Systemd vsock sshd

Greg Dahlman

Dec 28, 2025

CONTENTS:

1 Overview 1

2 Background 3

3 Problem 5

4 Possible Mitigations 9

5 Additional Information 13

Bibliography 17

CHAPTER

ONE

OVERVIEW

- **Systemd v256 change** - When the *openssh-server* package is installed on a VM with vsock support, systemd now automatically starts an *sshd* instance that listens on the **af_vsock** socket in the **global network namespace** without any manual configuration.
- **vsock exists in the global namespace** - Unlike `af_inet` sockets, vsock connections are not bound to a particular network namespace. By default they are visible to every namespace on the host.
- **Violation of namespace isolation** - Users normally expect that services bound in one namespace cannot be accessed from another namespace. The global-namespace vsock listener breaks this expectation, allowing processes in any namespace to reach the *sshd* instance.
- **Enables malware and lateral movement** - Malicious code that runs inside a container or sandbox can exploit the exposed vsock listener to connect to the host's SSH daemon, thereby **bypassing network-segmentation rules** and **moving laterally** across the host. This creates a powerful attack vector for malware that can spread from isolated workloads to the host or other guests without needing traditional network exposure.
- **Hard-to-audit data path** - vsock provides a low-level, kernel-backed IPC channel that is opaque to many security tools. It can be used by sandboxed programs or containers to send commands or data to sandboxed programs or containers in a way that is difficult to monitor or audit.
- **vsock ss/netstat invisibility** - The visibility feature is isolated in a network namespace, letting processes evade detection in an already hard-to-audit subsystem.
- **Trivial extension of active threats** - If not already being leveraged, it would be trivial to extend [BRICKSTORM] and [shai-hulud] to take advantage of vsock as described above. As [BRICKSTORM] is already known to be leveraging vsock on VMware, it is unlikely it is not already being used by APTs.

CHAPTER

TWO

BACKGROUND

What is [vsock]?

> The *vf_vsock* address family was introduced to help facilitate communication between virtual machines and the hypervisor. The project intentionally implemented a *vf_inet* like *socket()* interface to simplify development.
>
> The project delivered on that goal, which also makes it an attractive subsystem to use in ways that the original project may not have expected. Because it is so similar to *af_inet* those new uses can be problematic if they do not understand the very diffrent design assumptions and contract.
>
> Specificly, vsock is a project that was built under a trusted execution model, where a trusted process (hypervisor) had control over untrusted processes (virtual machines).
>
> As it is closer to IPC than network in implementation, many of the tools adminstrators would use either require additional flags or simply do not support *af_vsock*. As an example, most of the common audit and IDS systems, or tools like Fail2Ban, ebtables, etc… cannot be used to limit access or track traffic.

2.1 Vsock Characteristics

- Address Family int: 40
- syscall example: *connect(fd, AF=40 cid:2 port:22, 16)*

There are two actively used known address:

- VMADDR_CID_LOCAL (1)
- VMADDR_CID_HOST (2)

And a * bind address:

- VMADDR_CID_ANY (-1U)

There are a number of device files that are emmited

```
crw------- 1 root root 10, 137 Dec 11 13:53 /dev/vhci
crw-rw---- 1 root kvm  10, 238 Dec 11 13:55 /dev/vhost-net
crw-rw---- 1 root kvm  10, 241 Dec 20 18:15 /dev/vhost-vsock
crw------- 1 root root 10, 261 Dec 11 13:53 /dev/vmci
crw-rw-rw- 1 root root 10, 262 Dec 11 13:53 /dev/vsock
```

The device file of intrest in this case is */dev/vsock*. While a well behaved program should use device file permissions, and there is a IOCTL to find the VM's hypervisor assigned CID, it is possible to *connect()*, *bind()*, or *listen()* without access to, or in the complete absense of */dev/vsock*. Common tools like `socat` will produce a warning yet still function without our without access to */dev/vsock*.

CHAPTER

THREE

PROBLEM

Systemd ssh vsock listener

3.1 Context

Item	Description
Feature	Systemd Core >= v256 - automatically starts a socket activated SSH daemon on `af_vsock:22` on VMs when a openssh-server package is installed.
Target Port	`af_vsock` (40), port **22**.
Intended Benefit	Simplify user accessibility to containers/KVM guests.
Affected Environments	Linux VMs running systemd v256 or later that match the vm detection rules in `systemd-ssh-generator`
Key Stakeholders	Projects that depend on namespace isolation, corporate compliance officers, legal, Distros, DevOps, end-users, CSPs, customers subject to GDPR, HIPAA, PCI-DSS, etc.

3.2 Root Cause Analysis

Root Cause	Supporting Evidence
Use Case Mis-alignment	The design goal "Zero-Config" prioritized convenience over strict network isolation, causing a conflict with other use cases that may depend on network segmentation and namespace isolation.
Insufficient Threat Modeling	The vsock interface was treated as a non-exposed, "internal" channel. Threat modeling did not consider that containers with access to the global namespace `af_vsock` are effectively *trusted* by the host kernel, giving them access to resources that directly conflicts with many isolation and reduced privlage models.
Nonstandard Configuraiton / Default Behavior	The functionality is in the base `systemd` packages, with no clear guidance that the feature must be disabled for regulated workloads or to maintain netns isolation.
Excessive Permissions	LSM exceptions and common CRI seccomp profiles allow obsolete or namespace incompatible af_families and should reconsider defaults.

3.3 Impact Assessment

Area	Impact
Operational	The socket activated service may be missed in an audit, as hypervisor configuration, options, or cloud instance type may result in the interface appearing or disappearing in unexpected ways.
Security	Unrestricted global namespace SSH ingress + and L4 bridge from any pod → potential lateral movement, privilege escalation, data exfiltration.
Compliance	May violate GDPR Article 32 (security of processing), HIPAA Security Rule (Principle of Least Privilege), PCI-DSS Requirement 11.3.4, …
Legal	Some customers/vendors may have potential statutory damages or contractual penalties.
Business	Loss of customer trust, increased audit burden.

3.4 Technical Details & Detection Challenges

Issue	Detail	Why It Matters
vsock is not 'seen' by typical audit tools	- vsock sockets use the **AF_VSOCK** address family. Standard utilities often require extra flags or lack support, so a vsock-bound SSH daemon may never appear.	A security-operations team may not realize a listening SSH service is running on the host until an exploit is observed. The absence of visibility can delay incident detection.
Automatic activation via socket-activa unit	- The `openssh-server` package installation triggers the `systemd-ssh-generator` componant of `systemd`. When the `openssh-server` package is installed, the socket unit is enabled **by default**, and systemd automatically starts `sshd@*.service` whenever a connection attempt on vsock port 22 occurs. This activation happens *without* any explicit `systemctl enable` from the administrator.	Administrators may assume that installing `openssh-server` only gives them a network-bound SSH daemon, unaware that additional services also being provisioned in the global namespace.
Disabling requires a kernel-boot parameter or passing a credential file	- The upstream documentation for `systemd-ssh-generator` states that the supported way to disable the automatic binding to the AF_VSOCK and AF_UNIX sockets is by passing a kernel boot string `systemd.ssh_listen=` or passing in a credential file - This approach is a departure from typical configuration options.	Complicates adminstration and auditing.

3.4.1 Why These Technical Nuances Were Overlooked

Factor	Explanation	Root-Cause Connection
AF_VSOC is "invisible"	Most security tooling is built around AF_INET and AF_UNIX sockets; AF_VSOCK is a niche family that rarely appears in standard logs.	Root cause: *Insufficient threat modeling* - the team did not consider that a local socket family could be a covert channel with limited visibility.
AF_VSOC security is misunderstood	Most of AF_VSOCK security depends on a hypervisor controlling the allocation of VM CID assignments. mTLS may not be required to prevent MtM attacks but mutual authentication is required.	Root cause: *Kernel Feature Intent Matters* - Even a well-documented kernel API can have subtle restrictions. The sutability of a feature should be checked before re-using it in a different context.
Socket-acti is silent	Systemd's socket activation shows up in `systemctl list-units`; As socket activation is not expected many admins may never query it.	Root cause: *Unexpected behavior* - With the installation of `openssh-server` vsock listener is auto-enabled independent of the openssh configuration files that many would expect.
Process for disabling is obscure	Disabling the vsock sshd via a boot parameter is a hard-coded kernel setting; it is not a per-service toggle.	Root cause: *Policy mis-alignment* - the design did not provide a fine-grained way to disable the service for regulated workloads or users who leverage least privlages as defence in depth.

3.5 Final Take-away

The Systemd vsock-SSH feature inadvertently opened a **lateral-movement path** that was **invisible** to our typical standard monitoring and is difficult to disable.

CHAPTER

FOUR

POSSIBLE MITIGATIONS

Potential options for users who need to maintain the isolation and reduced privlages provided by namespaces and possibly legal, statutory, or regulitory obligations which are incompatable with all local executed code having access to the container host ssh daemon.

4.1 Disable systemd.ssh_auto

The offical way to disable vsock sshd is through a kernel command line parameters or system credentials logic. [systemd-ssh-generator]

```
KERNEL COMMAND LINE         top
   systemd-ssh-generator understands the following
   kernel-command-line(7) parameters:

   systemd.ssh_auto=
       This option takes an optional boolean argument, and defaults
       to yes. If enabled, the automatic binding to the AF_VSOCK and
       AF_UNIX sockets listed above is done. If disable, this is not
       done, except for those explicitly requested via
       systemd.ssh_listen= on the kernel command line or via the
       ssh.listen system credential.

       Added in version 256.
```

4.2 Partial mitigation options

Container runtimes and sandboxing utilities like `bwrap`, will need to use LSM and/or SecComp as kernel call blocking is the only effective option. As most CRIs allow normal users to run containers as unconfined, unlabled, and to modify SecComp profiles, a compromised container host VM or individual container requirements for elevated privlages may present challanges. As others will most likely leverage `af_vsock` for similar reasons it may possibly require blocking *af_vsock* by default.

Warning

af_vsock is intended to be zero conf, care should be taken when trying to block/allow specific, individual CIDs as it may be unreliable. In cloud situations do not depend on consistant behavior across instance types.

There are a few suggestions on the public internet related to *LOLPROX* which may be helpful but were not tested.

Note

Confinement with apparmor may be unreliable, as *af_vsock* is in *network* that is likely to be included by users by default. Additionaly many commands that are risky are *unconfined* to improve logging; including podman, lxc, flatpack, crun, runc etc…

CRIs and sandbox commands should probably follow the methods containerd used when containerd blocked vsock[7442]

I have been unable to find reliable ways to block the service using the typical systemd methods. At least that I feel would reliably work across package upgrades etc… As many workloads that need the isolation are also instantiated without interactive shell access, I have chosen to error on the side of caution.

In general CRIs and sandboxing utilities should take extra effort to either specificly enable address families they expect to use or at least remove legacy and non-namespace enabled address families from their defaults. There are many other non-namespaced shared kernel facilities that may have similar issues.

If LSMs offered namespace enabled default address family keywords that could possible help users and developers run with reduced privlages. This may allow willing users to apply policies that help maintain isolation without an undue burden on the user.

For example, many of these are not useful and risky in a sandbox/container context. `/etc/apparmor.d/abi/4.0`:

```
network {af_mask {unspec unix inet ax25 ipx appletalk netrom bridge atmpvc x25 inet6␣
↪rose netbeui security key netlink packet ash econet atmsvc rds sna irda pppox␣
↪wanpipe llc ib mpls can tipc bluetooth iucv rxrpc isdn phonet ieee802154 caif alg␣
↪nfc vsock kcm qipcrtr smc xdp mctp
```

These are just suggestions and obviously the projects should decide what is approprate and maintainable.

4.3 Logging

Adminstrators will need to make sure that they are monitoring the correct logs and may need to develop new patterns for alerting and intrusion detection.

The connection activated sshd logs can be viewed with:

```
journalctl -b -u sshd@*.service
```

As *vf_sock* was designed to bridge the host-vm divide, logs will have less information than may be expected, often limited to `CID:$Rand_port` and it may not be possible to determine if the origin was a container, malicious browser plugin, etc..

Logs that are collected by default are primarily conserned with the per-connection daemon start up and deactivation, expect failures and intrusion attempt to be counted as safe by common tools. The following was a connection by a malicious ssh client probing the server as an example:

```
Dec 20 21:14:25 debian systemd[1]: Started sshd@0-1:22-1:4036741787.service - OpenBSD␣
↪Secure Shell server per-connection daemon (vsock:1:  4036741787).
Dec 20 21:14:56 debian sshd-session[1578]: Connection closed by UNKNOWN port 65535
Dec 20 21:14:56 debian systemd[1]: sshd@0-1:22-1:4036741787.service: Deactivated␣
↪successfully.
```

If you have `vsock` channel configured on a VM, access attempts from the hypervisor to the guest will be from `2:` and VM local attempts will be from `1:`, however there are cases where VM local attemps may appear to be from `2:` and/or

[7442] https://github.com/containerd/containerd/issues/7442

`1:` at the same point of time. The CID and listening ports can change due to live migration or hot added VM featues, so it is not safe to assume persitant naming or that traffic from `2:` is from a trusted domain.

Examples are given later in the document.

CHAPTER

FIVE

ADDITIONAL INFORMATION

5.1 Vsock

5.1.1 Challanges with vsock and general purpose use are well known

- Vsock from a hypervisor or mixed container/hypervisor model is risky unless the hypervisor is a trusted environment.

[containerd7442]

> What is the problem you're trying to solve We want to use vsock for guest-host communication on KubeVirt (https://github.com/kubevirt/kubevirt). In KubeVirt we run VMs in pods.
>
> However since anyone can just connect from any pod to any VM with the default seccomp settings, we cannot limit connection attempts to our privileged node-agent.

- The VM vsock loopback has been known to be an attack vector for a long time.

[Virtio-Vsock] 4.5.2. Security Considerations:

> Loopback connectivity was **explicitly removed** by Google for ChromeOS [16] to prevent applications from connecting to other applications on the same machine. Loopback connectivity is present in the mainline Linux kernel and removing it would require a patched kernel.

5.1.2 CID is not reliable as an identifier

There are several programs using vsock for secure enclaves and code attestation, which do not leverage mechanisms to validate the listening service's identity, and there are multiple situations where a VM loopback may appear to be the Host CID==2 etc.. Program used below:

```
#!/usr/bin/env python3
# Will not work in containers
import fcntl
import socket
import struct

def get_cid(device: str) -> str:
    with open('/dev/vsock', 'rb') as f:
        res = fcntl.ioctl(f, socket.IOCTL_VM_SOCKETS_GET_LOCAL_CID, '    ')
        cid = struct.unpack('I', res)[0]
    return cid

if __name__ == "__main__":
    cid = get_cid('/dev/vsock')
    print(f'CID:{cid}')
```

1) Fresh VM install, never privisoned with virtio-vsock:

Note

This first example may not work based on hypervisor and VM config options, but it is always a possibility.

```
debian:~$ ./get_cid.py
CID:2

debian:~$ socat - VSOCK-CONNECT:2:22
SSH-2.0-OpenSSH_10.0p2 Debian-7
```

2) After attaching and detaching a vsock device with reboot:

```
debian:~$ ./get_cid.py
CID:1

debian:~$ socat - VSOCK-CONNECT:1:22
SSH-2.0-OpenSSH_10.0p2 Debian-7

debian:~$ socat - VSOCK-CONNECT:2:22
SSH-2.0-OpenSSH_10.0p2 Debian-7
```

3) Hot attach virtio-vsock:

```
debian:~./get_cid.py
CID:4
debian:~$ socat - VSOCK-CONNECT:1:22
SSH-2.0-OpenSSH_10.0p2 Debian-7

debian:~$ socat - VSOCK-CONNECT:2:22
2025/12/20 21:15:45 socat[1607] E connect(5, AF=40 cid:2 port:22, 16): Connection␣
↪reset by peer

debian:~$ socat - VSOCK-CONNECT:4:22
SSH-2.0-OpenSSH_10.0p2 Debian-7
```

4) Hot detach virtio-vsock:

```
debian:~$ ./get_cid.py
CID:1

debian:~$ socat - VSOCK-CONNECT:1:22
SSH-2.0-OpenSSH_10.0p2 Debian-7

debian:~$ socat - VSOCK-CONNECT:2:22
2025/12/20 21:16:37 socat[1621] E connect(5, AF=40 cid:2 port:22, 16): No such device

debian:~$ socat - VSOCK-CONNECT:4:22
2025/12/20 21:16:33 socat[1620] E connect(5, AF=40 cid:4 port:22, 16): No such device
```

5.1.3 Vsock is not namespaced but `ss/netcat` data is

This example demonstrates how easy it is to hide vsock listeners, which suffers from the interface `netstat/ss` use being namespaced away with netns, while the main functionality stays in the default namespace.

Global VM namespace vsock listner:

```
@debian:~$ socat - VSOCK-LISTEN:3000,fork
```

VM local user can see and connect to socket:

```
@debian:~$ ss -af vsock
Netid              State               Recv-Q               Send-Q               ␣
↪Local Address:Port                              Peer Address:Port
v_str              LISTEN              0                    0                        ␣
↪        *:22                                              *:*
v_str              LISTEN              0                    0                        ␣
↪        *:3000                                            *:*
@debian:~$ socat - VSOCK-CONNECT:1:3000
```

VM sees local connection

```
@debian:~$ ss -af vsock
Netid            State               Recv-Q              Send-Q                       Local␣
↪Address:Port                                Peer Address:Port
v_str            LISTEN              0                   0                                 ␣
↪   *:22                                              *:*
v_str            ESTAB               0                   0                                 ␣
↪   *:4036741804                                      1:3000
v_str            LISTEN              0                   0                                 ␣
↪   *:3000                                            *:*
v_str            ESTAB               0                   0                                 ␣
↪   1:3000                                            1:4036741804
```

Rootless netns namespace vsock listner:

Podman container with socat installed:

```
@debian:~$ podman run --rm -it 2fe4887726b3 /bin/bash
root@c6d61bf0b8a1:/# socat - VSOCK-LISTEN:3000,fork
2025/12/28 00:47:06 socat[2] W open("/dev/vsock", ...): No such file or directory
```

As the componant `ss` uses is namespaced away, there is a hidden listner in a global namespace address family:

```
@debian:~$ ss -af vsock
Netid              State               Recv-Q               Send-Q               ␣
↪Local Address:Port                              Peer Address:Port
v_str              LISTEN              0                    0                        ␣
↪        *:22                                              *:*
```

You can still connect from the default namespace, or any netns etc…

```
@debian:~$ socat - VSOCK-CONNECT:1:3000
```

And you can only see the default namespace portion of the connection:

```
@debian:~$ ss -af vsock
Netid            State            Recv-Q            Send-Q                        Local␣
↪Address:Port                               Peer Address:Port
v_str            LISTEN           0                 0                                   ␣
↪      *:22                                           *:*
v_str            ESTAB            0                 0                                   ␣
↪      *:4036741805                                   1:3000
```

Rootless netns namespace vsock listner and client:

You cannot see the listner, or the connected client!!!!!

Namespaced client:

```
@debian:~podman run --rm -it 2fe4887726b3 /bin/bash
root@524e90371231:/# socat - VSOCK-CONNECT:1:3000
2025/12/28 00:53:10 socat[2] W open("/dev/vsock", ...): No such file or directory
```

Note how the global namespace is blind to both listners and connections.

```
@debian:~$ ss -af vsock
Netid             State             Recv-Q             Send-Q                       ␣
↪Local Address:Port                               Peer Address:Port
v_str             LISTEN            0                  0                                ␣
↪         *:22                                       *:*
```

BIBLIOGRAPHY

[BRICKSTORM] https://www.cisa.gov/news-events/analysis-reports/ar25-338a#AppC

[shai-hulud] https://www.wiz.io/blog/shai-hulud-2-0-aftermath-ongoing-supply-chain-attack

[vsock] https://man7.org/linux/man-pages/man7/vsock.7.html

[systemd-ssh-generator] https://man7.org/linux/man-pages/man8/systemd-ssh-generator.8.html

[Virtio-Vsock] Virtio-Vsock - Configuration-Agnostic Guest/Host Communication: Johannes Wiesböck, Johannes Naab, Henning Stubbe https://www.net.in.tum.de/fileadmin/TUM/NET/NET-2019-10-1/NET-2019-10-1_14.pdf

[containerd7442] https://github.com/containerd/containerd/issues/7442