CSE 545 - Software Security

Spring 2016

Project Report

Prevention of Format Strings Vulnerability

Saili Bakare (1209547206)
Omkar Salvi (1209536104)

Problem addressed – Prevent format string vulnerability

The prototype of printf family functions converts, formats, and prints its arguments on a specific output stream. The format string contains ordinary characters, which are sent to the standard output stream, and conversion specifications or format specifiers, each of which causes conversion and printing of the corresponding argument to printf family functions. Each conversion specification begins with a % and ends with a conversion character.

The printf family functions are defined with variable length of arguments and format string can be specified statically or dynamically. Compilers like gcc cannot find dynamic format string vulnerabilities during compiling.

Printf family functions fetch the arguments corresponding to format specifiers from the stack. As the stack isn't marked with a boundary, printf family functions does not know if there are less number of arguments that are provided to it and will continue to fetch data from the stack. This results in information leakage, segmentation fault or running a malicious code by overwriting important program flags that control access privileges or overwriting return addresses on the stack, function pointers, etc.

In C, many format string vulnerabilities, particularly those with format specifiers' %n, %x, %p, %s etc. lead to traditional failures such as segmentation fault. Use of %n writes the number of characters printed before the %n to memory value. Inappropriate design or implementation of these formats specifiers can lead to a vulnerability.

In this project, we are providing a security layer that prevents format string vulnerabilities by hooking the printf family of functions to filter dangerous format strings like %n.

Approach to implement the project

We are preventing format string vulnerabilities that occur by use of printf family functions by overriding the C standard library's functions such as printf and its family functions with our own version of these functions.

For this we have written a separate wrapper function for each of the printf family of functions – printf(), fprintf(), sprint(), snprintf(), vprintf(), vfprintf(), vsprint() and vsnprintf() and included all of them in a common mylibrary.c file. We are sanitizing the format string in the wrapper function and calling the corresponding original function with this sanitized format string and corresponding argument list.

We are using the LD_PRELOAD environment variable so that whenever any printf family function is called in the application code, the wrapper function written by us will be called. This wrapper function will sanitize the format string and call the original function. Thus we are maintaining the original functionality of printf functions and also removing the vulnerability.

Implementation

The printf family of functions are hooked to filter and ignore individual %n characters from the format string. All possible combinations of %n as treated in same way as original function would treat. For more details refer README.txt. Doing just this is not enough and any argument corresponding to these placeholders have to be taken care of. Our design chooses to ignore the arguments corresponding to the %n, %x and %p placeholders. Format specifiers other than %n, %x and %p will print garbage value or result in segmentation fault in absence of corresponding arguments.

The wrapper function has the definition same as that of printf family functions. They take format string and variable number of arguments as the input. We are sanitizing the format string

before calling the original function from the overridden function. The wrapper function will scan through the format string character by character. If the characters read are not placeholders then the character will be printed using original function. When a placeholder is encountered the corresponding argument from variable argument list is extracted, interpreted as data type of corresponding placeholder and printed on specified output stream. This continues till end of string i.e. a '\0' (NULL) character is reached.

We have written our own version of printf family functions in a wrapper 'mylibrary.c' file. Developer must compile it as a shared library using command - 'gcc -Wall -fPIC -shared -o mylibrary.so mylibrary.c -ldl'.

Our shared library exports the vulnerable printf family function and then uses dlsym with the RTLD_NEXT pseudohandle to find the original printf family functions. We defined the _GNU_ SOURCE feature test macro in order to get the RTLD_NEXT definition from <dlfcn.h>. RTLD_NEXT finds the next occurrence of a function in the search order after the current library.

What a developer must do to use our approach

1) Developer must compile the wrapper 'mylibrary.c' file into a shared library 'mylibrary.so' file using below command:

gcc -Wall -fPIC -shared -o mylibrary.so mylibrary.c -ldl

2) Every time user program, say 'test1.c' is to be executed as './test1', developer needs to preload our library while invoking the executable of user program using command:

'LD_PRELOAD=./mylibrary.so ./test1'

Limitations of our library containing wrapper function:

Developer cannot use functionality of format specifiers like %n, %x and %p used in printf family of functions. For example, if developer has to count the number of bytes printed by

any of the printf family functions, he/she cannot use %n, instead a separate logic has to be written to do the same.

Future work that would improve our tool:

With current library containing printf family wrapper functions, the format string vulnerabilities related to %n are prevented to quite an extend. We are not considering the case of no or less arguments corresponding to place holders in format string in our design of wrapper functions. In future working on this case would improve our tool to prevent format string vulnerabilities.