<div align="center">

Assignment 8, Fall 2018
CS4630, Defense Against the Dark Arts
Buffer Overflow Attacks:
Arcus Injectus Mutatio Nota
and *Codice Injectio* Curses

</div>

Purpose

In this assignment you will learn how buffer overflow vulnerabilities can be exploited to perform an arc injection attack and a code injection attack.

Due

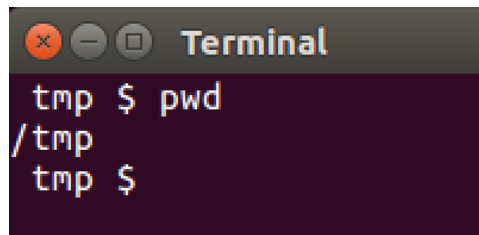This assignment is due on **Thursday, 15-NOV-2018 at 11:59 pm**

Assignment Prerequisites

1. NOTE: You will be developing your exploits such that they will run in `gdb` when invoked with an empty environment from the `/tmp` directory.

2. Review the "Stack Buffer Overflow" slides

3. Read the article "Detection and Prevention of Stack Buffer Overflow Attacks."

4. Download the code `dumbledore.exe` from the class webpage and examine its operation using `objdump` and `gdb`.

Assignment Details

1. This assignment must be completed using the **64-bit Ubuntu 18.04.1 LTS OS** you installed on your VM for Assignment #1. This environment is where we will test your submitted code. It is possible, due to the sensitivity of vulnerabilities to the operating environment, that exploit code developed for one environment will work not correctly in a slightly different environment.

2. The class Collab site has the file that you should download. You must attack the supplied version of `dumbledore.exe`.

3. **MANDATORY:** To control environment differences that will cause the stack addresses to shift, it is required that:

    (a) `dumbledore.exe` **must** be placed in the **top-level /tmp** directory on your VM for the reverse engineering steps you need to perform to develop your attack (`gdb`). Use

the `pwd` utility from within your `tmp` directory to verify that your `tmp` is at the top level. Your output should match the following screenshot.
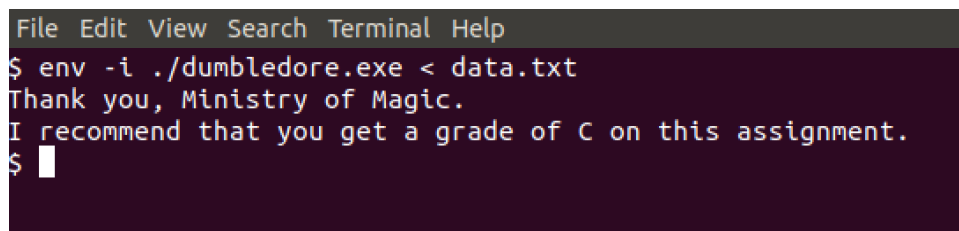

Figure 1: Testing correct location of `tmp` directory.

(b) **Use an empty environment** when you run `dumbledore.exe` in `gdb`. You can do so as follows:

```
cd /tmp
env -i gdb ./dumbledore.exe
```

4. Ubuntu has ASLR (Address Space Layout Randomization) turned on as a defense. To simplify your task, we will turn it off. The command
`setarch 'uname -m' -RL bash`
will disable ASLR for that shell. It does not affect any other shells.

5. Examine `dumbledore.exe`. It contains an obvious buffer overrun vulnerability in the `readString` function.

6. Create a file named `data.txt` that contains your name and run `dumbledore.exe`. Here are some sample runs with non-malicious input to `dumbledore.exe`. The file `data.txt` contains the string "`Ministry of Magic`" which when given to `dumbledore.exe` prints out the grade of "C". The file `data2.txt` contains the string "`Wizard in Training`". When given to `dumbledore.exe`, the program prints out a grade of "B".
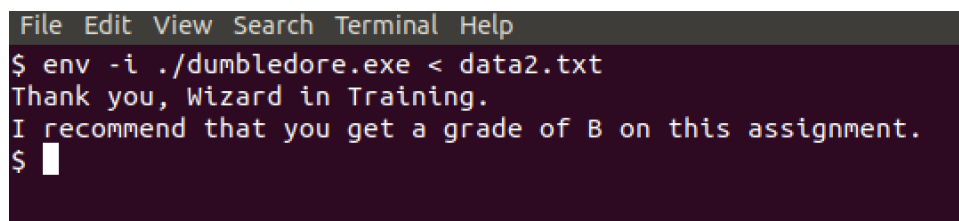

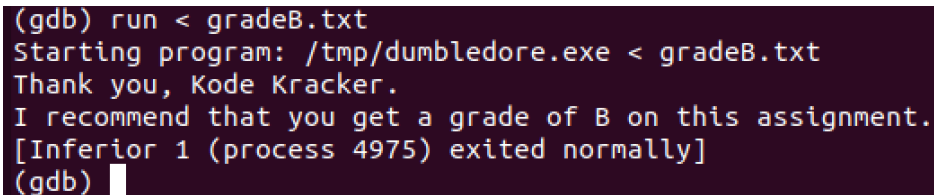Figure 2: Non-malicious input that gives user a grade of C.

Figure 3: Non-malicious input that gives user a grade of B.

7. There are two parts/goals in this assignment:

(a) ***Arcus Injectus Mutatio Nota* Curse.** Your goal for this part of the assignment is to attack this program by exploiting the buffer overrun vulnerability so that the program execution is as shown in Figure 4. Your exploit input should work when you run your program as follows:

```
env -i gdb ./dumbledore.exe
```

You must do so without injecting any additional code to execute (i.e., you will exploit the code by developing an arc-injection attack). Your exploit input should cause the program to print your full name instead of the name shown in the screenshot in Figure 4.
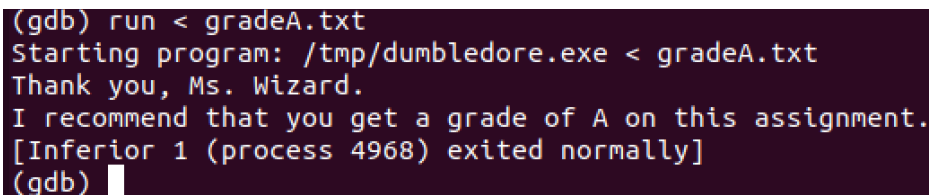
```
(gdb) run < gradeB.txt
Starting program: /tmp/dumbledore.exe < gradeB.txt
Thank you, Kode Kracker.
I recommend that you get a grade of B on this assignment.
[Inferior 1 (process 4975) exited normally]
(gdb)
```

Figure 4: Arc injection attack input that gives user a grade of B.

(b) ***Codice Injectio* Curse.** Your goal for this part of the assignment is to attack this program by exploiting the buffer overrun vulnerability so that the program execution is as follows. You must do so by injecting additional code (i.e., a code-injection attack) to alter your grade to "A". Your exploit input should cause the program to print your full name instead of the name shown in the screenshot in Figure 5. Your exploit input should work when you run your program as follows:

```
env -i gdb ./dumbledore.exe
```

```
(gdb) run < gradeA.txt
Starting program: /tmp/dumbledore.exe < gradeA.txt
Thank you, Ms. Wizard.
I recommend that you get a grade of A on this assignment.
[Inferior 1 (process 4968) exited normally]
(gdb)
```

Figure 5: Code injection attack input that gives user a grade of A.

Requirements

1. Your first step should be to analyze the `dumbledore` binary using `gdb` and `objdump`. Remember that `dumbledore.exe` should be placed in the top-level `/tmp` or the stack addresses you see will not match those on the grader's VM.

2. Next you should analyze `readString`s stack frame. To do this, you need to set a breakpoint in `readString`.

3. Your next task is to get the program to crash. Write a C program named `attack-crash.c` that produces an input to `dumbledore.exe`, as simple as possible, that causes dumbledore.exe to generate a segmentation fault. Make sure you understand why the program is crashing. To get you going, here is a program that creates a legal input. (NOTE: Your attack input generator should write the attack input to `stdout`.)

```
#include <stdio.h>
#include <string.h>
char attackString[] = "Bill Smith\n";
int main() {
 int i;
 char *p = attackString;
 for (i = 0; i < sizeof(attackString); i++) {
 putchar(*p);
 p++;
 }
 return 1;
}
```

Here is an example run.

```
$ ./attack-gradeC > data.txt
$ env -i gdb ./dumbledore.exe
(gdb) run  < data.txt
Thank you, Bill Smith.
I recommend that you get a grade of C on this assignment.
[Inferior 1 (process 4493) exited normally]
(gdb)
```

4. ***Arcus Injectus Mutatio Nota* curse**: The task for this step is to cause the program to give
   you a grade of "B". Write a program named `attack-gradeB.c`, that produces an input,
   as simple as possible, that causes `dumbledore.exe` program to print your name and
   recommend a grade of "B" without injecting additional code to execute (arc injection attack).
   You can see by reading the program that if your name is Wizard in Training, this task is
   easy. However, your name is probably not Wizard in Training, so you will need to figure
   out how to exploit the vulnerability to get a grade of "B". You should include comments in
   `attack-gradeB.c` that explains how your exploit works.

5. ***Codice Injectio* curse**: The task for this part of the assignmentis to cause the program to
   give you a grade of "A". Write a program named `attack-gradeA.c`, that produces an
   input, as simple as possible, that causes `dumbledore.exe` program to print your name
   and recommend a grade of "A". You should include comments in `attack-gradeA.c` that
   explains how your exploit works.

Miscellaneous Notes

1. Submit your attack generation programs and screenshots of your working exploits via the
   Collab. They must must be given the following names: `attack-gradeB.c` and `attack-gradeA.c`.

2. This assignment is pledged.

3. A useful command is `objdump -d`. This command will disassemble a file. This command is useful for seeing the binary encoding of instructions.

4. A useful `gdb` command is `si`. This single steps a program an instruction at a time.

5. The following command `display/i $pc` will display the current instruction and do so each time the program stops (e.g., after a single step command).

6. It is useful to draw a picture of the stack when the program is executing inside the function `readString`.

7. This assignment was adopted from one given at Princeton in COS 217 by Andrew Appel and one previously given in CS 4630 Spring 2010.

Items to Submit

This assignment is due on **Thursday, 15-NOV-2018 at 11:59 pm**. Upload screenshots of your working exploits, as well as your attack input generator C source files for the *Arcus Injectus Mutatio Nota* and *Codice Injectio* curses to Collab. Your code should be commented with your name, UVA ID, and descriptions of how you constructed the attack input.

1. *Arcus Injectus Mutatio Nota* **Curse**:

   - `attack-gradeB.c`
   - Screenshot of your working grade B exploit

2. *Codice Injectio* **Curse**:

   - `attack-gradeA.c`
   - Screenshot of your working grade A exploit

It is mandatory that you use the file names given and adhere to the given API to ease the task of grading many different submissions of this assignment. Throughout the semester, you will be given file names and sample execution output. All assignments will be submitted using the class Collab Website.