Assignment 3: Command Injection and Path Traversal

CPSC 348 - Computer Security

Fall 2021

100 pts

Due: Thurs., Oct. 21 at 4:40pm

Objectives

- Search for and identify vulnerabilities to command injection and path traversal in an existing system.
- Exploit these vulnerabilities.
- Defend against these vulnerabilities with both whitelists and blacklists.

Files in the Github Repo

You may not change any of the code I give you, especially function signatures or file names, and you may not add external dependencies.

- Executables:
    - `gather` : Allows the user to gather different foods and store them in one of three locations.
    - `check` : Allows the user to check one of three locations to see what foods are currently stored there.
    - `resetDataDirectory` : Empties all storage and resets the files and the directory structure to the default.
- Header files:
    - `gather.h` : When included, provides the function `gather(string, string)` (the C++ function that underlies the `gather` executable documented above).
    - `check.h` : When included, provides the function `check(string)` (the C++ function that underlies the `check` executable documented above).
    - `exploit.h` : The header file that you must implement with your exploit code.
    - `gatherWithValidation.h` : The header file that you must implement with your validation code.
- Object files:

    - `gather.o` : Any file with `#include<gather.h>` must be compiled together with this file.
    - `check.o` : Any file with `#include<check.h>` must be compiled together with this file.
    - `i.o` : Any file with `#include<gather.h>` or `#include<check.h>` must be compiled together with this file.
- Other:
    - `gradingScript.cpp` : This is a script that is a small subset of the script that I will use to grade you. It is compiled with the following command. **If your code doesn't compile and run correctly in this script, you will get a zero.**
        - `g++ gradingScript.cpp exploit.cpp gatherWithValidation.cpp -o gradingScript check.o gather.o i.o`
        - **Note**: Neither `exploit.cpp` nor `gatherWithValidation.cpp` may have a `main` function. If they do, the grading script won't compile. But if you want to create a separate file with a `main` function (e.g., `exploit_main.cpp`) to help you test your code while you're writing it, feel free. You may include it in your final submission if you want.

Finding and Exploiting the Vulnerabilities

Overview

The basic process of this assignment is to find vulnerabilities and exploit them using the command line, and then to save them in a C++ file. More specifically, your ultimate goals with be 1) to create a backdoor in the system by inserting a user with a username and password of your choosing and 2) to exfiltrate the users file (which in this context is a password file).

The instructions below will guide you through the process of finding and exploiting the vulnerabilities using the command line. You will need to create a file `exploit.cpp` that implements `exploit.h` exactly as documented and save the exploits you find in that file.

The file `exploit.cpp` must include `check.h` and `gather.h` , which will allow it to call `check(string)` and `gather(string, string)` . Thus, if your malicious command line input is `./gather "bad_stuff!@#$" pantry` , you will record it in the C++ file as `gather("bad_stuff!@#$", "pantry");` .

☠☠☠☠☠ Happy hacking! ☠☠☠☠☠

Instructions

1. Download the Github repo: https://classroom.github.com/a/_W6JCaka
2. Copy the files onto Ada.
    1. The executables and object files were compiled on Ada, so there are no guarantees that they will work anywhere else.
    2. Also, I will be grading your code on Ada, so if it doesn't run on Ada, it doesn't run.
3. Run `./gather` and read the docs.
4. Run `./check` and read the docs.
5. Tinker with the commands until you feel like you understand them.
6. If at any point you want to reset the "database" to its default state (i.e., the `data` directory and all its contents), run `./resetDataDirectory`.
    1. *Warning*: This will overwrite any changes you have made to `users.txt`.
    2. However, it will not delete any files or directories that you have added to `data` or its subdirectories, and it won't affect any files outside of `data`.
7. Find arguments to `./gather` that cause an error.
    1. Record the inputs in `exploit.cpp`.
8. Find an argument to `./check` that causes an error.
    1. Record the inputs in `exploit.cpp`.
9. *Question 1:* Based on the errors, which command do you think will be susceptible to command injection? Which to path traversal? Record your answer on the worksheet.
10. *&@$% Now let's get malicious! %$@&*
    1. If at any point you are trying to complete one of the following steps by exploiting a vulnerability in the `gather` command and it doesn't seem to be working, try exploiting it in the `check` command (or vice versa).
    2. If you want to challenge yourself, don't open the `data` directory or look at its structure; learn its structure exclusively by exploring it through the injection vulnerabilities you've found.
    3. *Question 2*: As you go through the rest of the steps, record on the worksheet all of the commands you try, successful or not.
11. Find a way to exploit the vulnerabilities in either `gather` or `check` to display the contents of the `data` directory on the console. (You only need to display the immediate contents, not nested/recursive.)
    1. Record the inputs in `exploit.cpp`.

    2. *Question 3*: Did you use command injection or path traversal? Record your answer on the worksheet.
12. Find a way to exploit the vulnerabilities in either `gather` or `check` to display the contents of the user file on the console.
    1. Record the inputs in `exploit.cpp`.
    2. *Question 4*: Did you use command injection or path traversal? Record your answer on the worksheet.
13. Find a way to exploit the vulnerabilities in either `gather` or `check` to insert a new user with the given username and password hash into the user file.
    1. Record the inputs in `exploit.cpp`.
    2. *Question 5*: Did you use command injection or path traversal? Record your answer on the worksheet.
14. Find a way to exploit the vulnerabilities in either `gather` or `check` to exfiltrate the user file to the given location.
    1. Record the inputs in `exploit.cpp`.
    2. *Question 6*: Did you use command injection or path traversal? Record your answer on the worksheet.
15. Last of all: Run `./resetDataDirectory`, compile the grading script using the command in the previous section, and run it (`./gradingScript`) to confirm that all your code works.
16. *Question 7*: There is some basic validation in the `gather` command. At any point, did your non-malicious input to gather a certain food get blocked? If so, what was the food? How does this relate to security vs. usability?

If You Get Stuck

If at any point you get stuck while trying to exploit the vulnerabilities, get inspired by looking at some examples of command injection, such as the following:

- CWE, under "Demonstrative Examples" (esp. #4): https://cwe.mitre.org/data/definitions/77.html
- More examples from PortSwigger: https://portswigger.net/web-security/os-command-injection
- More examples from Cobalt: https://cobalt.io/blog/a-pentesters-guide-to-command-injection
- Also remember that you can tinker with simple Linux commands to figure out which command will do what you want before you worry about how to *inject* that command.

    - Just remember to reset the `data` directory afterwards, so you know for sure what works and what doesn't.

Defending Against the Vulnerabilities with Validation

Now that you've hacked the planet, it's time to fix it. Create a file `gatherWithValidation.cpp` that implements `gatherWithValidation.h` exactly as documented. You will write two validation functions: one with a whitelist and one with a blacklist. Both of your validation functions must implement the principle of fail-safe defaults.

The whitelist must allow only lowercase letters, uppercase letters, and digits. The blacklist must disallow all the command separators listed in this article under "Ways of injecting OS commands": https://portswigger.net/web-security/os-command-injection#ways-of-injecting-os-commands

The ultimate result of your validation should be that *all legitimate inputs still work* and *no malicious inputs work anymore*.

Additionally, both of your functions must use regular expressions (regex). Here are some resources for getting started with regexes in general and/or regexes in C++:

- Here is one of many good online introductions to regexes in general: https://regexone.com/
- Here's a a great overall introduction to regexes in C++: https://www.softwaretestinghelp.com/regex-in-cpp/
    - One weird gotcha: To escape special regex characters, you need a double backslash, not a single one.
- This Stack Overflow answer has a good explanation of what each special character does: https://stackoverflow.com/questions/336210/regular-expression-for-alphanumeric-and-underscores
- More on special characters: https://docs.trifacta.com/display/DP/Supported+Special+Regular+Expression+Characters

Questions

A few final questions to fill out on the worksheet:

- *Question 8*: Read the following article: https://cobalt.io/blog/a-pentesters-guide-to-command-injection
    - What is in-band vs. out-of-band injection/exfiltration?

- *Question 9*: Read the following article: https://portswigger.net/web-security/os-command-injection#ways-of-injecting-os-commands
    - What is “blind” command injection? What is one of the tricks that attackers use to do it successfully?

Submission

Upload `exploit.cpp` and `gatherWithValidation.cpp` to your Github repo by the due date, and turn in the worksheet when you come to class.