#ret #hw

---

1 | User Input Validation

```
title: Assignment

For this assignment, show your understanding of at least one of these techniques by submitting:
-   code and notes demonstrating a successful break on a system meant for testing -- **_not on a produc
-   code and notes that demonstrate changes that would prevent your break
-   the name of a specific law you might be charged with if you were to do this on a system without perm
```

1.1 | Python 2 script

Suppose someone had a very simple Python 2 script that they ran on their computer:

```
# original script
favorite = input('What is your favorite number? ') # vulnerability in input
print 'I like the number {}, too!'.format(favorite)
```

Show examples of input that would give you access to information that user had access to (e.g. the contents of a file on their machine).

```
> python2 bad_input.py
What is your favorite number? open("/Users/huxmarv/super_secret_secrets.txt").read()
I like the number password123, too!
```

Updated script:

```
# secure script
favorite = raw_input('What is your favorite number? ') # change input to raw input
print 'I like the number {}, too!'.format(favorite)
```

```
> python2 better_input.py
What is your favorite number? open("/Users/huxmarv/super_secret_secrets.txt").read()
I like the number open("/Users/huxmarv/super_secret_secrets.txt").read(), too!
```

Breaks: California Legislative Information > (502c) Knowingly and without permission disrupts or causes the disruption of computer services or denies or causes the denial of computer services to an authorized user of a computer, computer system, or computer network

1.2 | Cross-site scripting

Vulnerable site

Write a comment that will cause some JavaScript to run.

```
<img src="!exist" onerror="alert('XSS')">; // alerts XSS
```

Updated site:

```
var textDiv = document.createElement('div');
//textDiv.innerHTML = document.getElementById('commentText').value;
textDiv.innerText = document.getElementById('commentText').value; // change innerHTML to innerText
newDiv.appendChild(textDiv);
```

Breaks: California Legislative Information > (502c) Knowingly and without permission disrupts or causes the disruption of computer services or denies or causes the denial of computer services to an authorized user of a computer, computer system, or computer network

1.3 | Buffer overflow

Try to give a password that is not the correct one but does grant you access.

```
// from http://stackoverflow.com/questions/34247068/buffer-overflow-does-not-work-on-mac-osx-el-capitan
#include "stdio.h"
#include "string.h"

int check_authentication(char *password) {
    int auth_flag = 0;
    char password_buffer[20]; // vulnerability


    strcpy(password_buffer, password);

    if (strcmp(password_buffer, "password") == 0) { // correct password: password
auth_flag = 1;
    }

    return auth_flag;
}

int main(int argc, char* argv[]) {
    if (argc < 2) {
printf("Usage: %s <password>\n", argv[0]);
    }

    if (check_authentication(argv[1])) {
printf("Access Granted.\n");
    } else {
printf("Access Denied.\n");
    }
}

compile with  `gcc -fno-stack-protector -D_FORTIFY_SOURCE=0 buffer_overflow.c` !

> ./a.out 123456789abcdefghijkl
Access Granted.

> ipython
In [1]: x = "123456789abcdefghijkl"
In [2]: len(x) # overflow with len > 20
Out[2]: 21
```

1. But how does this work? HR

Memory Layout	
Stack	High Address
↓	
↑	
Heap	
BSS Segment	
Data Segment	
Text Segment	Low Address

Buffer overflow can operate on the stack and the heap. Our example operates on the stack.

```
void func(int a, int b)
// when func is called, it allocates a block of mem at the top of the stack called a stack frame
// values of args are stored in the argument section of the stack frame
// next is return address, which is where the func needs to return to when it's done
// then the prev frame pointer
{
    int x, y; // and then finally local vars
    x=a+b;
    y=a-b;
}
```

To recap: Stack Frame:/ Arguments Return Address Previous Frame Pointer Local Variables/

When we enter the function, a stack frame is allocated at the top of the stack, and when we exit, it is released.

(a) Stack Buffer-Overflow Attack To copy, we need to allocate memory if we don't allocate enough, more data will be copied than can fit in the allocated space. **This gives us an overflow.**

our copying function:

```
strcpy(password_buffer, password); // used to copy strings: strcpy(to, from)
```

```
// password_buffer is
char password_buffer[20];
// so when the input is > than len 20,
// it will overwrite some of the stack above the buffer
// and we get buffer overflow.
```

Buffers grow from low address to high address, and `strcpy` will treat what is beyond `password_buffer[20]` as simply a continuation of the buffer, as in `password_buffer[21]`, `password_buffer[22]`, etc. `strcpy` simply copies until it hits the terminating character, '\0', which is generally added at the end of the string by the compiler.

Theoretically, we could insert arbitrary code by: 1. loading our code into memory by having it overwrite with overflow 2. overwrite the return address to jump to our code

However, **our example** is much simpler. By giving an input longer than 20, we overwrite what is allocated previously on the stack: `auth_flag` This makes `auth_flag` truthy, which leads to

```
if (check_authentication(argv[1])) { // check_authentication evaluting to true,
    printf("Access Granted.\n"); // and us gaining access.
} else {
    printf("Access Denied.\n");
}
```

(b) How to counteract simple answer:

```
strcpy(to, from); // don't use this!
strncpy(to, from, num); // use this.
// strncpy segfaults (sometimes) instead of overflowing
// if it doesn't reach a null terminator by the time it reaches num characters
```

we can just let it segfault, or we can

```
password_buffer[19] = "\0"; // set the last character to the null terminator
```

this effectively just cuts off the input.
Final Code:

```
// from http://stackoverflow.com/questions/34247068/buffer-overflow-does-not-work-on-mac-osx-el
#include <stdio.h>
#include <string.h>

int check_authentication(char *password) {
    int auth_flag = 0;
    char password_buffer[20];
    password_buffer[19] = "\0";
    strncpy(password_buffer, password, 20);

    if (strcmp(password_buffer, "password") == 0) {

auth_flag = 1;
    }

    return auth_flag;
}

int main(int argc, char* argv[]) {
    if (argc < 2) {
printf("Usage: %s <password>\n", argv[0]);
    }

    if (check_authentication(argv[1])) {
printf("Access Granted.\n");
    } else {
printf("Access Denied.\n");
    }
}
```

Breaks: California Legislative Information > (502c) Knowingly and without permission disrupts or causes the disruption of computer services or denies or causes the denial of computer services to an authorized user of a computer, computer system, or computer network

1.4 | To Explore: #todo #review

- KBxSideChannelAttacks
  - specter, heartbleed
  - "**Side-channel attacks** (SCAs) aim at extracting secrets from a chip or a system, through measurement and analysis of physical parameters. **Examples** of such parameters include supply current, execution time, and electromagnetic emission." – science direct

- KBxTimingAttacks
  - info from system based on the time it takes for the program to execute