# CIS 551 - Computer and Network Security Assignment #1 - Buffer Overflows

Consider the following program, which we might call `badbuf.c`:

```
#include <stdio.h>
int match(char *s1, char *s2) {
  while( *s1 != '\0' && *s2 != 0 && *s1 == *s2 ){
    s1++; s2++;
  }
  return( *s1 - *s2 );
}
void welcome(char *str) { printf(str); }
void goodbye(char *str) { void exit(); printf(str); exit(1); }
main(){
char name[123], pw[123]; /* passwords are short! */
char *good = "Welcome to The Machine!\n";
char *evil = "Invalid identity, exiting!\n";

printf("login: "); scanf("%s", name);
printf("password: "); scanf("%s", pw);
if( match(name,pw) == 0 )
welcome( good );
else
goodbye(evil );
}
```

Here is your assignment:

- Part 1: (due before class Sept. 10th) **Control** (25 points) Use a buffer overflow attack on this program so that it prints the welcome message for `name != pw`.
- Part 2: (due before class Sept. 17th) **Data payload** (25 points) Enhance your buffer overflow attack so that the program prints ``0wnz_U!''.
- Part 3: (due before class Sept. 24th) **General payload** (50 points) Further enhance your buffer overflow attack so that `/bin/sh` is executed and provides the attacker interactive access to the system on which `badbuf` is executing.

Turn in *all* source code used, including test cases and payload creation software. Turn in a demonstration log captured on a speclab machine using the Linux `script` command and run on an unmodified `badbuf`. Do not turn in executables. We suggest including a *makefile* so we can reproduce your setup - see `make(1)`) in the Linux documentation accessible by typing in `man make` at the command prompt. The easiest way to submit is to create a "tarball" with the Linux `tar(1)` command and submit the tarball using the `turnin` command on `eniac.seas.upenn.edu`. If needed, more details may be posted on Piazza, so stay tuned!

**Advice:**

- The assignment is tough, but feasible. Start early! If you finish one part early, begin building up skills for the next part - it takes a *lot* of experimentation.
- Read `http://insecure.org/stf/smashstack.html` for a readable introduction to the basic techniques.
- The speclab machines status is viewable at: `http://www.seas.upenn.edu/cets/checklab/index.php?lab=speclab`.
- These are 64-bit machines so Aleph One's 32-bit code *will not work*. The general techniques are still applicable.
- The speclab machines are configured *without* stack randomization; to allow execution of code on the stack various assembler or link time flags can be set - see `http://linux.die.net.man/8/execstack`.
- Contact the course TAs, Nikos Vasilakis - `nvas` at `cis.upenn.edu`, Rohit Dureja - `rohit@seas` at `seas.upenn.edu`, or Sibi Vijayakumar - `sibiv` at `seas.upenn.edu`, or Avinash Repaka - `avinashr` at `seas.upenn.edu` if you have any questions; they know the principles *and* the practice.