GLIBC REALPATH 缓冲区下溢漏洞

潘亦晟, 515021910384 06/04/2018

1 Introduction

GNU C 库没有正确处理 **getcwd()** 系统调用返回的相对路径，并且没有对缓冲区边界进行检查，造成 glibc 缓冲区下溢漏洞。。该漏洞涉及到 Linux 内核的 **getcwd()** 系统调用和 glibc 的 **realpath()** 函数，可以实现本地提权。

char *getcwd(char *buffer, int maxlen)

- 功能：获取当前工作目录;
- 参数说明：getcwd() 会将当前工作目录的绝对路径复制到参数 buffer 所指的内存空间中, 参数 maxlen 为 buffer 的空间大小;
- 返回值：成功则返回当前工作目录，失败返回 FALSE。

char *realpath(const char *path, char *resolved_path)

- 功能：将参数 path 所指的相对路径转换成绝对路径;
- 返回值：成功则返回指向 resolved_path 的指针，失败返回 NULL。

2 Loophole Analysis

漏洞产生的原因是 **getcwd()** 系统调用在 Linux-2.6.36 版本发生的一些变化，我们知道 **getcwd()** 用于返回当前工作目录的绝对路径，但如果当前目录不属于当前进程的根目录，即从当前根目录不能访问到该目录,**getcwd()** 会在返回的路径前加上 (unreachable)。如果 **realpath()** 使用 **getcwd()** 的返回值作为参数，会发生缓冲区下溢，从而导致任意代码执行，再利用 SUID 程序即可获得目标系统上的 root 权限。

HINT: 当进程使用 **chroot()** 设置了一个新的文件系统根目录，但没有将当前目录的根目录替换成新目录的时候，会出现当前目录不属于当前进程的根目录的情况。

Linux kernel side: 在 2.6.36 版本的 **vfs: show unreachable paths in getcwd and proc** 这次提交，使得当目录不可到达时，会在返回的目录字符串前面加上 (unreachable)。

Listing 1: fs/dcache.c

```
1  // fs/dcache.c
2
3  static int prepend_unreachable(char **buffer, int *buflen)
4  {
5      return prepend(buffer, buflen, "(unreachable)", 13);
6  }
7
8  static int prepend(char **buffer, int *buflen, const char *str, int ↩
       namelen)
9  {
10     *buflen -= namelen;
11     if (*buflen < 0)
12         return -ENAMETOOLONG;
13     *buffer -= namelen;
14     memcpy(*buffer, str, namelen);
15     return 0;
16 }
17
18 /*
19 * NOTE! The user-level library version returns a
20 * character pointer. The kernel system call just
21 * returns the length of the buffer filled (which
22 * includes the ending '\0' character), or a negative
23 * error value. So libc would do something like
24 *
25 *    char *getcwd(char * buf, size_t size)
26 *    {
27 *        int retval;
28 *
29 *        retval = sys_getcwd(buf, size);
30 *        if (retval >= 0)
31 *            return buf;
32 *        errno = -retval;
33 *        return NULL;
34 *    }
35 */
36 SYSCALL_DEFINE2(getcwd, char __user *, buf, unsigned long, size)
37 {
38     int error;
39     struct path pwd, root;
40     char *page = __getname();
41
42     if (!page)
43         return -ENOMEM;
44
45     rcu_read_lock();
46     get_fs_root_and_pwd_rcu(current->fs, &root, &pwd);
```

```
47
48      error = -ENOENT;
49      if (!d_unlinked(pwd.dentry)) {
50          unsigned long len;
51          char *cwd = page + PATH_MAX;
52          int buflen = PATH_MAX;
53
54          prepend(&cwd, &buflen, "\0", 1);
55          error = prepend_path(&pwd, &root, &cwd, &buflen);
56          rcu_read_unlock();
57
58          if (error < 0)
59              goto out;
60
61          /* Unreachable from current root */
62          if (error > 0) {
63              error = prepend_unreachable(&cwd, &buflen); // 当路径不可到达↩
                    时，添加前缀
64              if (error)
65                  goto out;
66          }
67
68          error = -ERANGE;
69          len = PATH_MAX + page - cwd;
70          if (len <= size) {
71              error = len;
72              if (copy_to_user(buf, cwd, len))
73              error = -EFAULT;
74          }
75      } else {
76          rcu_read_unlock();
77      }
78
79  out:
80      __putname(page);
81      return error;
82  }
```

Libc side: 看到在引进了 unreachable 这种情况后，仅仅判断返回值大于零是不够的，它并不能很好地区分开究竟是绝对路径还是不可到达路径。然而很可惜的是，glibc 就是这样做的，它默认了返回的 buf 就是绝对地址。当然也是由于历史原因，在修订 getcwd 系统调用之前，glibc 中的 getcwd() 库函数就已经写好了，于是遗留下了这个不匹配的问题。

由于 glibc 仍认为内核 **getcwd()**, 会返回绝对路径名，所以在函数 **realpath()** 中，仅仅依靠 **name[0] !=** '/' 就断定参数是一个相对路径，而忽略了以 **unreachable** 开头的不可到达路径。。

realpath() 展开所有符号链接，并解析对由 path 指定的以 null 结尾的字符串中的/./，/../和额外的'/' 字符的引用，以生成规范化的绝对路径名，最后将解析结果存放到 resolved_path 指向的地址中。

Listing 2: stdlib/canonicalize.c

```
 char *  __realpath (const char *name, char *resolved)
{
    ...
    # When resolving a relative pathname, getcwd() is called:
    if (name[0] != '/')
    {
        if (!__getcwd (rpath, path_max))
        {
           rpath[0] = '\0';
           goto error;
        }
       dest = __rawmemchr (rpath, '\0');
    }
   else
    ...
   # Loop over all name components:
   for (start = end = name; *start; start = end)
   {
    ...
    # If the name component is "..", remove it. This underflows the
    # buffer if rpath does not contain a starting slash.
      else if (end - start == 2 && start[0] == '.' && start[1] == '.')
      {
           /* Back up to previous component, ignore if at root already. ↩
                       */
           if (dest > rpath + 1)
             while ((--dest)[-1] != '/');
      }
      else
    # The name component is not ".", "..", so copy the name to dest.
      {
           size_t new_size;

           if (dest[-1] != '/')
             *dest++ = '/';
    ...
```

当传入的 name 不是一个绝对路径，比如../../x，realpath() 将会使用当前工作目录来进行解析，而且默认了它以 / 开头。解析过程是从后先前进行的，当遇到../ 的时候，就会跳到前一个

/，但这里存在一个问题，没有对缓冲区边界进行检查，如果缓冲区不是以 / 开头，则函数会越过缓冲区，发生溢出。所以当 getcwd 返回的是一个不可到达路径 (unreachable)/时，../../x 的第二个../ 就已经越过了缓冲区，然后 x 会被复制到这个越界的地址处。

3 Patch

Patch One

在发生溢出的地方加了一个判断，当 dest == rpath 的时候，如果 *dest != '/'，则说明该路径不是以 / 开头，便触发报错。

Listing 3: Patch 1

```
--- stdlib/canonicalize.c    2018-01-05 07:28:38.000000000 +0000
+++ stdlib/canonicalize.c    2018-01-05 14:06:22.000000000 +0000
@@ -91,6 +91,11 @@
         goto error;
     }
         dest = __rawmemchr (rpath, '\0');
+/* If path is empty, kernel failed in some ugly way. Realpath
+has no error code for that, so die here. Otherwise search later
+on would cause an underrun when getcwd() returns an empty string.
+Thanks Willy Tarreau for pointing that out. */
+      assert (dest != rpath);
     }
     else
     {
@@ -118,8 +123,17 @@
         else if (end - start == 2 && start[0] == '.' && start[1] == '.')
         {
         /* Back up to previous component, ignore if at root already.  */
-            if (dest > rpath + 1)
-            while ((--dest)[-1] != '/');
+            dest--;
+            while ((dest != rpath) && (*--dest != '/'));
+            if ((dest == rpath) && (*dest != '/') {
+            /* Return EACCES to stay compliant to current documentation:
+            "Read or search permission was denied for a component of the
+            path prefix." Unreachable root directories should not be
+            accessed, see https://www.halfdog.net/Security/2017/LibcRealpathBufferUnderflow/ */
+             __set_errno (EACCES);
+            goto error;
+      }
+      dest++;
 }
```

```
33      else
34      {
```

Patch Two

linux 最终采用的方案是对 getcwd() 返回的路径 path 进行检查，如果确定 **path[0] == '/'**，说明是绝对路径，返回。否则转到 **generic_getcwd()**(内部函数，源码里看不到) 进行处理：

Listing 4: Patch 2

```
1  --- a/sysdeps/unix/sysv/linux/getcwd.c
2  +++ b/sysdeps/unix/sysv/linux/getcwd.c
3  @@ -76,7 +76,7 @@ __getcwd (char *buf, size_t size)
4      int retval;
5
6      retval = INLINE_SYSCALL (getcwd, 2, path, alloc_size);
7  -   if (retval >= 0)
8  +   if (retval > 0 && path[0] == '/')
9      {
10     #ifndef NO_ALLOCATION
11         if (buf == NULL && size == 0)
12 @@ -92,10 +92,10 @@ __getcwd (char *buf, size_t size)
13         return buf;
14     }
15
16 -  /* The system call cannot handle paths longer than a page.
17 -     Neither can the magic symlink in /proc/self.  Just use the
18 +  /* The system call either cannot handle paths longer than a page
19 +     or can succeed without returning an absolute path.  Just use the
20        generic implementation right away.  */
21 -   if (errno == ENAMETOOLONG)
22 +   if (retval >= 0 || errno == ENAMETOOLONG)
23     {
24     #ifndef NO_ALLOCATION
25         if (buf == NULL && size == 0)
```

4 Exploit

未成功!

5 Danger Range

5.1 Package(eglibc)

Ubuntu Version	Effect
Upstream	needed
Ubuntu 12.04 ESM (Precise Pangolin)	released (2.15-0ubuntu10.21)
Ubuntu 14.04 LTS (Trusty Tahr)	released (2.19-0ubuntu6.14)
Ubuntu 16.04 LTS (Xenial Xerus)	DNE
Ubuntu 17.10 (Artful Aardvark)	DNE
Ubuntu 18.04 LTS (Bionic Beaver)	DNE

5.2 Package glibc

Ubuntu Version	Effect
Upstream	needed
Ubuntu 12.04 ESM (Precise Pangolin)	DNE
Ubuntu 14.04 LTS (Trusty Tahr)	DNE
Ubuntu 16.04 LTS (Xenial Xerus)	released (2.23-0ubuntu10)
Ubuntu 17.10 (Artful Aardvark)	released (2.26-0ubuntu2.1)
Ubuntu 18.04 LTS (Bionic Beaver)	not-affected (2.26-0ubuntu2.1)

5.3 Package dietlibc

Ubuntu Version	Effect
Upstream	needs-triage
Ubuntu 12.04 ESM (Precise Pangolin)	DNE
Ubuntu 14.04 LTS (Trusty Tahr)	needs-triage
Ubuntu 16.04 LTS (Xenial Xerus)	needs-triage
Ubuntu 17.10 (Artful Aardvark)	needs-triage
Ubuntu 18.04 LTS (Bionic Beaver)	needs-triage

5.4 Package musl

Ubuntu Version	Effect
Upstream	needs-triage
Ubuntu 12.04 ESM (Precise Pangolin)	DNE
Ubuntu 14.04 LTS (Trusty Tahr)	needs-triage
Ubuntu 16.04 LTS (Xenial Xerus)	needs-triage
Ubuntu 17.10 (Artful Aardvark)	needs-triage
Ubuntu 18.04 LTS (Bionic Beaver)	needs-triage