CS 491/591
Software Security

Assignment 2

Due by 11:59PM, Thursday, **April 21**

NOTE: Assignments are to be done in teams. Do not look at another team's answer and do not allow another team to look at your answer. Doing so will result in charges of academic misconduct.

This assignment is the natural follow-up to assignment 1.
Overview: In this second assignment you are to use the provided simple application that contains a vulnerability, identify the vulnerability, and then write an application that will exploit the vulnerability.

Description:
The assignment was heavily borrowed from Steve Zdancewic.
The blame service is a simple system for assigning blame. The program accepts on standard input the name of a scapegoat and prints on standard output a message asserting that person's universal culpability. For example:

```
$ echo "Bill Gates" | ./blame
        It's all Bill Gates's fault.
```

Source code for blame.c is attached below.

Simple-minded string processing aside (it would be more correct, after all, to say that "It's all Bill Gates' fault"), there is a serious problem with the blame program. Despite our best efforts, a bug allows anyone who can provide input to this program to run arbitrary code on the target machine. (What might happen if it is run as a network service under inetd?)

Your job is to create input that will cause the blame service to print out the helpful message "Now I pwn your computer" before it terminates. For example:

```
$ cat exploit_file | ./blame
...
Now I pwn your computer
```

Here, the "..." may be additional output caused as a side-effect of your attack.

Deliverables: 1) your exploit application, and 2) a document discussing what the vulnerability is, how you were able to exploit it, and what your recommendation is for fixing the vulnerability.

Blame source code (blame.c):

```
/*
 * Blame server.  Assigns blame to the person of your choice.
 *
 * Usage: blame
 *    (reads one line from standard input)
 *
 * To compile:
 *    gcc blame.c -o blame
 *
 * Copyright 2016 by Feckless C. Coder, PhD.
 */

#include <stdio.h>
#include <string.h>
#define INPUT_BUFFER 256  /* maximum name size */

/*
 * read input, copy into s
 * gets() is insecure and prints a warning
 *    so we use this instead
 */
void grabline(char *s)
{
      int c;

      while ((c=getchar()) != EOF)
            *s++ = c;
      *s = '\0';
}

/*
 * convert newlines to nulls in place
 */
void purgenewlines(char *s)
{
      int l;

      l = strlen(s);

      while (l--)
            if (s[l] == '\n')
                  s[l] = '\0';
}


int main()
{
      char scapegoat[INPUT_BUFFER];

      grabline(scapegoat);
      /* this check ensures there's no buffer overflow */
      if (strlen(scapegoat) < INPUT_BUFFER) {
            purgenewlines(scapegoat);
            printf("It's all %s's fault.\n", scapegoat);
      }
      return 0;
}
```