Path Safety “in the Trenches”

実地に踏んだパス安全の経験

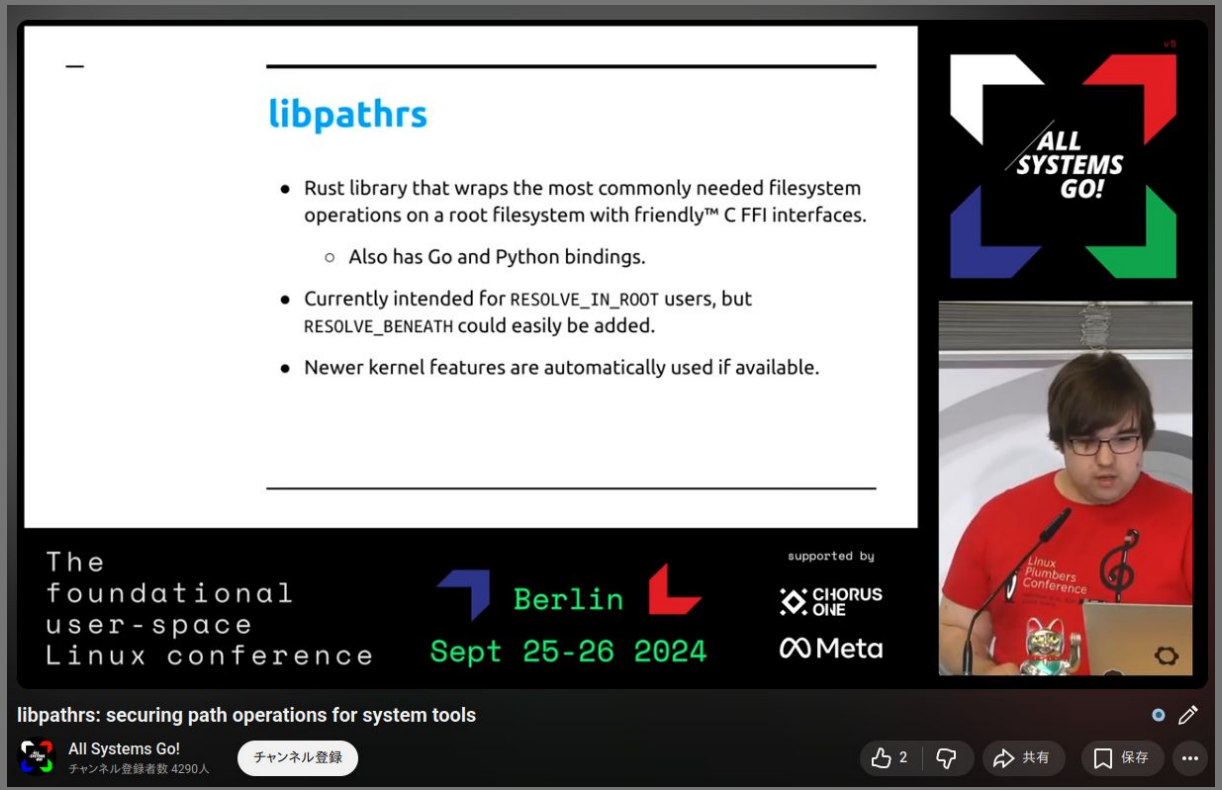
Aleksa Sarai
github.com/cyphar
2025年12月9日

—

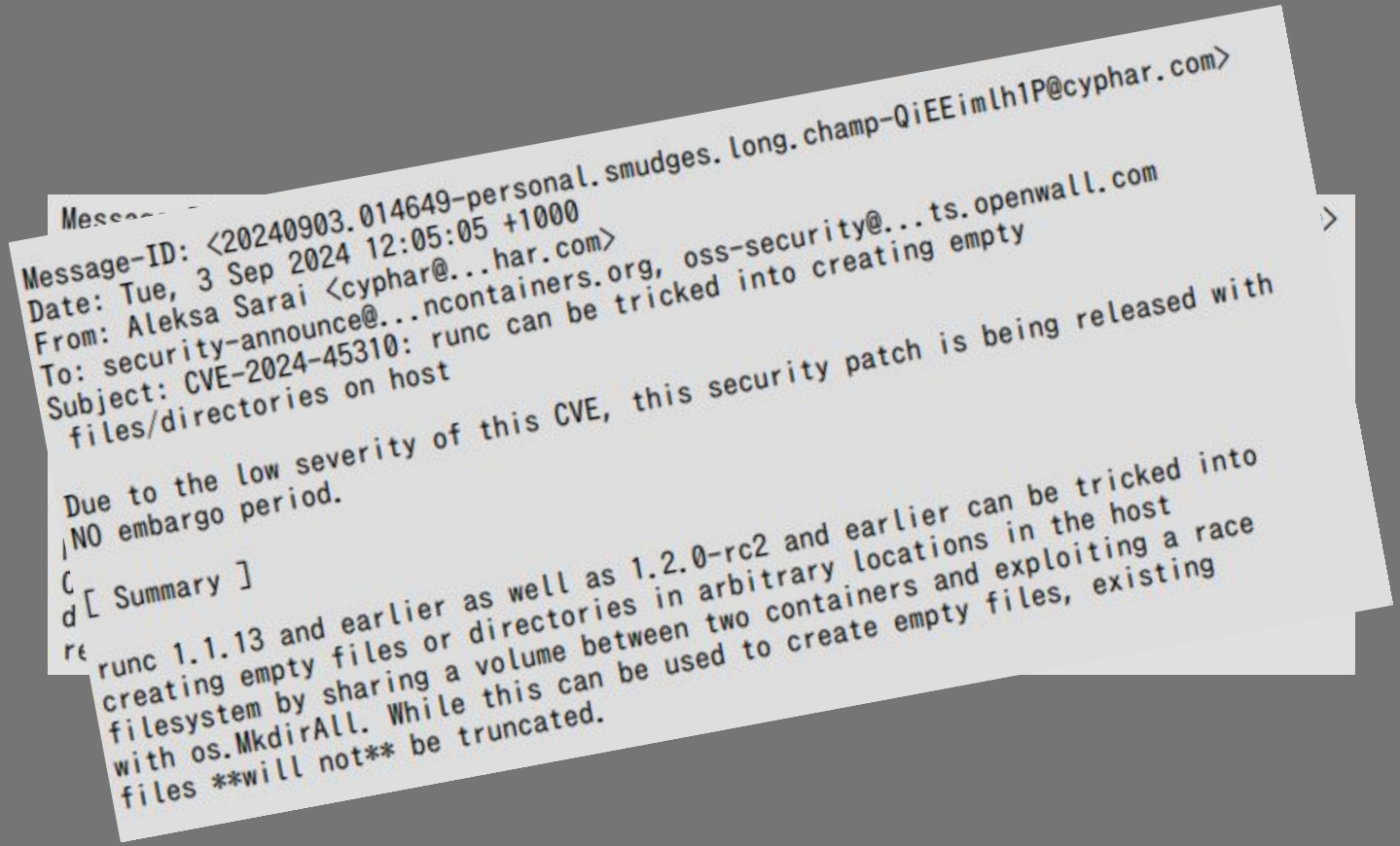
懐かしいなぁ…

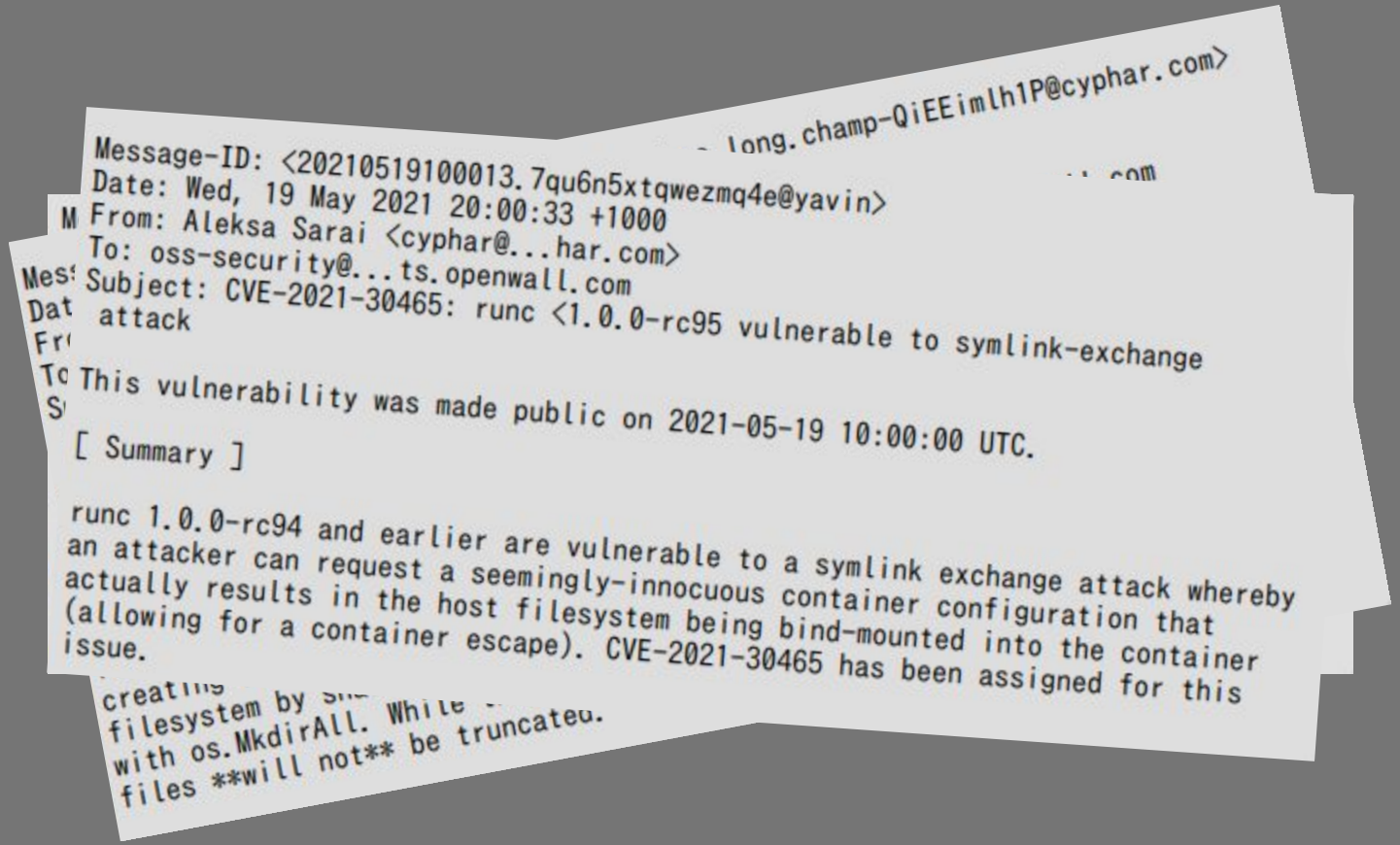

LINUXCON
containercon
RATION
ICK-UP
FERENCE
MAKI - 7F
ELYA - 5F
ON 3F
LINUX

runc


libpathrs
Rust library that wraps the most commonly needed filesystem operations on a root filesystem with friendly™ C FFI interfaces.
Also has Go and Python bindings.
Currently intended for RESOLVE_IN_ROOT users, but RESOLVE_BENEATH could easily be added.
Newer kernel features are automatically used if available.
ALL SYSTEMS GO!
The foundational user-space Linux conference
Berlin
Sept 25-26 2024
supported by
CHORUS ONE
Meta
Linux Plumbers Conference
libpathrs: securing path operations for system tools
All Systems Go!
チャンネル登録者数 4290人
チャンネル登録
2
共有
保存

```
Message-ID: <2025-11-05-remember-remember-the-fifth-of-november-3EtRdS@cyphar.com>
Date: Wed, 5 Nov 2025 20:53:08 +1100
From: Aleksa Sarai <cyphar@...har.com>
To: oss-security@...ts.openwall.com, fulldisclosure@...lists.org
Subject: runc container breakouts via procfs writes: CVE-2025-31133,
 CVE-2025-52565, and CVE-2025-52881

Hello,

This is a notification to vendors that use or ship runc about THREE (3)
high-severity vulnerabilities (CVE-2025-31133, CVE-2025-52565, and
CVE-2025-52881). All three vulnerabilities ultimately allow (through
different methods) for full container breakouts by bypassing runc's
restrictions for writing to arbitrary /proc files.
```


Message-ID: <20240903.014649-personal.smudges.long.champ-QiEEimlh1P@cyphar.com>
Date: Tue, 3 Sep 2024 12:05:05 +1000
From: Aleksa Sarai <cyphar@...har.com>
To: security-announce@...ncontainers.org, oss-security@...ts.openwall.com
Subject: CVE-2024-45310: runc can be tricked into creating empty
files/directories on host
Due to the low severity of this CVE, this security patch is being released with
NO embargo period.
[Summary]
runc 1.1.13 and earlier as well as 1.2.0-rc2 and earlier can be tricked into
creating empty files or directories in arbitrary locations in the host
filesystem by sharing a volume between two containers and exploiting a race
with os.MkdirAll. While this can be used to create empty files, existing
files **will not** be truncated.



long.champ-QiEEimlh1P@cyphar.com>

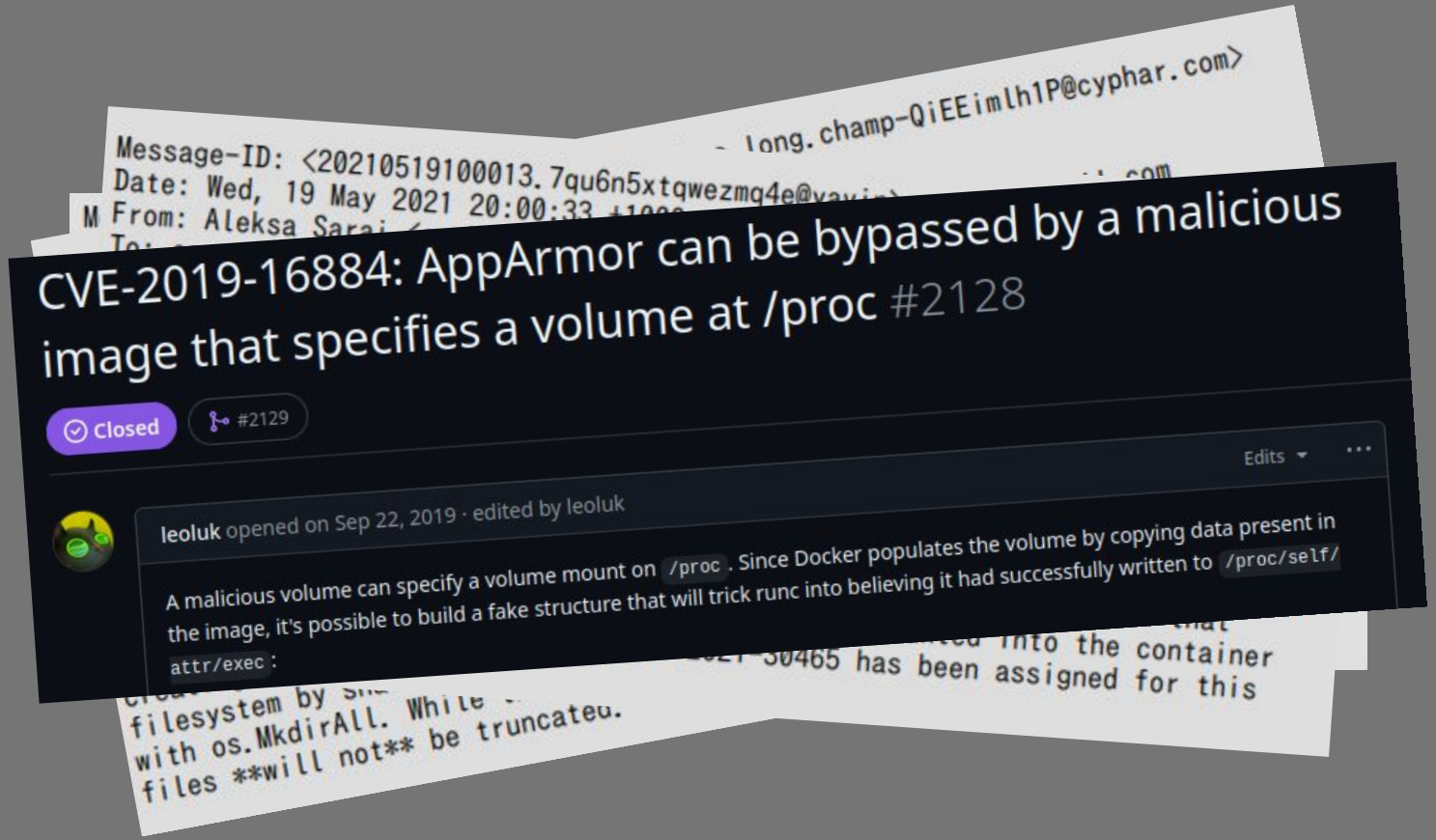
Message-ID: <20210519100013.7qu6n5xtqwezmq4e@yavin>
Date: Wed, 19 May 2021 20:00:33 +1000
From: Aleksa Sarai <cyphar@...har.com>
To: oss-security@...ts.openwall.com
Subject: CVE-2021-30465: runc <1.0.0-rc95 vulnerable to symlink-exchange attack

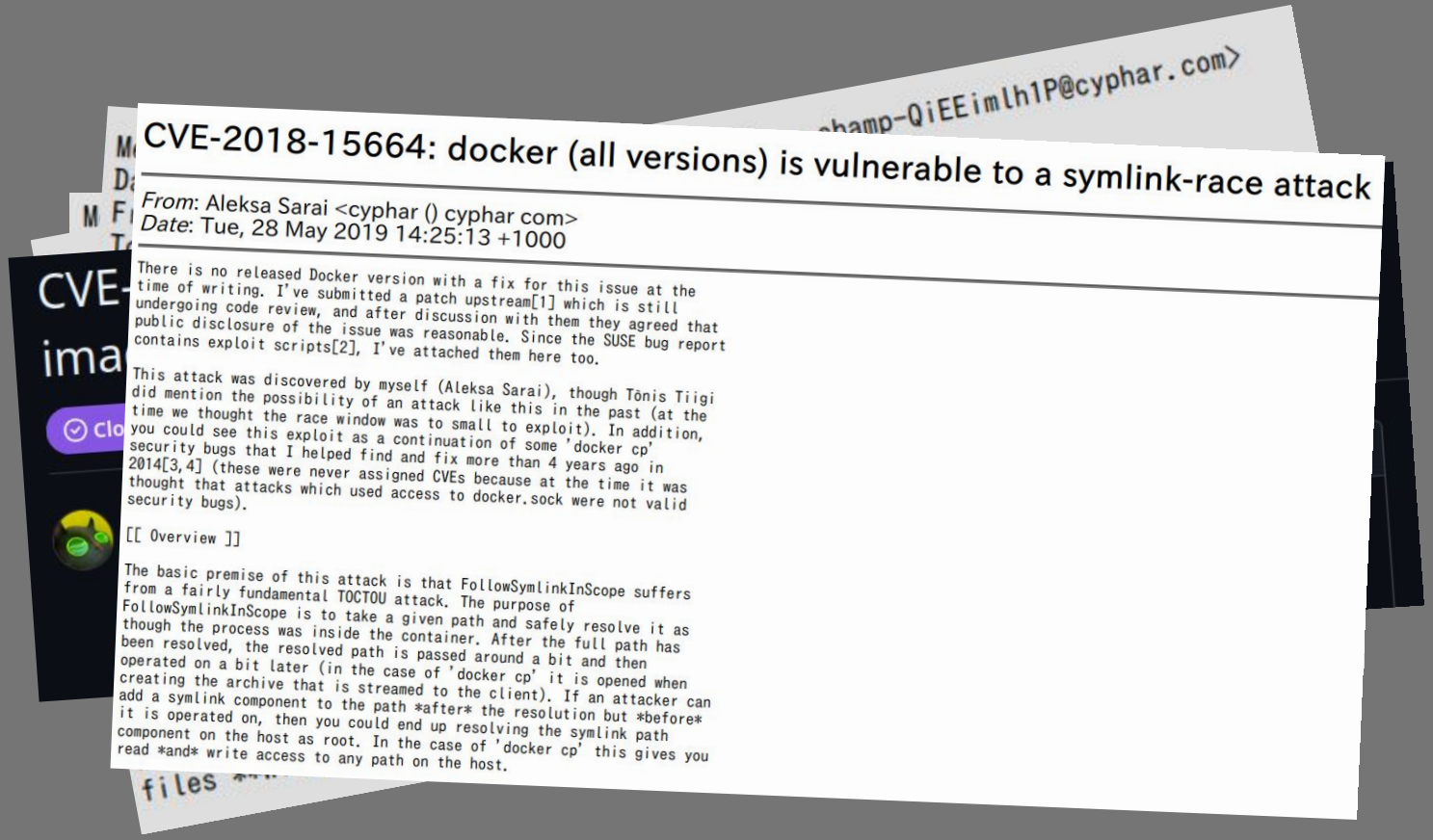
This vulnerability was made public on 2021-05-19 10:00:00 UTC.

[Summary]

runc 1.0.0-rc94 and earlier are vulnerable to a symlink exchange attack whereby an attacker can request a seemingly-innocuous container configuration that actually results in the host filesystem being bind-mounted into the container (allowing for a container escape). CVE-2021-30465 has been assigned for this issue.

filesystem by
with os.MkdirAll. While
files **will not** be truncated.


long.champ-QiEEimlh1P@cyphar.com>
Message-ID: <20210519100013.7qu6n5xtqwezmq4e@
Date: Wed, 19 May 2021 20:00:33
From: Aleksa Sarai
filesystem by
with os.MkdirAll. While
files **will not** be truncated.
into the container
30465 has been assigned for this
CVE-2019-16884: AppArmor can be bypassed by a malicious image that specifies a volume at /proc #2128
Closed
#2129
Edits
leoluk opened on Sep 22, 2019 · edited by leoluk
A malicious volume can specify a volume mount on /proc. Since Docker populates the volume by copying data present in the image, it's possible to build a fake structure that will trick runc into believing it had successfully written to /proc/self/attr/exec:



CVE-2018-15664: docker (all versions) is vulnerable to a symlink-race attack

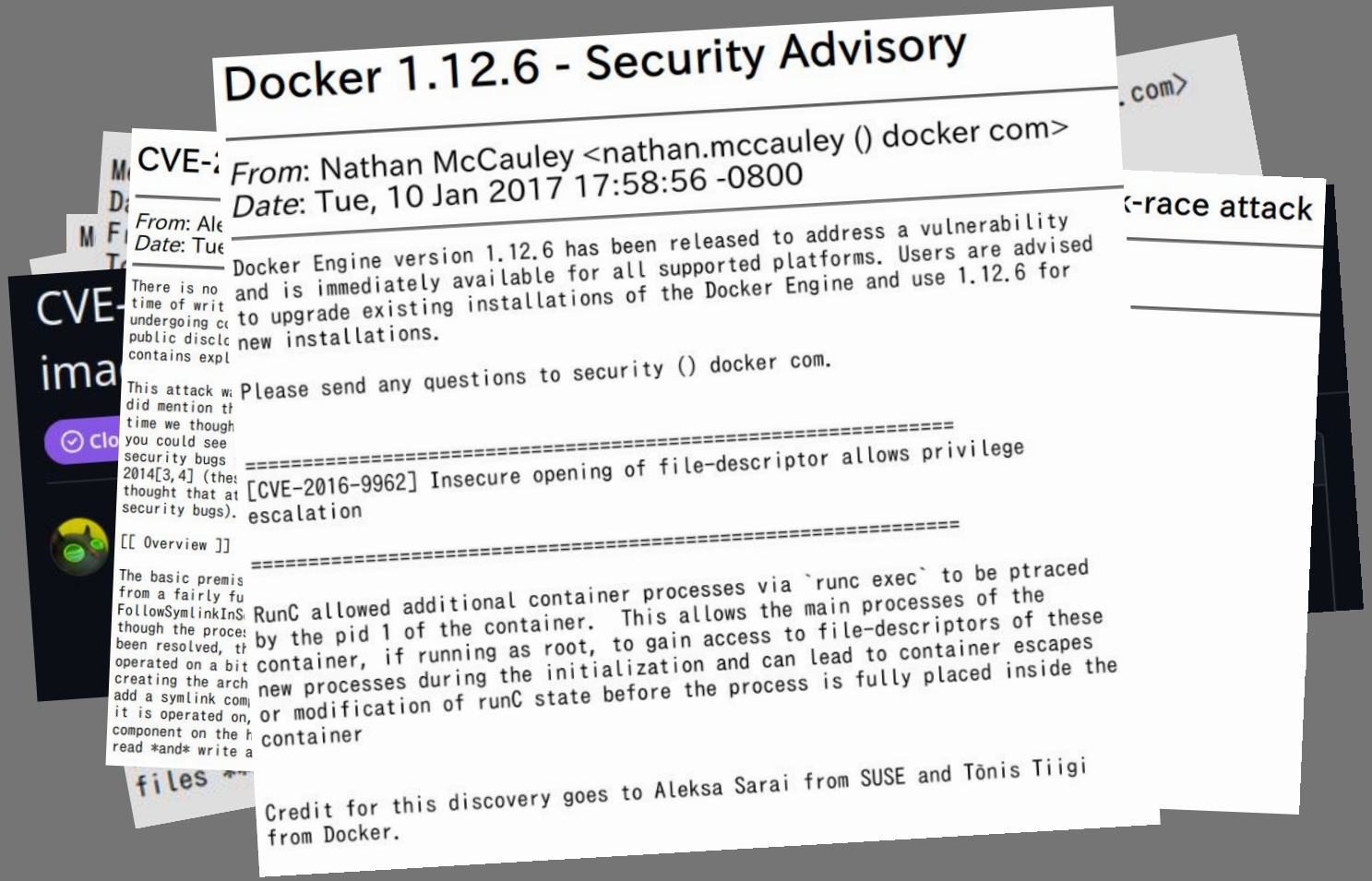
From: Aleksa Sarai <cyphar () cyphar com>
Date: Tue, 28 May 2019 14:25:13 +1000

There is no released Docker version with a fix for this issue at the time of writing. I've submitted a patch upstream[1] which is still undergoing code review, and after discussion with them they agreed that public disclosure of the issue was reasonable. Since the SUSE bug report contains exploit scripts[2], I've attached them here too.

This attack was discovered by myself (Aleksa Sarai), though Tõnis Tiigi did mention the possibility of an attack like this in the past (at the time we thought the race window was to small to exploit). In addition, you could see this exploit as a continuation of some 'docker cp' security bugs that I helped find and fix more than 4 years ago in 2014[3,4] (these were never assigned CVEs because at the time it was thought that attacks which used access to docker.sock were not valid security bugs).

[[Overview]]

The basic premise of this attack is that FollowSymlinkInScope suffers from a fairly fundamental TOCTOU attack. The purpose of FollowSymlinkInScope is to take a given path and safely resolve it as though the process was inside the container. After the full path has been resolved, the resolved path is passed around a bit and then operated on a bit later (in the case of 'docker cp' it is opened when creating the archive that is streamed to the client). If an attacker can add a symlink component to the path *after* the resolution but *before* it is operated on, then you could end up resolving the symlink path component on the host as root. In the case of 'docker cp' this gives you read *and* write access to any path on the host.



Docker 1.12.6 - Security Advisory

From: Nathan McCauley <nathan.mccauley () docker com>
Date: Tue, 10 Jan 2017 17:58:56 -0800

Docker Engine version 1.12.6 has been released to address a vulnerability and is immediately available for all supported platforms. Users are advised to upgrade existing installations of the Docker Engine and use 1.12.6 for new installations.

Please send any questions to security () docker com.

==========

[CVE-2016-9962] Insecure opening of file-descriptor allows privilege escalation

==========

RunC allowed additional container processes via `runc exec` to be ptraced by the pid 1 of the container. This allows the main processes of the container, if running as root, to gain access to file-descriptors of these new processes during the initialization and can lead to container escapes or modification of runC state before the process is fully placed inside the container

Credit for this discovery goes to Aleksa Sarai from SUSE and Tõnis Tiigi from Docker.

PLEASE USE USER NAMESPACES

path safety
「パス安全」とは

- “**regular**” path safety
  「**普通の**」パス安全
- “**strict**” path safety (procfs)
  「**厳しい**」パス安全（特にprocfsの為）

regular path safety
普通のパス安全

regular path safety
普通のパス安全

- Almost all system tools need to interact with unsafe paths.
- When operating on a path, a path component might be swapped with a symlink.
- Path sanitisation is useless because it just becomes TOCTTOU.

regular path (un)safety
普通のパス安全のない一例

```
open("$rootfs/foo/bar/baz", ...)
```

regular path (un)safety

普通のパス安全のない一例

```
open("$rootfs/foo/bar/baz", ...)
```

↓

/host/path/

regular path (un)safety
普通のパス安全のない一例

```
open("$rootfs/foo/bar/baz", ...)
```

↓

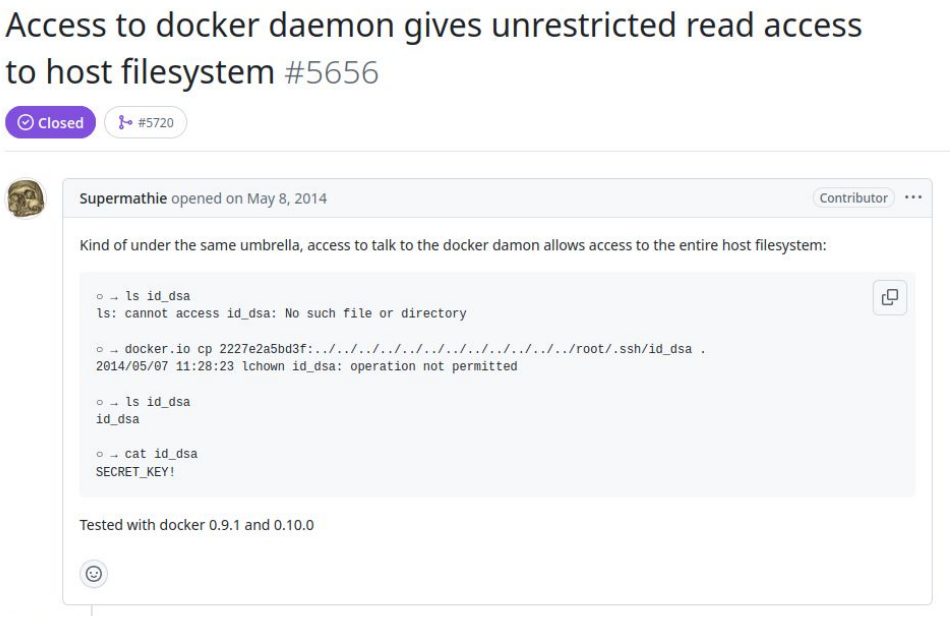
/host/path/

(broken) previous approaches
将来の駄目なやり方

`filepath.Join`

- Trivially exploitable with `../../../../../../foo`.
- Symlink components are resolved *in the host*.

example vulnerabilities (i)
脆弱性の一例（其ノ壱）



Access to docker daemon gives unrestricted read access to host filesystem #5656

Closed #5720

Supermathie opened on May 8, 2014 Contributor

Kind of under the same umbrella, access to talk to the docker damon allows access to the entire host filesystem:

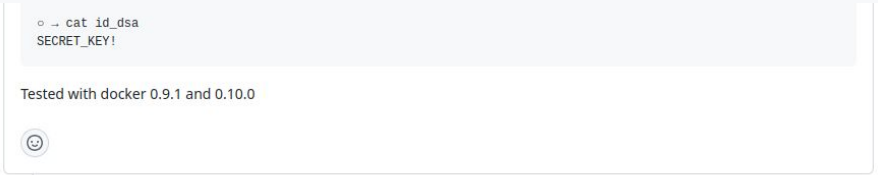
```
○ → ls id_dsa
ls: cannot access id_dsa: No such file or directory

○ → docker.io cp 2227e2a5bd3f:../../../../../../../../../../../root/.ssh/id_dsa .
2014/05/07 11:28:23 lchown id_dsa: operation not permitted

○ → ls id_dsa
id_dsa

○ → cat id_dsa
SECRET_KEY!
```

Tested with docker 0.9.1 and 0.10.0

example vulnerabilities (i)
脆弱性の一例（其ノ壱）

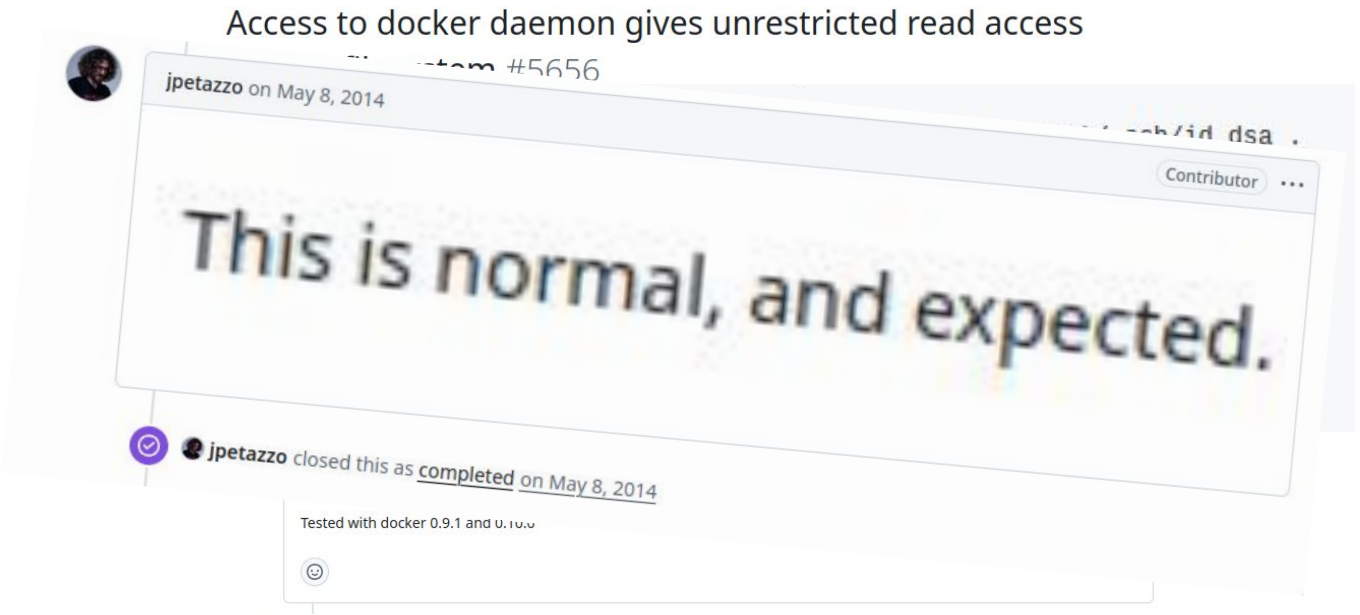
Access to docker daemon gives unrestricted read access to host filesystem #5656

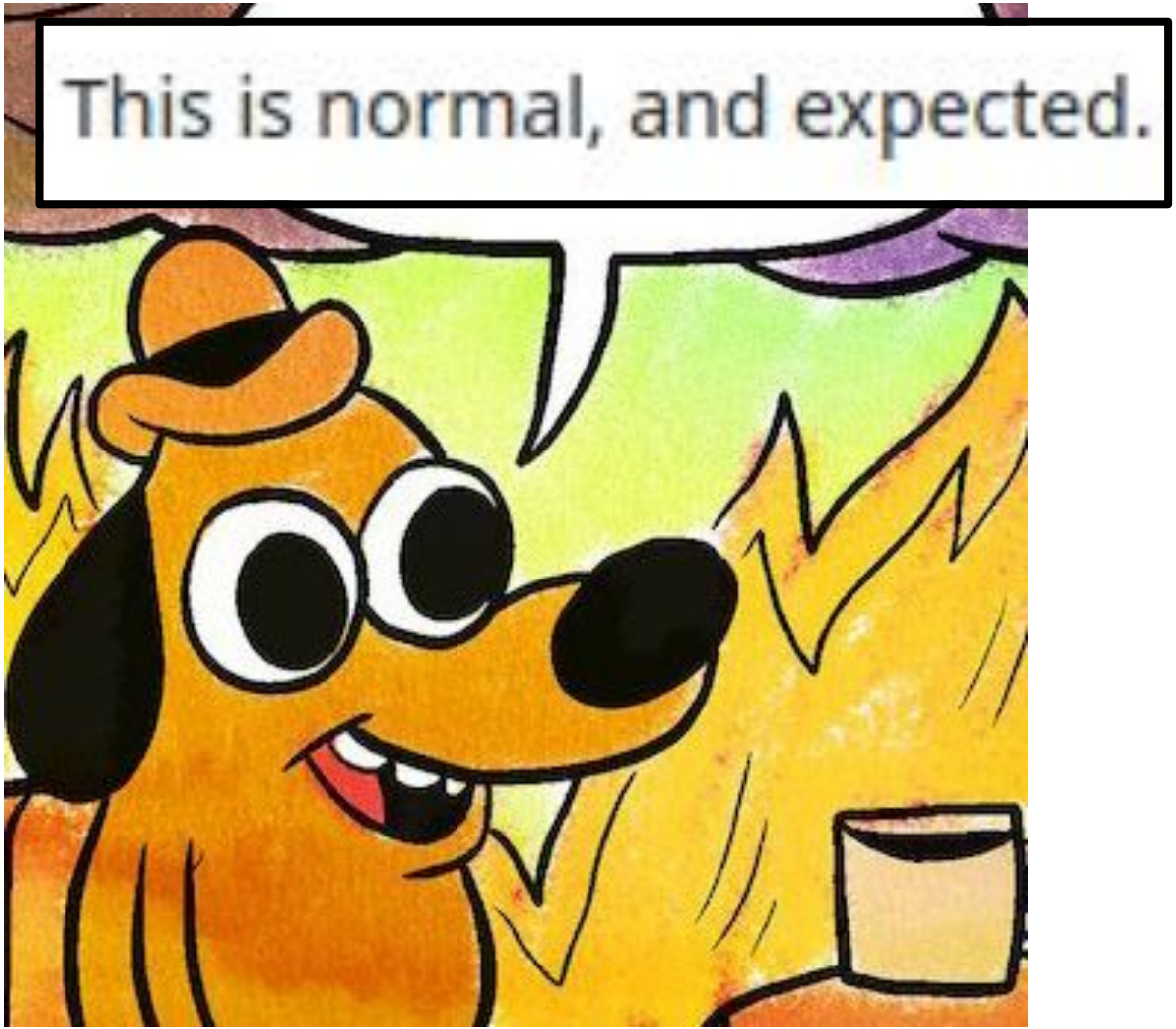
```
○ → docker.io cp 2227e2a5bd3f:../../../../../../../../../../../../root/.ssh/id_dsa .
2014/05/07 11:28:23 lchown id_dsa: operation not permitted

○ → ls id_dsa
id_dsa

○ → cat id_dsa
SECRET_KEY!
```



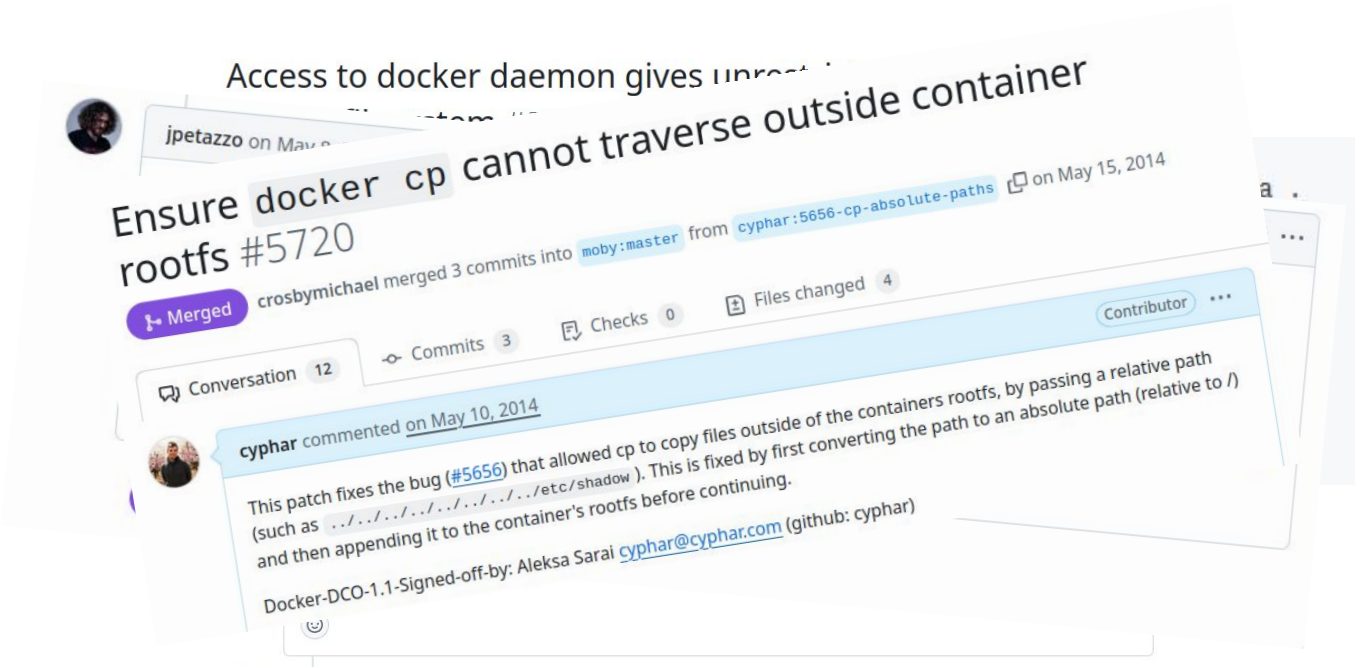
example vulnerabilities (i)
脆弱性の一例（其ノ壱）

example vulnerabilities (i)
脆弱性の一例（其ノ壱）




This is normal, and expected.

example vulnerabilities (i)
脆弱性の一例（其ノ壱）



filepath-securejoin

- Path lookups in userspace with `chroot(2)`-like scoping.

still broken previous approaches
まだ駄目なやり方

filepath-securejoin

- Path lookups in userspace with `chroot(2)`-like scoping.

still broken previous approaches
まだ駄目なやり方

filepath-securejoin

- Path lookups in userspace with `chroot(2)`-like scoping.
- Returns a path string – not safe against races!

example vulnerabilities (ii)

脆弱性の一例（其ノ弐）

CVE-2018-15664

- `docker cp` used `chroot(2)` to try to block path-based attacks.
- Unfortunately, Docker rooted the `chroot(2)` within the attacker-controlled path and so the same issue was present.
- (This was one example of a larger pattern within Docker that AFAIK is still not fully resolved.)

example vulnerabilities (iii)

脆弱性の一例（其ノ参）

CVE-2024-45310

- Mountpoint targets would be created with `os.MkdirAll` or `O_CREAT` of a `filepath-securejoin` path.
- Racing attackers could trick us into creating host inodes.

solutions (i)
解決（其ノ壱）

`openat2(2)`

(since Linux 5.6)

- `chroot(2)`-like `IN_ROOT` Just Works™ (mostly).
- `NO_SYMLINKS`, `BENEATH` for everything else.
- `NO_MAGICLINKS`, `NO_XDEV` are particularly useful.
- Requires moving to a more file-descriptor based approach.

solutions (ii)
解決（其ノ弐）

`openat(O_PATH)`

(since Linux 2.6.39-ish)

- Manually do lookup with `O_PATH` handles, emulating what `openat2(2)` does.
- `..` and `/` components are usually verified through `/proc/self/fd`.
- Still requires moving to a more file-descriptor based approach.

example vulnerabilities (iv)

脆弱性の一例（其ノ肆）

CVE-2025-31133 & CVE-2025-52565

- Special bind-mount sources like `/dev/null` and `/dev/pts/$n` were sourced from the container.
- Racing attackers could replace these with symlinks, causing us to bind-mount procfs files that would normally be masked, allowing for breakouts.

strict path safety
厳しいパス安全

strict path safety
厳しいパス安全

- Some filesystem operations need to operate on a specific file, the most common example is operations on procfs.
- In these cases, bind-mounts and symlinks are big trouble.

strict path (un)safety

厳しいパス安全のない一例

```
open("/proc/self/attr/exec", ...)
open("/proc/sys/some/sysctl", ...)
open("/proc/self/fd/$fd", ...)
execve("/proc/self/exe", ...)
```

strict path (un)safety

厳しいパス安全のない一例

/tmp/bad_binary

/proc/self/sched

```
open("/proc/self/attr/exec", ...)
open("/proc/sys/some/sysctl", ...)
open("/proc/self/fd/$fd", ...)
execve("/proc/self/exe", ...)
```

/proc/sys/kernel/core_pattern

example vulnerabilities (v)
脆弱性の一例（其ノ伍）
CVE-2019-19921

- runc would previously just write to `/proc` normally.
- Attackers could trick runc into configuring a `tmpfs` mount on top of `/proc` with fake procfs files.
- LSM labels are set through `/proc/self/attr/$lsm/exec`, fake files make this a no-op.

temporary fix
暫定手段

- Use `fstatfs` to check that we are operating on a procfs file.
- Only a temporary fix – we can’t be sure which procfs file it is!

example vulnerabilities (vi)

脆弱性の一例（其ノ陸）

CVE-2025-5288

- Rather than using `tmpfs`, trick runc into bind-mounting bits of `procfs`.
  - `/proc/self/attr/exec` ← `/proc/self/sched` (no-op)
  - `/proc/self/attr/exec` ← `/proc/sysrq-trigger` (crash)
  - `/proc/sys/...` ← `/proc/sys/kernel/core_pattern` 😈

solutions (iii)
解決（其ノ参）

`fsopen(2)` · `open_tree(2)`

(since Linux 5.1)

- Provides a mechanism for a private procfs mount that cannot have overmounts or racing attackers doing mounts.
- For regular procfs files, `openat2(NO_XDEV)` is usually enough.
    - (If you check that `/proc` is `PROC_ROOT_INO`.)

```
Message-ID: <2025-11-05-remember-remember-the-fifth-of-november-3EtRdS@cyphar.com>
Date: Wed, 5 Nov 2025 20:53:08 +1100
From: Aleksa Sarai <cyphar@...har.com>
To: oss-security@...ts.openwall.com, fulldisclosure@...lists.org
Subject: runc container breakouts via procfs writes: CVE-2025-31133,
 CVE-2025-52565, and CVE-2025-52881

Hello,

This is a notification to vendors that use or ship runc about THREE (3)
high-severity vulnerabilities (CVE-2025-31133, CVE-2025-52565, and
CVE-2025-52881). All three vulnerabilities ultimately allow (through
different methods) for full container breakouts by bypassing runc's
restrictions for writing to arbitrary /proc files.
```

```
commit 2f74f4ae1b483b3a4d197f574a37c7b768baf96f
Merge: fb01482d128f 3f925525b44d
Author: Aleksa Sarai <cyphar@cyphar.com>
Date:   Wed Nov 5 20:19:30 2025 +1100

        merge private security patches into opencontainers/runc:main

        Aleksa Sarai (21):
          rootfs: re-allow dangling symlinks in mount targets
          openat2: improve resilience on busy systems
          selinux: use safe procfs API for labels
          rootfs: switch to fd-based handling of mountpoint targets
          libct/system: use securejoin for /proc/$pid/stat
          init: use securejoin for /proc/self/setgroups
          init: write sysctls using safe procfs API
          utils: remove unneeded EnsureProcHandle
          utils: use safe procfs for /proc/self/fd loop code
          apparmor: use safe procfs API for labels
          ci: add lint to forbid the usage of os.Create
          rootfs: avoid using os.Create for new device inodes
          internal: add wrappers for securejoin.Proc*
          go.mod: update to github.com/cyphar/filepath-securejoin@v0.5.0
          console: verify /dev/pts/ptmx before use
          console: avoid trivial symlink attacks for /dev/console
          console: add fallback for pre-TIOCGPTPEER kernels
          console: use TIOCGPTPEER when allocating peer PTY
          *: switch to safer securejoin.Reopen
          internal: move utils.MkdirAllInRoot to internal/pathrs
          internal/sys: add VerifyInode helper

        Li Fubang (1):
          libct: align param type for mountCgroupV1/V2 functions

        Kir Kolyshkin (3):
          libct: maskPaths: don't rely on ENOTDIR for mount
          libct: maskPaths: only ignore ENOENT on mount dest
          libct: add/use isDevNull, verifyDevNull

        Fixes: CVE-2025-31133 GHSA-9493-h29p-rfm2
        Fixes: CVE-2025-52565 GHSA-qw9x-cqr3-wc7r
        Fixes: CVE-2025-52881 GHSA-cgrx-mc8f-2prm
        Reported-by: Lei Wang <ssst0n3@gmail.com>
        Reported-by: Li Fubang <lifubang@acmcoder.com>
        Reported-by: Tõnis Tiigi <tonistiigi@gmail.com>
        Reported-by: Aleksa Sarai <cyphar@cyphar.com>
        Signed-off-by: Aleksa Sarai <cyphar@cyphar.com>

 111 files changed, 9452 insertions(+), 1338 deletions(-)
```

```
111 files changed, 9452 insertions(+), 1338 deletions(-)
```

our approach
我らの解決手段

- Switch to pathrs-lite.
  - `openat2(2)` support (with `O_PATH` fallback).
  - Provides helpers for safe procfs operations.
  - A pure-Go version of `libpathrs`.
- Hardern verification of all special inodes we interact with.
- Audit all write paths – can they write to `/proc`?
- Switch to file descriptors as much as possible.

general takeaways
解決手段

- Most system tools need at least regular path safety.
- Switch to tools like `libpathrs` (or `pathrs-lite`) and move to a move file-descriptor-based design.
    - If you don't need to support old kernels, feel free to use `openat2(2)` directly.
- If you use procfs, you should **really** use `libpathrs`.

remaining work
まだまだ・・・

- Switch runc to `libpathrs`.
  - Polish `libpathrs` some more.
- Move even more things to use file descriptor based.
  - Switch to the `fsopen(2)` mount API.
- Come up with a decent threat model for runc.

questions?

(rants, pitchforks…?)

libpathrs

- Rust library that wraps the most commonly needed filesystem operations on a root filesystem with friendly™ C FFI interfaces.
  - Also has Go and Python bindings.
- Currently intended for `RESOLVE_IN_ROOT` users, but `RESOLVE_BENEATH` could easily be added.
- Newer kernel features are automatically used if available.

libpathrs (rust)

```rust
use pathrs::{Root, flags::OpenFlags};
let root = Root::open("/path/to/root")?;

// Resolve and open a file.
let passwd = root
        .resolve("/etc/passwd")?
        .reopen(OpenFlags::O_RDONLY)?;
// ... or ...
let passwd = root.open_subpath(
        "/etc/passwd", OpenFlags::O_RDONLY
)?;

// Create a new file and open it (O_CREAT).
let newfile = root.create_file(
        "/etc/newfile",
        OpenFlags::O_RDWR,
        &Permissions::from_mode(0o755),
)?;
```

```rust
// Create a symlink.
let newfile = root.create(
        "/link",
        &InodeType::Symlink("/target".into()),
)?;

// mkdir -p
let dir = root.mkdir_all(
        "/foo/bar/baz",
        &Permissions::from_mode(0o755),
)?;

// rm -r
root.remove_all("/foo/bar")?;

// See the docs for more info.
```

libpathrs (C)

```c
int root = pathrs_open_root("/path/to/root");
if (root < 0) {
        liberr = root;
        goto err;
}
int handle = pathrs_inroot_resolve(root,
                                   "/etc/passwd");
if (handle < 0) {
        liberr = handle;
        goto err;
}
int fd = pathrs_reopen(handle, O_RDONLY);
if (fd < 0) {
        liberr = fd;
        goto err;
}
// or pathrs_inroot_open(root, "/etc/passwd", ...)
```

```c
err:
if (liberr < 0) {
        pathrs_error_t *error =
                pathrs_errorinfo(liberr);
        fprintf(stderr,
                "Uh-oh: %s (errno=%d)\n",
                error->description,
                error->saved_errno);
        pathrs_errorinfo_free(error);
}

close(root);
close(handle);
/* ... do something with fd ... */
```

“**re**open”?

```rust
use pathrs::{Root, flags::OpenFlags};
let root = Root::open("/path/to/root")?;

// Get a reusable (O_PATH) ptmx handle.
let ptmx = root
       .resolve("/dev/pts/ptmx")?;

// Create several new console instances.
// They are all independent instances.
let console1 = ptmx
       .reopen(OpenFlags::O_RDWR)?;
let console2 = ptmx
       .reopen(OpenFlags::O_RDWR)?;
let console3 = ptmx
       .reopen(OpenFlags::O_RDWR)?;
```

- Sometimes you need to re-open the same file multiple times.
- This is not just dup! It’s a proper *race-free* open.
- Implementing lookups entirely with O_PATH and re-opening is simpler…

`procfs` api

- Detecting attackers is the primary goal, followed by resiliency.
- Private procfs instance with `fsopen` and `open_tree` if possible.
  - Can’t be used for unprivileged programs…
- `openat2` or very restrictive `O_PATH` resolver for lookups.
  - *(This resolver doesn’t do reopens!)*
- Used internally by `libpathrs` to implement the main API.

procfs api (rust)

```rust
use pathrs::{flags::OpenFlags, procfs::*};

// Open *regular* file.
let attr_file = ProcfsHandle::new()?
        .open(
                ProcfsBase::ProcThreadSelf,
                "attr/exec",
                OpenFlags::O_WRONLY,
        )?;

// Create your own private handle.
let handle =
        ProcfsHandleBuilder::new()
                .unmasked(true)
                .build()?;
```

```rust
// Open a magic-link.
let exe = ProcfsHandle::new()?.open_follow(
        ProcfsBase::ProcSelf,
        "exe",
        OpenFlags::O_RDONLY,
)?;

// Equivalent to readlinkat(fd, "").
let fd_path = ProcfsHandle::new()?.readlink(
        ProcfsBase::ProcThreadSelf,
        format!("fd/{}", file.as_raw_fd()),
        OpenFlags::O_RDONLY,
)?;
```

procfs api (C)

```c
int write_apparmor_label(const char *label)
{
    /* Open *regular* file. */
    int fd = pathrs_proc_open(
        PATHRS_PROC_THREAD_SELF,
        "attr/apparmor/exec",
        O_WRONLY|O_NOFOLLOW
    );
    if (fd < 0) {
        pathrs_error_t *e = pathrs_errorinfo(fd);
        /* ... print the error ... */
        pathrs_errorinfo_free(e);
        return -1;
    }
    int err = write(fd, label, strlen(label));
    close(fd);
    return err;
}
```

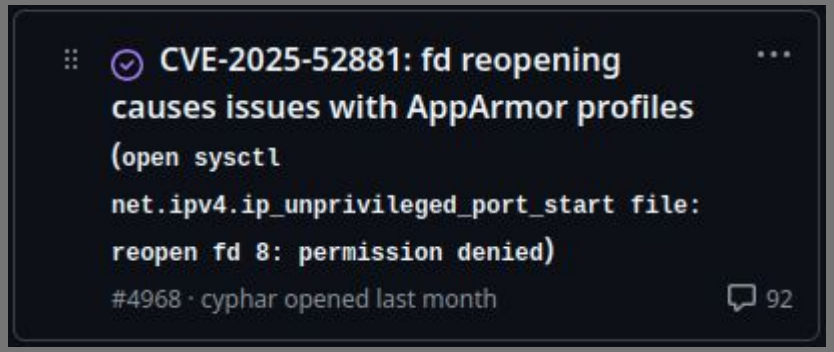
```c
int get_self_exe(void)
{
    /* Follows the magic-link! */
    int fd = pathrs_proc_open(PATHRS_PROC_SELF,
                              "exe", O_PATH);
    if (fd < 0) {
        pathrs_error_t *e = pathrs_errorinfo(fd);
        /* ... print the error ... */
        pathrs_errorinfo_free(e);
        return -1;
    }
    return fd;
}
```

`procfs` limitations

- For non-magic-links, `openat2` (Linux 5.6) is sufficient.
  - (`openat2` might be blocked due to seccomp limitations.)
- For magic-links, we need:
  - `fsopen` or `open_tree` (Linux 5.1) for race safety.

questions?

(rants, pitchforks…?)

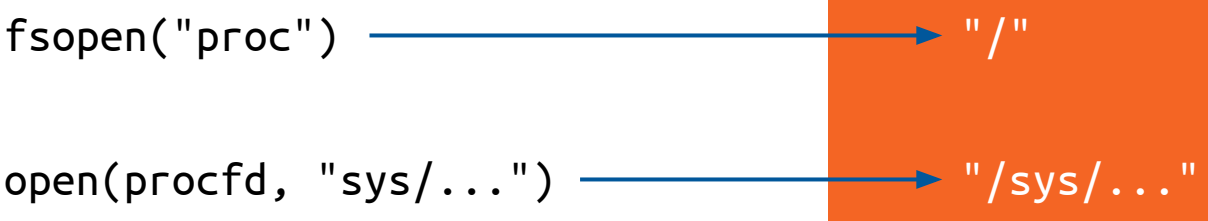

CVE-2025-52881: fd reopening causes issues with AppArmor profiles (open sysctl net.ipv4.ip_unprivileged_port_start file: reopen fd 8: permission denied)
#4968 · cyphar opened last month
92

runc

AppArmor

runc

AppArmor



runc

AppArmor

`fsopen("proc")` → `"/"`

`open(procfd, "sys/...")` → `"/sys/..."`

questions?

(rants, pitchforks…?)

kernel hardening – magic-links

- Magic-links can be used to break out of containers.
  - Userspace needs to be careful about leaks.
  - Container runtimes need to use `PR_SET_DUMPABLE`!
- `CVE-2019-5736`: Overwrite host binary with `/proc/self/exe`.
  - Solution: restrict reopening with extra permissions.
  - (Also allow users to specify fd restrictions with `openat2`.)
- Restrict mounts on top of all magic-links.

kernel work – openat2

- Some more things we might want to add:
  - `RESOLVE_NO_BLOCK` (`NO_REMOTE`?) to avoid DoSes.
  - Restrict types of files we want to open (another DoS).
  - `RESOLVE_NO_DOTDOT` for extreme lookup restrictions.
- What about atomic `mknod` combined with `open`? (`O_MKNOD`?)
  - If it could take `RESOLVE_*` flags that would be even nicer!