COMP26020 Programming Languages and Paradigms -- Part 1

Memory Safety

```
└─$ ./basic-test-suite
basic-test-suite.c:344:allocate:PASS
basic-test-suite.c:347:init_rand:PASS
basic-test-suite.c:348:init_zeros:PASS
basic-test-suite.c:349:init_identity:PASS
basic-test-suite.c:352:equal:PASS
basic-test-suite.c:355:sum:PASS
basic-test-suite.c:356:scalar_product:PASS
basic-test-suite.c:357:transposition:PASS
basic-test-suite.c:358:product:PASS
basic-test-suite.c:363:dump_file:PASS
basic-test-suite.c:366:load_from_file:PASS
basic-test-suite.c:369:dump_and_load_from_file:PASS
basic-test-suite.c:372:equal_file:PASS
basic-test-suite.c:375:sum_file:PASS
basic-test-suite.c:376:scalar_product_file:PASS
[1]    15440 segmentation fault  ./basic-test-suite
┌─pierre@pop ~/Desktop/comp26020/lab/lab1/code ‹master●›
└─$
```

Memory Unsafety in C Programs

- C, and other languages e.g. C++ **are inherently memory unsafe**
- There is absolutely no check at runtime if memory accessed is initialised, allocated, or mapped
- In most cases the compiler won't warn you either

Memory Unsafety in C Programs

- C, and other languages e.g. C++ **are inherently memory unsafe**
- There is absolutely no check at runtime if memory accessed is initialised, allocated, or mapped
- In most cases the compiler won't warn you either
- **A few examples of memory errors:**
  - Out of bounds accesses on buffers/arrays accesses

Memory Unsafety in C Programs

- C, and other languages e.g. C++ **are inherently memory unsafe**
- There is absolutely no check at runtime if memory accessed is initialised, allocated, or mapped
- In most cases the compiler won't warn you either
- **A few examples of memory errors:**
  - Out of bounds accesses on buffers/arrays accesses
  - Failure to initialise stack/heap-allocated variables before read access

Memory Unsafety in C Programs

- C, and other languages e.g. C++ **are inherently memory unsafe**
- There is absolutely no check at runtime if memory accessed is initialised, allocated, or mapped
- In most cases the compiler won't warn you either
- **A few examples of memory errors:**
  - Out of bounds accesses on buffers/arrays accesses
  - Failure to initialise stack/heap-allocated variables before read access
  - Leaks, double free, use-after-free

Memory Unsafety in C Programs

- C, and other languages e.g. C++ **are inherently memory unsafe**
- There is absolutely no check at runtime if memory accessed is initialised, allocated, or mapped
- In most cases the compiler won't warn you either
- **A few examples of memory errors:**
  - Out of bounds accesses on buffers/arrays accesses
  - Failure to initialise stack/heap-allocated variables before read access
  - Leaks, double free, use-after-free
- Memory errors are **hard to detect, hard to debug**
  - No warning/error at compile-time
  - Undefined behaviour: at runtime sometimes it crashes, sometimes not...

Memory Unsafety in C Programs

- C, and other languages e.g. C++ **are inherently memory unsafe**
- There is absolutely no check at runtime if memory accessed is initialised, allocated, or mapped
- In most cases the compiler won't warn you either
- **A few examples of memory errors:**
  - Out of bounds accesses on buffers/arrays accesses
  - Failure to initialise stack/heap-allocated variables before read access
  - Leaks, double free, use-after-free
- Memory errors are **hard to detect, hard to debug**
  - No warning/error at compile-time
  - Undefined behaviour: at runtime sometimes it crashes, sometimes not...

> **These bugs have huge security implications!**

Microsoft: 70 percent of all security bugs are memory safety issues

Percentage of memory safety issues has been hovering at 70 percent for the past 12 years.

By Catalin Cimpanu for Zero Day | February 11, 2019 | Topic: Security

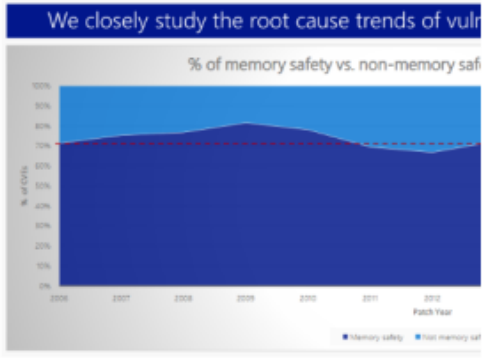


Image: Matt Miller

Chrome: 70% of all security bugs are memory safety issues

Google software engineers are looking into ways of eliminating memory management-related bugs from Chrome.

By Catalin Cimpanu for Zero Day | May 23, 2020 | Topic: Security

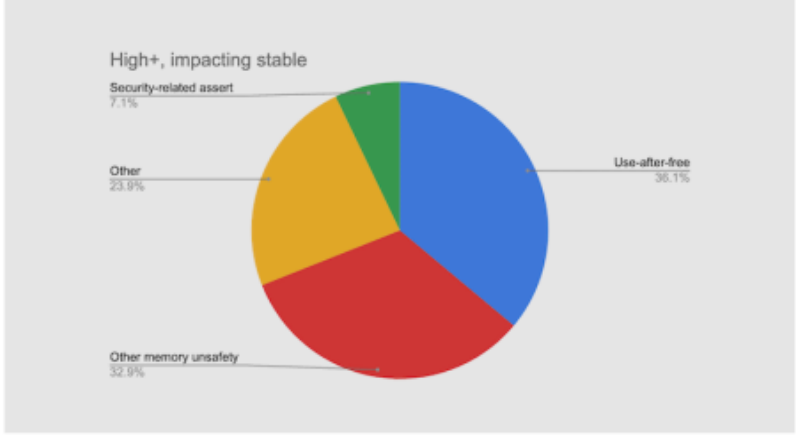


Image: Google

RELATED


Apple: Side-loading on iOS would open the malware floodgates
Security


Work to earn several highly respected CompTIA certifications with these self-paced courses
Security


US indicts UK resident 'PlugwalkJoe' for cryptocurrency theft
Security

NEWSLETTERS

ZDNet Security

Your weekly update on security around the globe, featuring research, threats, and more.

Sources: https://zd.net/3q8axgo, https://zd.net/3wfFHU7

Example 1: Infoleak

Example 1: Infoleak

Original code:

```c
char *welcome_message = "Hi there! How is it going?\n"; // 27 char
char *password = "secret";
char entered_password[128];

int main(int argc, char **argv) {
    for(int i=0; i<27; i++)  // Print welcome message character by character
        printf("%c", welcome_message[i]);

    printf("Please input the password:\n");
    scanf("%s", entered_password);

    if(!strcmp(entered_password, password)) { printf("Passowrd ok!\n"); /* ... */ }
    else { printf("Wrong password! aborting\n"); }

    return 0;
}
```

23-memory-safety/infoleak-orig.c

Example 1: Infoleak

Updated code (`welcome_message` shortened):

```
char *welcome_message = "Hi there!\n"; // shortened welcome message, only 11 chars now
char *password = "secret";
char entered_password[128];

int main(int argc, char **argv) {
    for(int i=0; i<27; i++)
        printf("%c", welcome_message[i]);

    printf("Please input the password:\n");
    scanf("%s", entered_password);

    if(!strcmp(entered_password, password)) { printf("Passowrd ok!\n"); /* ... */ }
    else { printf("Wrong password! aborting\n"); }

    return 0;
}
```

23-memory-safety/infoleak-updated.c

Example 1: Infoleak

Updated code (`welcome_message` shortened):

```c
char *welcome_message = "Hi there!\n"; // shortened welcome message, only 11 chars now
char *password = "secret";
char entered_password[128];

int main(int argc, char **argv) {
    for(int i=0; i<27; i++)  // Oopsie! forgot to update that bit of the code
        printf("%c", welcome_message[i]);

    printf("Please input the password:\n");
    scanf("%s", entered_password);

    if(!strcmp(entered_password, password)) { printf("Passowrd ok!\n"); /* ... */ }
    else { printf("Wrong password! aborting\n"); }

    return 0;
}
```

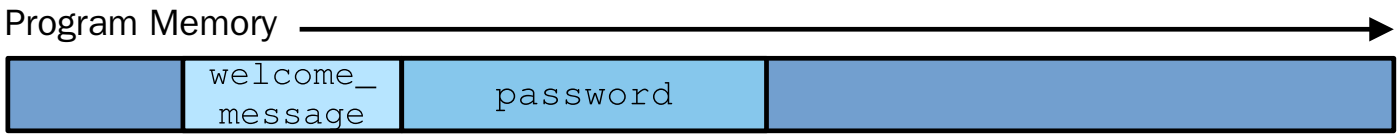
23-memory-safety/infoleak-updated.c

Example 1: Infoleak

Updated code (`welcome_message` shortened):

```
char *welcome_message = "Hi there!\n"; // shortened welcome message, only 11 chars now
char *password = "secret";
char entered_password[128];

int main(int argc, char **argv) {
    for(int i=0; i<27; i++)  // Oopsie! forgot to update that bit of the code
        printf("%c", welcome_message[i]);

    printf("Please input the password:\n");
    scanf("%s", entered_password);

    if(!strcmp(entered_password, password)) { printf("Passowrd ok!\n"); /* ... */ }
    else { printf("Wrong password! aborting\n"); }

    return 0;
}
```

23-memory-safety/infoleak-updated.c

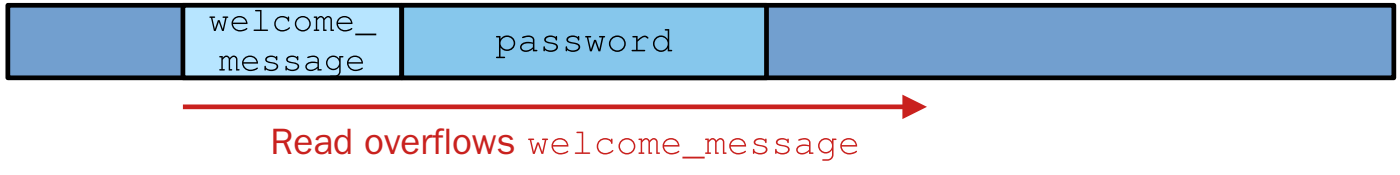


Example 1: Infoleak

Updated code (`welcome_message` shortened):

```
char *welcome_message = "Hi there!\n"; // shortened welcome message, only 11 chars now
char *password = "secret";
char entered_password[128];

int main(int argc, char **argv) {
    for(int i=0; i<27; i++)  // Oopsie! forgot to update that bit of the code
        printf("%c", welcome_message[i]);

    printf("Please input the password:\n");
    scanf("%s", entered_password);

    if(!strcmp(entered_password, password)) { printf("Passowrd ok!\n"); /* ... */ }
    else { printf("Wrong password! aborting\n"); }

    return 0;
}
```

23-memory-safety/infoleak-updated.c

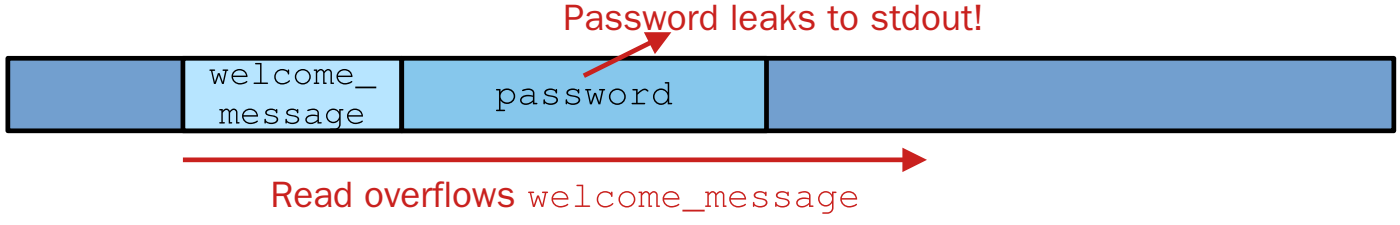


Example 1: Infoleak

Updated code (`welcome_message` shortened):

```c
char *welcome_message = "Hi there!\n"; // shortened welcome message, only 11 chars now
char *password = "secret";
char entered_password[128];

int main(int argc, char **argv) {
    for(int i=0; i<27; i++)  // Oopsie! forgot to update that bit of the code
        printf("%c", welcome_message[i]);

    printf("Please input the password:\n");
    scanf("%s", entered_password);

    if(!strcmp(entered_password, password)) { printf("Passowrd ok!\n"); /* ... */ }
    else { printf("Wrong password! aborting\n"); }

    return 0;
}
```

23-memory-safety/infoleak-updated.c



Example 2: Sensitive Data Tampering

Example 2: Sensitive Data Tampering

```c
char user_input[32] = "0000000000";
char password[32] = "secret";

int main(int argc, char **argv) {

    if(argc != 2) { printf("Usage: %s <password>\n", argv[0]); return 0; }

    strcpy(user_input, argv[1]);

    if(!strncmp(password, user_input, strlen(password))) {
        printf("login success!\n");
        /* do important stuff  ... */
    } else {
        printf("wrong password!\n");
    }

    return 0;
}
```

23-memory-safety/tampering.c

Example 2: Sensitive Data Tampering

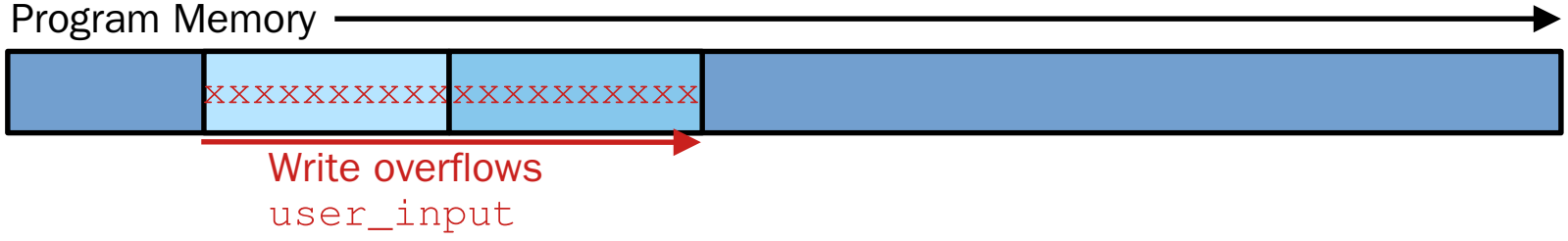
```c
char user_input[32] = "0000000000";
char password[32] = "secret";

int main(int argc, char **argv) {

    if(argc != 2) { printf("Usage: %s <password>\n", argv[0]); return 0; }

    strcpy(user_input, argv[1]);

    if(!strncmp(password, user_input, strlen(password))) {
        printf("login success!\n");
        /* do important stuff  ... */
    } else {
        printf("wrong password!\n");
    }

    return 0;
}
```

23-memory-safety/tampering.c

Example 2: Sensitive Data Tampering

```c
char user_input[32] = "0000000000";
char password[32] = "secret";

int main(int argc, char **argv) {

    if(argc != 2) { printf("Usage: %s <password>\n", argv[0]); return 0; }

    strcpy(user_input, argv[1]);

    if(!strncmp(password, user_input, strlen(password))) {
        printf("login success!\n");
        /* do important stuff  ... */
    } else {
        printf("wrong password!\n");
    }

    return 0;
}
```

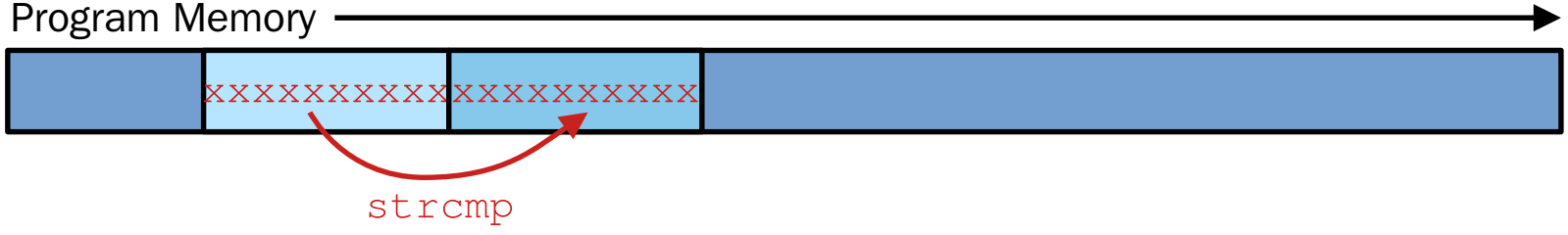
23-memory-safety/tampering.c



Example 2: Sensitive Data Tampering

```c
char user_input[32] = "0000000000";
char password[32] = "secret";

int main(int argc, char **argv) {

    if(argc != 2) { printf("Usage: %s <password>\n", argv[0]); return 0; }

    strcpy(user_input, argv[1]);

    if(!strncmp(password, user_input, strlen(password))) {
        printf("login success!\n");
        /* do important stuff  ... */
    } else {
        printf("wrong password!\n");
    }

    return 0;
}
```

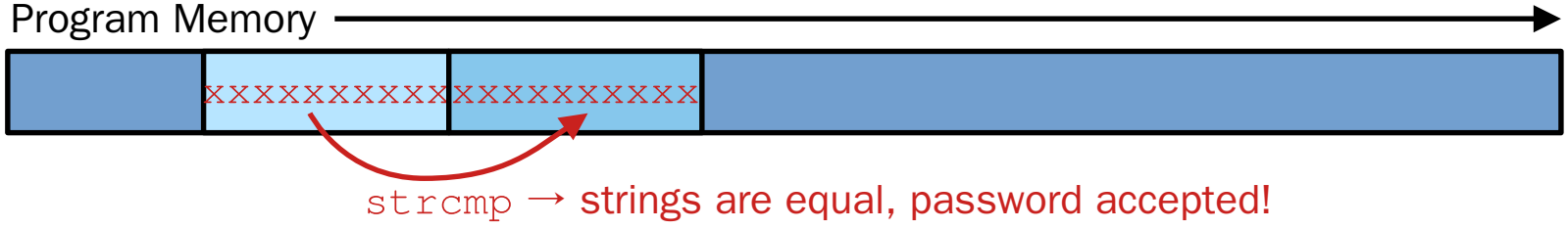
23-memory-safety/tampering.c



Example 2: Sensitive Data Tampering

```c
char user_input[32] = "0000000000";
char password[32] = "secret";

int main(int argc, char **argv) {

    if(argc != 2) { printf("Usage: %s <password>\n", argv[0]); return 0; }

    strcpy(user_input, argv[1]);

    if(!strncmp(password, user_input, strlen(password))) {
        printf("login success!\n");
        /* do important stuff  ... */
    } else {
        printf("wrong password!\n");
    }

    return 0;
}
```

23-memory-safety/tampering.c



Example 3: Stack Smashing

Example 3: Stack Smashing

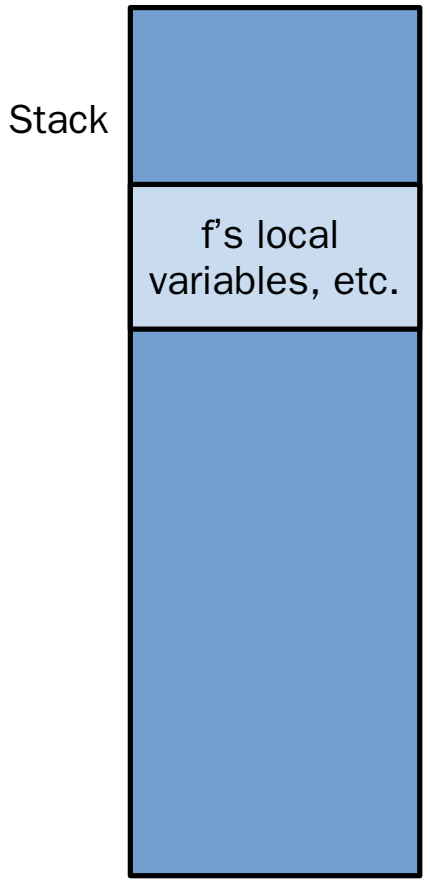
- Classic control flow hijacking attack
- Originally proposed in 1996 here: http://www.phrack.org/archives/issues/49/14.txt
- First let's see how the CPU manage function calls/returns

Example 3: Stack Smashing

C CODE

```
int f(double param)
{
    /* ... */
    g(x);
    /* ... */
}

void g(int param) {
    /* ... */
    return;
}
```

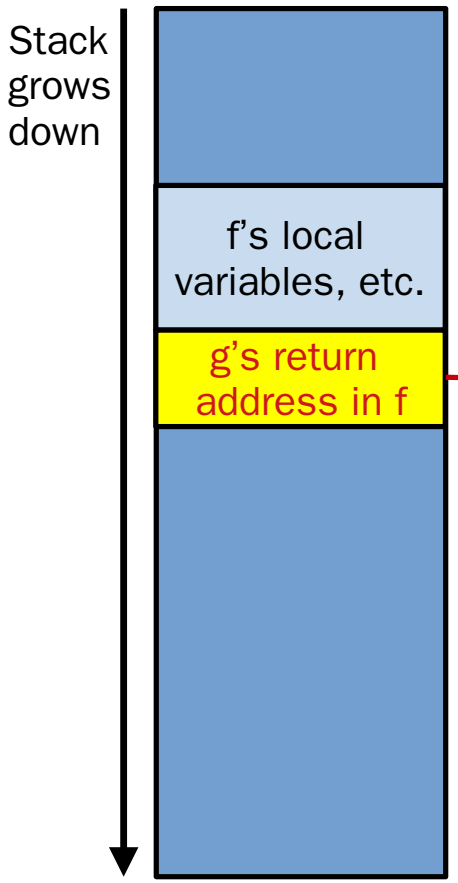
Example 3: Stack Smashing



C CODE

```
int f(double param)
{
    /* ... */
    g(x);
    /* ... */
}

void g(int param) {
    /* ... */
    return;
}
```

Example 3: Stack Smashing



C CODE

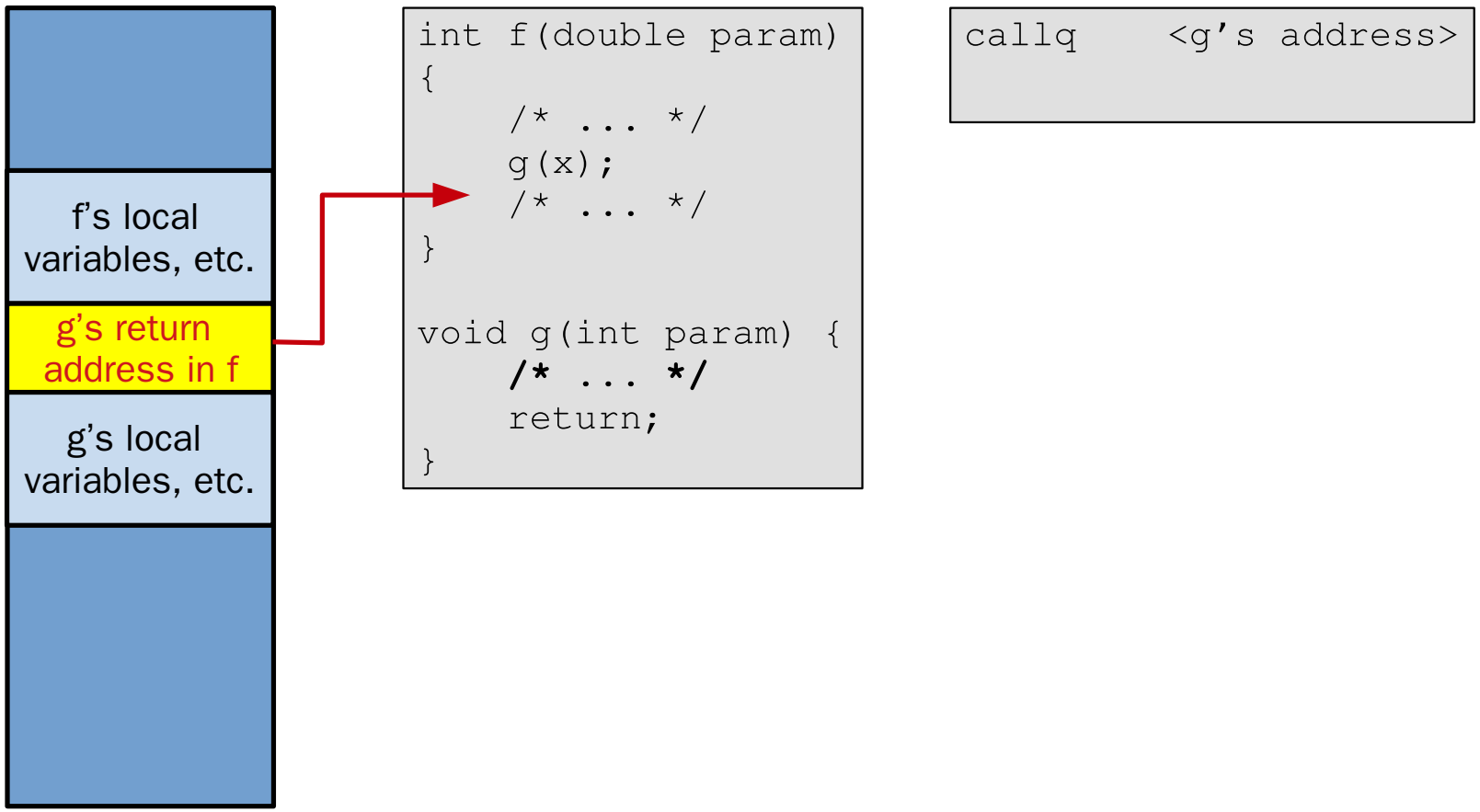
```
int f(double param)
{
    /* ... */
    g(x);
    /* ... */
}

void g(int param) {
    /* ... */
    return;
}
```

MACHINE INSTRUCTIONS

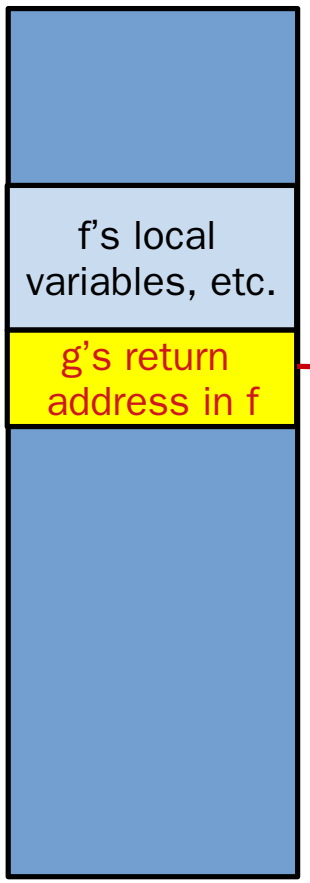
```
callq    <g's address>
```

Push return address on the stack and jump to g()

Example 3: Stack Smashing



Example 3: Stack Smashing



C CODE

```
int f(double param)
{
    /* ... */
    g(x);
    /* ... */
}

void g(int param) {
    /* ... */
    return;
}
```

MACHINE INSTRUCTIONS

```
callq    <g's address>
```

```
ret
```

CPU pops return address from the stack and jumps there

Example 3: Stack Smashing

```c
char *password = "super-secret-password";

void security_critical_function() { printf("launching nukes!!\n"); }

void preprocess_input(char *string) {
    char local_buffer[16];
    strcpy(local_buffer, string);
    /* work on local buffer ... */
    return;
}

int main(int argc, char **argv) {
    if (argc != 2) {
        printf("usage: %s <password>\n", argv[0]);
        return -1;
    }

   preprocess_input(argv[1]);

    if(!strncmp(password, argv[1], strlen(password)))
        security_critical_function();
    else printf("Unauthorised user!\n");

    return 0;
}
```

23-memory-safety/stack-smashing.c

Example 3: Stack Smashing

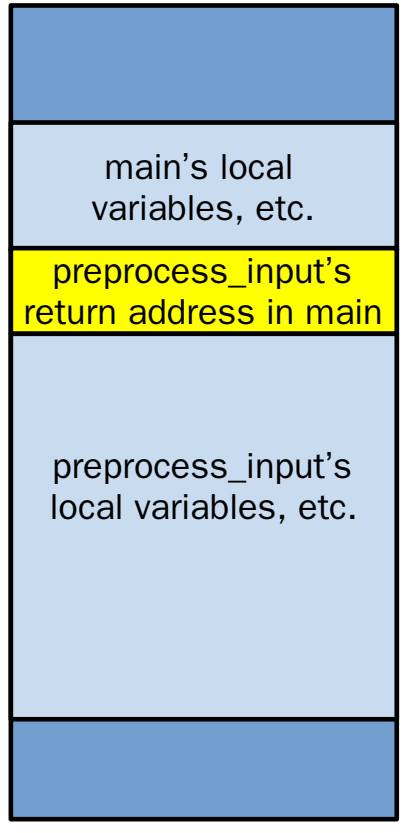
```c
char *password = "super-secret-password";

void security_critical_function() {
    printf("launching nukes!!\n");
}

void preprocess_input(char *string) {
    char local_buffer[16];
    strcpy(local_buffer, string);
    /* work on local buffer ... */
}

int main(int argc, char **argv) {
    if (argc != 2) { /* ... */ }

    preprocess_input(argv[1]);

    if(!strncmp(password, argv[1],
            strlen(password)))
        security_critical_function();
    else
        printf("Unauthorised user!\n");

    return 0;
}
```

23-memory-safety/stack-smashing.c

Example 3: Stack Smashing

```c
char *password = "super-secret-password";

void security_critical_function() {
    printf("launching nukes!!\n");
}

void preprocess_input(char *string) {
    char local_buffer[16];
    strcpy(local_buffer, string);
    /* work on local buffer ... */
}

int main(int argc, char **argv) {
    if (argc != 2) { /* ... */ }

    preprocess_input(argv[1]);

    if(!strncmp(password, argv[1],
            strlen(password)))
        security_critical_function();
    else
        printf("Unauthorised user!\n");

    return 0;
}
```

23-memory-safety/stack-smashing.c

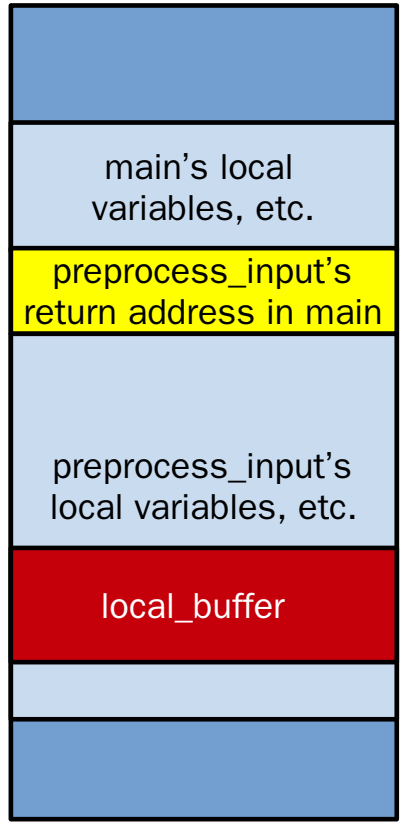


Stack layout in memory when preprocess_input runs

Example 3: Stack Smashing

```c
char *password = "super-secret-password";

void security_critical_function() {
    printf("launching nukes!!\n");
}

void preprocess_input(char *string) {
    char local_buffer[16];
    strcpy(local_buffer, string);
    /* work on local buffer ... */
}

int main(int argc, char **argv) {
    if (argc != 2) { /* ... */ }

    preprocess_input(argv[1]);

    if(!strncmp(password, argv[1],
            strlen(password)))
        security_critical_function();
    else
        printf("Unauthorised user!\n");

    return 0;
}
```

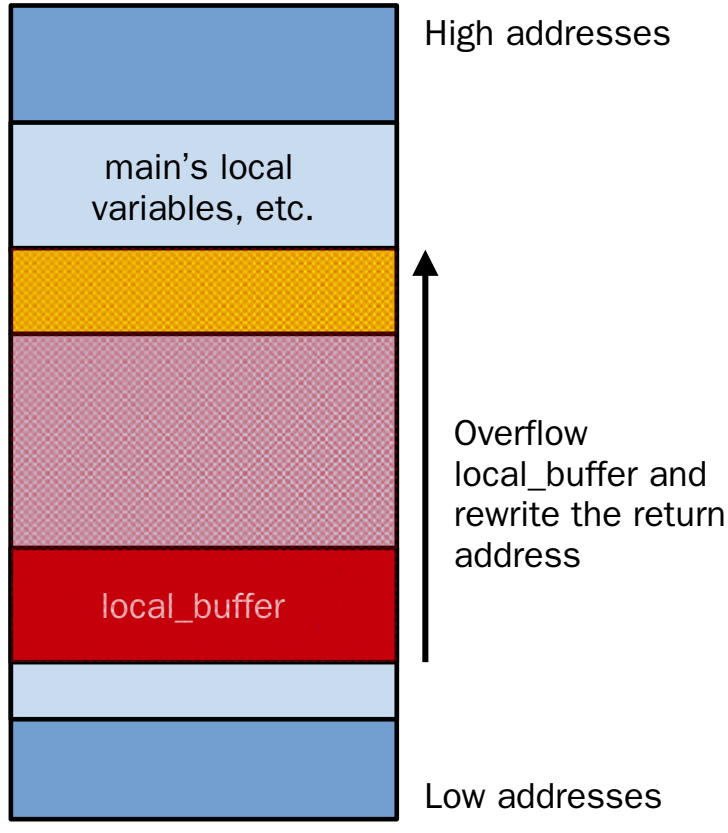
23-memory-safety/stack-smashing.c



Example 3: Stack Smashing

```c
char *password = "super-secret-password";

void security_critical_function() {
    printf("launching nukes!!\n");
}

void preprocess_input(char *string) {
    char local_buffer[16];
    strcpy(local_buffer, string);
    /* work on local buffer ... */
}

int main(int argc, char **argv) {
    if (argc != 2) { /* ... */ }

    preprocess_input(argv[1]);

    if(!strncmp(password, argv[1],
            strlen(password)))
        security_critical_function();
    else
        printf("Unauthorised user!\n");

    return 0;
}
```

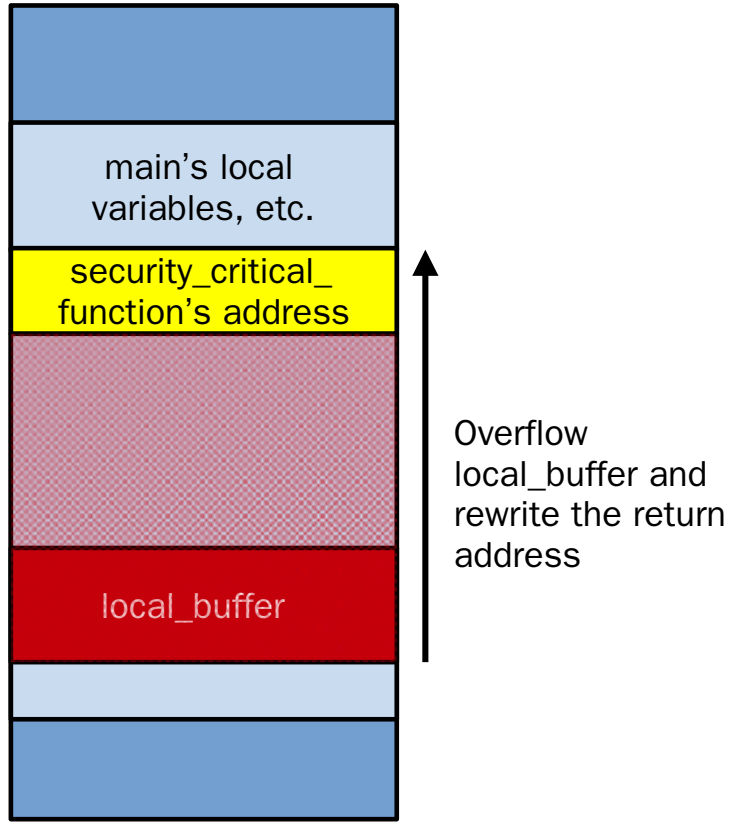
23-memory-safety/stack-smashing.c



Example 3: Stack Smashing

```c
char *password = "super-secret-password";

void security_critical_function() {
    printf("launching nukes!!\n");
}

void preprocess_input(char *string) {
    char local_buffer[16];
    strcpy(local_buffer, string);
    /* work on local buffer ... */
}

int main(int argc, char **argv) {
    if (argc != 2) { /* ... */ }

    preprocess_input(argv[1]);

    if(!strncmp(password, argv[1],
            strlen(password)))
        security_critical_function();
    else
        printf("Unauthorised user!\n");

    return 0;
}
```

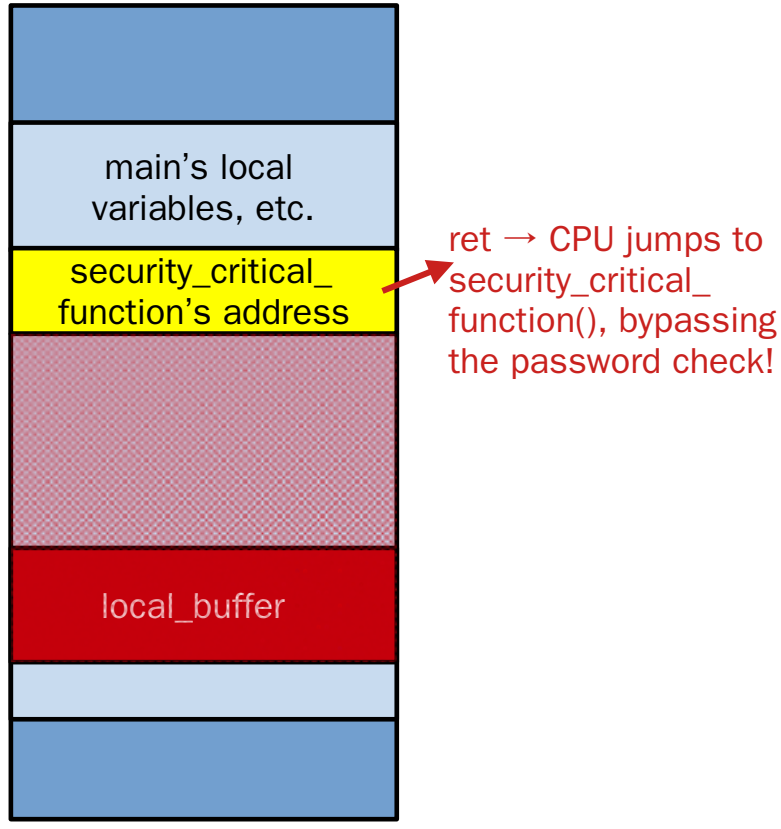
23-memory-safety/stack-smashing.c



Example 3: Stack Smashing

```c
char *password = "super-secret-password";

void security_critical_function() {
    printf("launching nukes!!\n");
}

void preprocess_input(char *string) {
    char local_buffer[16];
    strcpy(local_buffer, string);
    /* work on local buffer ... */
}

int main(int argc, char **argv) {
    if (argc != 2) { /* ... */ }

    preprocess_input(argv[1]);

    if(!strncmp(password, argv[1],
            strlen(password)))
        security_critical_function();
    else
        printf("Unauthorised user!\n");

    return 0;
}
```

23-memory-safety/stack-smashing.c



Example 4: Use-After-Free

Example 4: Use-After-Free

```c
typedef struct {
    double member1; double member2;
    void (*member3)(int);
} my_struct;

void print_hello(int x) {
    printf("Hello, parameter: %d\n", x);
}

void security_critical_function() {
    printf("Launching nukes!\n");
    /* ... */
}

//
```

```c
int main(int argc, char **argv) {
    /* allocate and init ms */
    my_struct *ms = malloc(sizeof(my_struct));
    ms->member1 = 42.0; ms->member2 = 42.0;
    ms->member3 = &print_hello;
    /* call the function pointer */
    ms->member3(12);

    free(ms);
    char *buffer = malloc(12);
    strcpy(buffer, argv[1]);

    ms->member3(12);
    /* check a password, runs sec_crit_fn */
}
```

23-memory-safety/stack-smashing.c

Example 4: Use-After-Free

```c
typedef struct {
    double member1; double member2;
    void (*member3)(int);
} my_struct;

void print_hello(int x) {
    printf("Hello, parameter: %d\n", x);
}

void security_critical_function() {
    printf("Launching nukes!\n");
    /* ... */
}

//
```

```c
int main(int argc, char **argv) {
    /* allocate and init ms */
    my_struct *ms = malloc(sizeof(my_struct));
    ms->member1 = 42.0; ms->member2 = 42.0;
    ms->member3 = &print_hello;
    /* call the function pointer */
    ms->member3(12);

    free(ms);
    char *buffer = malloc(12);
    strcpy(buffer, argv[1]);

    ms->member3(12);
    exit(0);
}
```

23-memory-safety/stack-smashing.c

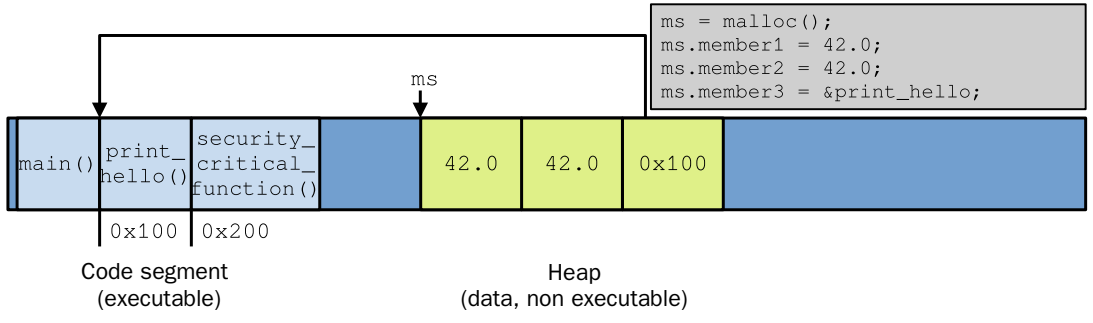
- `member2` is a **function pointer**
  - Can be dynamically set and called at runtime

Example 4: Use-After-Free

```c
typedef struct {
    double member1; double member2;
    void (*member3)(int);
} my_struct;

void print_hello(int x) {
    printf("Hello, parameter: %d\n", x);
}

void security_critical_function() {
    printf("Launching nukes!\n");
    /* ... */
}

//
```

```c
int main(int argc, char **argv) {
    /* allocate and init ms */
    my_struct *ms = malloc(sizeof(my_struct));
    ms->member1 = 42.0; ms->member2 = 42.0;
    ms->member3 = &print_hello;
    /* call the function pointer */
    ms->member3(12);

    free(ms);
    char *buffer = malloc(12);
    strcpy(buffer, argv[1]);

    ms->member3(12);
    /* check a password, runs sec_crit_fn */
}
```

23-memory-safety/stack-smashing.c

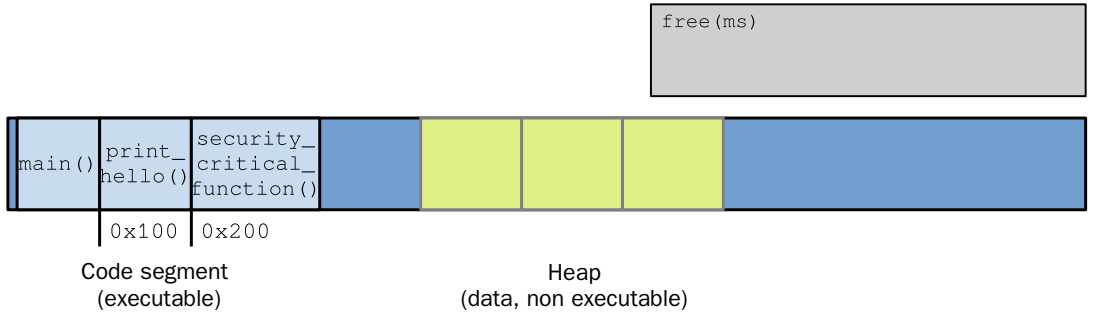


Example 4: Use-After-Free

```c
typedef struct {
    double member1; double member2;
    void (*member3)(int);
} my_struct;

void print_hello(int x) {
    printf("Hello, parameter: %d\n", x);
}

void security_critical_function() {
    printf("Launching nukes!\n");
    /* ... */
}

//
```

```c
int main(int argc, char **argv) {
    /* allocate and init ms */
    my_struct *ms = malloc(sizeof(my_struct));
    ms->member1 = 42.0; ms->member2 = 42.0;
    ms->member3 = &print_hello;
    /* call the function pointer */
    ms->member3(12);

    free(ms);
    char *buffer = malloc(12);
    strcpy(buffer, argv[1]);

    ms->member3(12);
    /* check a password, runs sec_crit_fn */
}
```

23-memory-safety/stack-smashing.c

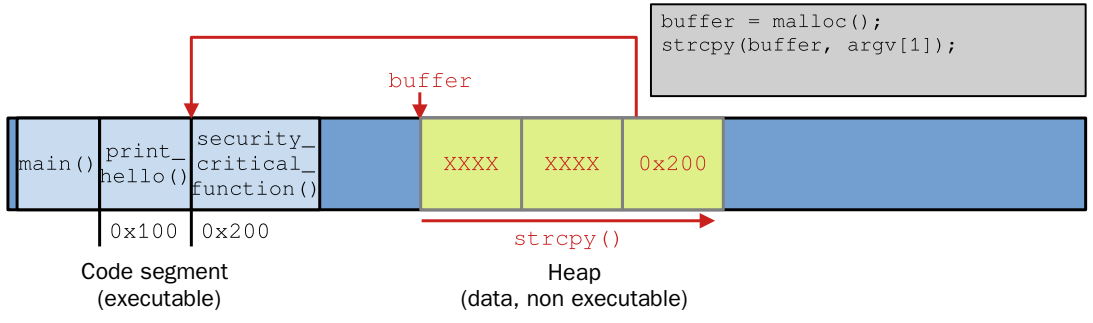


Example 4: Use-After-Free

```c
typedef struct {
    double member1; double member2;
    void (*member3)(int);
} my_struct;

void print_hello(int x) {
    printf("Hello, parameter: %d\n", x);
}

void security_critical_function() {
    printf("Launching nukes!\n");
    /* ... */
}

//
```

```c
int main(int argc, char **argv) {
    /* allocate and init ms */
    my_struct *ms = malloc(sizeof(my_struct));
    ms->member1 = 42.0; ms->member2 = 42.0;
    ms->member3 = &print_hello;
    /* call the function pointer */
    ms->member3(12);

    free(ms);
    char *buffer = malloc(12);
    strcpy(buffer, argv[1]);

    ms->member3(12);
    /* check a password, runs sec_crit_fn */
}
```

23-memory-safety/stack-smashing.c

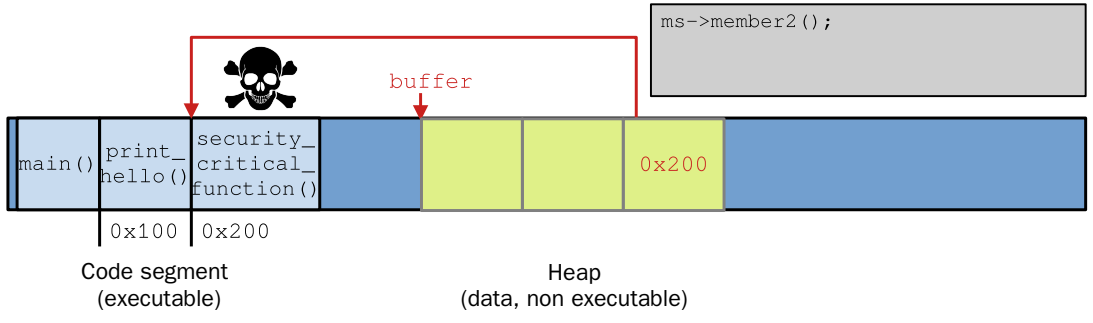


Example 4: Use-After-Free

```c
typedef struct {
    double member1; double member2;
    void (*member3)(int);
} my_struct;

void print_hello(int x) {
    printf("Hello, parameter: %d\n", x);
}

void security_critical_function() {
    printf("Launching nukes!\n");
    /* ... */
}

//
```

```c
int main(int argc, char **argv) {
    /* allocate and init ms */
    my_struct *ms = malloc(sizeof(my_struct));
    ms->member1 = 42.0; ms->member2 = 42.0;
    ms->member3 = &print_hello;
    /* call the function pointer */
    ms->member3(12);

    free(ms);
    char *buffer = malloc(12);
    strcpy(buffer, argv[1]);

    ms->member3(12);
    /* check a password, runs sec_crit_fn */
}
```

23-memory-safety/stack-smashing.c



Summary

- C is **not memory safe**
- Memory issues benign at a first glance can have **huge security consequences**
- How to avoid these?

Feedback form: https://bit.ly/3iybv0Y