Total points: 100 (+0 bonus)

CMPT403 (2025 Fall)
Assignment 1

Written assignment

1. [12 points] Read each of the following news stories about malware:
   (a) ENISA reports a 30% increase in crypto-jacking incidents year-on-year in 2020, and they have only increased since. Crypto-jacking uses the victim's computing resources (usually CPU) to mine cryptocurrencies for the attacker. Crypto-jacking can be done by background scripts on a webpage. Webpages that are otherwise useful to visit are particularly powerful attack vectors. The resources stolen is not large enough to be noticeable to a victim.
   (b) In 2013, the New York Times reported that the Dual_EC_DRBG random number generator has a potential backdoor. The NSA is the sole editor of this algorithm's standard. The backdoor allows an attacker to fully compromise cryptography based on this tool. RSA Security started using Dual_EC_DRBG as its standard RNG for some of its software after accepting $10 million from the NSA.
   (c) Pegasus is a powerful piece of malware developed by the NSO Group that has frequently been used for surveillance on high-profile targets such as politicians and human-rights activists. Attackers exploited a zero-day vulnerability in the Safari Webkit by sending a file to a victim that appears to be a GIF file, but clicking on it would cause surveillance software to be installed on their iPhone. A 2021 report by Amnesty International shows that it has been used in thousands of attacks over the three preceding years.
   (d) CVE-2023-28771 is a new vulnerability in Zyxel firewall devices being exploited by major botnets such as Mirai. A single packet is sufficient to trigger the exploit and give attacker full control over the target device. The vulnerability exists in an error logging function, which attempts to echo (print) an error message into logfile, but the attacker can change the error message to terminate the echo and cause an arbitrary command to be run as root. (For further details see https://attackerkb.com/topics/N3i8dxpFKS/cve-2023-28771/rapid7-analysis) The Mirai botnet is widely used as a DDoS attack network.

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

   i. [4 points] Which of the CIA principles is being violated? Several answers may be possible.
   ii. [4 points] Classify the malware by method of spread (not payload).

iii. [4 points] Suggest a reasonable counter-measure to defeat or prevent the attack.

Please organize your answer so that by answering all three questions for 1a, then all three questions for 1b, and so on.

2. [10 points] State whether each of the following statements is true or false. For each, explain why.
   (a) [2 points] Some buffer overflow attacks do not overwrite any return address at all.
   (b) [2 points] Minimizing privileges in critical programs can help mitigate the impact of buffer overflow attacks.
   (c) [2 points] Return-Oriented Programming is able to defeat stack canaries.
   (d) [2 points] XSS attacks usually require the attacker to gain full control over the web server first.
   (e) [2 points] If there is a format string vulnerability in OpenSSL, it would be a more serious bug than Heartbleed.

3. [18 points] In 2015, Ion et al. investigated the differences between security practices recommended by experts and non-experts. Experts included hacker conference attendees, professionals and researchers, while non-experts were recruited from MTurk.
   i. [6 points] The biggest difference in security practices between experts and non-experts was to "update software". 35% of experts included this in their top three suggestions while only 2% of non-experts did so. Give two examples of real attacks that could have been prevented if software updates were taken more seriously.
   ii. [6 points] The top advice from non-experts was to use antivirus software, but experts do not agree. Experts rate the effectiveness of antivirus software much lower than non-experts. Explain this by describing how malware can defeat antivirus software.
   iii. [6 points] Experts strongly recommend using a password manager. Explain the benefit of a password manager by describing an attack vector for login compromise from the webmaster's perspective: this attack vector would work even if your web server's defenses are strong enough to resist the attacker. (Hint: Think like an attacker. If you know someone's login name but not their password, and they do not use a password manager, what could you do to obtain their password?)

Programming assignment

Buffer Overflow Vulnerabilities [60 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 check your input against 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 provided one.

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 only checks if a code (shared by all users) is correct, so it does not check your username.
2. `./login -j <username> <password>` will check your username and password against the secret `password.txt` file. This time, it uses a hardcoded canary which is randomized.
3. `./login -k <username> <password>` will check your username and password against the secret `password.txt` file. This time, you cannot expect to overwrite the canary 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 @sfu.ca username**. For example, if my email is wujl@sfu.ca, I can choose wujl123 or wujl666 as my username for this assignment.

(a) [20 points] Using a buffer overflow exploit, log in to the program using the first method. Submit your username and password in a text file called `a1a.txt`, with exactly **two lines**, the first line being the username, and the second line being the password.

(b) [20 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). (Your username can be different from part a, but it must start with your @sfu.ca username.)

(c) [20 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). (Your username can be different from part a, but it must start with your @sfu.ca username.)

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

The exact command I will use to mark your code in (a) is (Do not directly copy this command as the single quote marks will not be copied correctly.):

```
./login -i $(sed -n '1p' a1a.txt) $(sed -n '2p' a1a.txt)
```

You need to make sure this command would execute your submission correctly. The same command is used for marking b and c (replacing the filename/option as appropriate).

Your submission should not rely on any information in the `password.txt` file (it will be different during marking). 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, i.e. the compiler would not re-arrange the order of variables.

Submission instructions

You should submit the following files to Canvas by the deadline:

- a1.pdf, with your answers to the written component and high-level ideas of how you solve the programming assignment.
- a1a.txt, a1b.txt, a1c.txt, with your answers to the programming component. These must be textfiles, not, for example, Word documents renamed with a .txt extension.
- Create a folder with name your @sfu.ca user name appended by '-hm1' and put all the above files in this folder. For example, my sfu username is wujl, and my folder name would be `wujl-hm1`.
- Compress the file and submit. Use the tar command to compress the folder. For example, `tar czvf wujl-hm1.tgz wujl-hm1`. Upload the `tgz` file in Canvas.

You can submit the `tgz` 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. Please remember to name your files correctly.

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.

(Important) LLM Usage. If you have used LLM (Large Language Model, such as ChatGPT) during the assignment, write a section titled with `LLM Usage` to describe how LLM is used in your assignment in `a1.pdf`. If LLM is used while not stated in the pdf file, it will be considered as plagiarism.

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
```