Week 2 Discussion

Zijie Zhao

Karthik Mudda

Sumanth Rao

Stack Layout

```
              High Memory

        ┌─────────────────────┐
        │                     │
        ├─────────────────────┤
        │                     │
        │      arguments      │
        │                     │
        ├─────────────────────┤
        │     saved %eip      │
  ebp ──┼──▶ saved %ebp       │
        ├─────────────────────┤
        │    other saved      │
        │     registers       │
        ├─────────────────────┤
        │                     │
        │   local variables   │
        │                     │
        ├─────────────────────┤
        │                     │
        └─────────────────────┘

              Low Memory
```
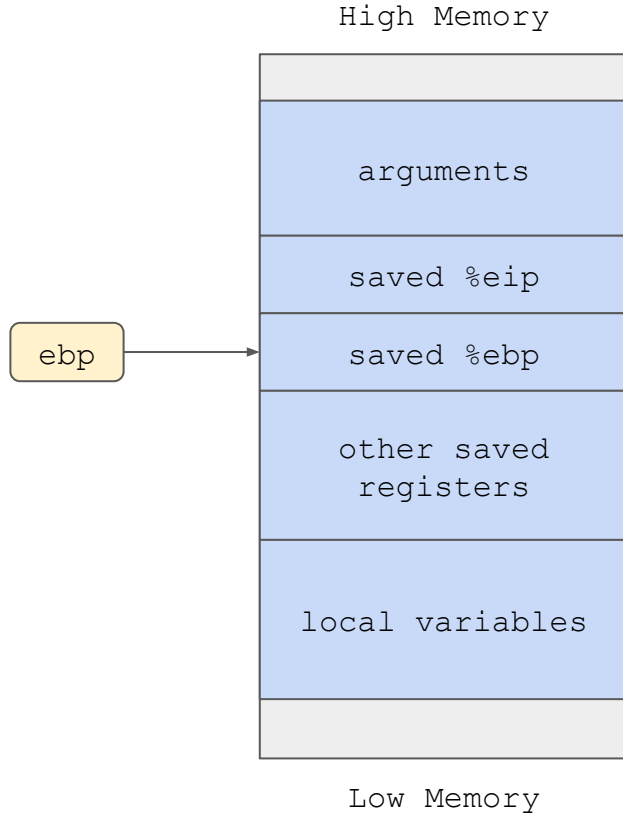
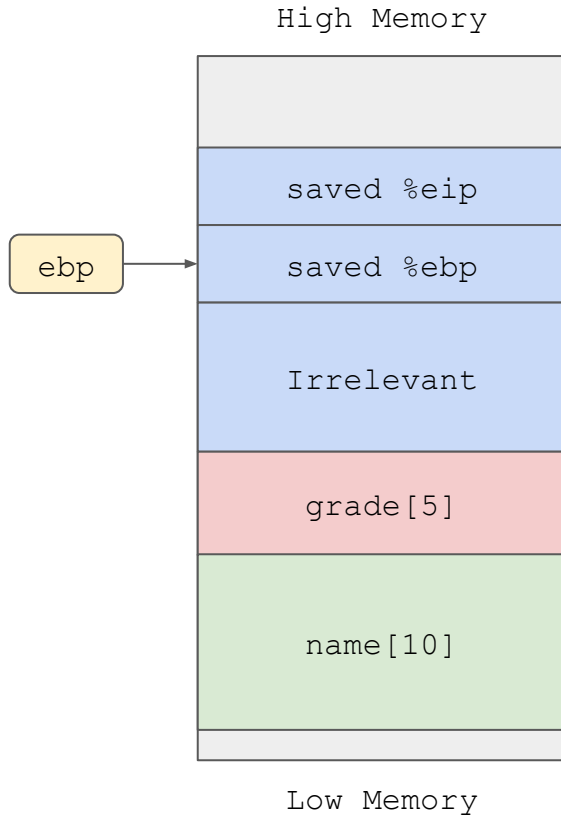`%eip` is a register pointing to the instruction that CPU will execute in next cycle

saved `%eip` references to a 4-byte address value stored on stack

saved `%eip` is stored on stack when a function call is made. It has the address of where to resume execution in the caller function

When a function returns, the saved `%eip` value will be popped into the register `%eip` → control will transfer to where saved `%eip` points to

return address == saved return address == saved `%eip` == `%ebp+4`

Target 0

High Memory

ebp

```
int _main(int argc, char *argv[])
{
        char grade[5];
        char name[10];

        strcpy(grade, "nil");

        gets(name);

        printf("Hi %s! Your grade is %s.\n", name,
grade);

        exit(0);
}
```

- How does `gets` copy stdin into `name`?
- When does `gets` stop copying?
- Will `gets` check the boundary between `name` and `grade`?

Low Memory

Be careful, when `name` is printed, no extra garbage content should be printed. Think about when `printf` stops prints given a pointer.

Target 0
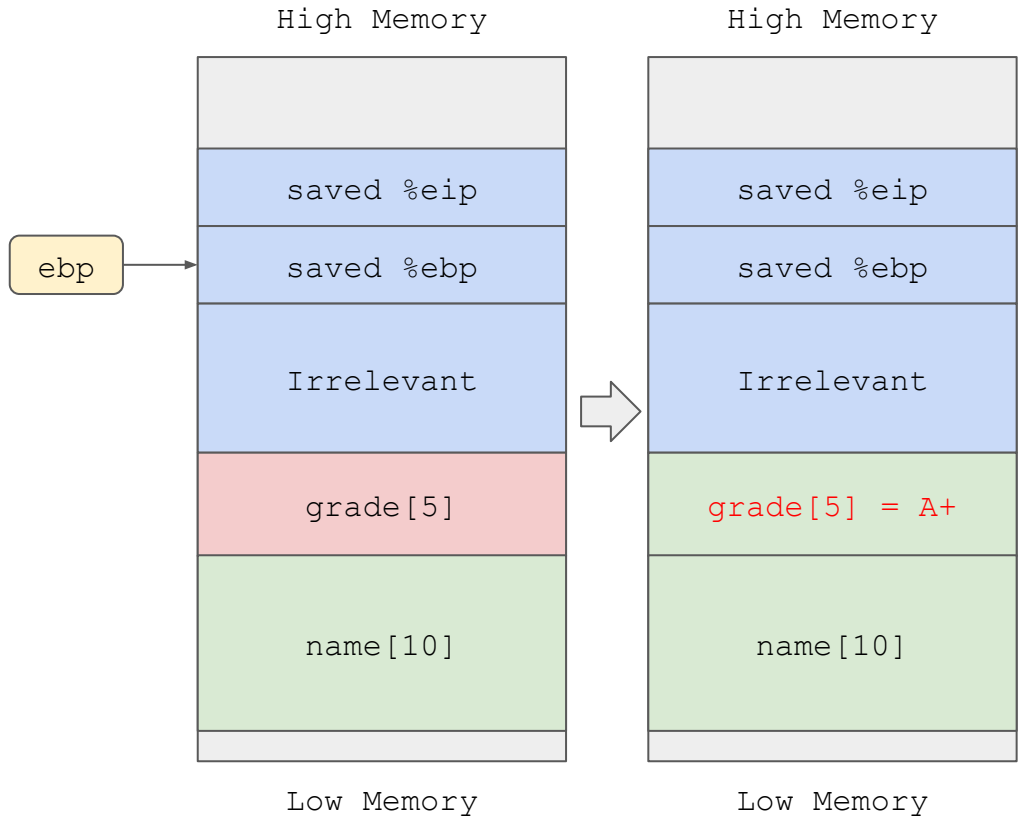
High Memory

| saved %eip |
| saved %ebp |
| Irrelevant |
| grade[5] |
| name[10] |

ebp →

Low Memory

High Memory

| saved %eip |
| saved %ebp |
| Irrelevant |
| grade[5] = A+ |
| name[10] |

Low Memory

```
int _main(int argc, char *argv[])
{
        char grade[5];
        char name[10];

        strcpy(grade, "nil");

        gets(name);

        printf("Hi %s! Your grade is %s.\n", name,
grade);

        exit(0);
}
```

UCSDCSE
Computer Science and Engineering

Target 1

High Memory

saved %eip
saved %ebp
Irrelevant
input[4]

ebp →

Low Memory
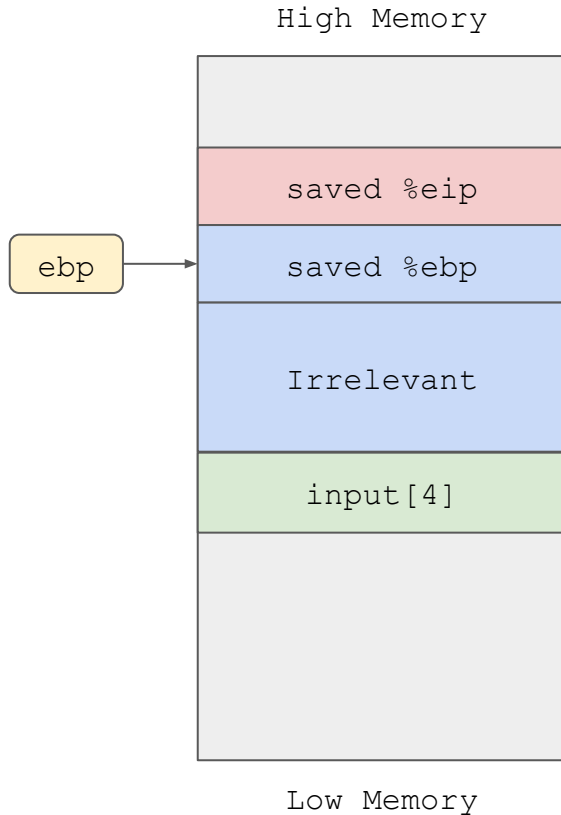
- What is the distance between `input` and saved `%eip`?
- How do you fill the gap in between?
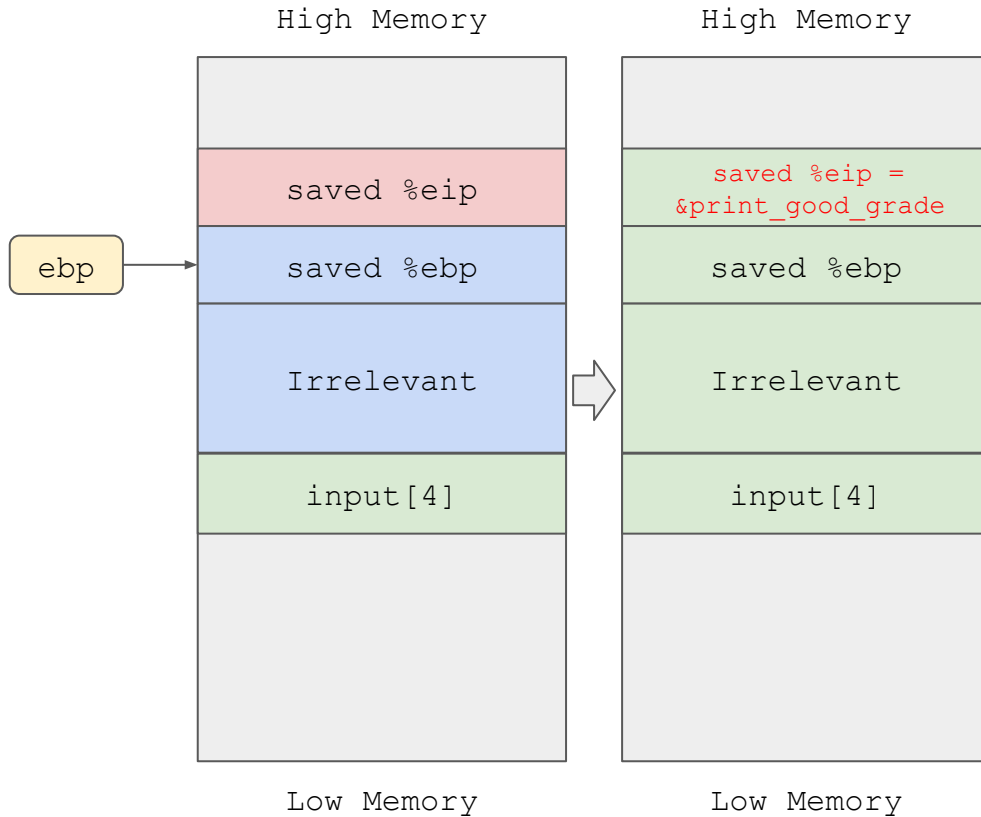- Where do you want to redirect control to?

```
void print_bad_grade(void)
{
        puts("Your grade is nil.");
        exit(0);
}

void print_good_grade(void)
{
        puts("Your grade is perfect.");
        exit(1);
}

void vulnerable()
{
        char input[4];
        gets(input);
}

int _main()
{
        vulnerable();
        print_bad_grade();
        return 0;
}
```
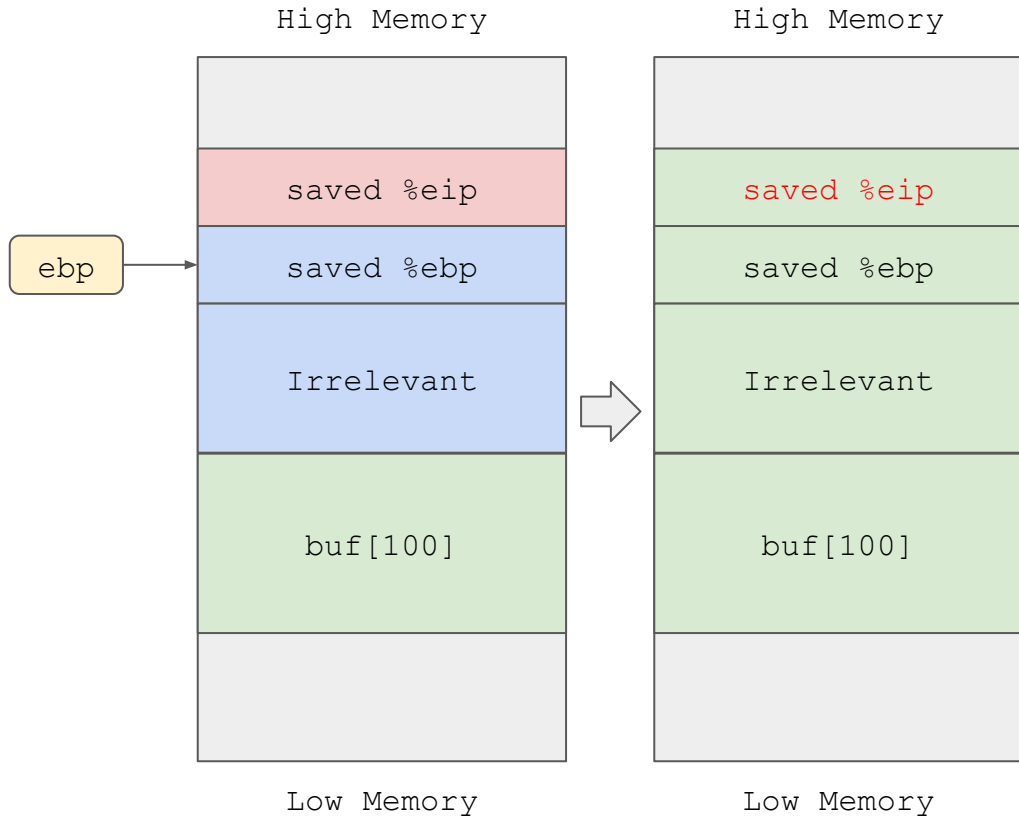
Target 1

High Memory

saved %eip
saved %ebp
Irrelevant
input[4]

ebp →

Low Memory

High Memory

saved %eip = &print_good_grade
saved %ebp
Irrelevant
input[4]

Low Memory

```
void print_bad_grade(void)
{
        puts("Your grade is nil.");
        exit(0);
}

void print_good_grade(void)
{
        puts("Your grade is perfect.");
        exit(1);
}

void vulnerable()
{
        char input[4];
        gets(input);
}

int _main()
{
        vulnerable();
        print_bad_grade();
        return 0;
}
```

Target 2

High Memory
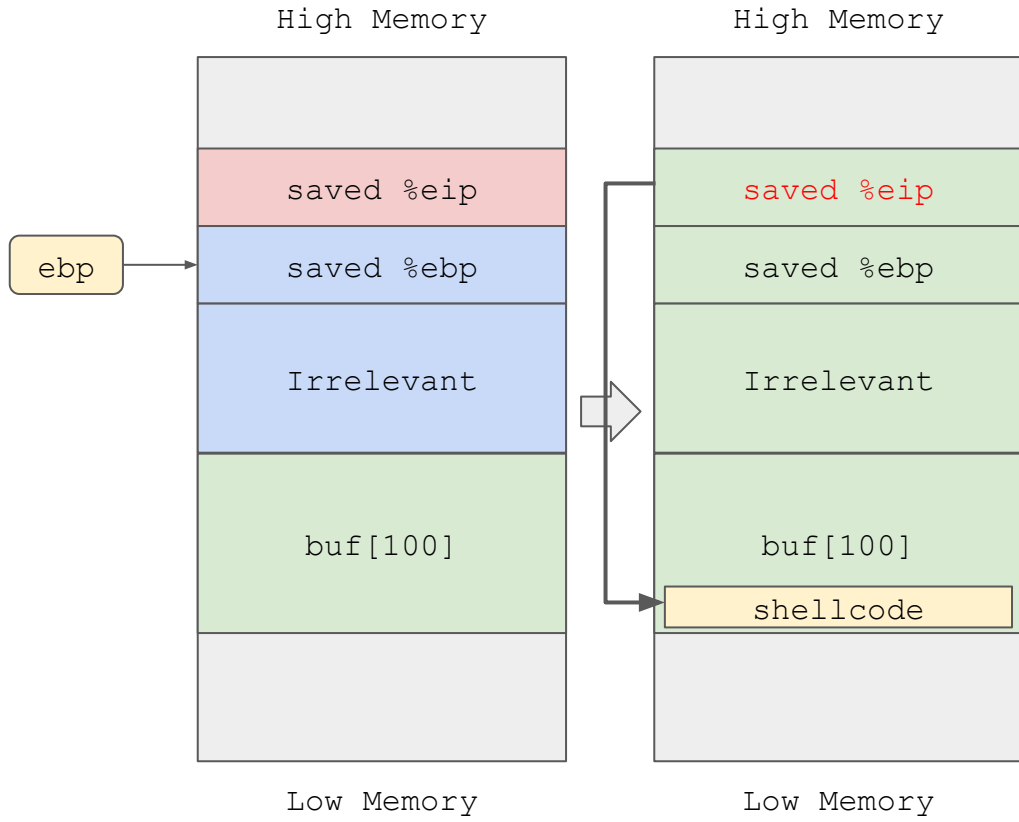

High Memory

```
void vulnerable(char *arg)
{
        char buf[100];
        strcpy(buf, arg);
}

int _main(int argc, char **argv)
{
        if (argc != 2) {
                fprintf(stderr, "Error: need a command-line
argument\n");
                return 1;
        }
        vulnerable(argv[1]);
        return 0;
}
```

Where do you want to transfer control to?

Target 2

High Memory

Low Memory

High Memory

Low Memory

```
void vulnerable(char *arg)
{
        char buf[100];
        strcpy(buf, arg);
}

int _main(int argc, char **argv)
{
        if (argc != 2) {
                fprintf(stderr, "Error: need a command-line
argument\n");
                return 1;
        }
        vulnerable(argv[1]);
        return 0;
}
```
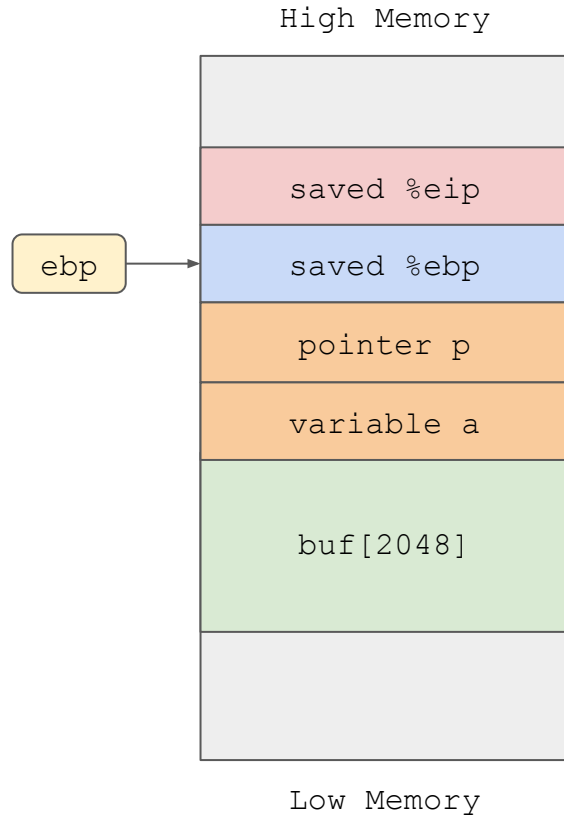
Where do you want to transfer control to?

Shellcode - Simple examples and explanations

- https://www.youtube.com/watch?v=1S0aBV-Waeo
- https://www.youtube.com/watch?v=hJ8IwyhqzD4

Target 3 (Hint)

High Memory


```
void vulnerable(char *arg)
{
        int *p;
        int a;
        char buf[2048];

        strncpy(buf, arg, sizeof(buf) + 8);

        *p = a;
}
```

Low Memory

- Technique: "Pointer subterfuge"

- Used to bypass stack canaries

Target 4 (Hint)

- count: 32-bit unsigned integer

- What happens when I overflow an unsigned int?

```
int main() {
    unsigned int a = pow(2, 30) + 127;
    unsigned int b = pow(2, 31) + 100;

    printf("%u\n", a * 4);
    printf("%u\n", b * 2);
}
```

```
void read_elements(FILE *f, int *buf, unsigned int count)
{
        unsigned int i;
        for (i=0; i < count; i++) {
                if (fread(&buf[i], sizeof(unsigned int), 1, f) < 1) {
                        break;
                }
        }
}

void read_file(char *name)
{
        FILE *f = fopen(name, "rb");
        …..

        unsigned int count;
        fread(&count, sizeof(unsigned int), 1, f);

        unsigned int *buf = alloca(count * sizeof(unsigned int));
        ….

        read_elements(f, buf, count);
}
```

Target 5 (Hint)

- Remember - Compiled with DEP enabled

- What can't you do - put shellcode in buffer and point to it from return address.

- What can you do?

- **"Return-to-libc" style of attack**

```
// Compiled with DEP enabled.

void greetings(void)
{
        system("echo Hello World");
}

void vulnerable(char *arg)
{
        char buf[10];
        strcpy(buf, arg);
}

int _main(int argc, char *argv[])
{
        ….
        setuid(0);
        vulnerable(argv[1]);
        greetings();
        ….
}
```

Target 6 (Hint)

- Remember ASLR is enabled.

- Stack can be placed anywhere in memory (0–255 bytes)

- How can I make sure my shell code is always hit?

```
// Compiled with -DMINIASLR.

void vulnerable(char *arg)
{
        char buf[1024];
        strcpy(buf, arg);
}


int _main(int argc, char *argv[])
{
        ….

        vulnerable(argv[1]);

        return 0;
}
```