



Security Vulnerabilities - Crash course

Holistic Software Security

Aravind Machiry



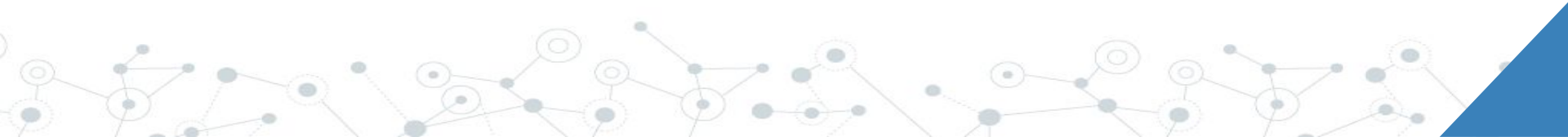
Stack Based Buffer Overflow

Stack Buffer Overflow

example2.c

```
void function(char *str) {  
    char buffer[16];  
  
    strcpy(buffer, str);  
}
```

```
void main() {  
    char large_string[256];  
    int i;  
  
    for( i = 0; i < 255; i++)  
        large_string[i] = 'A';  
  
    function(large_string);  
}
```



Stack Buffer Overflow

example2.c

```
-----  
void function(char *str) {  
    char buffer[16];  
  
    strcpy(buffer, str);  
}
```

```
void main() {  
    char large_string[256];  
    int i;  
  
    for( i = 0; i < 255; i++)  
        large_string[i] = 'A';  
  
    function(large_string);  
}
```

bottom of
memory

top of
memory

<----- buffer sfp ret *str
 [][][][]

top of
stack

bottom of
stack



Stack Buffer Overflow

example2.c

```
-----  
void function(char *str) {  
    char buffer[16];  
  
    strcpy(buffer, str);  
}
```

```
void main() {  
    char large_string[256];  
    int i;  
  
    for( i = 0; i < 255; i++)  
        large_string[i] = 'A';  
  
    function(large_string);  
}
```

bottom of
memory

top of
memory

<----- buffer sfp ret *str
 [AAAAAAAAAAAAAAAA] [] [] []

top of
stack

bottom of
stack

Stack Buffer Overflow

example2.c

```
-----  
void function(char *str) {  
    char buffer[16];  
  
    strcpy(buffer, str);  
}
```

```
void main() {  
    char large_string[256];  
    int i;  
  
    for( i = 0; i < 255; i++)  
        large_string[i] = 'A';  
  
    function(large_string);  
}
```

bottom of
memory

top of
memory

buffer sfp ret *str
[AAAAAAAAAAAAAAAAAAAA] [AAAA] [AAAA] [AAAA] AAAAAAAAAAAAAAAAAAAAAA/

top of
stack

bottom of
stack



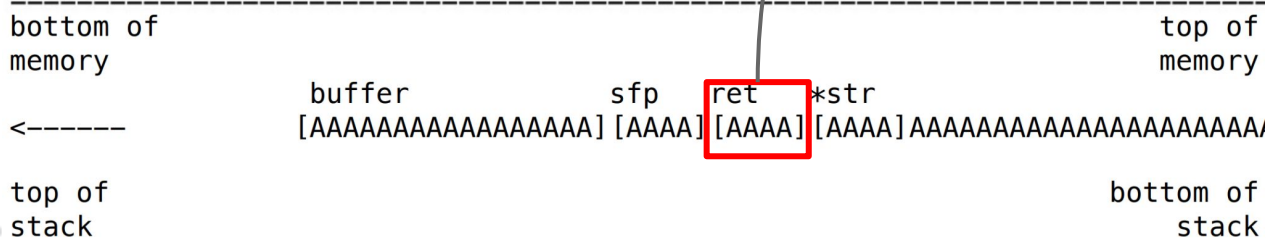
Stack Buffer Overflow

example2.c

```
void function(char *str) {  
    char buffer[16];  
  
    strcpy(buffer, str);  
}
```

```
void main() {  
    char large_string[256];  
    int i;  
  
    for( i = 0; i < 255; i++)  
        large_string[i] = 'A';  
  
    function(large_string);  
}
```

We can control where function returns, i.e., control the execution of the program.



Stack Buffer Overflow

example2.c

```
void function(char *str) {  
    char buffer[16];  
  
    strcpy(buffer, str);  
}
```

```
void main() {  
    char large_string[256];  
    int i;  
  
    for( i = 0; i < 255; i++)  
        large_string[i] = 'A';  
  
    function(large_string);  
}
```

We can make return address point to the data we just provided (SSSS..), i.e., make the program execute our code.

bottom of memory	DDDDDDDD	EEEE	FFFF	FFFF	FFFF	FFFF	top of memory
	89ABCDEF0123456789AB	CDEF	0123	4567	89AB	CDEF	
	buffer	sfp	ret	a	b	c	

<----- [SSSSSSSSSSSSSSSSSSSSSS] [SSSS] [0xD8] [0x01] [0x02] [0x03]

top of stack

bottom of stack

Integer Overflow

```
int get_two_vars(int sock, char *out, int len){
    char buf1[512], buf2[512];
    unsigned int size1, size2;
    int size;

    if(recv(sock, buf1, sizeof(buf1), 0) < 0){
        return -1;
    }
    if(recv(sock, buf2, sizeof(buf2), 0) < 0){
        return -1;
    }

    /* packet begins with length information */
    memcpy(&size1, buf1, sizeof(int));
    memcpy(&size2, buf2, sizeof(int));

    size = size1 + size2;          /* [1] */

    if(size > len){               /* [2] */
        return -1;
    }

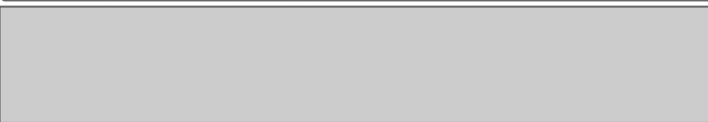
    memcpy(out, buf1, size1);
    memcpy(out + size1, buf2, size2);

    return size;
}
```

Integer Overflow

```
int get_two_vars(int sock, char *out, int len){  
    char buf1[512], buf2[512];  
    unsigned int size1, size2;  
    int size;
```

```
    if(recv(sock, buf1, sizeof(buf1), 0) < 0){  
        return -1;  
    }  
    if(recv(sock, buf2, sizeof(buf2), 0) < 0){  
        return -1;  
    }
```



```
    return size;
```

```
}
```

Read 2 pieces of data from socket.

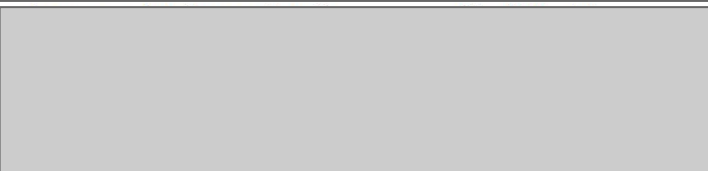


Integer Overflow

```
int get_two_vars(int sock, char *out, int len){
    char buf1[512], buf2[512];
    unsigned int size1, size2;
    int size;

    if(recv(sock, buf1, sizeof(buf1), 0) < 0){
        return -1;
    }
    if(recv(sock, buf2, sizeof(buf2), 0) < 0){
        return -1;
    }
```

```
/* packet begins with length information */
memcpy(&size1, buf1, sizeof(int));
memcpy(&size2, buf2, sizeof(int));
```



```
return size;
```

```
}
```

Convert first 4 bytes into integers.

Integer Overflow

```
int get_two_vars(int sock, char *out, int len){
    char buf1[512], buf2[512];
    unsigned int size1, size2;
    int size;

    if(recv(sock, buf1, sizeof(buf1), 0) < 0){
        return -1;
    }
    if(recv(sock, buf2, sizeof(buf2), 0) < 0){
        return -1;
    }

    /* packet begins with length information */
    memcpy(&size1, buf1, sizeof(int));
    memcpy(&size2, buf2, sizeof(int));

    size = size1 + size2;          /* [1] */

    if(size > len){               /* [2] */
        return -1;
    }

    return size;
}
```

Add both the numbers and check that sum is less than len.

Integer Overflow

```
int get_two_vars(int sock, char *out, int len){
    char buf1[512], buf2[512];
    unsigned int size1, size2;
    int size;

    if(recv(sock, buf1, sizeof(buf1), 0) < 0){
        return -1;
    }
    if(recv(sock, buf2, sizeof(buf2), 0) < 0){
        return -1;
    }

    /* packet begins with length information */
    memcpy(&size1, buf1, sizeof(int));
    memcpy(&size2, buf2, sizeof(int));

    size = size1 + size2;          /* [1] */

    if(size > len){               /* [2] */
        return -1;
    }

    memcpy(out, buf1, size1);
    memcpy(out + size1, buf2, size2);

    return size;
}
```

Copy the data into out buffer.

What's wrong?

```
int get_two_vars(int sock, char *out, int len){
    char buf1[512], buf2[512];
    unsigned int size1, size2;
    int size;

    if(recv(sock, buf1, sizeof(buf1), 0) < 0){
        return -1;
    }
    if(recv(sock, buf2, sizeof(buf2), 0) < 0){
        return -1;
    }

    /* packet begins with length information */
    memcpy(&size1, buf1, sizeof(int));
    memcpy(&size2, buf2, sizeof(int));

    size = size1 + size2;          /* [1] */

    if(size > len){                /* [2] */
        return -1;
    }

    return size;
}
```

Say len = 16

What happens when:
size1 = 0x7fffffff
size2 = 0x7fffffff

What's wrong? Integer Overflow

```
int get_two_vars(int sock, char *out, int len){
    char buf1[512], buf2[512];
    unsigned int size1, size2;
    int size;

    if(recv(sock, buf1, sizeof(buf1), 0) < 0){
        return -1;
    }
    if(recv(sock, buf2, sizeof(buf2), 0) < 0){
        return -1;
    }

    /* packet begins with length information */
    memcpy(&size1, buf1, sizeof(int));
    memcpy(&size2, buf2, sizeof(int));

    size = size1 + size2;          /* [1] */

    if(size > len){                /* [2] */
        return -1;
    }

    return size;
}
```

Say len = 16

What happens when:

size1 = 0x7fffffff

size2 = 0x7fffffff

size = (0x7fffffff + 0x7fffffff =
0xffffffffe (-2)) and (-2 < 16)

Integer Overflow

```
int get_two_vars(int sock, char *out, int len){
    char buf1[512], buf2[512];
    unsigned int size1, size2;
    int size;

    if(recv(sock, buf1, sizeof(buf1), 0) < 0){
        return -1;
    }
    if(recv(sock, buf2, sizeof(buf2), 0) < 0){
        return -1;
    }

    /* packet begins with length information */
    memcpy(&size1, buf1, sizeof(int));
    memcpy(&size2, buf2, sizeof(int));

    size = size1 + size2;          /* [1] */

    if(size > len){               /* [2] */
        return -1;
    }

    memcpy(out, buf1, size1);
    memcpy(out + size1, buf2, size2);

    return size;
}
```

Say len = 16

Overflowing the buffer pointed by out

We are copying 2*0x7fffffff bytes into out whose size is just 16.

Integer Overflow

```
int get_two_vars(int sock, char *out, int len){
    char buf1[512], buf2[512];
    unsigned int size1, size2;
    int size;

    if(recv(sock, buf1, sizeof(buf1), 0) < 0){
        return -1;
    }
    if(recv(sock, buf2, sizeof(buf2), 0) < 0){
        return -1;
    }

    /* packet begins with length information */
    memcpy(&size1, buf1, sizeof(int));
    memcpy(&size2, buf2, sizeof(int));

    size = size1 + size2;          /* [1] */

    if(size > len){               /* [2] */
        return -1;
    }

    memcpy(out, buf1, size1);
    memcpy(out + size1, buf2, size2);

    return size;
}
```

Say len = 16

Overflowing the buffer pointed by out


What if out is on stack? **Stack based buffer-overflow!!**

We are copying 2*0x7fffffff bytes into out whose size is just 16.



Temporal Vulnerabilities

(use-after-free, double free)



Race Condition to Memory Corruption



References

Basics: <https://www.youtube.com/watch?v=VroEiMOJPm8>

Wargames: <https://overthewire.org/wargames/>, <https://pwnable.kr/>

Temporal vulnerabilities exploitation: <https://github.com/shellphish/how2heap>