Ordinary Wizarding Level Examination
Fall 2018, CS4630
Defense Against the Dark Arts

As a young wizard, you are required to take the Ordinary Wizarding Level Examination (O.W.L.) to earn your wizard title and hone your skills for the battle you are sure to face. In the O.W.L. examination, you must tackle six challenges. If you complete all six challenges you will be awarded a grade of 100 (out of 100) for the final examination grade. Glory is within your grasp! Complete details of the grading of the O.W.L. are provided at the end of this document.

Due

This O.W.L. examination is due on **Friday, 7-DEC-2018 at 5:00 pm** . **No late submissions will be accepted.**

O.W.L. Examination Rules

1. You may complete the challenges in any order.
2. You may not work together.
3. The O.W.L. examination is a pledged activity.
4. You may use any non-human resource. For example, you may use any resource available on the Internet. (Remember: Google is a Wizard's friend.)

Examination Details

1. Make sure that ASLR is turned off:

   ```
   setarch `uname -m` -RL bash
   ```

   will disable ASLR for the current shell. It does not affect any other shells.

2. To keep the necessary files segregated according to each Challenge, the files are provided in a `tar`'d, `gzip`'d file named `OWL.tgz`.

3. The `tgz` file should be copied and extracted in the **top-level `/tmp`** directory as follows:

   ```
   ~/Downloads$ cp OWL.tgz /tmp
   ~/Downloads$ cd /tmp
   /tmp$ tar xvzf OWL.tgz
   < ... list of extracted files ... >
   ```

(**Note**: If the `pwd` utility when executed in your `tmp` directory returns a path similar to: `/home/<your userid>/tmp`, instead of simply `/tmp`, then `OWL.tgz` has not been extracted to the correct location). Your attack spells should be developed with `OWL.tgz` extracted at the root-level (top-level) `tmp` directory. You should do your work in each challenge directory (i.e,. `Challenge1`, `Challenge2`, `Challenge3`, `Challenge4`, `Challenge5`, and `Challenge6`.

4. **IMPORTANT**: Use the executables provided. **DO NOT** compile the source provided to produce new executables for attacking. The Ministry of Magic will be using the executables they provided to test your spells.

5. 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.

6. When you develop your exploits for each challenge, make sure to run and test the program with an empty environment, using `env -i` as in previous assignments. We have provided screenshots throughout the O.W.L. examination demonstrating how to use it.

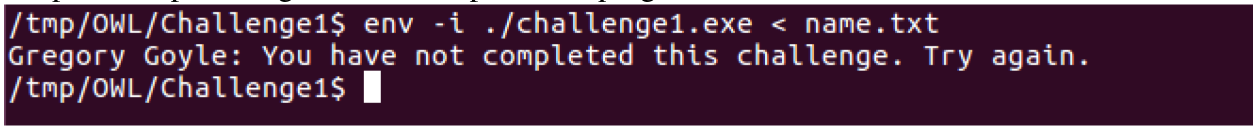
7. Challenges will be tested using `env -i gdb ./challengeX < <inputFile>`
Make sure that your exploit solution works when invoked as listed in this bullet.
NOTE: Screenshots for each challenge do not show the invocation of `gdb`. We chose to show you the output this way to highlight the correct output for a successful exploit, rather than confuse you with our automated gdb batch script commandline.

Challenge 1: Arc injection via buffer overrun (easy)

For this challenge, you will use the code `challenge1.c` and `challenge1.exe`. This first challenge is very similar to Assignment 8. **Remember:** use the executable provided by the Ministry of Magic as the executable to develop your spell inputs. Do not recompile the source code to produce a new executable. The Challenge 1 code contains a buffer overrun vulnerability in the `readString` function.

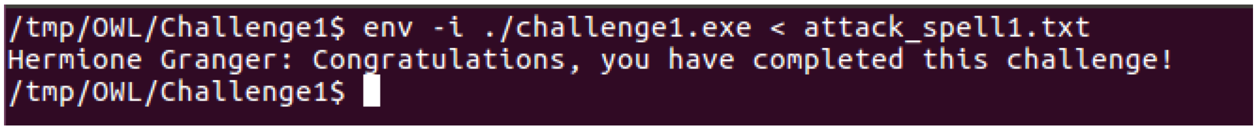
You should write a program to generate the attack string. Your attack input should print your name and output similar to the sample output shown in the screenshot of a successful sample attack below. Save the attack string into a file named `attack_spell1.txt`. Here are two sample runs that demonstrate the execution of the program. Assume the file `name.txt` contains only the name Gregory Goyle.

Output from providing non-attack input to the program:



```
/tmp/OWL/Challenge1$ env -i ./challenge1.exe < name.txt
Gregory Goyle: You have not completed this challenge. Try again.
/tmp/OWL/Challenge1$
```

Output from providing successful attack input to the program:



```
/tmp/OWL/Challenge1$ env -i ./challenge1.exe < attack_spell1.txt
Hermione Granger: Congratulations, you have completed this challenge!
/tmp/OWL/Challenge1$ █
```

For success on this challenge, your output must be identical to the above screenshot (except with your name instead of Hermione's).

If you complete this challenge, submit the following files to Collab:

1. Your attack input file named `attack_spell1.txt` and

2. your attack generator file named `attack_spell1.c`.

Challenge 2: ROP attack via buffer overrun

For this challenge, you will use the following files: `challenge2.c`, `challenge2.exe`, `hacked.sh`, and `ROPgadget.tgz`.

A ROP-attack, or return-oriented programming attack, is a type of arc-injection attack where the attacker causes gadgets already existing in the code to be executed. If sufficient gadgets exist, the attacker can cause arbitrary programs to be executed. The goal of our attack is to exec a shell and execute a shell command (`hacked.sh`).

You first need to download and install the `ROPgadget` tool (the same one used in the ROP assignment). We have provided a tarball of `ROPgadget` for this Challenge (the same tarball as in the ROP assignment). Some of the following instructions are the same as those used in the ROP assignment. These instructions have been included for the sake of completeness.

Download the tarball and extract the files in the directory where you will be doing your work (`/tmp`). Change to the `ROPgadget` directory and issue the following commands from the terminal:

```
$ make clean
$ make
```

Make sure that `ROPgadget` is built properly by issuing the following command:

```
$ ./ROPgadget --help
```

You will need to read the document to understand how to use `ROPgadget`.

You will notice that the `readString` function in `challenge2.c` is similar to the one used in Challenge 1. Craft an attack string that will kick off the execution of a sequence of gadgets. The file `hacked.sh` needs to be in the directory where you carry out the attack (i.e., same directory as the binary you are attacking). You first need to build `ROPgadget`. Untar the package in your directory containing the challenge files.

```
$ tar xvzf ROPgadget.tgz
< ... list of files extracted ... >
$ cd ROPgadget
$ make clean
$ make
```

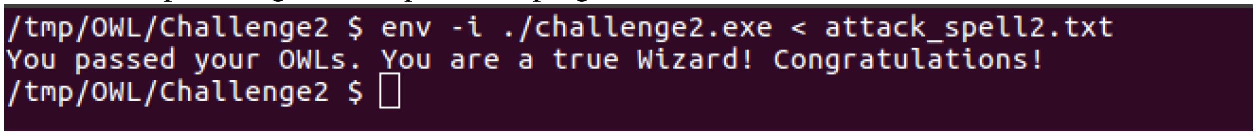
The tool ROPgadget will create most of the attack string. Here is the command:

```
$ ./ROPgadget/ROPgadget -csyn challenge2.exe /bin/sh
hacked.sh > attack_spell2.c
```

You will need to add some code to the program that ROPgadget generates (in this case attack_spell2.c) where it says Padding goes here. This is the beginning of the attack string your name and some padding to fill up the buffer so that the first address that ROPgadget generated overwrites the return address in function readStrings activation record. The attack input essentially performs an arc-injection attack to the first gadget and then ROPgadget has set up successive addresses in the attack input to invoke the appropriate gadgets.

Call this modified program (where you have inserted the necessary padding), attack_spell2.c. The following illustrates an attack run assuming the attack string generated by ROPgadget was written to a file called attack_spell2.c and it was compiled to produce attack_spell2.exe.

Result from providing attack input to the program:



```
/tmp/OWL/Challenge2 $ env -i ./challenge2.exe < attack_spell2.txt
You passed your OWLs. You are a true Wizard! Congratulations!
/tmp/OWL/Challenge2 $
```

Notice that because the attack code causes a shell to be execed (causing the shell script contained in hacked.sh to be executed), the normal output of the program does not appear. Your attack input should produce output identical to the reference output above.

If you complete this challenge, submit the following to Collab:

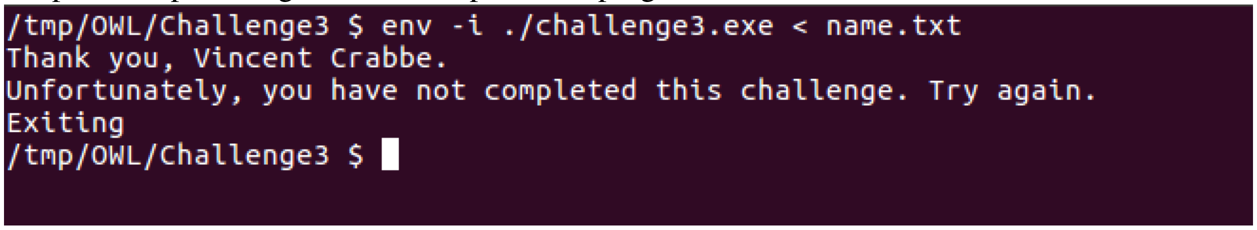
1. Your attack string named attack_spell2.txt and
2. your C source named attack_spell2.c

Challenge 3: Code injection via buffer overrun

For this challenge, you will use the code challenge3.c and challenge3.exe. Your goal is to craft an attack input which causes the program to produce output like the screenshot of running the program with the attack input shown below (except with your name instead of Ginny's). Unfortunately, an arc-injection attack is not effective against this slightly modified version of the program. You must craft a code-injection attack (i.e., inject code on the run-time stack and cause it to be executed). You must inject code that will change your grade and then cause that code to be executed.

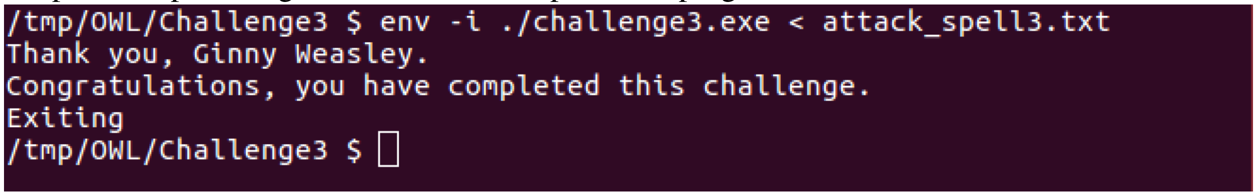
The Ministry of Magic has prepared an executable that will permit code injection attacks. As before, you should write a program to generate the attack string. Save the attack string into a file named attack_spell3.txt. Here are two sample runs that demonstrate the sample runs of the program. Assume the file name.txt contains the name Vincent Crabbe.

Output from providing non-attack input to the program:



```
/tmp/OWL/Challenge3 $ env -i ./challenge3.exe < name.txt
Thank you, Vincent Crabbe.
Unfortunately, you have not completed this challenge. Try again.
Exiting
/tmp/OWL/Challenge3 $
```

Output from providing successful attack input to the program:



```
/tmp/OWL/Challenge3 $ env -i ./challenge3.exe < attack_spell3.txt
Thank you, Ginny Weasley.
Congratulations, you have completed this challenge.
Exiting
/tmp/OWL/Challenge3 $ 
```

For success on this challenge, your output must be like the above screenshot (of course with your name – not Ginny Weasley's). No additional characters should be printed after your name for the first printed sentence. The remainder of the program's output should be identical to the attack input shown above.

If you complete this challenge, submit the following files to Collab:

1. Your attack string named `attack_spell3.txt` and
2. the C source for your attack string generator named `attack_spell3.c`.

Challenge 4: Reverse Engineering

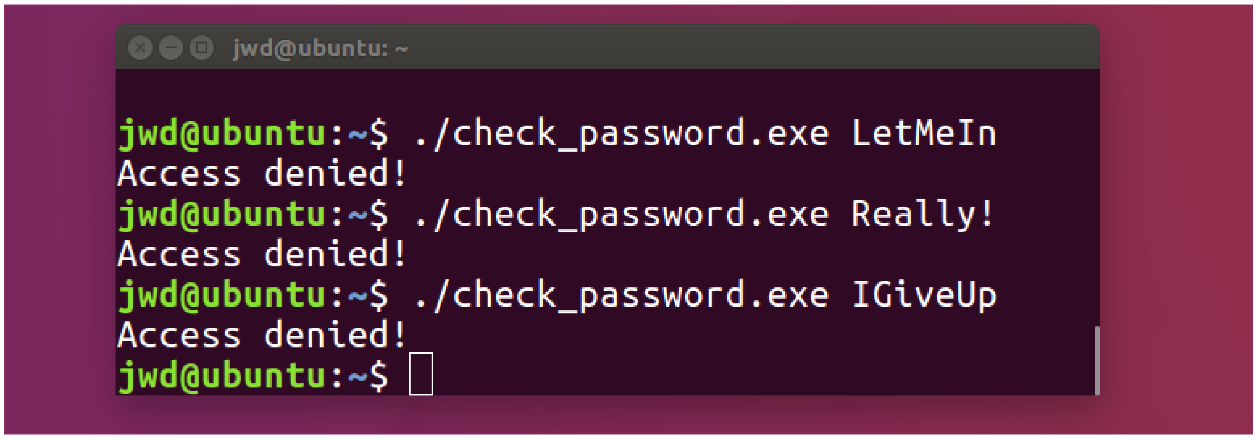
For this challenge, you will need to reverse engineer a binary to find a hidden password in the binary. The program's operation is simple. It takes a single argument that is a password (i.e., an ASCII string) and checks if it is correct. If the password is correct, the program prints

```
Access allowed!
```

If the password entered is incorrect, the program prints

```
Access denied!
```

Here is a screenshot showing someone trying to use brute force find the password.



```
jwd@ubuntu:~$ ./check_password.exe LetMeIn
Access denied!
jwd@ubuntu:~$ ./check_password.exe Really!
Access denied!
jwd@ubuntu:~$ ./check_password.exe IGiveUp
Access denied!
jwd@ubuntu:~$ 
```

If you complete this challenge, submit a text file named `password.txt` which consists of a single line (no leading or tailing spaces) with the password.

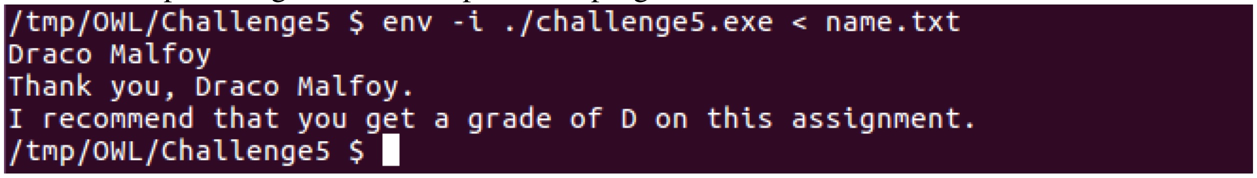
Challenge 5: Format String attack

For this challenge, you will use the code `challenge5.c` and `challenge5.exe`. Again, your goal is to give yourself a grade of “A”. This challenge contains a format string vulnerability.

You should write a program to generate the attack string. Save the attack string into a file named `attack_spell5.txt`.

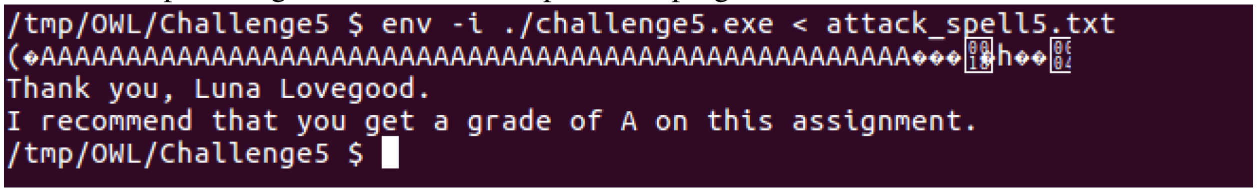
Here are two sample runs that demonstrate the sample runs of the program. Assume the file `name.txt` contains two lines where each line contains the name Draco Malfoy.

Result from providing non-attack input to the program:



```
/tmp/OWL/Challenge5 $ env -i ./challenge5.exe < name.txt
Draco Malfoy
Thank you, Draco Malfoy.
I recommend that you get a grade of D on this assignment.
/tmp/OWL/Challenge5 $ 
```

Result from providing successful attack input to the program:



```
/tmp/OWL/Challenge5 $ env -i ./challenge5.exe < attack_spell5.txt
(�AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA����h���
Thank you, Luna Lovegood.
I recommend that you get a grade of A on this assignment.
/tmp/OWL/Challenge5 $ 
```

Your attack input should cause the program to print your full name with no additional characters. The remainder of the program output should be identical to the program output shown when the reference attack input is given. (The characters printed above the ”Thank you” sentence may differ.)

If you complete this challenge, submit the following files to Collab:

1. Your attack string file named `attack_spell5.txt` and
2. your C attack string generator source file named `attack_spell5.c`.

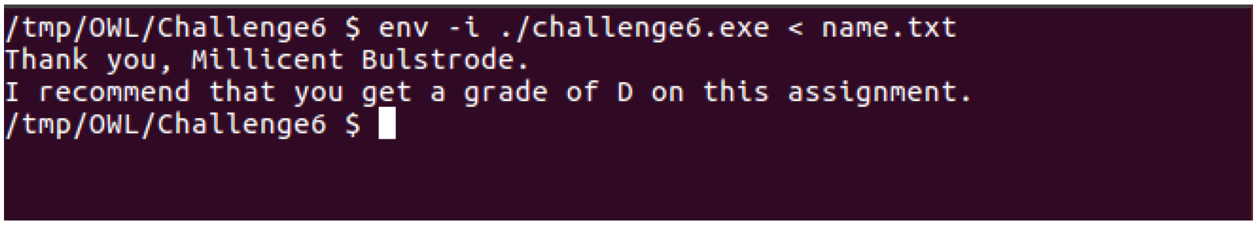
Challenge 6: Arc injection via buffer overrun (hard)

For this challenge, you will use the code `challenge6.c` and `challenge6.exe`. This challenge is also very similar to Assignment 8, but with a difference which makes it more difficult to solve. Remember: use the executable provided by the Ministry of Magic. It is the one the Ministry will use to test your attack input. Do not recompile the source code to produce a new executable.

The code contains a buffer overrun vulnerability in the `readString` function. You are to exploit this vulnerability by performing an arc-injection spell to change your grade from a “D” to an “A”.

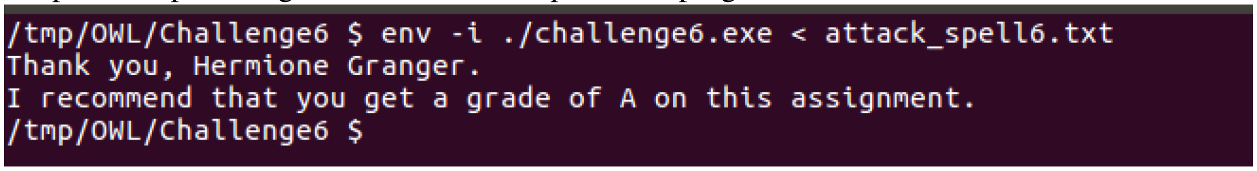
As before, you should write a C program, `attack_spell6.c`, to generate the attack string. Save the attack string into a file named `attack_spell6.txt`. Here are two sample runs that demonstrate the execution of the program. Assume the file `name.txt` only has the name Millicent Bulstrode.

Output from providing non-malicious input to the program:



```
/tmp/OWL/Challenge6 $ env -i ./challenge6.exe < name.txt
Thank you, Millicent Bulstrode.
I recommend that you get a grade of D on this assignment.
/tmp/OWL/Challenge6 $
```

Output from providing successful attack input to the program:



```
/tmp/OWL/Challenge6 $ env -i ./challenge6.exe < attack_spell6.txt
Thank you, Hermione Granger.
I recommend that you get a grade of A on this assignment.
/tmp/OWL/Challenge6 $
```

For success on this challenge, your output must be identical to the above screenshot (of course with your name – not Hermione's). The remainder of the program output should be identical to the reference output for successful attack above.

If you complete this challenge, submit the following files to Collab:

1. Your attack string file named `attack_spell6.txt` and
2. the C source code file for your attack string generator named `attack_spell6.c`.

Wizards of Distinction

The first two wizards to finish all six challenges will be deemed the champions to signify their stellar achievement. At the Ministry of Magic's direction, the champions will receive Wizard's Cups and at least an "A" for the final course grade.

If you have finished all six challenges and champions have not been announced, send e-mail to `jwd@virginia.edu`, `mc2zk@virginia.edu`, `whh8b@virginia.edu`, and `abiusx@virginia.edu`. This e-mail serves to stake your claim and further timestamp your submission. Note: you must still submit your code and attacks via Collab and the Collab timestamp will be the official timestamp to determine the two *Wizards of Distinction*. When two champions have been verified, the class will be notified via e-mail.

Grading

For grading purposes, completion of challenges will be scored as indicated in the following table. The total point value of the O.W.L. exam is 100 points.

Challenge Description	**Point Value**
1: Arc injection via buffer overrun (easy)	62 points
2: ROP attack	6 points
3: Code injection via buffer overrun	6 points
4: Reverse Engineering	6 points
5: Format string attack	10 points
6: Arc injection via buffer overrun (hard)	10 points
Point Total	**100 points**

Items to Submit

The following files must be submitted to Collab:

Challenge	**Files to submit**
1: Arc injection via buffer overrun (easy)	`attack_spell1.txt` `attack_spell1.c`
2: ROP attack	`attack_spell2.txt` `attack_spell2.c`
3: Code injection via buffer overrun	`attack_spell3.txt` `attack_spell3.c`
4: The hidden password	`password.txt`
5: Format string attack	`attack_spell5.txt` `attack_spell5.c`
6: Arc injection via buffer overrun (hard)	`attack_spell6.txt` `attack_spell6.c`

The O.W.L. examination is due on **Friday, 7-DEC-2018 at 5:00 pm** . There are no extensions!!

It is mandatory that you use the file names given and adhere to the given API to ease the task of grading multiple different student submissions of this examination.