OS Assignment 1 Writeup

Palaash Goel - 2021547

October 2022

1 Introduction

This write-up serves as the documentation for the shell code written by me in the C language. There are 3 internal commands - `cd, echo` and `pwd` - and 5 external commands - `ls, cat, date, rm` and `mkdir` - that are handled by the shell.
Internal Commands - As is evident by their classification, these commands are handled by the shell itself without a call to any other binaries.
External Commands - These commands require the use of external binaries. The shell can use both the `fork(), exec() and wait()` family of system calls as well as the `pthread() and system()` family of system calls.
The following sections give a description of what task each command performs along with the options (henceforth also referred to as 'flags') available for each command. Potential vulnerabilities in the form of bugs, attacks and other errors will also be outlined.
Lastly, I have also created multiple test cases that can help the user test the functioning of the shell.

2 Internal Commands

2.1 echo

Outputs the subsequent string argument to the shell command line verbatim (option-based modifications are supported)
Usage: `echo [-option1] [-option2] string_argument`
Available Options:

1. `-n`: Does not print the trailing `\n` character at the end of the string argument
2. `-v`: Informs the user every time echo outputs a message

Vulnerabilities Defended Against:

1. Multiple string input: White-space has not been used as a delimiter for the input string which allows the command to function as intended even if the user enters multiple strings as input.
2. Invalid options entered: In case the user enters an invalid option, the shell gives an error to the user informing them about the same.

2.2 pwd

Outputs the path of the current working directory
Usage: `pwd [-option]` (Both options cannot be used simultaneously)
Available Options:

1. `-L`: Outputs the contents of the `$PWD` environment variable
2. `-P`: Outputs the absolute path of the current working directory after resolving all symbolic links (if any)

Vulnerabilities Defended Against:

1. Use of `getenv($PWD)` leading to a buffer overflow: `getenv($PWD)` has a return type of `char *` due to which it might be possible for a malicious user to write something to the address returned by it and even cause a buffer overflow. Thus, `getenv($PWD)` has been type-casted to `const char *` to resolve that potential threat.

2. Handling the absence of a symbolic link when the `-P` option is used: In order to read symbolic links, the `readlink()` function has been used which returns different integer values based on whether a symbolic link exists or not. While it returns the absolute path of the current working directory if it finds a symbolic link, it raises an error in the absence of a symbolic link. Thus, in case the user's current working directory does not have a symbolic link and the user has still used the `-P` option, the command will function as if the `-L` option was given instead.

2.3 cd

Used to change the current working directory of the current process to another directory, hereby referred to as `path/to/directory`
Usage: `cd [-option] path/to/directory` (Only one flag is allowed at a time for this command)
Available Options:

1. `-L`: Symbolic links are followed while changing directories
2. `-P`: Symbolic links are resolved and absolute paths are used while changing directories

Vulnerabilities Defended Against:

1. Using `chdir(path)` only changes the path of the current process, that is, the `$PWD` environment variable stays the same. However, updating this environment variable ensures a more robust shell especially in the context of its external commands (which are run as different processes altogether) and avoids any potential discrepancies in getting the current working directory for any process that the shell runs.
2. Usage of dot-dot notation: A user might try using this command using the dot-dot notation and thus, the functionality for the same is supported by the shell.

3 External Commands

3.1 ls

Lists the files and directories in the current working directory
Usage: `ls [-option1] [-option2]`
Available Options:

1. `-i`: Outputs the contents of the current working directory along with their index numbers. Output follows the format: `i1 f1 i2 f2 ...`
2. `-m`: Outputs the a comma-separated list of the contents of the current working directory

Vulnerabilities Defended Against:

1. Irretrievable file statistics (relevant for the `-i` option): When a user uses the `-i` option, there can be cases where it is not possible to retrieve a file's statistics (in this case the index number) and the shell gives the user an error informing them of the same.
2. Portability and Thread Safety: According to the GNU C Library, `readdir()` is more portable than `readdir_r()` due to the latter's inability to function in the absence of a caller-specified buffer length as well as its inability to read file names that are "too long". Moreover, even though `readdir_r()` is completely thread safe and `readdir()` is not (as deemed by POSIX-1.2008), in current implementations, the use of `readdir()` is safe enough in most multi-threaded programs and should be completely safe for the use-case at hand considering this shell works with at most 1 thread at a time.

3.2 cat

Lists the contents of a specified file, hereby referred to as `file_name.ext`
Usage: `cat [-option1] [-option2] file_name.ext`
Available Options:

1. `-T`: Outputs the contents of `file_name.ext` and escapes the tab character to display `^I` anytime it is encountered
2. `-E`: Outputs the contents of `file_name.ext` and escapes the newline character to display `\n` anytime it is encountered

Vulnerabilities Defended Against:

1. Buffer overflow using the file name as a vector: The use of `fgets()` with a maximum character limit (`maxChar`) ensures that the input file name containing more characters than this limit does not lead to a potential buffer overflow exploit.

2. Using `feof(fp)` to control a loop: `feof(fp)` should not be used to control a loop (such as a while loop) to read a file till the program reaches the `EOF` character. This is because when such a loop actually reaches the `EOF` character, the loop still needs to finish its last iteration and keeps reading the buffer which may contain garbage/null values. This may cause the input to be malformed, in turn leading to bugs in the program. A simple alternative is to use

   ```
   if (feof(fp)) {
       break;
   }
   ```

   which breaks the loop the moment an `EOF` character is encountered, thus ensuring that no such garbage values are read.

3.3 date

Outputs the date, day and time of a particular timezone (local timezone by default)
Usage: `date [-option1] [-option2]`
Available Options:

1. `-u`: Outputs the UTC (Coordinated Universal Time) date, day and time

2. `-R`: Outputs the local date, day and time in the RFC-3339 format

Vulnerabilities Defended Against:

1. Change of timezone environment variable to UTC for the `-u` option: In order to implement the `-u` option, the timzone environment variable needs to be changed. The program would have a bug if this environment variable was not changed back to the local timezone once the UTC time had been retrieved. Thus, the appropriate measure were taken by ensuring that the timezone environment variable is changed back to the local timezone once the UTC time has been output.

2. Usage of unsupported options: In case, any options other than `-u` and `-R` are entered, the program gives the user an error informing them of the same.

3.4 rm

Removes/deletes a file, say `file_name.ext`, from the current working directory
Usage: `rm [-option] file_name.ext`
Available Options:

1. `-i`: Always gives the user a prompt confirming whether they want to delete the given file or not

2. `-R`: Never prompts the user to confirm whether they want to delete the file, does not do anything in case the user entered the name of a non-existent file

Vulnerabilities Defended Against:

1. User inputs the name of a file that does not exist: The program gives an error to the user in case they pass a string that does not correspond to the name of any file in the current working directory (exception: `-R` has been used).

2. Handling user input if `-i` is used: In case the user uses the `-i` option, it is necessary to control the input that the user can enter. This has been done by not only giving the user an indication of the valid inputs (`printf("Remove file ’%s’ (y/n)? ", file);`), but support for similar inputs including `Y` and `yes` has also been added.

3.5 mkdir

Creates a sub-directory in the current working directory
Usage: `mkdir [-option] dir_name`
Available Options:

1. `-m`: Creates the directory, prompts the user for a mode for it, and sets the directory to that mode
2. `-v`: Outputs a message everytime a directory is created

Vulnerabilities Defended Against:

1. Access to non-creator users and groups: It is important to maintain which user/group can access which file/folder. Not doing so can result in some user accessing files which should only have administrator privileges and using this confidential information in a malicious manner. This is prevented by making sure that by default, only the user who has created the file has full permission to read, write, execute and search for it whereas other users and groups can only read and search for it.
2. User inputs the name of a directory that already exists: In case the user enters the name of a directory that already exists in the current working directory, the program shows an error to the user informing them of the same.

4 Test Cases

In order to check the functioning of all the commands and their options, I came up with the following test cases which, in my opinion, test some of the major potential bugs in the shell. I have tested each test case on my system and the shell passes all of them.
Prerequisites:

- A file named `file.txt` in the directory one level above the current working directory
- A directory named `dir` in the current working directory

Test Case 1: Testing to see if absolute paths for the binaries have been used or not. If relative paths are used, the binaries may not work after a `cd` operation.

```
cd ..
pwd
echo -n This is a test
cd /home
pwd -P
echo hello world
exit
```

Test Case 2: Testing for invalid options

```
echo -k hello world
pwd -L
pwd -L -P
cd ..
cd ..
pwd
exit
```

Test Case 3: Testing `cat` and it's capability to detect whether a file exists in the current working directory or not

```
ls
echo -n hello world
cat file.txt
cd ..
cat file.txt
ls -i -m
exit
```

Test Case 4: Testing date with all flags

```
date&t -u
date -R
date -u -R
date&t -R -u
exit
```

Test Case 5: Testing `mkdir` and if the user can go into the new directory created using it

```
mkdir&t -v newdir
ls&t
cd newdir
echo&t -n hello world
cd&t ..
ls
exit
```

Test Case 6: Testing the functioning of `rm` and its options

```
rm -i dir
rm -f randomdirthatdoesnotexist
cd ..
ls&t -m
exit
```