Total points: 80 (+0 bonus)

COMP3632 (2020 Spring)
Assignment 1

Due: 11:59 PM, 25th March, 2020 (Wed)

Written assignment

1. [20 points] Read each of the following news stories about malware:

   (a) In 2019, a new type of Linux malware called Skidmap is found which makes the victim's computer mine Bitcoins for the attacker. Bitcoin mining consumes computational cycles to generate profit for the attacker. The malware is delivered by tricking the user into running it and giving it administrator privileges. Once installed, Skidmap hides from the user by overwriting the "rm" (delete file) binary, so that if the user attempts to remove it, Skidmap would schedule a re-installation of itself again at a later date. Users usually cannot detect the presence of Skidmap.

   (b) The UK's National Health Service has suffered severe disruption to its services after its computers were infected with the WannaCry ransomware in May 2017. WannaCry spreads with the EternalBlue exploit, which can infect computers remotely using a specially crafted packet targeting vulnerable background daemons. Once infected, the computer's files are encrypted and cannot be encrypted without paying money to the attackers, although the attackers often did not give the decryption key even when paid. As of 2019, more than 1 million devices are sitll vulnerable to this exploit.

   (c) Two men were charged with bribing AT&T employees with more than 1 million USD to install malware in the company's computers. The malware recorded the victim employees' actions on AT&T computers and sends the data remotely to the attackers to help the attackers penetrate its infrastructure. It appears that the ultimate goal of the attackers was to steal unlock codes for AT&T phones.

   For **each** of the above news stories, answer the following questions:

   i. [6 points] Which of the CIA principles is being violated? Explain why.

   ii. [7 points] Classify the malware by method of spread, and explain.

   iii. [7 points] Classify the malware by payload, and explain.

2. [15 points] State whether each of the following statements is true or false. For each, explain why.

   (a) [3 points] Cryptographic protocols generally rely on open design, even the ones that use secret keys.

   (b) [3 points] Buffer overflow attacks are made more powerful because of poor implementation of the principle of least common mechanism.

   (c) [3 points] Heartbleed can be considered as a threat against privacy.

   (d) [3 points] XSRF attacks usually require the attacker to gain full control over the web server first.

   (e) [3 points] A format string vulnerability gives both buffer overread and buffer overflow capabilities.

Programming assignment

Buffer Overflow Vulnerabilities [45 points]

You have been given `login.cpp`, a simple program to check if the user's login matches a stored username and password using three different methods. Compile it and test it with user input. Normally, the program checks a secret `password.txt` file to match your input with the stored username and password, and grant access if they match. For your convenience, you have been provided with such a `password.txt` file to test your code, but you should assume the true `password.txt` file is **different** from the one you were provided when you write your exploit.

There are three ways to log in using `login.cpp`:

1. `./login -i <username> <password>` will check your username and password against the secret `password.txt` file. It uses a hardcoded canary to detect buffer overflows, just like stack canaries. However, since this canary is hardcoded and not randomized, you can see it in the code and use it for your attack.
2. `./login -j <username> <password>` will check your username and password against the secret `password.txt` file. This time, its hardcoded canary is dynamically computed against your username. Furthermore, the hardcoded canary is placed in a different location in memory.
3. `./login -k <username> <password>` will check your username and password against the secret `password.txt` file. Its canary is randomized, so you cannot expect to overwrite it correctly.

The `login` program is simplified: upon a successful login, it shows a congratulatory message and does not do anything else. You are free to imagine that it will then allow you to perform some privileged action, such as allowing access to confidential data or launching a missile.

Your username must **start with your own @connect.ust.hk username**. For example, if my e-mail is taow@connect.ust.hk, I can choose taow123 as my username for this assignment. This is meant to discourage plagiarism so that each student's answer will be unique.

(a) [15 points] Using a buffer overflow exploit, log in to the program using the first method. Submit your username and password in a file called `a1a.txt`, with exactly two lines, the first line being the username, and the second line being the password. (Do not write anything else in your submitted file.) To repeat the above, the username you choose must start with your own @connect.ust.hk username.

(b) [15 points] Using a buffer overflow exploit, log in to the program using the second method. Submit your username and password in a file called `a1b.txt` exactly as in part (a). The above restrictions apply.

(c) [15 points] Using a buffer overflow exploit, log in to the program using the third method. Submit your username and password in a file called `a1c.txt` exactly as in part (a). The above restrictions apply. For this part, some students may achieve an answer in another architecture to the marking computer. For marking convenience, submit a screenshot showing success as **a1c_screenshot.png**. The screenshot should show your command line arguments and the results on the screen.

For each part, as long as "Login successful!" appears, the attempt is considered successful even if other warning messages appear.

All of your exploits should work on your own computer, considering all the buffer overflow defenses. Therefore, you should not be trying to overwrite the return address. To emphasize, you should not use any information in the `password.txt` file: it is there only for your convenience.

Hints

To study the `login.cpp` code, you can try to modify it, for example, to log the values of the variables at different lines in the code. It may also be a good idea to learn how to use a memory disassembler like `gdb` to find where the variables are. Just remember to remove your modifications to test your username and password against the original `login.cpp` file.

`struct` is used in this code to ensure that the variables are always placed in the right order; the compiler would not re-arrange the order of variables.

Submission instructions

All submissions should be done through the CASS system.
You should submit the following files by the deadline:

- a1.pdf, with your answers to the written component.
- a1a.txt, a1b.txt, a1c.txt, with your answers to the programming component.
- a1c_screenshot.png, showing success with 1(c). It will be used to help grading if your a1c.txt answer does not work on the marking computer.

Do not zip any files. You can submit these files any number of times and the system will accept the last submission for each file, which overwrites previous submissions. You are encouraged to make submissions as early as possible. It is important to name your files correctly. Otherwise, we may not mark them!

Keep in mind that plagiarism is a serious academic offense; you may discuss the assignment, but write your assignment alone and do not show anyone your answers and code.

The submission system will be closed exactly 48 hours after the due date of the assignment. The 48 hours act as a no-penalty grace period. Submissions after then will not be accepted unless you have requested an extension before the due date of the assignment. You will receive no marks if there is no submission within 48 hours after the due date.

Command Line Arguments

C++ (and most other languages) can accept *command line arguments*, which are additional commands given while running the code. Suppose we compiled `mycode` from `mycode.cpp`. If we run the binary code file `mycode` as such:

```
./mycode abc 3 1
```

We say that `mycode` is run with 3 command line arguments; the first one is `abc`, the second one is `3`, and the third one is `1`.

In `mycode.cpp`, the main function header will be written as follows:

```
int main(int argc, char ** argv) {
        ...
}
```

In this code, `argc` is an integer that counts the number of arguments (plus one, because `mycode` also counts), and `argv` is a list of character arrays that contains the arguments. For example:

```
int main(int argc, char ** argv) {
        printf("Number of arguments is: %d, first argument is %s\n", argc, argv[1]);
        return 0;
}
```

The output will be:

```
Number of arguments is 4, first argument is abc
```