

# Secure Coding in C and C++

## Exercise #1: String Review

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](mailto: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](mailto: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

---

## Signal program

- runs with elevated privileges
- sends requested signal to programs

Program is table driven, reading in the list of support signals for any given system from a database.

For now, we have only implemented the help feature, which outputs a description of a specified signal.

# Usage

---

The program accepts a command line argument:

**Usage: %s database\_file**

The **database\_file** argument specifies the name of the file containing the signal database.

The program also accepts an environmental variable **DATA\_PATH**.

- If **DATA\_PATH** is set, the program reads the specified input file from the **DATA\_PATH** directory.
- If not, the program reads the input file from the current working directory.

# The Database File

---

The database is just a character file.

- The first line contains the integral number of entries.
- The remaining lines contain
  - the signal number (small positive integer value)
  - the signal ID (a small string of up to 6 alphanumeric characters)
  - a short string with a description of the signal
- Fields are white-space delimited except for the description.
- The description can contain white space and is delimited by the EOL.

# Example Database File (data.txt)

---

31

1 HUP Hangup

2 INT Interrupt

3 QUIT Quit

4 ILL Illegal instruction

5 TRAP Trace trap

...

# Exercise

---

Review the code.

- manual code reading
- compile and test

Use reference material.

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

Identify defects involving string operations and

- note line number of defect
- note specific problem
- optionally reference C or CERT Secure Coding standard

# Exercise

---

Find string defects  
(30 minutes)

When you are done,  
take a break.





# The `getenv()` function

---

If the specified **name** cannot be found, a null pointer is returned.

```
file = getenv("DATA_PATH");  
if (file != '\0') {
```



Incorrect test

# Getting the File Name

```
char *full_path;
```

Why  
1000?

`malloc()` does  
not clear memory.

```
full_path = (char *)malloc(1000 + strlen(argv[1])+1);
```

```
strncpy(full_path, file, 999);
```

Path may be truncated  
and not null-terminated.

```
if (full_path[strlen(full_path)-1] != '/') {
```

```
    full_path=strcat(full_path, "/");
```

Size returned by  
`strlen()` could be  
arbitrarily long.

```
}    Where is this memory accounted for?
```

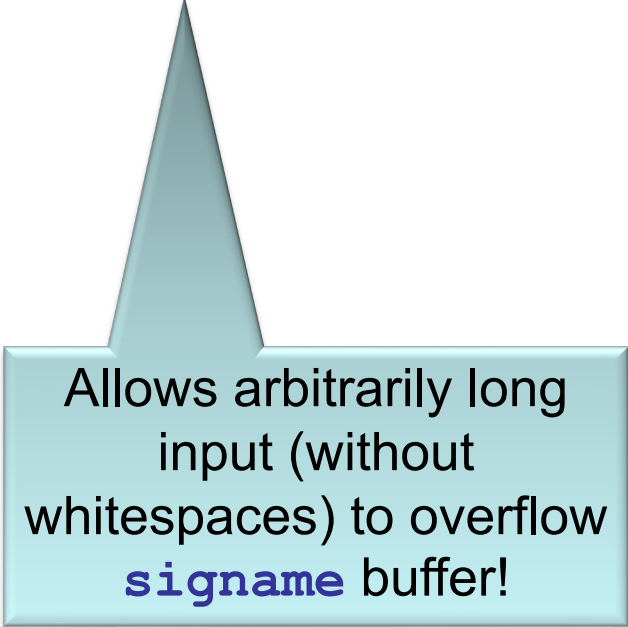
```
full_path=strcat(full_path, argv[1]);
```

Possible buffer  
overflow

# Getting the File Name

---

```
fscanf(in, "%i", &sigdb[i].signum) ;  
fscanf(in, "%s", sigdb[i].signame) ;
```



Allows arbitrarily long  
input (without  
whitespaces) to overflow  
**signame** buffer!

# Other Problems

---

There are many other problems with this code

- integer problems
- file IO problems

We'll talk about integer problems tomorrow.

Memory management issues are covered on day three using a different example

File I/O issues are covered on the fourth day using a different example.

# Questions

