Total points: 80 (+0 bonus)

CMPT479 (2022 Summer) Assignment 1

Due: 11:59 PM, 17th June 2021 (Fri)

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) The Meris botnet broke several records for DDoS volume in 2021. It compromises MikroTik routers with a directory traversal vulnerability that allows remote attackers to steal the admin password of the device to gain full control over it. With 250,000 such routers, Cloudflare estimates that Meris targeted approximately 50 different websites a day, demanding ransoms and DDoSing websites that refused to pay.

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

   i. [4 points] Which of the CIA principles is being violated?
   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.

2. [10 points] State whether each of the following statements is true or false. For each, explain why.

   (a) [2 points] One major reason in why we continue to use C and C++ despite its vulnerabilities is the Saltzer-Schroeder principle of "work factor".

   (b) [2 points] Buffer overflow attacks are made more powerful because of poor implementation of the principle of least privilege.

   (c) [2 points] Stack canaries are an example of the principle of fail-safe defaults.

   (d) [2 points] XSS attacks usually require the attacker to gain full control over the web server first.

   (e) [2 points] The vulnerability behind Heartbleed is more serious than what would have been caused by a format string vulnerability.

3. [13 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. [4 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. [5 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. Why is that? Explain with reference to current malware capabilities.

   iii. [4 points] Experts and non-experts viewed password management differently. Generally, experts recommend using a password manager, while non-experts advocated for rotating passwords frequently, choosing strong passwords, and writing down passwords. Many more experts say they don't remember all their passwords than non-experts (83% vs 48%). Explain the difference in practices with reference to two Saltzer-Schroeder design principles.

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 randomized canary to detect buffer overflows, just like stack canaries.
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 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. 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 e-mail is taowang@sfu.ca, I can choose taowang123 as my username for this assignment. This is meant to discourage plagiarism so that each student's answer will be unique.

The assignment will be marked on 20th June.

(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.)

(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). (Your username can be different from part a, but most start with your @sfu.ca username.)

(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). (Your username can be different from part a, but most 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.

You may choose to use escape characters in your submitted answer or not; the marker will consider both cases.

All of your exploits should work on your own computer, not a dated architecture. Therefore, overwriting the return address is not part of the assignment. 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

You should submit the following files to Coursys 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. These must be textfiles, not, for example, Word documents renamed with a .txt extension.

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 may 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
```