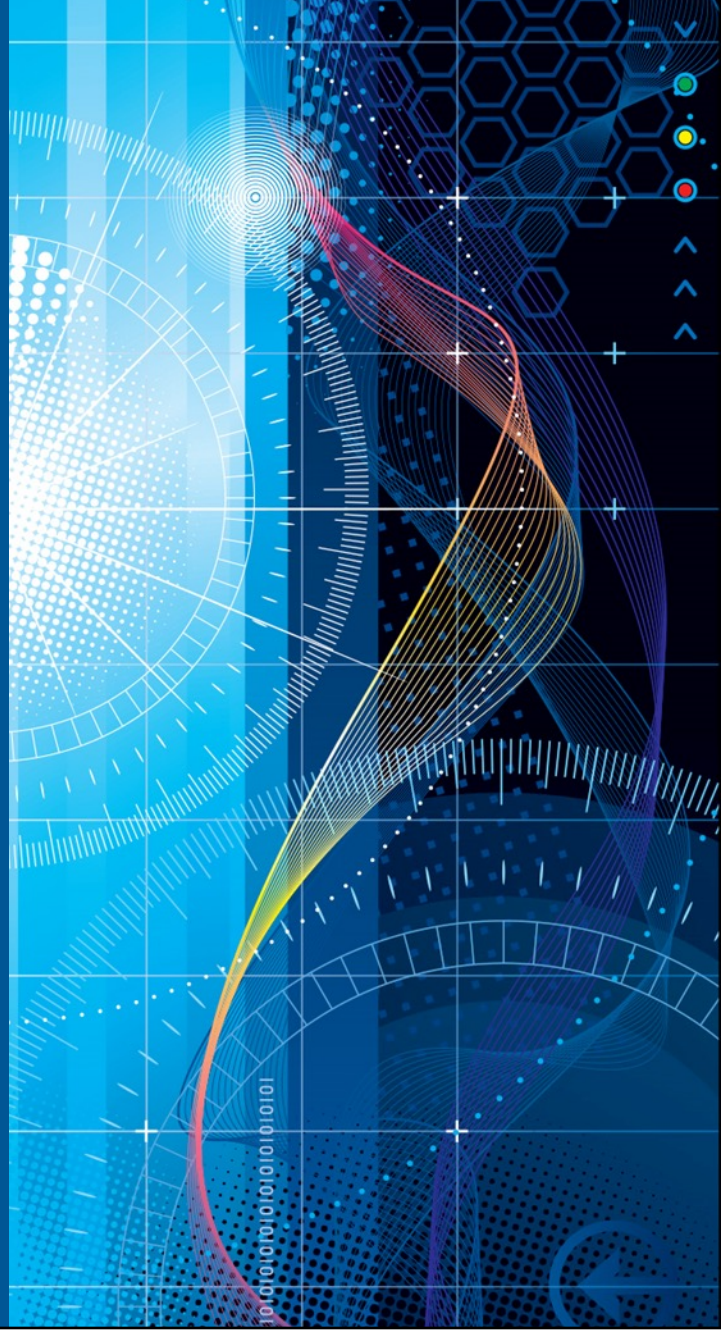
Secure Coding in C and C++

Exercise #1: String Review

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213



Notices

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

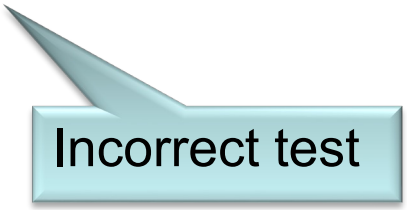
(30 minutes)

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') {
```



Getting the File Name

```
char *full_path;

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

strncpy(full_path, file, 999);

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

  full_path=strcat(full_path,"/");

}

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

Why 1000?

`malloc()` does not clear memory.

Path may be truncated and not null-terminated.

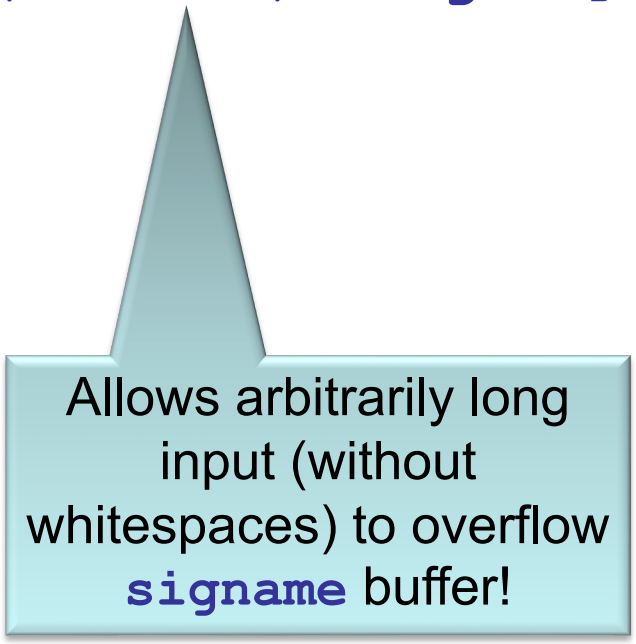
Size returned by `strlen()` could be arbitrarily long.

Where is this memory accounted for?

Possible buffer overflow

Getting the File Name

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



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