Total points: 75 (+5 bonus)

COMP3632 (2017 Fall)
Assignment 1

Due: 11:55 PM, 6th October, 2017 (Fri)

Written assignment

1. [8 points] OnePass offers a password management service. Users of the service can store all of their passwords securely on the OnePass servers for $5 per month. They can have easy access to all of their own passwords by entering a master password. However, one possible vulnerability is that attackers can access any user's password if they can guess the master password. OnePass engineers estimate that every year, 2% of their 10,000 user accounts will be hacked by a password-guessing attacker. The problem can be eliminated by enabling two-factor authentication. It will cost the company an extra $3,000 a year to run a server to support two-factor authentication.
   (a) [3 points] For the above scenario, state the System, Asset, Vulnerability, Attack, and Defense.
   (b) [3 points] Assume that the only financial damage caused by the above account-hacking attack is that the hacked user has a 10% chance of noticing right after the hack, after which the hacked user will terminate the service and stop paying the fee. Using Quantitative Risk Analysis, determine if it is worth enabling two-factor authentication.
   (c) [2 points] Suggest one other type of financial damage that may affect the company, besides hacked users terminating the service.

2. [12 points] Read each of the following news stories about malware:
   (a) A new type of Sundown malware is found which makes the victim's computer mine Bitcoins for the attacker. Mining Bitcoins is CPU-intensive, and it is a way to make money for the attacker. The malware is delivered from Javascript code, which can be hosted on an innocuous-looking web site that the attacker invites the victim to access.
   (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 without any involvement from the user. Once infected, the computer's files are encrypted and cannot be encrypted without paying a ransom. Often, paying the ransom did not cause the key to be released.

(c) The Angler exploit kit has been shut down after the arrests of several hacking gangs. The kit is often used to collect passwords and credit card information from exploited computers. Angler is capable of hacking legitimate websites and inserting its own code into those sites, thus infecting people who are visiting those web sites.

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. [6 points] Classify the malware by stating which class it belongs to, and explain.

3. [10 points] For each of the following Saltzer and Schroeder's Principles of Secure Design:

(a) Open Design

(b) Economy of Mechanism

(c) Least Common Mechanism

(d) Complete Mediation

(e) Psychological Acceptability

Answer the following questions:

i. [10 points] Explain the principle.

ii. [5 points (bonus)] Give an example from fiction (movie, TV, books, etc.) where the principle was violated, or where the principle was useful. The asset being protected does not have to be computer-related. You are not allowed to use the original *Star Wars* trilogy as it is readily available online.

Keep in mind that you should use your own words; do not copy and paste from anywhere.

Assignment Due: 11:55 PM, 6th October, 2017 (Fri)

Programming assignment

Format String Vulnerabilities [45 points]

Formal string vulnerabilities are a special type of buffer overflow that occurs when the attacker has control over the input of `printf()` or `fprintf()`. It is possible to leverage a format string vulnerability into a buffer overflow gaining full control over the program's code by overwriting the return address. In this assignment, we will not attempt to overwrite the return address as this would only cause a program crash with stack canaries. Instead, we will overwrite other parts of the program memory to cause unexpected behavior and bypass security.

You have been given `login.cpp`, a simple program to check if the user's login matches a stored username and password. Compile it and test it with user input. Normally, the program checks a secret `passwd` file to match your input with the stored username and password, and grant access if they match. (It is not a good idea to store passwords in plaintext.) For your convenience, you have been provided with such a `passwd` file to test your code, but you should assume the true `passwd` 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 `passwd` 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 `passwd` file. It uses a hardcoded canary again, which is dynmically 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 `passwd` file. This time, there is simply a bounds check to prevent the buffer overflow error.

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.

You can choose any input username as long as it **starts 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 restriction is set to discourage plagiarism.

(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.

You may not assume that the login program was compiled on an old computer with buffer overflow defenses. In other words, all of your exploits should work on your own computer.

To emphasize, you should not use any information in the `passwd` file: it is there only for your convenience. Therefore, your username and password combination should work no matter what the true username and password is, **without accessing the passwd file**.

Hints

It is a good idea to modify the `login.cpp` code, especially to log the values of the variables at different stages 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.

As mentioned in class, I use `struct` liberally in this code to ensure that the variables are always placed in the right order; the compiler should not be able to re-arrange the order of variables.

Assignment Due: 11:55 PM, 6th October, 2017 (Fri)

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.

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.

Assignment Due: 11:55 PM, 6th October, 2017 (Fri)

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
```

Assignment Due: 11:55 PM, 6th October, 2017 (Fri)