```
Date: Mon, 24 Jan 2022 14:05:01 +0000
From: Qualys Security Advisory <qsa@...lys.com>
To: "oss-security@...ts.openwall.com" <oss-security@...ts.openwall.com>
Subject: CVE-2021-3998 and CVE-2021-3999 in glibc's realpath() and getcwd()
```

Hi all,

We discovered two vulnerabilities in the glibc, CVE-2021-3998 in realpath() and CVE-2021-3999 in getcwd(). Patches are now available at (many thanks to Siddhesh Poyarekar and Red Hat Product Security):

https://sourceware.org/git/gitweb.cgi?p=glibc.git;h=ee8d5e33adb284601c00c94687bc907e10aec9bb
https://sourceware.org/git/gitweb.cgi?p=glibc.git;h=f7a79879c0b2bef0dadd6caaaeeb0d26423e04e5

https://sourceware.org/git/gitweb.cgi?p=glibc.git;h=23e0e8f5f1fb5ed150253d986ecccdc90c2dcd5e
https://sourceware.org/git/gitweb.cgi?p=glibc.git;h=472e799a5f2102bc0c3206dbd5a801765fceb39c

Below is a short write-up (which is part of a longer advisory that is mostly unrelated to the glibc and that we will publish at a later date):

CVE-2021-3998: Unexpected return value from glibc's realpath()

While auditing umount and fusermount, we also discovered a vulnerability in the glibc's realpath() function, which is used internally by various programs. Normally, when the output buffer "resolved" that is passed to realpath() is not NULL, then realpath() either returns NULL on failure, or it returns the output buffer "resolved" on success. Unfortunately, since commit c6e0b0b ("stdlib: Sync canonicalize with gnulib") from January 2021, realpath() can mistakenly return a malloc()ated buffer that is neither NULL nor the output buffer "resolved":

```
------------------------------------------------------------------------
430 char *
431 __realpath (const char *name, char *resolved)
432 {
...
437   struct scratch_buffer rname_buffer;
438   return realpath_stk (name, resolved, &rname_buffer);
439 }
------------------------------------------------------------------------
197 static char *
198 realpath_stk (const char *name, char *resolved,
199               struct scratch_buffer *rname_buf)
200 {
...
399   failed = false;
...
403   if (resolved != NULL && dest - rname <= get_path_max ())
404     rname = strcpy (resolved, rname);
...
410   if (failed || rname == resolved)
411     {
412       scratch_buffer_free (rname_buf);
413       return failed ? NULL : resolved;
414     }
415
416   return scratch_buffer_dupfree (rname_buf, dest - rname);
417 }
------------------------------------------------------------------------
```

For example, if the input path "name" is "." and if the current working directory is longer than PATH_MAX, then:

- at line 399, "failed" is set to false;

- at lines 403-404, "rname" is NOT set to "resolved" and "resolved" is left untouched and uninitialized (because "dest - rname" is longer than PATH_MAX);

- the code block at lines 410-414 is skipped (because "failed" is false and "rname" is not "resolved");

- at line 416, scratch_buffer_dupfree() returns a malloc()ated buffer that is NOT the output buffer "resolved".

The consequences of this vulnerability depend on the affected programs; for example, fusermount (a SUID-root program) can disclose sensitive information (pointers) when displaying the contents of a stack-based buffer that is mistakenly left uninitialized by realpath() (we tested this proof of concept on Ubuntu 21.04):

```
------------------------------------------------------------------------
$ gcc -o CVE-2021-3998-fusermount CVE-2021-3998-fusermount.c
$ ./CVE-2021-3998-fusermount > CVE-2021-3998-fusermount.output
...

$ hexdump -C CVE-2021-3998-fusermount.output
00000000  2f 75 73 72 2f 62 69 6e  2f 66 75 73 65 72 6d 6f  |/usr/bin/fusermo|
00000010  75 6e 74 3a 20 65 6e 74  72 79 20 66 6f 72 20 f0  |unt: entry for .|
00000020  83 9b 99 ff 7f 20 6e 6f  74 20 66 6f 75 6e 64 20  |..... not found |
00000030  69 6e 20 2f 65 74 63 2f  6d 74 61 62 0a 0a 2f 75  |in /etc/mtab../u|
00000040  73 72 2f 62 69 6e 2f 66  75 73 65 72 6d 6f 75 6e  |sr/bin/fusermoun|
00000050  74 3a 20 65 6e 74 72 79  20 66 6f 72 20 39 ac b7  |t: entry for 9..|
00000060  a5 a2 7f 20 6e 6f 74 20  66 6f 75 6e 64 20 69 6e  |... not found in|
00000070  20 2f 65 74 63 2f 6d 74  61 62 0a 0a              | /etc/mtab..|
------------------------------------------------------------------------
```

CVE-2021-3999: Off-by-one buffer overflow/underflow in glibc's getcwd()

While studying the vulnerability in realpath(), we also discovered a vulnerability in the glibc's getcwd() function (which is used internally by realpath() to resolve relative pathnames) -- an off-by-one buffer overflow and underflow, but if and only if the "size" of "buf" is exactly 1:

```
------------------------------------------------------------------------
 48 __getcwd (char *buf, size_t size)
 49 {
 ..
 54   size_t alloc_size = size;
 ..
 76     path = buf;
 ..
 80   retval = INLINE_SYSCALL (getcwd, 2, path, alloc_size);
...
100   if (retval >= 0 || errno == ENAMETOOLONG)
101     {
...
110       result = __getcwd_generic (path, size);
------------------------------------------------------------------------
158 __getcwd_generic (char *buf, size_t size)
159 {
...
```

```
187   size_t allocated = size;
...
247     dir = buf;
248
249   dirp = dir + allocated;
250   *--dirp = '\0';
...
262   while (!(thisdev == rootdev && thisino == rootino))
263     {
...
441     }
...
449   if (dirp == &dir[allocated - 1])
450     *--dirp = '/';
...
457   used = dir + allocated - dirp;
458   memmove (dir, dirp, used);
```

If, at line 48, the "size" of "buf" is exactly 1:

- and if, at line 80, the kernel's getcwd() syscall fails with the error ENAMETOOLONG (because the current working directory is longer than PATH_MAX),

- then, at line 110, a generic implementation of getcwd() is called;

- at line 250, a null byte is written to "dirp", which points exactly to "buf" (because "size", and hence "allocated", are exactly 1);

- if the code block at lines 262-441 is skipped entirely (if the current working directory corresponds to the "/" directory),

- then, at lines 449-450, a slash is written to "buf-1" (an off-by-one buffer underflow, because at line 449 "dirp" was still pointing exactly to "buf"),

- and, at lines 457-458, a null byte is written to "buf+1" (an off-by-one buffer overflow, because at line 457 "used" is exactly 2).

It may seem impossible to satisfy the condition at line 100 (the current working directory is longer than PATH_MAX) and the condition at line 262 (the current working directory corresponds to the "/" directory), but in reality we can:

- in a child process:

  - create an unprivileged mount namespace;

  - create a directory longer than PATH_MAX;

  - bind-mount "/" onto this directory;

  - open() this directory and send its file descriptor to the parent process (outside the unprivileged mount namespace);

- in the parent process:

  - receive the file descriptor of this directory (which corresponds to "/" and is longer than PATH_MAX) and fchdir() to it;

  - execute a SUID program that calls getcwd() with a buffer of size 1, which triggers the off-by-one buffer overflow and underflow.

Apparently, this vulnerability was introduced in February 1995 by the very first commit in the glibc's git history (28f540f, "initial import") and could be triggered without an unprivileged mount namespace, by simply chdir()ing to the "/" directory:

```
190 getcwd (buf, size)
...
218     path = buf;
...
226   pathp = path + size;
227   *--pathp = '\0';
...
242   while (!(thisdev == rootdev && thisino == rootino))
243     {
...
351     }
352
353   if (pathp == &path[size - 1])
354     *--pathp = '/';
...
359   memmove (path, pathp, path + size - pathp);
```

Although "the size of buf is exactly 1" is a strong requirement, vulnerable code like the following may exist in the wild:

```
#include <unistd.h>
#include <stdio.h>

int main(int argc, char * argv[]) {
    char buf[4096];
    int len = snprintf(buf, sizeof(buf), "%s: cwd is ", argv[0]);
    if (len <= 0 || (unsigned)len >= sizeof(buf)) return __LINE__;
    if (!getcwd(buf + len, sizeof(buf) - len)) return __LINE__;
    puts(buf);
    return 0;
}
```

Thank you very much! We are at your disposal for questions, comments, and further discussions.

With best regards,

--
the Qualys Security Advisory team