Authorization bypass and symlink attack in multipathd (CVE-2022-41974 and CVE-2022-41973)

From: Qualys Security Advisory via Fulldisclosure <fulldisclosure () seclists org>
Date: Mon, 24 Oct 2022 15:18:41 +0000

```
Qualys Security Advisory

Leeloo Multipath: Authorization bypass and symlink attack in multipathd
(CVE-2022-41974 and CVE-2022-41973)


=========================================================================
Contents
=========================================================================

Summary
CVE-2022-41974: Authorization bypass
CVE-2022-41973: Symlink attack
Acknowledgments
Timeline


=========================================================================
Summary
=========================================================================

We discovered two local vulnerabilities (an authorization bypass and a
symlink attack) in multipathd, a daemon that is running as root in the
default installation of (for example) Ubuntu Server:

https://ubuntu.com/server/docs/device-mapper-multipathing-introduction
https://github.com/opensvc/multipath-tools

We combined these two vulnerabilities with a third vulnerability, in
another package that is also installed by default on Ubuntu Server, and
obtained full root privileges on Ubuntu Server 22.04; other releases are
probably also exploitable. We will publish this third vulnerability, and
the complete details of this local privilege escalation, in an upcoming
advisory.

The authorization bypass (CVE-2022-41974) was introduced in February
2017 (version 0.7.0) by commit 9acda0c ("Perform socket client uid check
on IPC commands"), but earlier versions perform no authorization checks
```

at all: any unprivileged local user can issue any privileged command to
multipathd.

The symlink attack (CVE-2022-41973) was introduced in May 2018 (version
0.7.7) by commit 65d0a63 ("functions to indicate mapping failure in
/dev/shm"); the vulnerable code was hardened significantly in May 2020
(version 0.8.5) by commit 40ee3ea ("simplify failed wwid code"), but it
remains exploitable nonetheless.

CVE-2022-41974: Authorization bypass

The multipathd daemon listens for client connections on an abstract Unix
socket (conveniently, the multipathd binary itself can act as a client,
if executed with non-option arguments; we use this feature extensively
in this advisory to connect and send commands to the multipathd daemon):

```
$ ps -ef | grep 'multipath[d]'
root         377       1  0 13:55 ?        00:00:00 /sbin/multipathd -d -s

$ ss -l -x | grep 'multipathd'
u_str LISTEN 0      4096        @/org/kernel/linux/storage/multipathd 18105
```

The commands sent by a client to multipathd are composed of keywords,
and internally, each keyword is identified by a different bit; for
example, "list" is 1 (1<<0), "add" is 2 (1<<1), and "path" (which
requires a parameter) is 65536 (1<<16):

```
155 load_keys (void)
...
163         r += add_key(keys, "list", LIST, 0);
164         r += add_key(keys, "show", LIST, 0);
165         r += add_key(keys, "add", ADD, 0);
...
183         r += add_key(keys, "path", PATH, 1);
```

```
 53 #define LIST            (1ULL << __LIST)
 54 #define ADD             (1ULL << __ADD)
 ..
 69 #define PATH            (1ULL << __PATH)
```

```
  6 enum {
  7         __LIST,                 /*  0 */
  8         __ADD,
 ..
 23         __PATH,
```

In turn, each command is associated with a handler (a C function) by its
fingerprint -- the bitwise OR of its constituent keywords; for example,
the command "list path PARAM" is associated with cli_list_path() by the
fingerprint 65537 (LIST+PATH=1+65536), and the command "add path PARAM"
is associated with cli_add_path() by the fingerprint 65538
(ADD+PATH=2+65536):

```
1522 void init_handler_callbacks(void)
....
1527         set_handler_callback(LIST+PATH, cli_list_path);
....
```

```
1549         set_handler_callback(ADD+PATH, cli_add_path);
```

```
321 static uint64_t
322 fingerprint(const struct _vector *vec)
...
325         uint64_t fp = 0;
...
331         vector_foreach_slot(vec, kw, i)
332                 fp += kw->code;
333
334         return fp;
```

```
 89 static struct handler *
 90 find_handler (uint64_t fp)
 ..
 95         vector_foreach_slot (handlers, h, i)
 96                 if (h->fingerprint == fp)
 97                         return h;
 98
 99         return NULL;
```

When multipathd receives a command from a client, it first performs an authentication check and an authorization check (both at line 491):

```
431 static int client_state_machine(struct client *c, struct vectors *vecs,
...
485         case CLT_PARSE:
486                 c->error = parse_cmd(c);
487                 if (!c->error) {
...
491                         if (!c->is_root && kw->code != LIST) {
492                                 c->error = -EPERM;
...
495                         }
496                 }
497                 if (c->error)
...
501                 else
502                         set_client_state(c, CLT_WORK);
...
522         case CLT_WORK:
523                 c->error = execute_handler(c, vecs);
```

- Authentication: if the client's UID (obtained from SO_PEERCRED) is 0 (i.e., if is_root is true), then the client is privileged; otherwise, it is unprivileged.

- Authorization: if the client is privileged, it is allowed to execute any commands; otherwise, only unprivileged LIST commands are allowed (i.e., commands whose first keyword is either "list" or "show").

Attentive readers may have noticed that multipathd does not, in fact, calculate the fingerprint of a command by bitwise-ORing its constituent keywords, but by arithmetic-ADDing them (at line 332). While these two operations are equivalent if no keyword is repeated, we (attackers) can send a seemingly unprivileged command (whose first keyword is "list") but whose fingerprint matches a privileged command (by repeating the "list" keyword): we can exploit this flaw to bypass multipathd's authorization check.

For example, we are not allowed to execute "add path PARAM" (whose fingerprint is 2+65536=65538) because the first keyword is not "list",

but we are allowed to execute the equivalent "list list path PARAM"
(whose fingerprint is also 1+1+65536=65538, instead of 1|1|65536=65537)
because the first keyword is "list" (the multipathd daemon below replies
"blacklisted" because PARAM is an invalid path, not because the command
is denied):

```
------------------------------------------------------------------------
$ multipathd add path PARAM
permission deny: need to be root

$ multipathd list list path PARAM
blacklisted
------------------------------------------------------------------------
```

This authorization bypass greatly enlarges the attack surface of
multipathd: 34 privileged command handlers become available to local
attackers, in addition to the 23 unprivileged command handlers that are
normally available. We audited only a few of these command handlers,
because we quickly discovered a low-hanging vulnerability (a symlink
attack) in one of them.

```
========================================================================
CVE-2022-41973: Symlink attack
========================================================================
```

multipathd operates insecurely, as root, in /dev/shm (a sticky,
world-writable directory similar to /tmp). The vulnerable code (in
mark_failed_wwid()) may be executed during the normal lifetime of
multipathd, but a local attacker can force its execution by exploiting
the authorization bypass CVE-2022-41974; for example, by adding a
"whitelisted, unmonitored" device to multipathd:

```
------------------------------------------------------------------------
$ multipathd list devices | grep 'whitelisted, unmonitored'
    sda1 devnode whitelisted, unmonitored
    ...

$ multipathd list list path sda1
fail
------------------------------------------------------------------------
```

This command, which is equivalent to "add path sda1", results in the
following system-call trace (strace) of the multipathd daemon:

```
------------------------------------------------------------------------
386 openat(AT_FDCWD, "/dev/shm/multipath/failed_wwids", O_RDONLY|O_DIRECTORY) = -1 ENOENT
(No such file or directory)
387 mkdir("/dev", 0700)                     = -1 EEXIST (File exists)
388 mkdir("/dev/shm", 0700)                 = -1 EEXIST (File exists)
389 mkdir("/dev/shm/multipath", 0700)       = 0
390 mkdir("/dev/shm/multipath/failed_wwids", 0700) = 0
391 openat(AT_FDCWD, "/dev/shm/multipath/failed_wwids", O_RDONLY|O_DIRECTORY) = 12
392 getpid()                                = 375
393 openat(12, "VBOX_HARDDISK_VB60265ca5-df119cb6.177", O_RDONLY|O_CREAT|O_EXCL, 0400) = 13
394 close(13)                              = 0
395 linkat(12, "VBOX_HARDDISK_VB60265ca5-df119cb6.177", 12, "VBOX_HARDDISK_VB60265ca5-
df119cb6", 0) = 0
396 unlinkat(12, "VBOX_HARDDISK_VB60265ca5-df119cb6.177", 0) = 0
397 close(12)                              = 0
------------------------------------------------------------------------
```

- at line 389, the directory "/dev/shm/multipath" is created, if it does
  not exist already;

- at line 390, the directory "/dev/shm/multipath/failed_wwids" is
  created, if it does not exist already;

- at lines 391-397, the empty file
  "/dev/shm/multipath/failed_wwids/VBOX_HARDDISK_VB60265ca5-df119cb6" is
  created, if it does not exist already (its name is the "World Wide ID"
  of the added device).

multipathd is therefore vulnerable to two different symlink attacks:

1/ if we (attackers) create an arbitrary symlink "/dev/shm/multipath",
then we can create a directory named "failed_wwids" (user root, group
root, mode 0700) anywhere in the filesystem;

2/ if we create an arbitrary symlink "/dev/shm/multipath/failed_wwids",
then we can create a file named "VBOX_HARDDISK_VB60265ca5-df119cb6"
(user root, group root, mode 0400, size 0) anywhere in the filesystem.

These two symlink attacks are very weak, because we do not control the
name, user, group, mode, or contents of the directory or file that we
create; only its location. Despite these limitations, we were able to
combine multipathd's vulnerabilities (authorization bypass and symlink
attack) with a third vulnerability (in another package), and obtained
full root privileges on Ubuntu Server 22.04; we will publish this third
vulnerability in an upcoming advisory.

Side note: initially, we thought that the symlink attack 1/ would fail,
because /dev/shm is a sticky world-writable directory, and the kernel's
fs.protected_symlinks is 1 by default; to our great surprise, however,
it succeeded. Eventually, we understood that only the final component of
a path is protected, not its intermediate components; for example, if
/tmp/foo is a symlink, then an access to /tmp/foo itself is protected,
but an access to /tmp/foo/bar is not. Interestingly, this weakness was
already pointed out in 2017 by Solar Designer, and the original
Openwall, grsecurity, and Yama protections are not affected:

https://www.openwall.com/lists/kernel-hardening/2017/06/06/74

==========
Acknowledgments
==========

We thank Martin Wilck and Benjamin Marzinski for their hard work on this
release, and the SUSE Security Team for their help with this disclosure.
We also thank the members of linux-distros@openwall.

==========
Timeline
==========

2022-08-24: Advisory sent to security@suse.

2022-10-10: Advisory and patches sent to linux-distros@openwall.

2022-10-24: Coordinated Release Date (15:00 UTC).

Current thread:

Authorization bypass and symlink attack in multipathd (CVE-2022-41974 and CVE-2022-41973)
Qualys Security Advisory via Fulldisclosure (Oct 30)