Binary Vulnerability Assessment Report

Ioannis Christodoulakos

22/11/2021

CENSUS

Table of Contents

[Executive Summary 2](#_Toc88408450)

[Binary Analysis 2](#_Toc88408451)

[Vulnerability scan results 3](#_Toc88408452)

[Privilege Escalation 3](#_Toc88408453)

[Intercept Posix Message Queue Channel 4](#_Toc88408454)

# Executive Summary

The purpose of this vulnerability scan is to identify potential weaknesses on binary file pwv. The binary is running from administrators to find users with weak passwords. Binary detect passwords which are same with username, ‘1234’ and special characters.

# Binary Analysis

The binary initially checks if the user has root rights for further execution. Then it creates two Posix message queue channels ‘/pwv-hashes’ and ‘/pwv-results’. After that, it creates two child processes with fork. The first child calls a function which pulls the results from channel ‘/pwv-results’ and prints it. The second child call a function which grabs shadow file information and compare the hash values from it with the generated hash values of the weak passwords. For example, if the binary gets username ‘test’ and the hashed password, it tries to generate hash values of weak passwords like ‘test’, ‘1234’ etc. If the password hash value is equal with one of the generated hash, it means that binary has found a weak password and prints it in plain text.

# Vulnerability scan results

## Privilege Escalation

**Description:**

The binary tries to see if user has root rights by checking the euid (Effective ID) to be 0. If the user has root rights it continues the execution, otherwise it exits with printed error ‘This program must run as root’.

Text

Description automatically generated

Checking the effective id of a user to see if has root rights is really bad decision. User can tricky the binary to run it with root privileges if can find an application which can run system commands and has the seuid bit enabled. In more details, the user will run the binary through the application, which has root rights, so the binary would be exploit by the user.

**Demonstration of attack:**

Suppose we have a system with enabled suid bit on vim editor. Vim is editor which accepts commands like :qa to exit and save. Also you can run system commands using the character ‘!’ ex. ‘ :!ls -l ’. So a non-root user can run ‘:! ./pwv 2’

without sudo and get the weak passwords of the system.

**Impact:** Critical Severity Vulnerability**.** A non-root user can run the binary and get credentials of the users in the system.

**Solution:** In order to avoid the exploit, the binary should check the real user id with getuid( ) function. As a result, users can’t exploit the binary and run it through applications with suid bit enabled.

## Spoof Posix Message Queue Channel

**Description:**

The binary after some process, send the results of weak passwords in plain text from channel /pwv-results. After that, a function receives the results from the channel and prints it out in the terminal. The exploit in this situation can occur when a user with low privileges access the channel which the binary created and steal the passwords in plain text.

**Text

Description automatically generated**

**Demonstration of attack:**

#include <stdio.h>

#include <mqueue.h>

#include <assert.h>

#include <stdlib.h>

int main(){

while(1){

/\* Open the message queue for reading \*/

mqd\_t mqd = mq\_open ("/pwv-results", O\_RDONLY);

/\* Get the message queue attributes \*/

struct mq\_attr attr;

char \*buffer = calloc (attr.mq\_msgsize, 1);

assert (buffer != NULL);

/\* Retrieve message from the queue and get its priority level \*/

unsigned int priority = 0;

if ((mq\_receive (mqd, buffer, attr.mq\_msgsize, &priority)) == -1)

printf ("Failed to receive message\n");

else

printf ("Received [priority %u]: '%s'\n", priority, buffer);

/\* Clean up the allocated memory and message queue \*/

free (buffer);

buffer = NULL;

mq\_close (mqd);

}

return 0;

}

Running this program written in C with non-root privileges, the attacker through a continues loop is trying to receive information from the channel. If a root user run the binary and at the same time a user is running this code can intercept the channel and steals the credentials.

**Impact:** Critical Severity Vulnerability**.** A non-root user running this simple exploit can steal passwords in plain text. The worst case scenario would be if a user with weak password is root. The attacker can gain access in the whole system.

**Solution:** In order to avoid the exploit, the binary should change the permissions