord ( unsigned int id, field_t field, int data ){
    record_t database [ MAX_DB_SIZE ];
    if ( id <= MAX_DB_SIZE ) {
        if ( field == SALARY ) database [ id ].salary = data;
        if ( field == AGE ) database [ id ].age = data;
        return true;
    } else return false; //invalid argument
}
```

```
updateRecord(64, AGE, 0xdeadbeef)
```

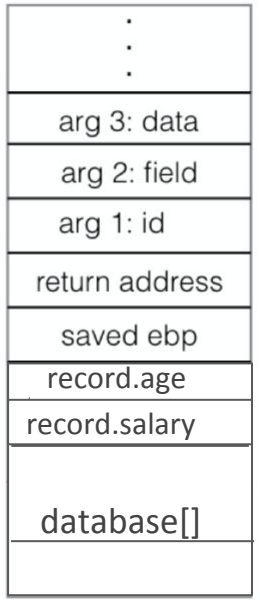


(b) Stack Frame Layout

FA16 - Midterm1 - Q6

```
typedef struct {
unsigned int salary;
unsigned int age;
} record_t;

# define MAX_DB_SIZE 64
enum field_t { SALARY = 0 , AGE = 1 };

bool updateRecord ( unsigned int id, field_t field, int data ){
    record_t database [ MAX_DB_SIZE ];
    if ( id <= MAX_DB_SIZE ) {
        if ( field == SALARY ) database [ id ].salary = data;
        if ( field == AGE ) database [ id ].age = data;
        return true;
    } else return false; //invalid argument
}
```

```
updateRecord(64, AGE, 0xdeadbeef)
```

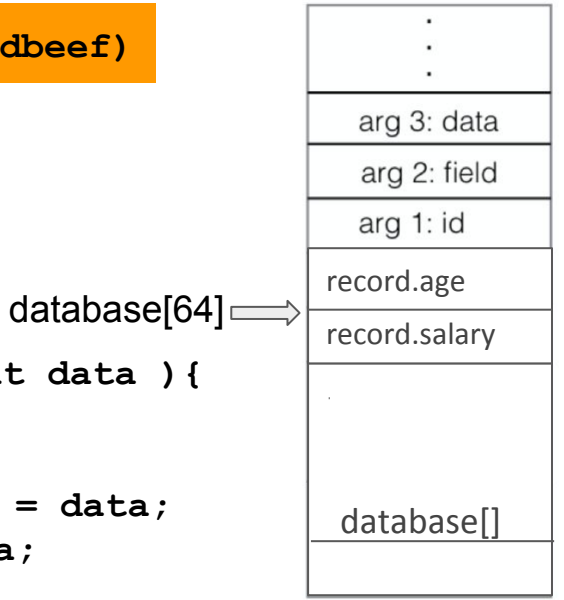


(b) Stack Frame Layout

FA16 - Midterm1 - Q6

```
typedef struct {
unsigned int salary;
unsigned int age;
} record_t;

# define MAX_DB_SIZE 64
enum field_t { SALARY = 0 , AGE = 1 };

bool updateRecord ( unsigned int id, field_t field, int data ){
    record_t database [ MAX_DB_SIZE ];
    if ( id <= MAX_DB_SIZE ) {
        if ( field == SALARY ) database [ id ].salary = data;
        if ( field == AGE ) database [ id ].age = data;
        return true;
    } else return false; //invalid argument
}
```

```
updateRecord(64, AGE, 0xdeadbeef)
```

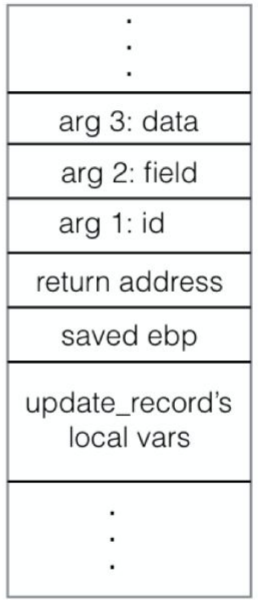


(b) Stack Frame Layout

FA16 - Midterm1 - Q6

```
typedef struct {
unsigned int salary;
unsigned int age;
} record_t;

# define MAX_DB_SIZE 64
enum field_t { SALARY = 0 , AGE = 1 };

bool updateRecord ( unsigned int id, field_t field, int data ){
    record_t database [ MAX_DB_SIZE ];
    if ( id <= MAX_DB_SIZE ) {
        if ( field == SALARY ) database [ id ].salary = data;
        if ( field == AGE ) database [ id ].age = data;
        return true;
    } else return false; //invalid argument
}
```

Your colleague tells you to enable stack canaries to avoid an attack of this form. Is the advice sound? Explain why or why not?

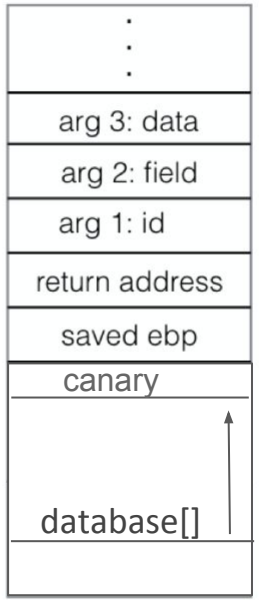


(b) Stack Frame Layout

FA16 - Midterm1 - Q6

```
typedef struct {
unsigned int salary;
unsigned int age;
} record_t;

# define MAX_DB_SIZE 64
enum field_t { SALARY = 0 , AGE = 1 };

bool updateRecord ( unsigned int id, field_t field, int data ){
    record_t database [ MAX_DB_SIZE ];
    if ( id <= MAX_DB_SIZE ) {
        if ( field == SALARY ) database [ id ].salary = data;
        if ( field == AGE ) database [ id ].age = data;
        return true;
    } else return false; //invalid argument
}
```

Yes! A stack canary would not allow us to overwrite the return instruction pointer.



(b) Stack Frame Layout

Defenses

- Stack Canaries - add a random value between the return address and ebp which is checked in the function epilogue during execution
- Address Space Layout Randomization - the OS randomizes the starting base of each section (stack, heap, etc)
- Non-executable Stack - mark the stack as non-executable

- Bounds checking - check the length of the buffer and ensure it can fit the user input

Other Attacks

- Return to Libc
- Return Oriented Programming

Return to Libc

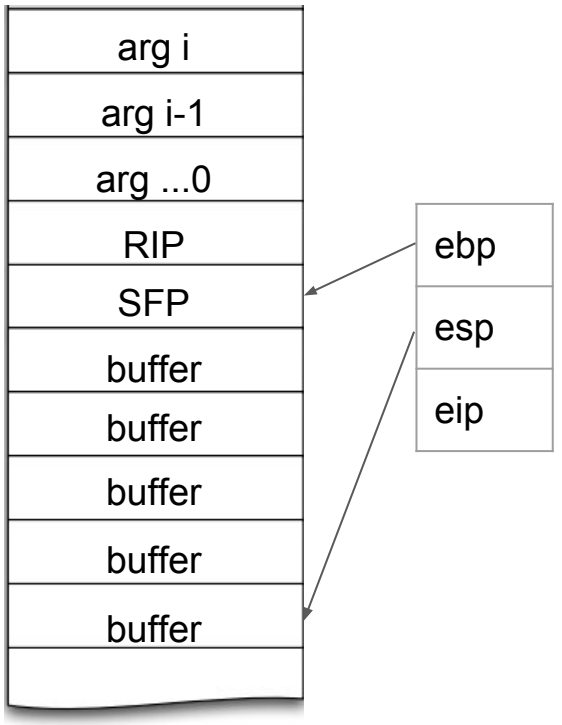
Main Idea: **`ret`** into the libc function **`system()`** (or a different libc function)

Remember:
`call` - pushes the current code location onto the stack and then performs an unconditional jump to the code location indicated by the label operand
`ret` - pops a code location off the stack. It then performs an unconditional jump to the retrieved code location.

Return to Libc

Main Idea: **`ret`** into the libc function **`system()`**

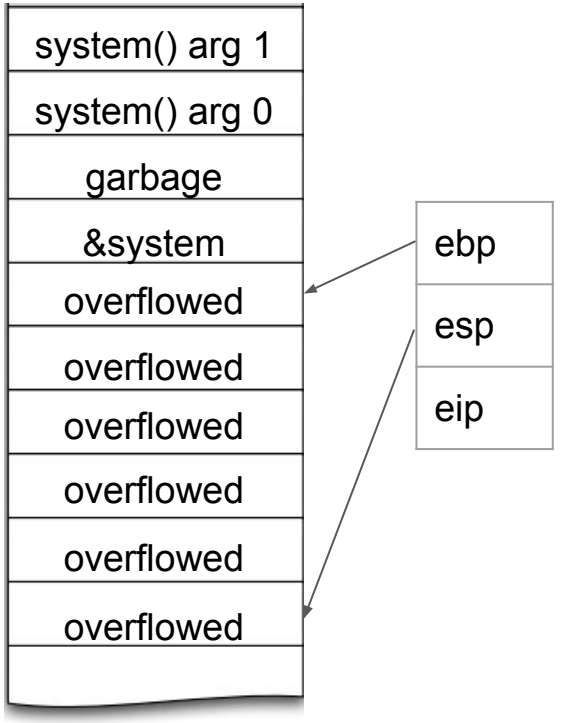
Remember:
`call` - pushes the current code location onto the stack and then performs an unconditional jump to the code location indicated by the label operand
`ret` - pops a code location off the stack. It then performs an unconditional jump to the retrieved code location.



Return to Libc

Main Idea: **`ret`** into the libc function **`system()`**

Remember:
`call` - pushes the current code location onto the stack and then performs an unconditional jump to the code location indicated by the label operand
`ret` - pops a code location off the stack. It then performs an unconditional jump to the retrieved code location.

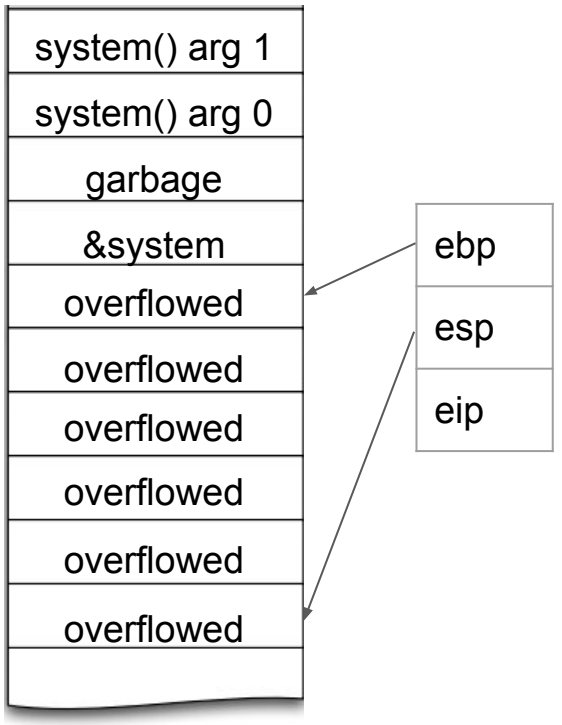


Return to Libc

Main Idea: `ret` into the libc function `system()`

Function epilogue:

```
move %esp %ebp
pop %ebp
ret
```

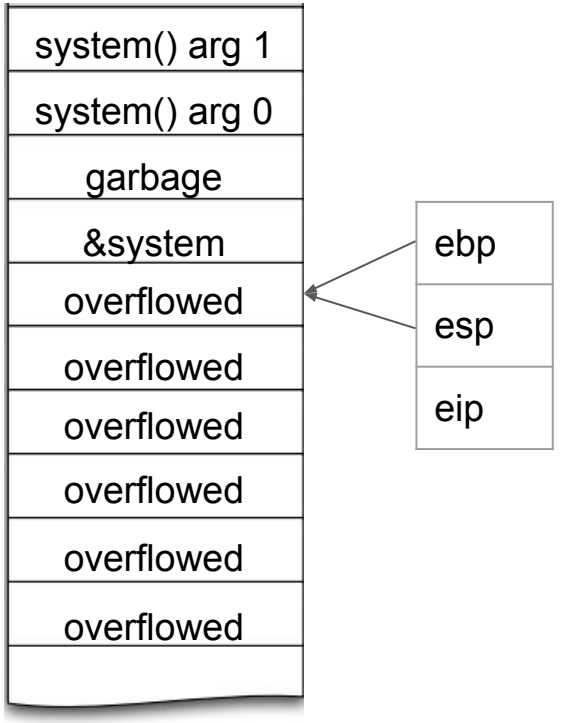


Return to Libc

Main Idea: `ret` into the libc function `system()`

Function epilogue:

```
move %esp %ebp
pop %ebp
ret
```

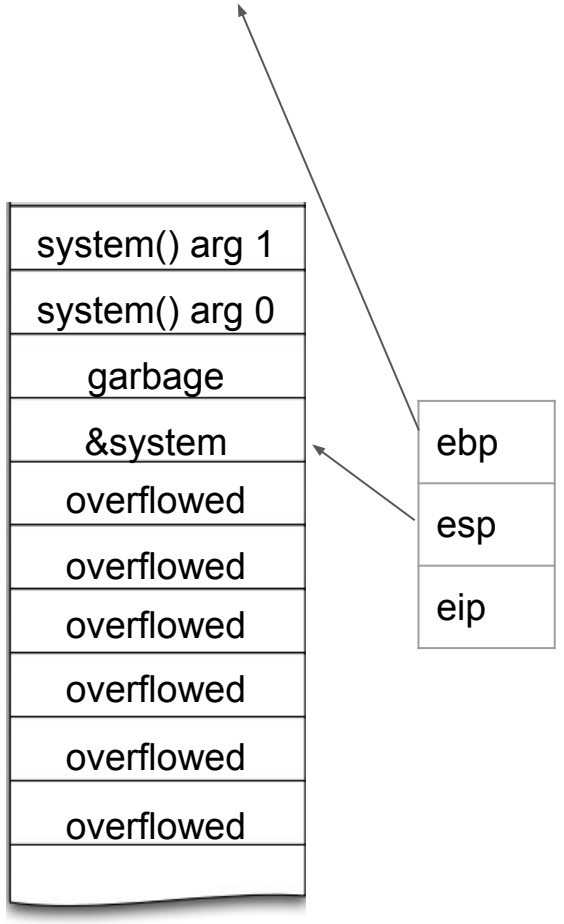


Return to Libc

Main Idea: `ret` into the libc function `system()`

Function epilogue:

```
move %esp %ebp
pop %ebp
ret
```



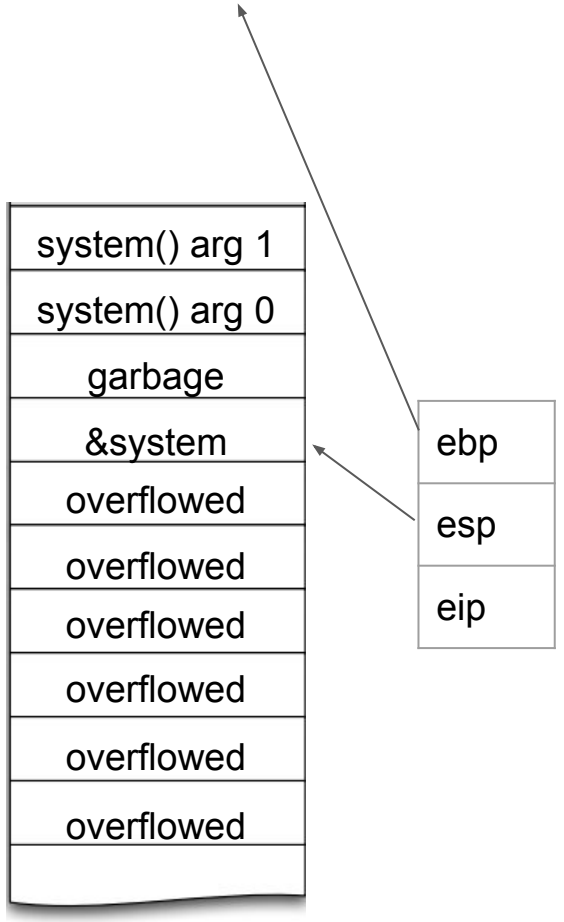
Return to Libc

Main Idea: `ret` into the libc function `system()`

Function epilogue:

```
move %esp %ebp
pop %ebp
ret
```

Ret:
Pops a code location off the stack.
Unconditional jump to the retrieved code location



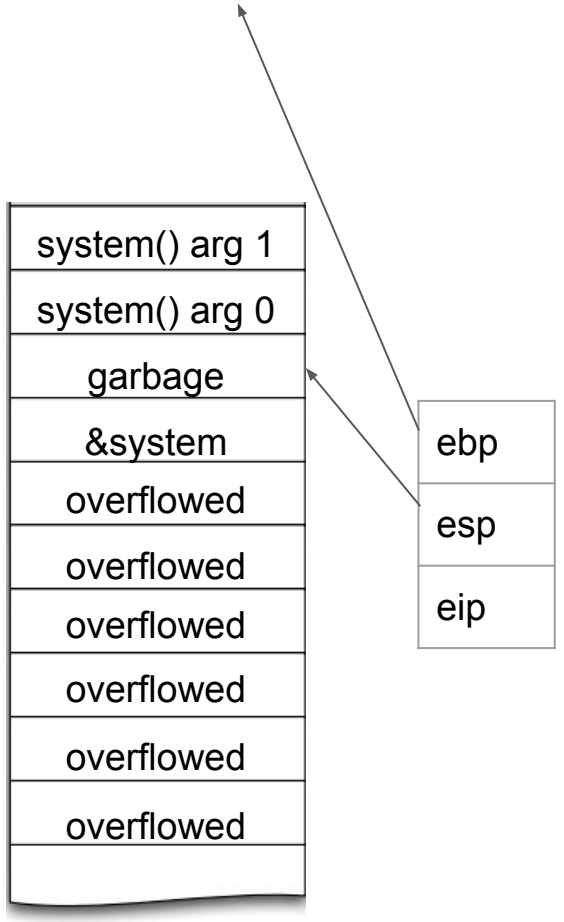
Return to Libc

Main Idea: `ret` into the libc function `system()`

Function epilogue:

```
move %esp %ebp
pop %ebp
ret
```

Ret:
Pops a code location off the stack.
Unconditional jump to the retrieved code location

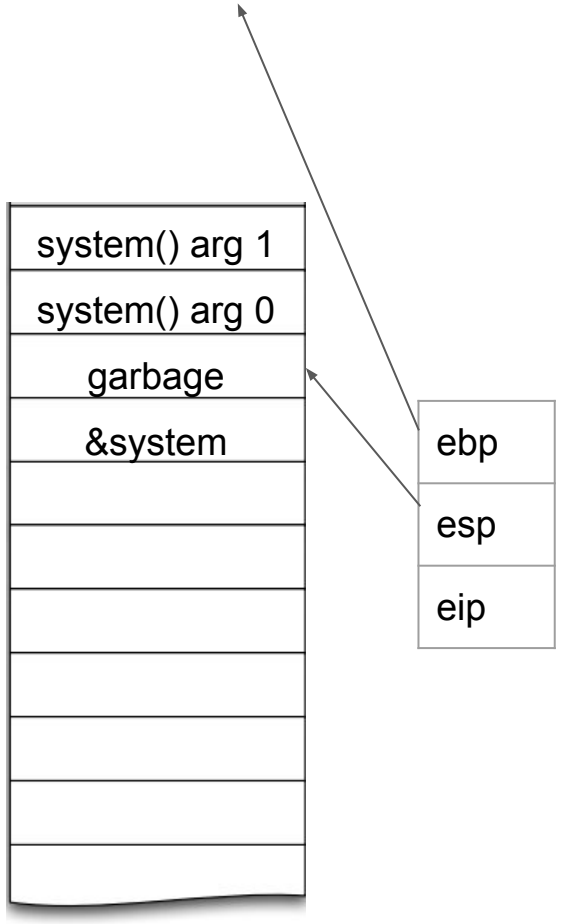


Return to Libc

Main Idea: `ret` into the libc function `system()`

```
system:
     push %ebp
     mov %esp,%ebp
     // continue func
```

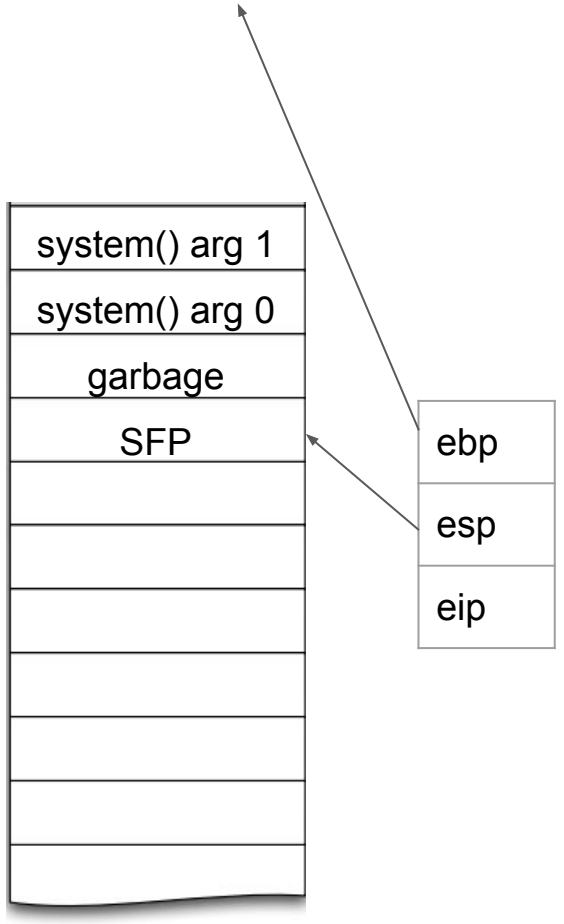
Ret:
Pops a code location off the stack.
Unconditional jump to the retrieved code location



Return to Libc

Main Idea: `ret` into the libc function `system()`

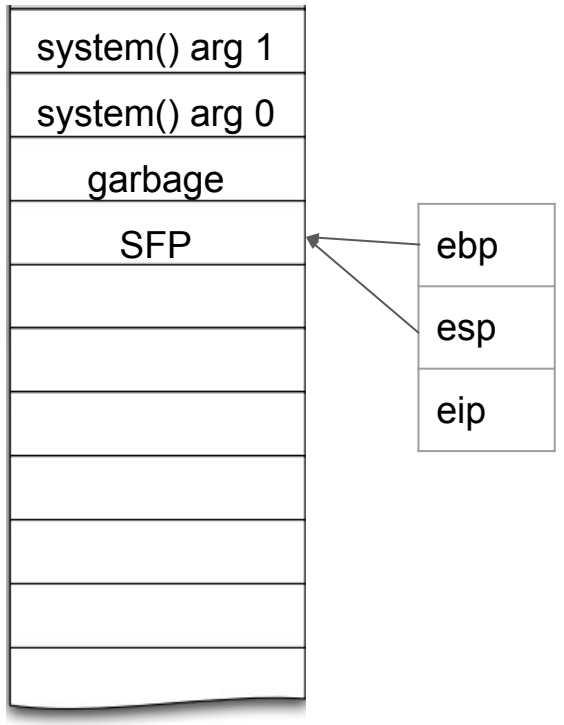
```
system:
    push %ebp
    mov %esp,%ebp
    // continue func
```



Return to Libc

Main Idea: `ret` into the libc function `system()`

```
system:
     push %ebp
     mov %esp,%ebp
     // continue func
```

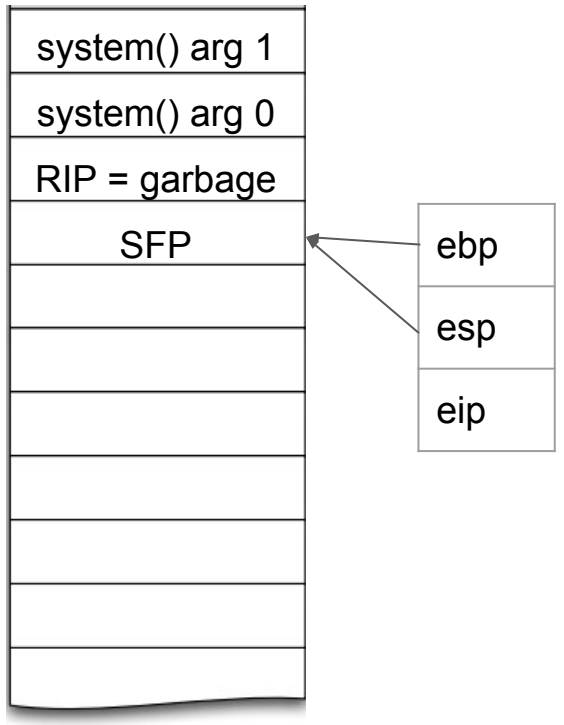


Return to Libc

Main Idea: `ret` into the libc function `system()`

```
system:
     push %ebp
     mov %esp,%ebp
     // continue func
```

It looks like this function was called normally!



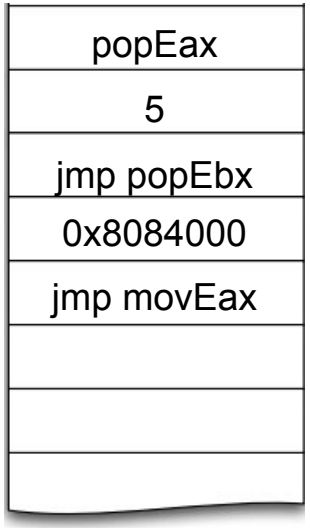
Return Oriented Programming

Main Idea: return to chains of **gadgets**, which are instruction sequences ending in `ret`. This allows us to perform arbitrary code execution **without actually introducing new code.**

Return Oriented Programming

Main Idea: return to chains of **gadgets**, which are instruction sequences ending in `ret`. This allows us to perform arbitrary code execution **without actually introducing new code.**

```
popEax:
     pop %eax
     ret

popEbx:
     pop %ebx
     ret

movEax:
     mov %eax, (%ebx)
     ret
```

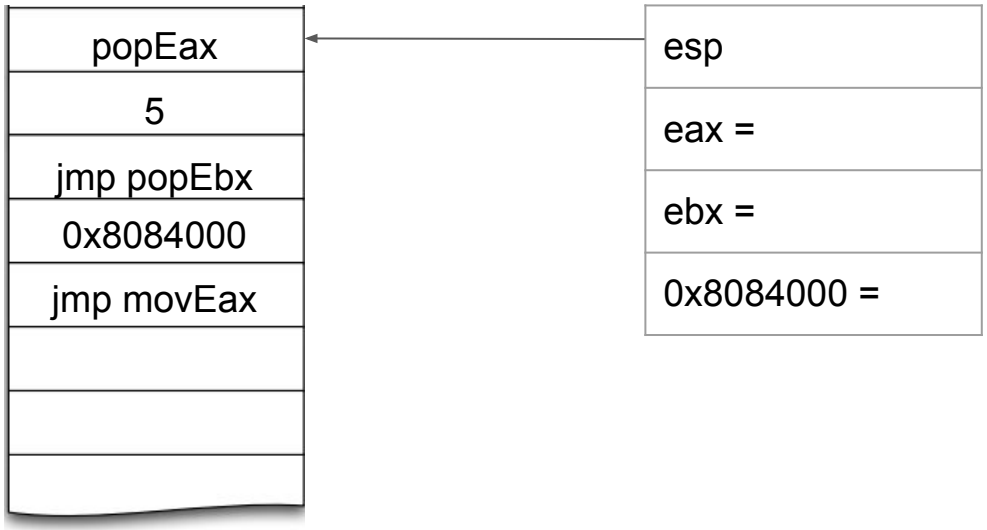


Our goal is to set 0x8084000 equal to 5

Return Oriented Programming

Main Idea: return to chains of **gadgets**, which are instruction sequences ending in `ret`. This allows us to perform arbitrary code execution **without actually introducing new code.**

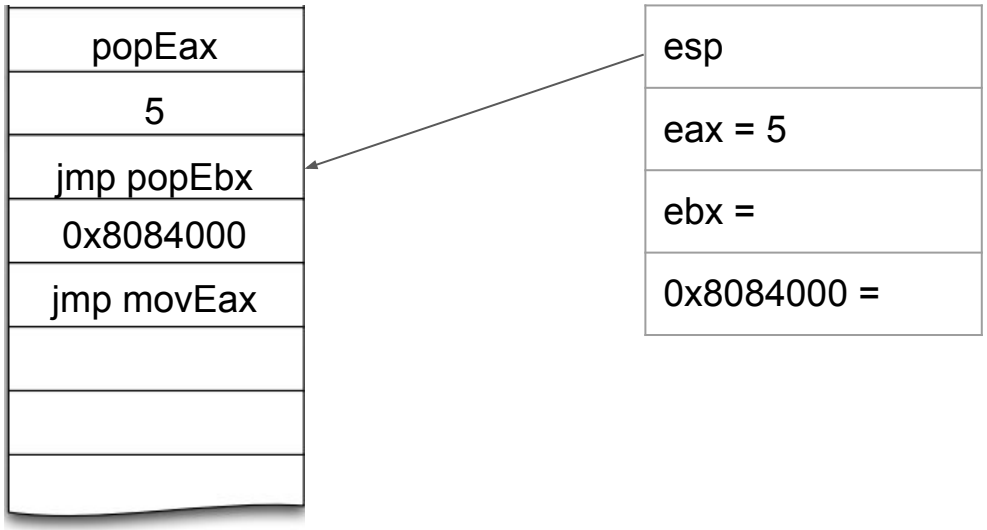
```
popEax:
     pop %eax
     ret

popEbx:
     pop %ebx
     ret

movEax:
     mov %eax, (%ebx)
     ret
```



Return Oriented Programming

Main Idea: return to chains of **gadgets**, which are instruction sequences ending in `ret`. This allows us to perform arbitrary code execution **without actually introducing new code.**

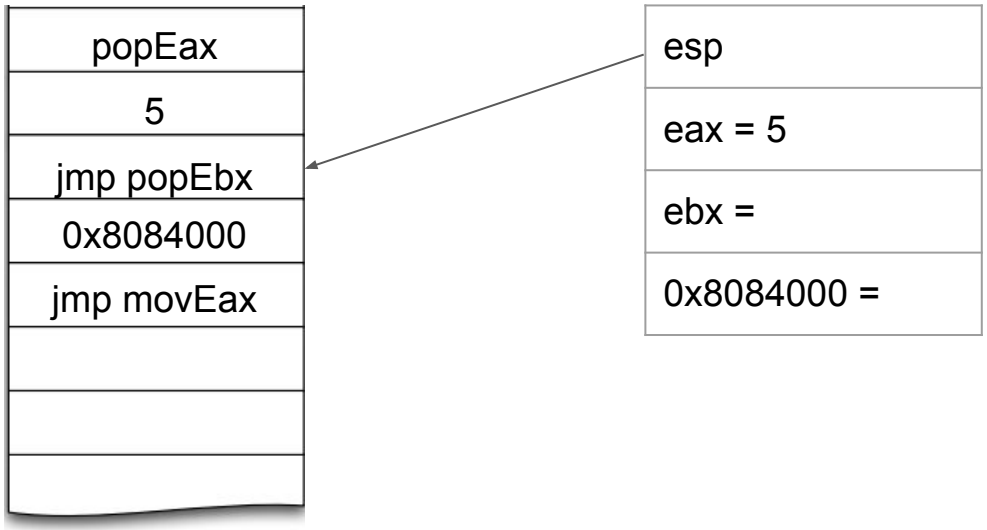
```
popEax:
     pop %eax
     ret

popEbx:
     pop %ebx
     ret

movEax:
     mov %eax, (%ebx)
     ret
```



Return Oriented Programming

Main Idea: return to chains of **gadgets**, which are instruction sequences ending in `ret`. This allows us to perform arbitrary code execution **without actually introducing new code.**

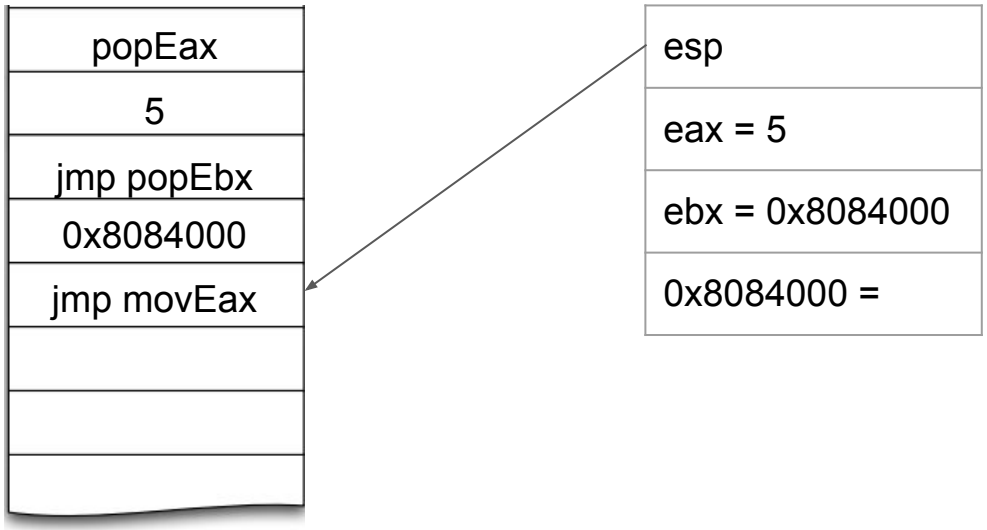
```
popEax:
     pop %eax
     ret

popEbx:
     pop %ebx
     ret

movEax:
     mov %eax, (%ebx)
     ret
```



Return Oriented Programming

Main Idea: return to chains of **gadgets**, which are instruction sequences ending in `ret`. This allows us to perform arbitrary code execution **without actually introducing new code.**

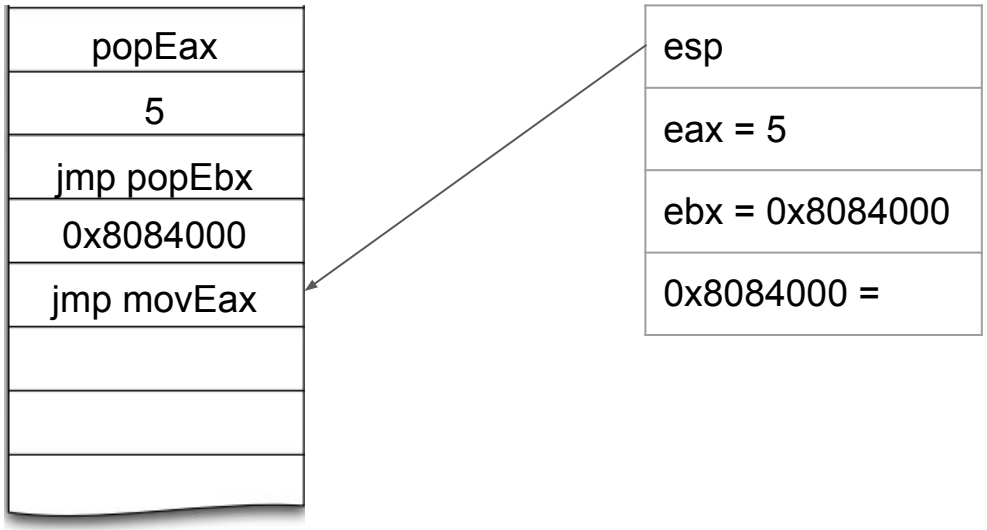
```
popEax:
     pop %eax
     ret

popEbx:
     pop %ebx
     ret

movEax:
     mov %eax, (%ebx)
     ret
```



Return Oriented Programming

Main Idea: return to chains of **gadgets**, which are instruction sequences ending in `ret`. This allows us to perform arbitrary code execution **without actually introducing new code.**

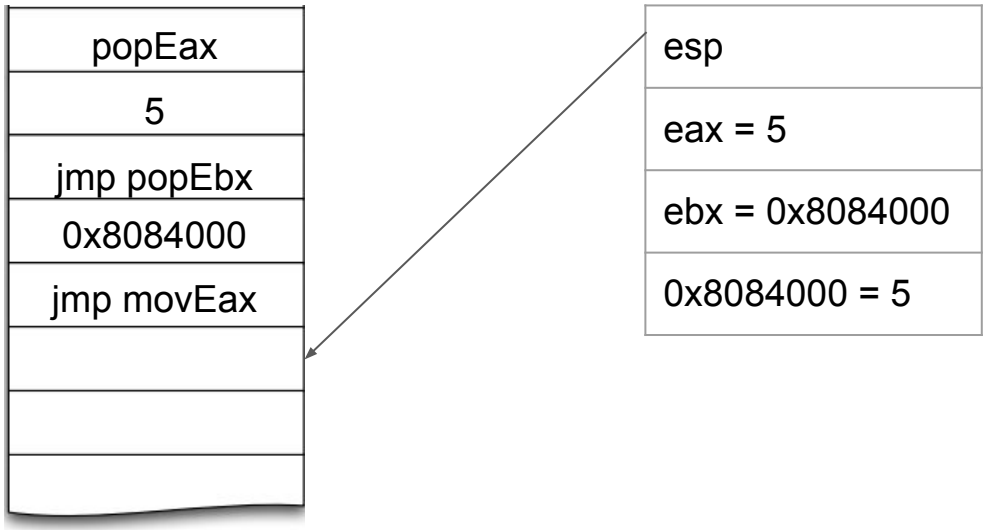
```
popEax:
     pop %eax
     ret

popEbx:
     pop %ebx
     ret

movEax:
     mov %eax, (%ebx)
     ret
```



Return Oriented Programming

Main Idea: return to chains of **gadgets**, which are instruction sequences ending in `ret`. This allows us to perform arbitrary code execution **without actually introducing new code.**

```
popEax:
     pop %eax
     ret

popEbx:
     pop %ebx
     ret

movEax:
     mov %eax, (%ebx)
     ret
```



Sp15 - Midterm 1 - Q2

Mark each defense that would detect, prevent, or significantly reduce the probability of the success of each attack attempt.

Assumptions:
1. Whenever the attacker needs to overwrite a saved return address, he does so by writing past the end of the buffer onto the stack.

2. The attacker only overwrites the saved return addresses with absolute addresses.

(a) The attacker overwrites the saved return address pointer. He then rewrites this return address to point to code that he injected into the vulnerable buffer on the stack.

- StackGuard (StackCanary)
- Address Space Layout Randomization (ASLR)
- Non-executable (NX) Stack
- Bounds Checking
- None of the above

Sp15 - Midterm 1 - Q2

Mark each defense that would detect, prevent, or significantly reduce the probability of the success of each attack attempt.

Assumptions:
1. Whenever the attacker needs to overwrite a saved return address, he does so by writing past the end of the buffer onto the stack.

2. The attacker only overwrites the saved return addresses with absolute addresses.

(a) The attacker overwrites the saved return address pointer. He then rewrites this return address to point to code that he injected into the vulnerable buffer on the stack.

- StackGuard (StackCanary)
- Address Space Layout Randomization (ASLR)
- Non-executable (NX) Stack
- Bounds Checking
- None of the above

Sp15 - Midterm 1 - Q2

Mark each defense that would detect, prevent, or significantly reduce the probability of the success of each attack attempt.

Assumptions:
1. Whenever the attacker needs to overwrite a saved return address, he does so by writing past the end of the buffer onto the stack.

2. The attacker only overwrites the saved return addresses with absolute addresses.

(b) The attacker overwrites a function pointer on the stack to point to malicious code stored in stack memory. Assume the malicious code is already in place.

- StackGuard (StackCanary)
- Address Space Layout Randomization (ASLR)
- Non-executable (NX) Stack
- Bounds Checking
- None of the above

Sp15 - Midterm 1 - Q2

Mark each defense that would detect, prevent, or significantly reduce the probability of the success of each attack attempt.

Assumptions:
1. Whenever the attacker needs to overwrite a saved return address, he does so by writing past the end of the buffer onto the stack.

2. The attacker only overwrites the saved return addresses with absolute addresses.

(b) The attacker overwrites a function pointer on the stack to point to malicious code stored in stack memory. Assume the malicious code is already in place.

- StackGuard (StackCanary)
- Address Space Layout Randomization (ASLR)
- Non-executable (NX) Stack
- Bounds Checking
- None of the above

Sp15 - Midterm 1 - Q2

Mark each defense that would detect, prevent, or significantly reduce the probability of the success of each attack attempt.

Assumptions:
1. Whenever the attacker needs to overwrite a saved return address, he does so by writing past the end of the buffer onto the stack.

2. The attacker only overwrites the saved return addresses with absolute addresses.

(c) The attacker performs a return to libc (ret2libc) by overwriting the saved return address to execute code from libc.

- StackGuard (StackCanary)
- Address Space Layout Randomization (ASLR)
- Non-executable (NX) Stack
- Bounds Checking
- None of the above

Sp15 - Midterm 1 - Q2

Mark each defense that would detect, prevent, or significantly reduce the probability of the success of each attack attempt.

Assumptions:
1. Whenever the attacker needs to overwrite a saved return address, he does so by writing past the end of the buffer onto the stack.

2. The attacker only overwrites the saved return addresses with absolute addresses.

(c) The attacker performs a return to libc (ret2libc) by overwriting the saved return address to execute code from libc.

- StackGuard (StackCanary)
- Address Space Layout Randomization (ASLR)
- Non-executable (NX) Stack
- Bounds Checking
- None of the above

Sp15 - Midterm 1 - Q2

Mark each defense that would detect, prevent, or significantly reduce the probability of the success of each attack attempt.

Assumptions:
1. Whenever the attacker needs to overwrite a saved return address, he does so by writing past the end of the buffer onto the stack.

2. The attacker only overwrites the saved return addresses with absolute addresses.

(d) The attacker performs a return-oriented programming (ROP) attack by chaining gadgets stored in the code portion of memory by overwriting the saved return address.

- StackGuard (StackCanary)
- Address Space Layout Randomization (ASLR)
- Non-executable (NX) Stack
- Bounds Checking
- None of the above

Sp15 - Midterm 1 - Q2

Mark each defense that would detect, prevent, or significantly reduce the probability of the success of each attack attempt.

Assumptions:
1. Whenever the attacker needs to overwrite a saved return address, he does so by writing past the end of the buffer onto the stack.

2. The attacker only overwrites the saved return addresses with absolute addresses.

(d) The attacker performs a return-oriented programming (ROP) attack by chaining gadgets stored in the code portion of memory by overwriting the saved return address.

- StackGuard (StackCanary)
- Address Space Layout Randomization (ASLR)
- Non-executable (NX) Stack
- Bounds Checking
- None of the above

Intro to Web, SOP, SQL Injection

Web Security Risks

Risk #1: we don't want a malicious site to be able to trash my files/programs on my computer

- Browsing to `awesomevids.com` (or `evil.com`) should not infect my computer with malware, read or write files on my computer, etc.

Web Security Risks

Risk #1: we don't want a malicious site to be able to trash my files/programs on my computer

- Browsing to `awesomevids.com` (or `evil.com`) should not infect my computer with malware, read or write files on my computer, etc.

Defense: Javascript is sandboxed;
try to avoid security bugs in browser code;
privilege separation; automatic updates; etc.

Web Security Risks

Risk #2: we don’t want a malicious site to be able to spy on or tamper with my information or interactions with other websites

- Browsing to `evil.com` should not let `evil.com` spy on my emails in Gmail or buy stuff with my Amazon account

Web Security Risks

Risk #2: we don't want a malicious site to be able to spy on or tamper with my information or interactions with other websites

- Browsing to `evil.com` should not let `evil.com` spy on my emails in Gmail or buy stuff with my Amazon account

Defense: the same-origin policy

- A security policy grafted on after-the-fact, and enforced by web browsers

Web Security Risks

Risk #3: we want data stored on a web server to be protected from unauthorized access

Web Security Risks

Risk #3: we want data stored on a web server to be protected from unauthorized access

Defense: server-side security

SQL Injection

```
$recipient = $_POST['recipient'];
$sql = "SELECT AcctNum FROM Customer
        WHERE Balance < 100 AND
         Username='$recipient' ";
$result = $db->executeQuery($sql);
```

- How can $recipient cause trouble here?

SQL Injection

```
$recipient = $_POST['recipient'];
$sql = "SELECT AcctNum FROM Customer
        WHERE Balance < 100 AND
         Username='$recipient' ";
$result = $db->executeQuery($sql);
```

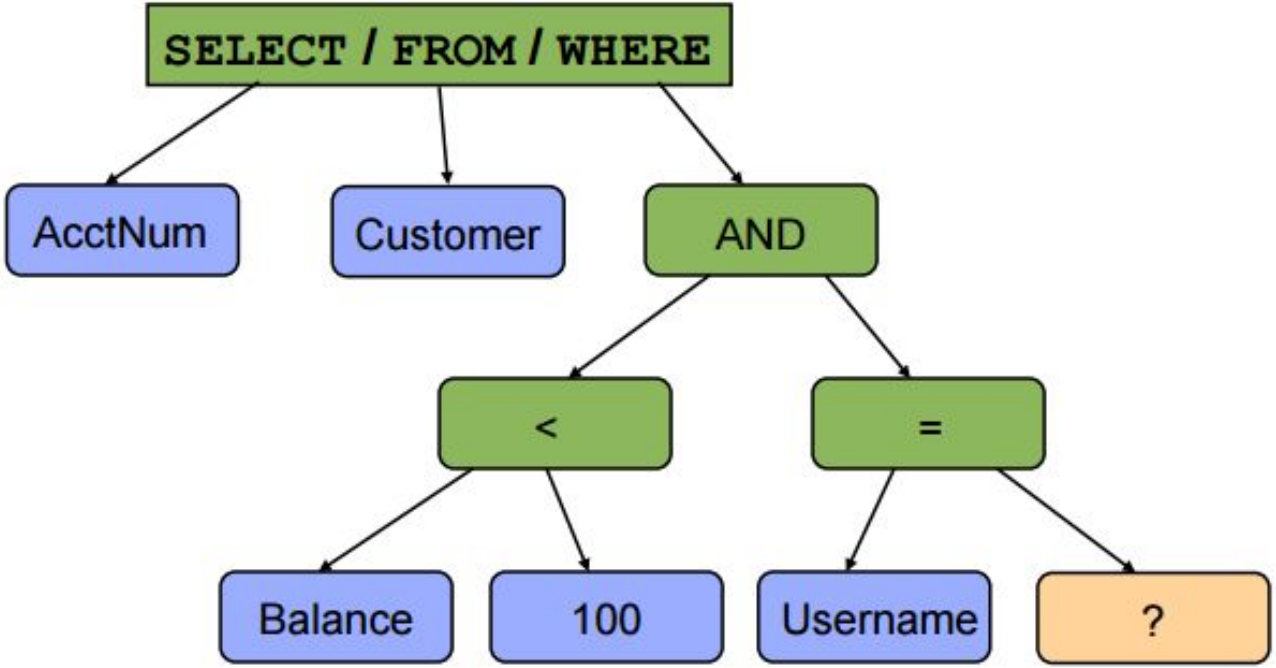
- How can $recipient cause trouble here?

```
foo'; DROP TABLE Customer; --
```

SQL Injection Defenses

- Sanitize Input
- Escape Input
- Use Prepared Statements
  - Language support for constructing queries
  - Input is confined to a single SQL data value

Prepared Statement



Note: **`prepared`** statement only allows ?'s at leaves, not internal nodes. So *structure* of tree is *fixed*.

Fall 2016 Final Q5

Problem 5 ***Tasty but insecure*** **(16 points)**

Oski was hired as a security consultant for www.tastytreats.com. The developer of this website has not taken CS 161 so none of the user input is validated or escaped. For each question, state a way that Oski can test for the specified vulnerability.

(a) After talking to an engineer at Tasty Treats, Oski finds the following line of code running at the web server:

```
treat = "SELECT * FROM TREATS WHERE Name = ' " + treatName + "';";
```

What could Oski enter in the place of treatName to erase all treats from the database?

Fall 2016 Final Q5

Problem 5 ***Tasty but insecure*** **(16 points)**

Oski was hired as a security consultant for www.tastytreats.com. The developer of this website has not taken CS 161 so none of the user input is validated or escaped. For each question, state a way that Oski can test for the specified vulnerability.

(a) After talking to an engineer at Tasty Treats, Oski finds the following line of code running at the web server:

```
treat = "SELECT * FROM TREATS WHERE Name = ' " + treatName + "';";
```

What could Oski enter in the place of treatName to erase all treats from the database?

Solution: x'; DROP TREATS; --

Fall 2016 Final Q5

(b) What should Oski recommend that Tasty Treats do to prevent the attack in part (a)? Write exactly two different defenses, and not more.

Fall 2016 Final Q5

(b) What should Oski recommend that Tasty Treats do to prevent the attack in part (a)?

Solution: Use Parametrized SQL statements, or escape, or sanitize inputs.

Same Origin Policy

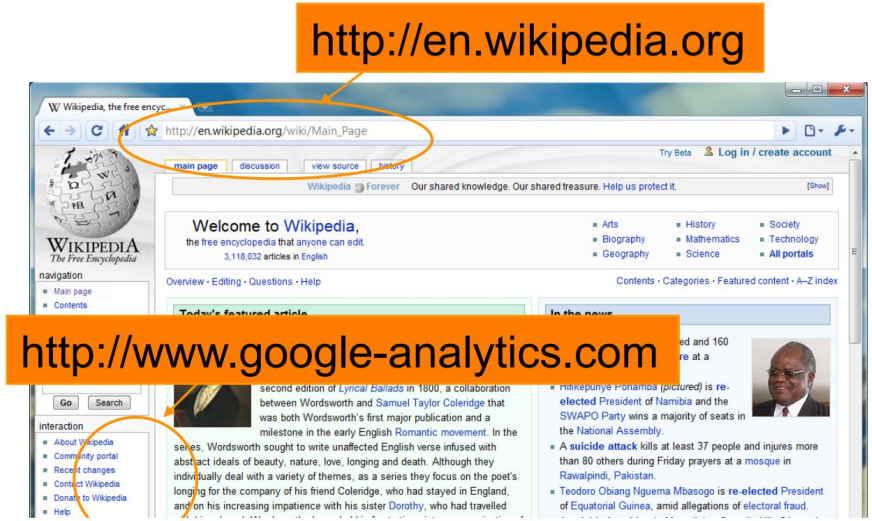
One origin should not be able to access the resources of another origin

Javascript on one page cannot read or modify pages from different origins.

The contents of an *iframe* have the origin of the URL from which the iframe is served; *not* the loading website.

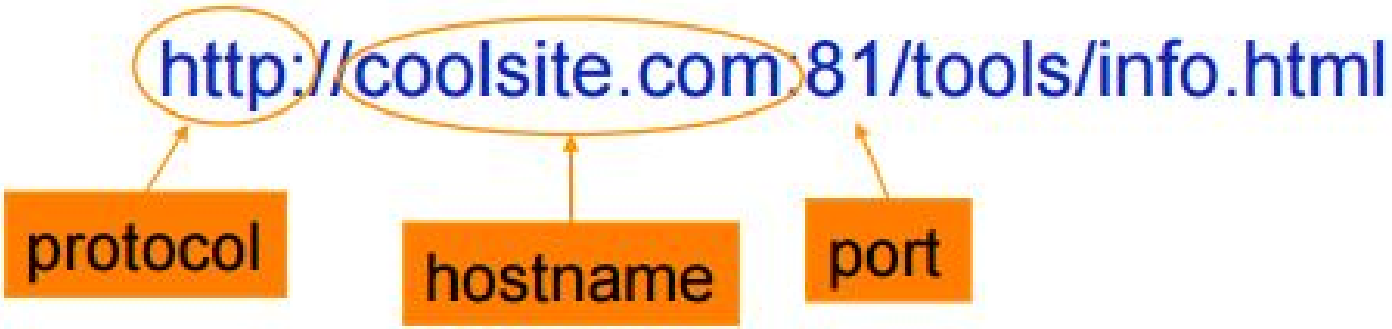
Same Origin Policy

- The origin of a page is derived from the URL it was loaded from
- Special case: Javascript runs with the origin of the page that loaded it



Same Origin Policy

- Origin = protocol + hostname + port



Spring 2016 Final Q6

Problem 6 ***Web security*** **(15 points)**

(a) When users of `bank.com` are logged in, a request to `bank.com/session.js` returns a Javascript file containing

```
var session_id = "0123456789";
```

except that 0123456789 is replaced with the session ID for the user who made the request.

An attacker controls `evil.com` and would like to learn Alice's session ID for `bank.com`. How can the attacker do this? Explain why the same-origin policy doesn't stop this attack. (Assume the attacker can get Alice to visit `evil.com`.)

Spring 2016 Final Q6

Problem 6 ***Web security*** **(15 points)**

(a) When users of `bank.com` are logged in, a request to `bank.com/session.js` returns a Javascript file containing

```
var session_id = "0123456789";
```

except that 0123456789 is replaced with the session ID for the user who made the request.

An attacker controls `evil.com` and would like to learn Alice's session ID for `bank.com`. How can the attacker do this? Explain why the same-origin policy doesn't stop this attack. (Assume the attacker can get Alice to visit `evil.com`.)

Solution: Put `<SCRIPT SRC="http://bank.com/session.js">` into `evil.com`'s page, followed by Javascript that reads the `session_id` variable and sends it back to `evil.com`. Allowed because of the special exception in the same-origin policy for scripts.

(It's fine to assume Alice is logged in for this part; otherwise, no attack is possible.)

Spring 2016 Final Q6

(b) When `bank.com` learns of this problem, they fix it by beginning all Javascript files with

```
if (!document.location.includes("http://bank.com")) {
    while (1) {}   // infinite loop
}
```

Explain why this doesn't work. How could an attacker defeat this defense?

Spring 2016 Final Q6

(b) When `bank.com` learns of this problem, they fix it by beginning all Javascript files with

```
if (!document.location.includes("http://bank.com")) {
    while (1) {}   // infinite loop
}
```

Explain why this doesn't work. How could an attacker defeat this defense?

Solution: An attacker could put malicious stuff on `http://bank.com.evil.com/`, or on `http://evil.com/foohttp://bank.com/`.

Spring 2016 Final Q6

(c) Propose better Javascript code to put at the start of all Javascript files.

Spring 2016 Final Q6

(c) Propose better Javascript code to put at the start of all Javascript files.

Solution:

```
if (!document.location.startsWith("http://bank.com/")) {
    while (1) {}   // infinite loop
}
```

or

```
if (document.domain != "bank.com")) {
    while (1) {}   // infinite loop
}
```

Clickjacking

- By placing an invisible iframe of `target.com` *over* some enticing content, a malicious web server can fool a user into taking unintended action on `target.com` …
- … By placing a visible iframe of `target.com` *under* the *attacker's own invisible iframe*, a malicious web server can "steal" user input – in particular, keystrokes

Clickjacking Defenses

a. Framebusting: Web site ensures that its pages can’t be included as a frame inside another browser frame
b. X-Frame-Options (HTTP header options): Allows whitelisting of what domains – if any – are allowed to frame a given page a server returns

More web attacks!

You developed the web-based payment form for a new fancy payments startup, CashMo. When a user clicks submit, the following request is made:

```
https://www.cashmo.com/payment?amount=<dollar amount>
&recipient=<username>
```

Soon after, your friend Eve sends you this message:

```
Hey, check out this funny cat picture.
http://MyAwesomeUrlShortener.com/as3fsjg
```

You click on this link and later find out that you have paid Eve 1 dollar via CashMo.

(a) Name the type of vulnerability that Eve exploited to steal your money.

(b) What did the link redirect to?

Cookies

GET ...

Server

HTTP Header:

Set-cookie: NAME=VALUE ;

domain = (when to send) ;

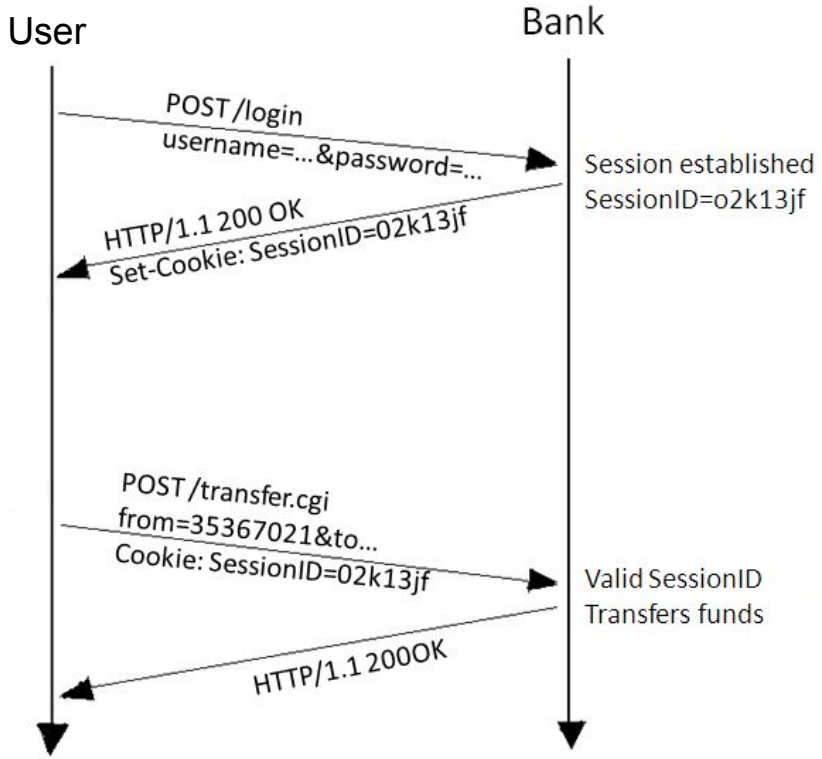
path = (when to send)

secure = (only send over HTTPS);

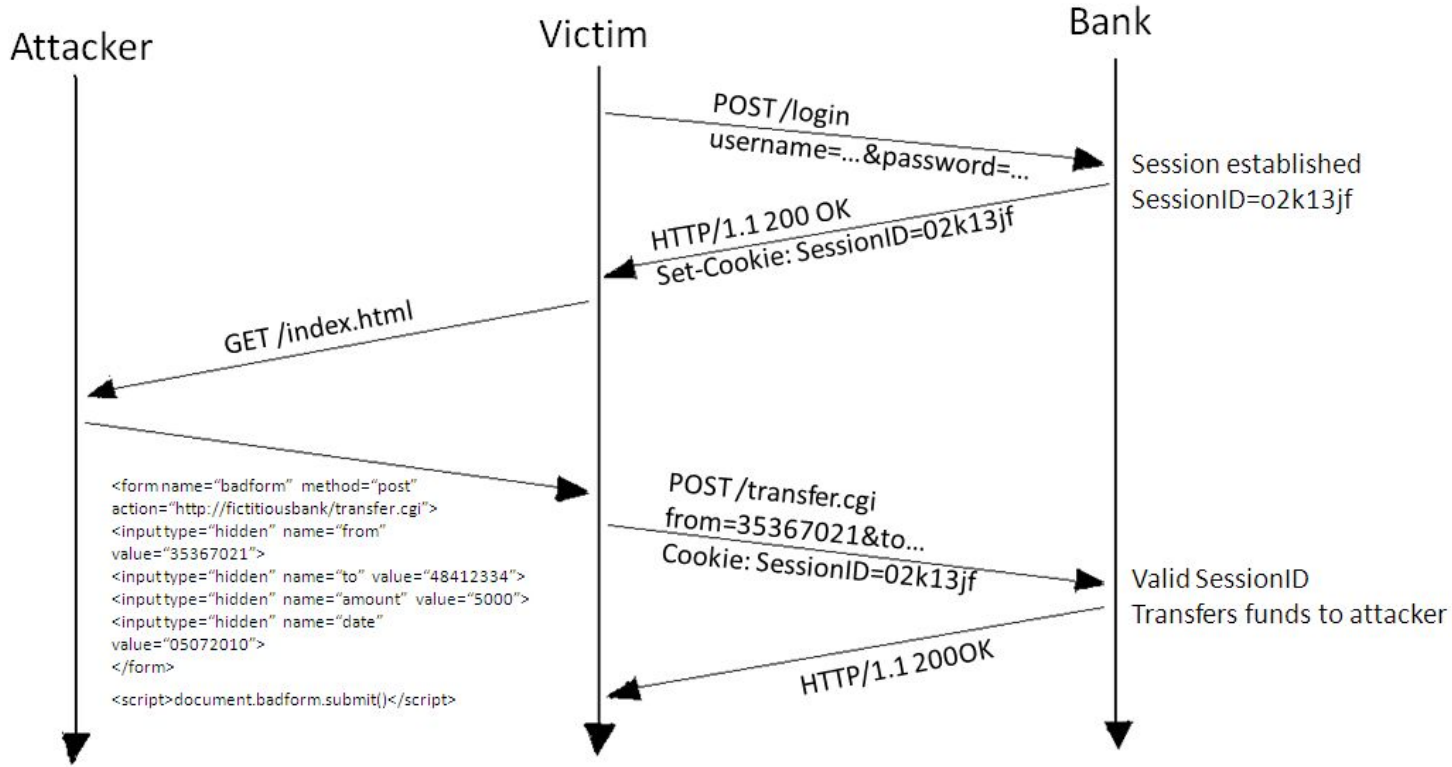
expires = (when expires) ;

HttpOnly

Cross-site request forgery (CSRF)



Cross-site request forgery (CSRF)

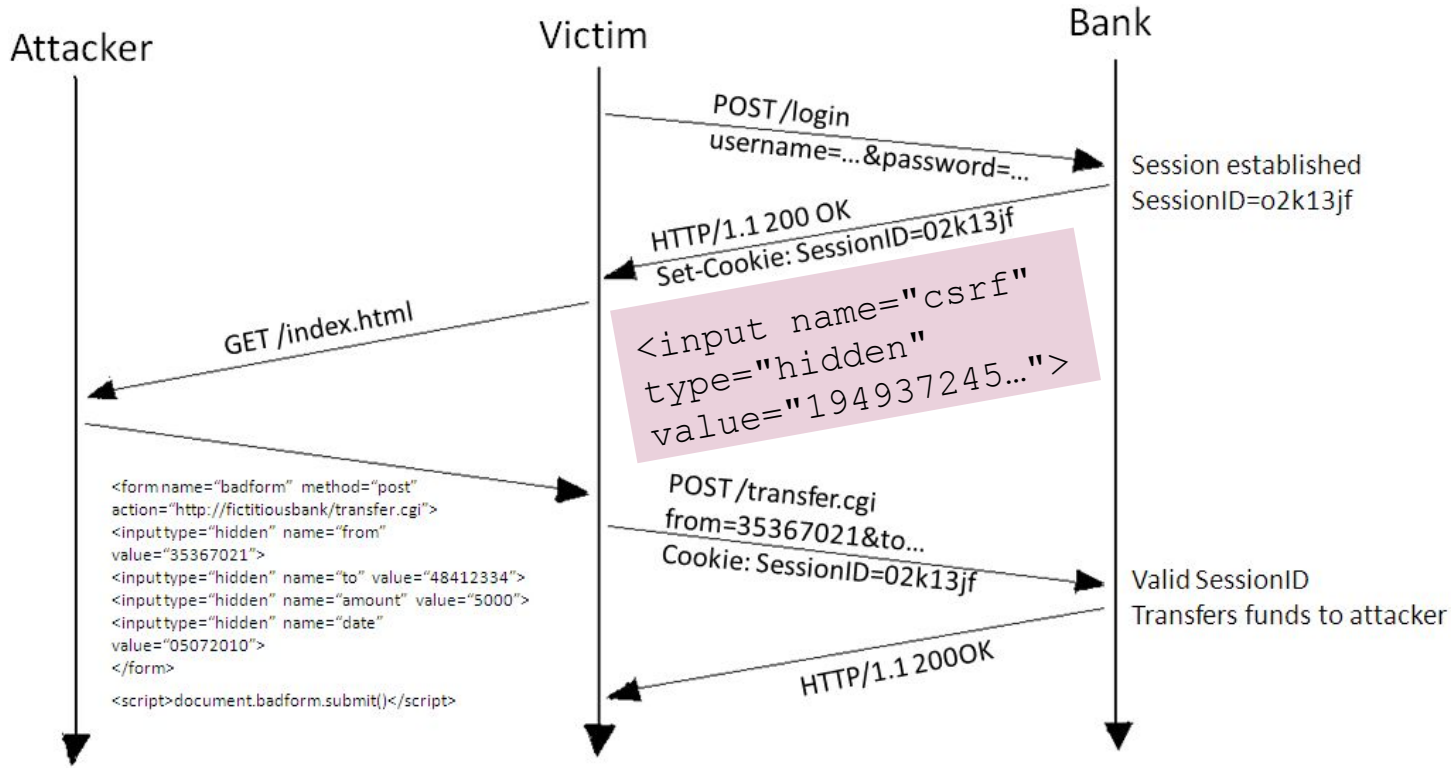


(c) How could you, as the developer of CashMo, defend your web service from this sort of attack?

(c) How could you, as the developer of CashMo, defend your web service from this sort of attack?

- Require the user to re-authenticate
  - e.g., re-enter username and password explicitly for every sensitive transaction
- Check the `Referer` header
- Check the `Origin` header
- CSRF tokens

CSRF tokens


Attacker
Victim
Bank
POST /login
username=...&password=...
Session established
SessionID=o2k13jf
HTTP/1.1 200 OK
Set-Cookie: SessionID=02k13jf
GET /index.html
<input name="csrf"
type="hidden"
value="194937245...">
<form name="badform" method="post"
action="http://fictitiousbank/transfer.cgi">
<input type="hidden" name="from"
value="35367021">
<input type="hidden" name="to" value="48412334">
<input type="hidden" name="amount" value="5000">
<input type="hidden" name="date"
value="05072010">
</form>
<script>document.badform.submit()</script>
POST /transfer.cgi
from=35367021&to...
Cookie: SessionID=02k13jf
Valid SessionID
Transfers funds to attacker
HTTP/1.1 200OK

Squigler provides a way to search for Squigs. When presented with a URL such as:

```
http://www.squigler.com/search?query=cats
```

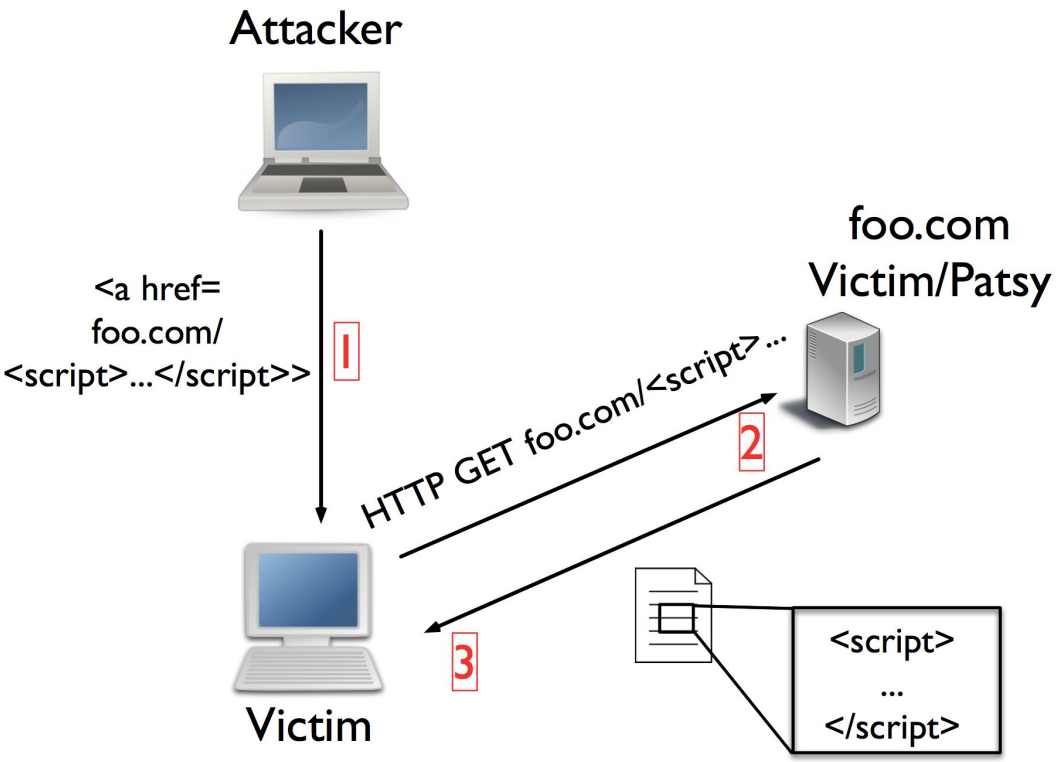
The server will return an HTML search results page containing:

```
...searched for: <b>cats</b> ...
```

In particular, the search phrase from the URL parameter is always included into the HTML exactly as found in the URL, without any changes.

(a) The site has a vulnerability. Describe it, in a sentence or two.

Reflected XSS



Squigler provides a way to post Squigs. When presented with a URL such as:

```
http://www.squigler.com/post?squig=I love cats!
```

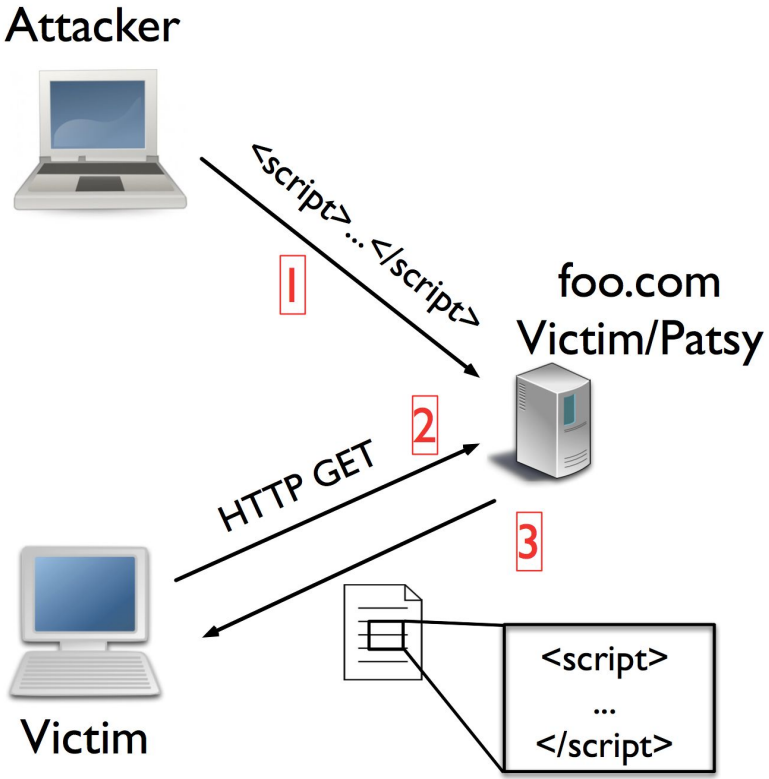
Every Squigler user will see an HTML page containing:

```
...posted: <b>I love cats!</b> ...
```

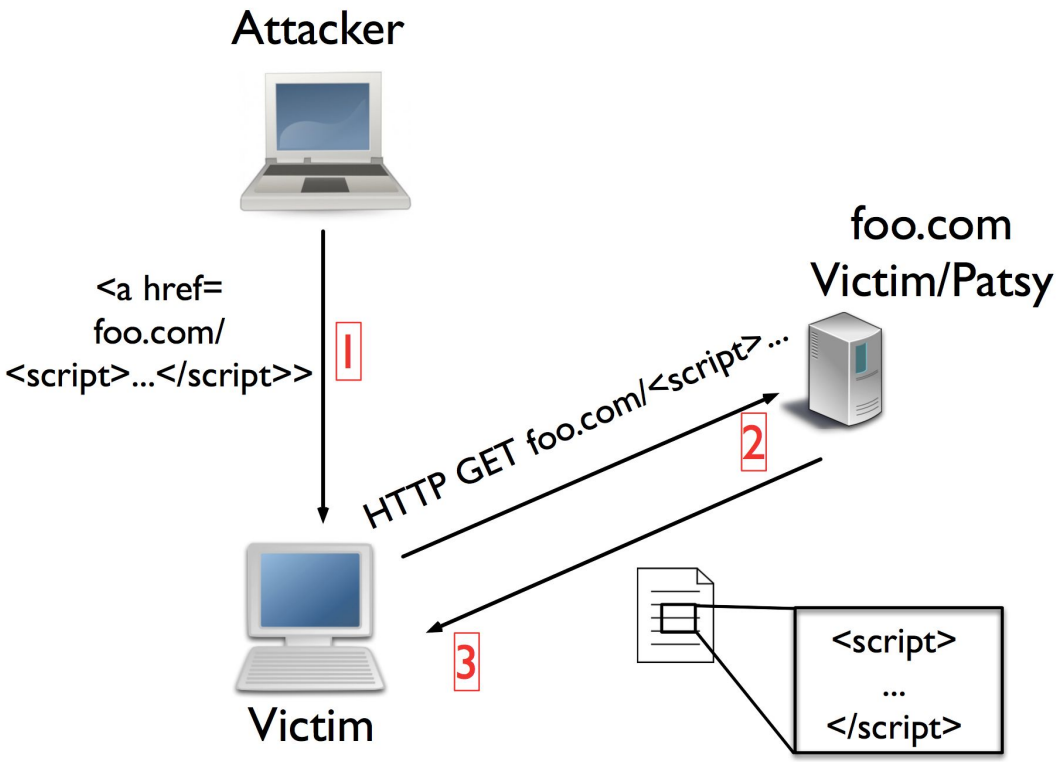
In particular, the search phrase from the URL parameter is always included into the HTML exactly as found in the URL, without any changes.

(a) The site has a vulnerability. Describe it, in a sentence or two.

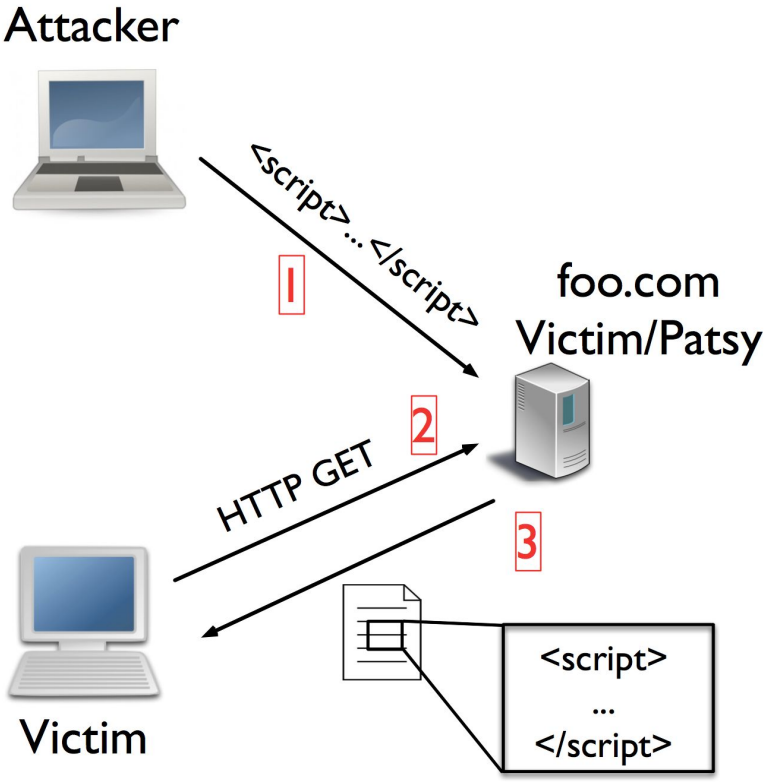
Stored XSS

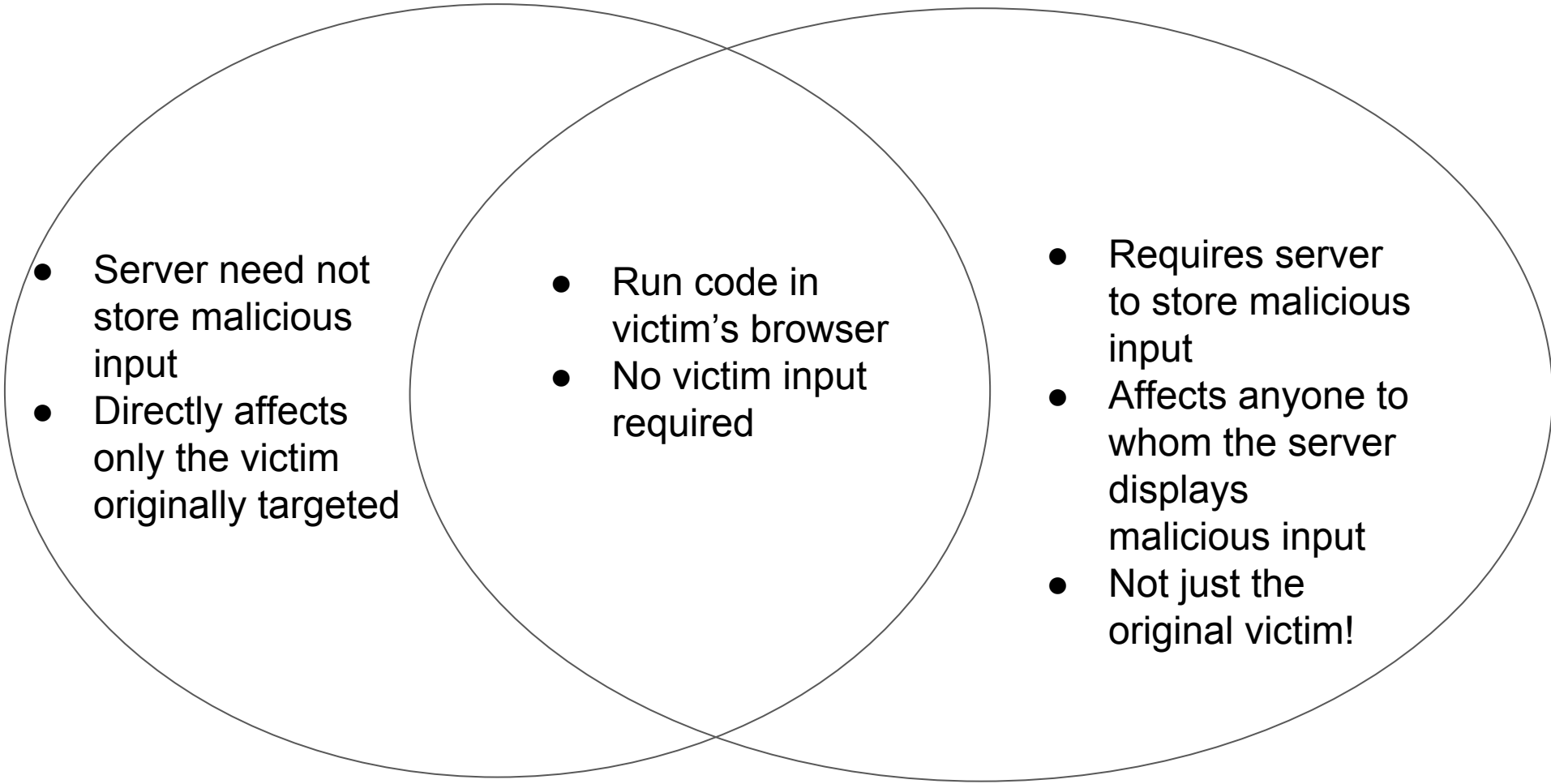

Attacker
<script>… </script>
1
foo.com
Victim/Patsy
2
HTTP GET
3
<script>
…
</script>
Victim

Reflected XSS



Stored XSS




Reflected XSS
Stored XSS
Server need not store malicious input
Directly affects only the victim originally targeted
Run code in victim's browser
No victim input required
Requires server to store malicious input
Affects anyone to whom the server displays malicious input
Not just the original victim!

XSS Defenses

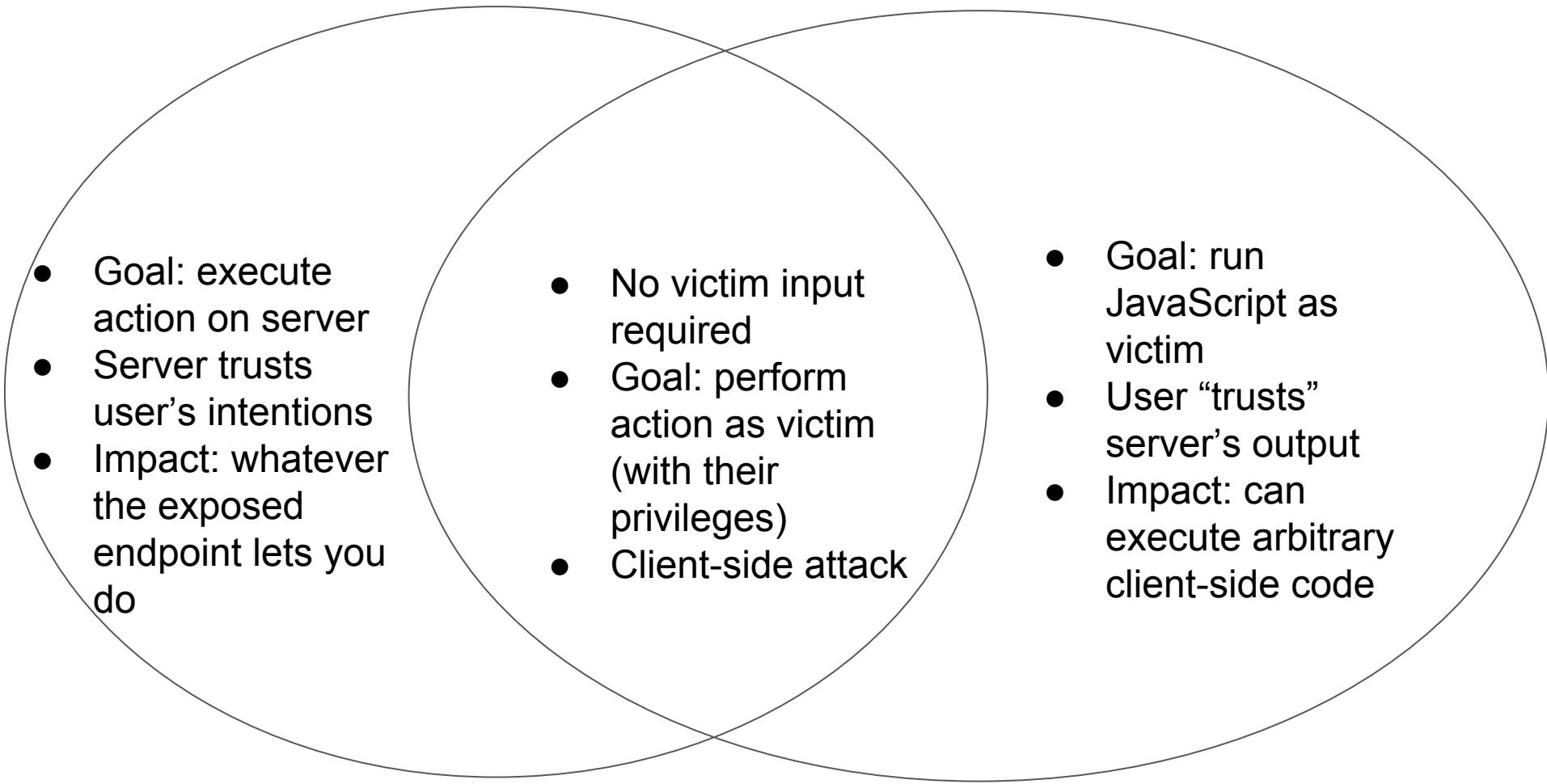
a. Never insert untrusted data except in allowed locations
b. HTML-escape user input
c. Content-Security-Policy HTTP header: allows reply to specify white-list, instructs the browser to only execute or render resources from those sources

Content Security Policy

Goal: prevent XSS by specifying a whitelist from where a browser can load resources (Javascript scripts, images, frames, …) for a given web page

Approach:

- Prohibits inline scripts
- Content-Security-Policy HTTP header allows reply to specify white-list, instructs the browser to only execute or render resources from those sources
  - e.g., `script-src 'self' http://b.com; img-src *`
- – Relies on browser to enforce


CSRF
XSS
Goal: execute action on server
Server trusts user's intentions
Impact: whatever the exposed endpoint lets you do
No victim input required
Goal: perform action as victim (with their privileges)
Client-side attack
Goal: run JavaScript as victim
User "trusts" server's output
Impact: can execute arbitrary client-side code

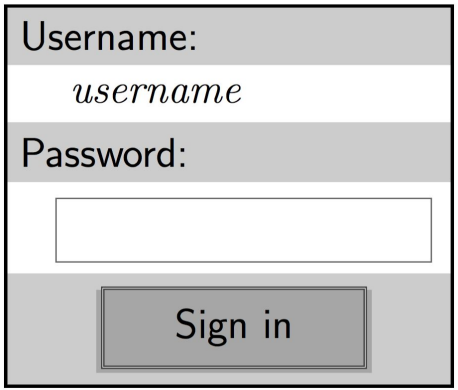
EasyWeb Inc.’s web server functions as follows: it logs in users whenever it receives a URL of the following form:

```
http://easyweb.com/login?user=username&pass=password
```

(assuming that the provided username & password are correct)

(a) Name and briefly describe a vulnerability in EasyWeb Inc.’s use of this approach. Briefly sketch a scenario in which an attacker would exploit the vulnerability.

EasyWeb Inc. now requires that all login requests go through a specific login form page. When a user of the service first surfs to the URL `http://www.easyweb.com/login?user=username`, the website returns a web page that conveniently pre-fills part of the login form for the user, like this:



(c) In using this approach, EasyWeb Inc. has introduced a new vulnerability while fixing the previous one. Name the vulnerability and briefly describe it.

(d) Explain how an attacker can use this new vulnerability to perform an attack. Briefly sketch how the attacker sets up the attack and what happens when the attack occurs.

(e) Briefly sketch a potential defense that will prevent at least some instances of the attacks enabled by the vulnerability in part (d), even if not all of them. Discuss a drawback or limitation of the defense.

The End

Good luck! 

Bonus questions

Oski wants to look up the treat, samosas, on the Tasty Treats website. When he enters samosas in the search bar he notices that his browser issues an http request to the url

> http://www.tastytreats.com/search.html?term=samosas

and that the web server embeds the term with no validation in the page returned. For example, he gets the following message:

> The treat "samosas" was not found.

What kind of vulnerability has Oski found?

Consider an attacker who wants to obtain the cookies of Alice for tastytreats.com.

Write down the URL that Oski (pretending he is an attacker) can send to Alice in an email such that, when she clicks on that URL, Oski's site (www.oski.com/getcookie) ends up obtaining her cookies. If you don't know the exact name of a HTML or Javascript command, write a name with a suggestive meaning.

www.awesomevids.com provides a way to search for cool videos. When presented with a URL such as:

http://www.awesomevids.com/search.php?search=cats

The server will return an HTML search results page containing: ...searched for: cats ...

In particular, the search phrase from the URL parameter is always included into the HTML exactly as found in the URL, without any changes.

(a) The site has a vulnerability. Describe it, in a sentence or two

(b) Alice is a user of www.awesomevids.com. Describe how an attacker might be able to use this vulnerability to steal the cookies that Alice’s browser has for www.awesomevids.com. You can assume that the attacker knows Alice’s email address.

(c) The developers of www.awesomevids.com hear rumors of this vulnerability in their site, so they deploy framebusting on all of their pages. Does this prevent exploitation of the vulnerability? Why or why not?