CS 491/591 Project 1

Buffer Overflows

(This project was designed by Steve Zdancewic, University of Pennsylvania. Used by permission.)

Consider the following C program, which we 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[128], pw[128]; /* passwords are eight characters - double this */
  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 );
}
```

Your assignment is in two parts:

Part 1: Control. Due Monday, Oct. 8 (30 %). Find input strings for `name` and `pw` that are not equal, but that cause the program to print the message "`Welcome to The Machine!`" anyway.

Part 2: Data Payload. Due Monday, Oct. 15 (70 %). Find input strings for `name` and `pw` that cause the program to print the message "`ownz_u!`".

This assignment is to be done in teams of two.

Detailed instructions:

0. The program must be compiled and executed on the CS linux lab machines (running Ubuntu and `gcc 4.6.3`). You can `slogin` to these machines at `linuxlab.cs.pdx.edu`.

1. Do your work within a sub-shell that you start with the command

```
setarch i386 -RL bash
```

This disables address randomization and enforces legacy memory layout on programs run in the subshell, which makes your job much easier!

2. Compile the program using something like

```
gcc -fno-stack-protector -o badbuf badbuf.c
```

The `-fno-stack-protector` flag prevents the compiler from applying various stack protection mechanisms in the generated program; again, this simplifies your job at lot. *Don't* use -O optimization flags. The format-security warning messages are expected.

3. For each part of the assignment, you should submit a file containing your exploit strings. The desired (buggy) behaviors should be produced by redirecting standard input from this file, e.g.

```
./badbuf < myexploit
```

4. Your strings are likely to contain non-printing characters. You will need to find some way of generating such strings from hexadecimal values. Shell scripts, C programs, and perl or python programs are handy for this. If you use any such programs, you must submit the code for them in addition to the exploit files themselves.

5. Finally, you must also submit a short text file called `README` describing how each exploit works, and giving any necessary instructions for executing the generation code.

6. Each team should create a single zip file containing all the elements of your submission, and email it as an attachment to `apt@cs.pdx.edu`. The body of the email should give the names of all team members.

Note: The course web page contains pointers to some useful information on the x86-64 architecture.