

Secure Coding in C and C++

Exercise #6: File I/O Vulnerabilities

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213



Software Engineering Institute

Carnegie Mellon University

© 2015 Carnegie Mellon University

Notices

Copyright 2022 Carnegie Mellon University.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material was prepared for the exclusive use of Trainees of online & offline course and may not be used for any other purpose without the written consent of permission@sei.cmu.edu.

This material is distributed by the Software Engineering Institute (SEI) only to course attendees for their own individual study.

Except for any U.S. government purposes described herein, this material SHALL NOT be reproduced or used in any other manner without requesting formal permission from the Software Engineering Institute at permission@sei.cmu.edu.

Although the rights granted by contract do not require course attendance to use this material for U.S. Government purposes, the SEI recommends attendance to ensure proper understanding.

This material and exercises include and/or can make use of certain third party software ("Third Party Software"), which is subject to its own license. The Third Party Software that is used by this material and exercises are dependent upon your system configuration, but typically includes the software identified in the documentation and/or ReadMe files. By using this material and exercises, You agree to comply with any and all relevant Third Party Software terms and conditions contained in any such Third Party Software or separate license file distributed with such Third Party Software. The parties who own the Third Party Software ("Third Party Licensors") are intended third party beneficiaries to this License with respect to the terms applicable to their Third Party Software. Third Party Software licenses only apply to the Third Party Software and not any other portion of the material as a whole.

References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.

CERT® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM22-0263

Sample Program

Caesar cipher decryption program

- Implements simple rotation cipher
- Takes input from files

In a real usage scenario, the decrypted file and the keys file must be kept secret from unauthorized users.

- Should only be usable by intended user
- The secret file can be used by anyone; it's protected by encryption!

Usage

The program accepts a command line argument:

Usage: %s **secret_file** **keys_file** [**output_file**]

The **secret_file** argument specifies the name of the file containing the encrypted text.

The **keys_file** argument specifies the name of the file containing the corresponding “keys” to decrypt each line of the encrypted text.

- **keys_file** must live in home directory, or a subdirectory.
- **keys_file** can only be read with **root** privileges

The program also accepts an optional **output_file** argument.

- If **output_file** is not specified, the program prints the output to **stdout**.
- Otherwise **output_file** must be placed in home directory, or a subdirectory.

The Input Files

All of the files involved are just character files.

Each line contains the ciphertext and corresponding “key” (number of chars to rotate). For example:

Ciphertext	Key	Plaintext
Lzak ak s lwk1	8	This is a test

The lines are delimited by the EOL.

A working set of example files is included.

Exercise Tools

Find and fix any I/O security vulnerabilities in the **caesar** program:

- manual code reading
- compile and test

Use reference material.

- C99 standard
- POSIX standard
- man / help pages
- CERT Secure Coding standards
- *Secure Coding in C and C++*

Assume the program runs with root privileges

Exercises

Find and fix:

- I/O security vulnerabilities

(45 minutes)



Format String Vulnerability

In `usage()`:

Recall that `errmsg` is constructed from `getenv("USER")`

```
fprintf(stderr, errmsg);
```

Missing format specifier

File Validations

Input file and key file should exist and be readable.

- Accomplished by `fopen(filename, "r")`

Keys file should exist and live in home directory (or subdirectory)

- Accomplished using `realpath()`

Output file should not exist but be specified to live in home directory (or subdirectory)

- Accomplished using `realpath()`

Verifying Path lives in Home Directory

```
int in_homedir(char *const filename) {  
  
    struct passwd *pwd = getpwuid(getuid());  
    if (pwd == NULL) {  
        return 0;  
    }  
  
    const size_t len = strlen( pwd->pw_dir);  
    if (strncmp( filename, pwd->pw_dir, len) != 0) {  
        return 0;  
    }  
  
    return 1;  
}
```

Learns home directory
from `/etc/passwd`

Returns true if
path begins with
home directory.

Requires **filename** to be absolute path and canonical!

Validating Canonical Path

```
FILE* canonicalize_and_open(char *const filename, char* mode) {
    char *realpath_res = NULL;
    char *canonical_filename = NULL;
    size_t path_size = (size_t)PATH_MAX;
    if (path_size > 0) {
        canonical_filename = malloc(path_size);
        if (canonical_filename == NULL) {
            errx(1, "Out of memory");
        }
        realpath_res = realpath(filename, canonical_filename);
    }
    char* errstr = NULL;
    if (realpath_res == NULL) {
        errstr = "Realpath failure";
        goto error;
    }
    if (!in_homedir(realpath_res)) {
        errstr = "Not in home directory";
        goto error;
    }
}
```

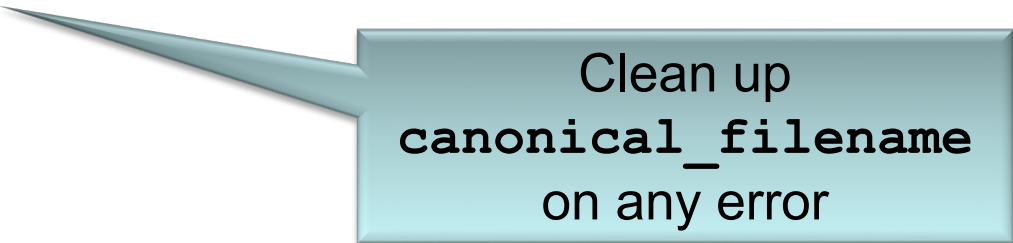
...

Perform home
directory check
on canonical path

Validating Canonical Path, cont.

```
FILE* canonicalize_and_open(char *const filename, char* mode)
{
...
    FILE* fd;
    if ((fd = fopen(realpath_res, mode)) == NULL ) {
        errstr = "File not found";
    }

error:
    free(canonical_filename);
    if (errstr != NULL) {
        errx(1, errstr);
    }
    return fd;
}
```



Clean up
canonical_filename
on any error

Putting It All Together

```
int main(int argc, char *argv[])
{
...
if ((infile = fopen(argv[1], "r")) == NULL)
    errx(1, "Cannot open input file.");

keyfile = canonicalize_and_open(argv[2], "r");

if (argc == 4) {
    outfile = canonicalize_and_open(argv[3], "w");
    oflag=1;
}

...
}
```

Use canonicalization to verify keys and output files

Questions

