KubeCon | CloudNativeCon

North America 2019

On the Security of Copying To and From Live Containers

Yuval Avrahami & Ariel Zelivansky

Palo Alto Networks

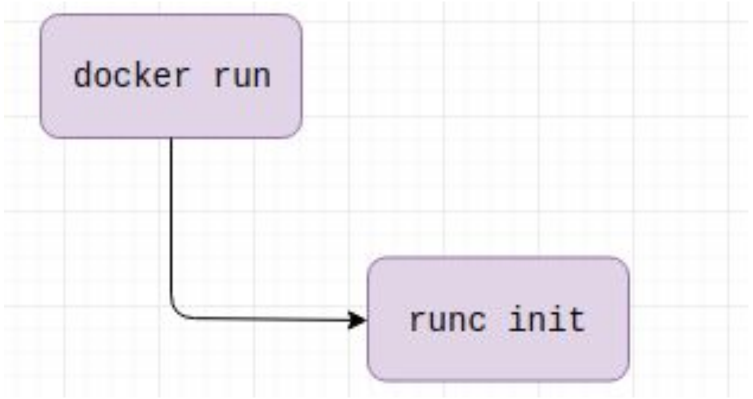
Agenda

- Background
- Copy Vulnerabilities
  - Podman
  - Docker
  - K8s
- Takeaways

Containers 101

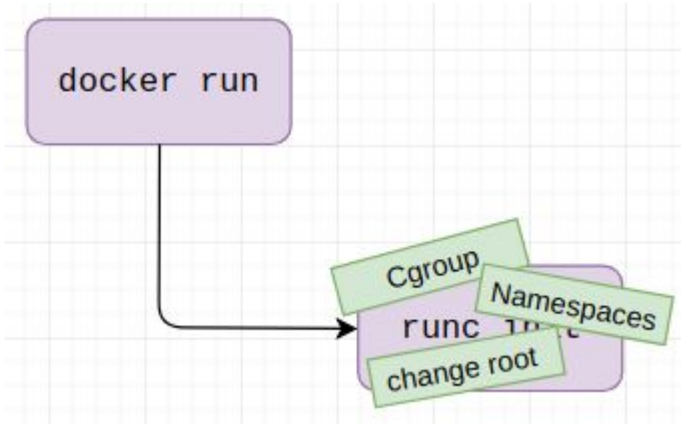
- Restricted **processes** chrooted to a separate filesystem

Starting a Container



- runC - the industry standard tool for running containers

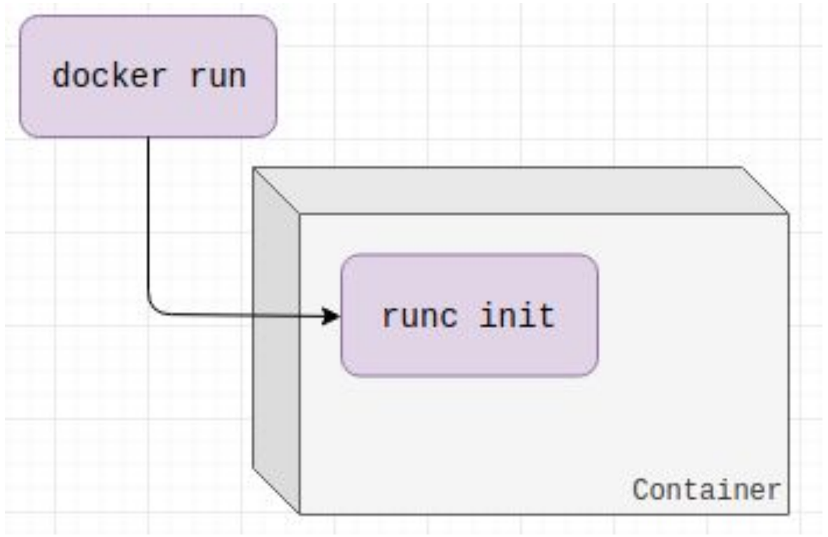
Starting a Container



- Namespaces
- Cgroups
- Chroot to image fs (/var/.../docker/$ctrid/merged)
- Drop capabilities
- LSMs (AppArmor)

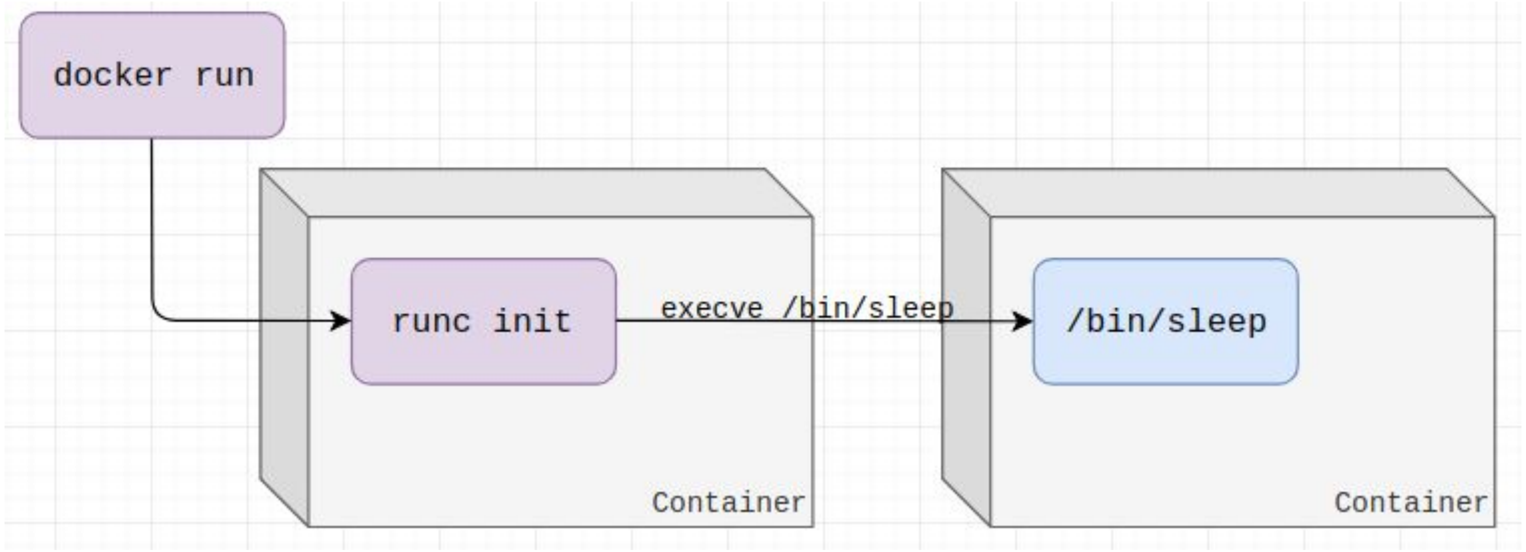
  ...

Starting a Container



- It’s alive!

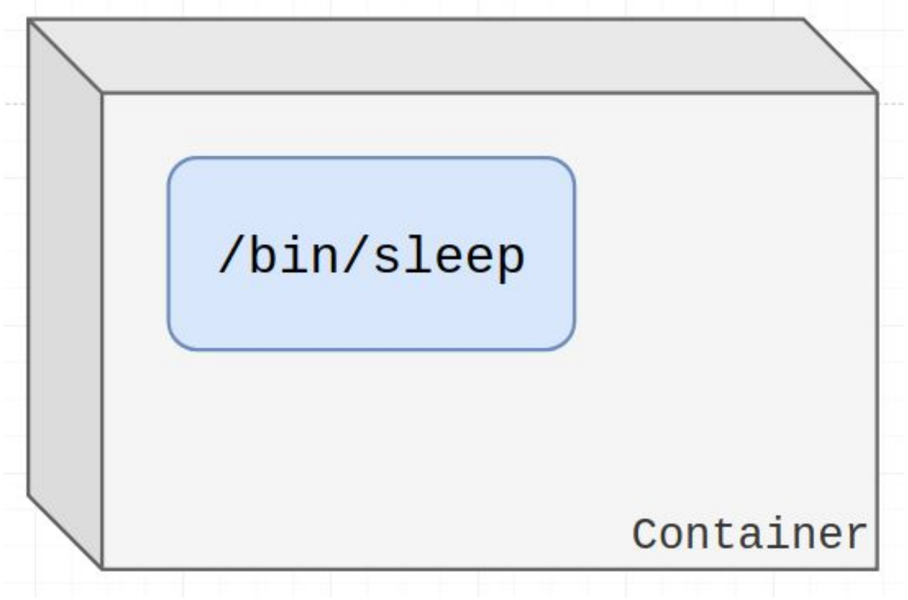
Starting a Container



```
> docker run ubuntu sleep
```

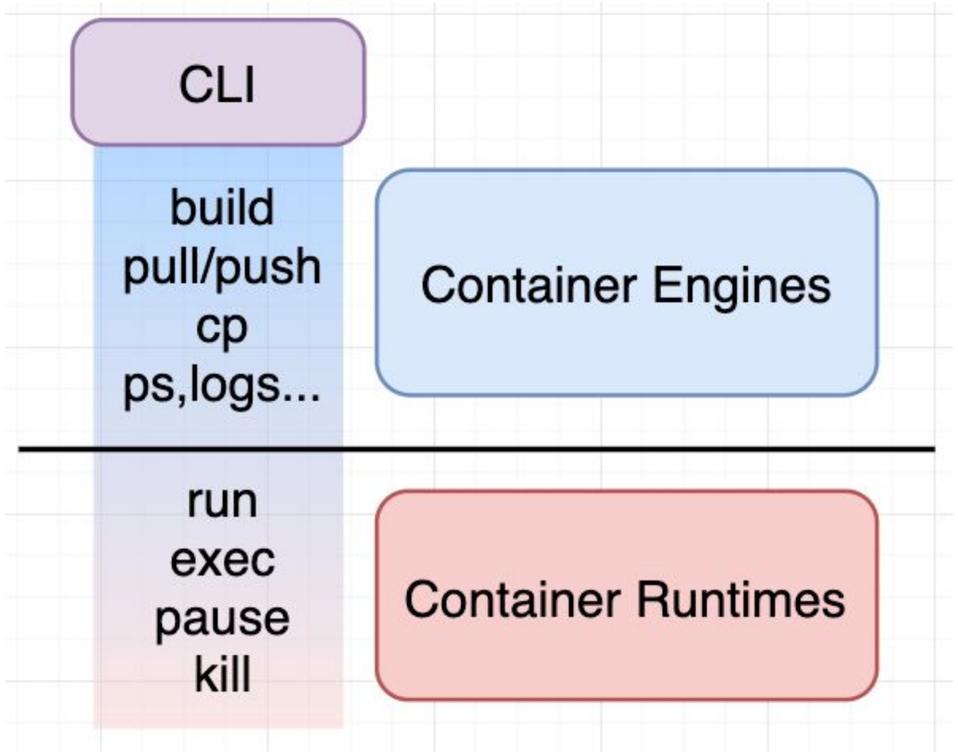
Starting a Container

- Result:



```
> docker run ubuntu sleep
```

Engine or Runtime?



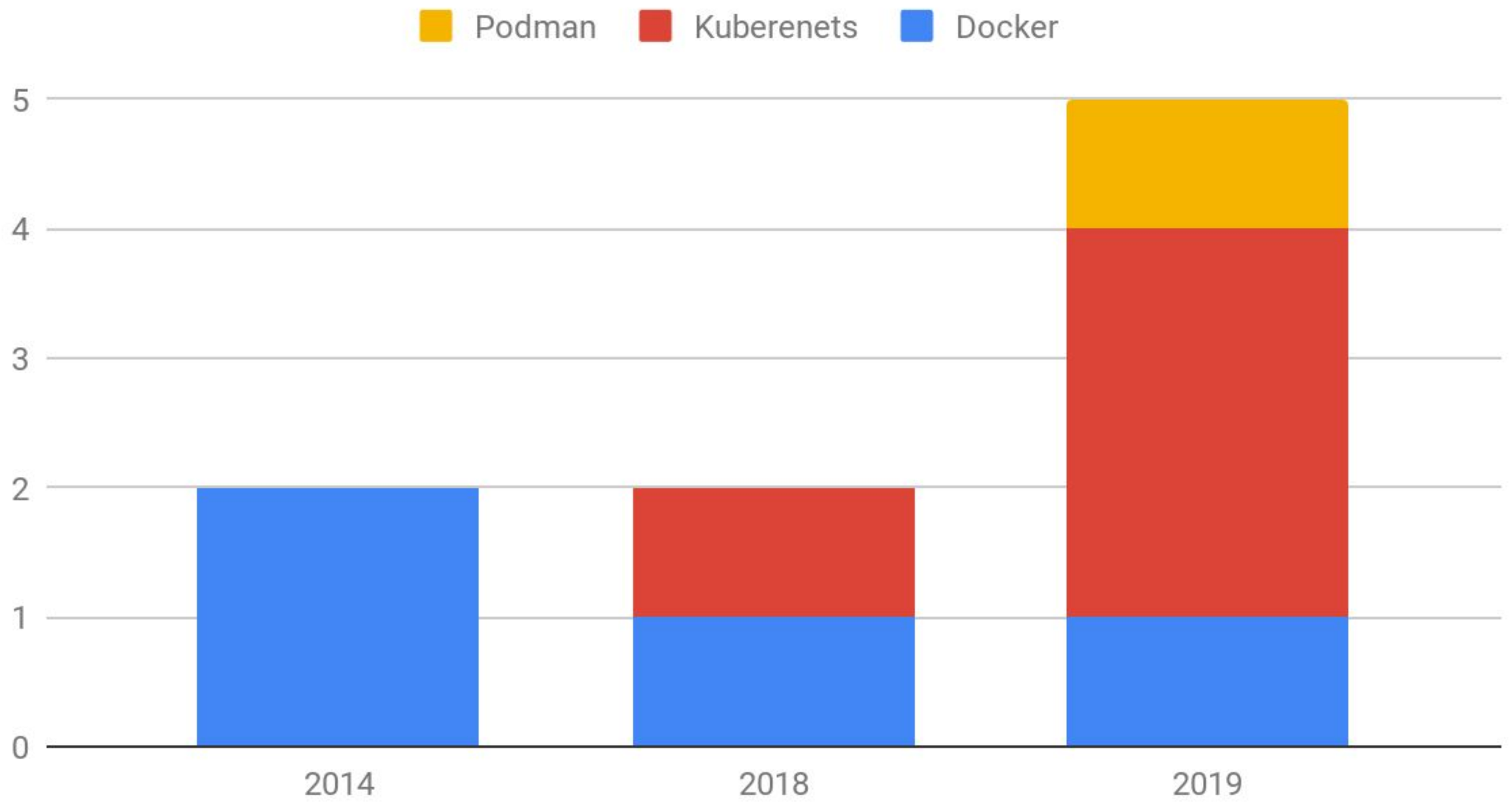
podman
cri-o
docker
lxd
RUNC
LXC

Copy Command

- Copy from a container to host
- Copy from host to container
- Copy between containers

```
> docker cp /tmp/file ubuntu_container:/tmp/file
```

Copy Vulnerabilities Discovered



Podman

```
> podman cp host_file ctr:/dir/abc
```

- Build container path (from host's view)
  - `/var/lib/…/$ctrid/merged + /dir/abc`
- Then copy

```
> cp host_file /var/lib/…/$ctrid/merged/dir/abc
```

So What Could Go Wrong?

- Symlinks!

Case #1 - Podman CVE-2019-10152

- Symlinks resolved under host root

```
fake_dir -> /critical/path
> podman cp host_file ctr:/fake_dir/ab

/critical/path/ab
```

Docker - Copying In

1. Resolve container path in container root
2. Add resolved path to container mount point
3. Copy

```
fake_dir -> /critical/path
> docker cp host_file ctr:/fake_dir/abc
```

```
1. /critical/path/abc
2. /var/lib/.../$ctrid/merged + /critical/path/abc
3. cp host_file /var/.../merged/critical/path/abc
```

Case #2 - Docker CVE-2018-15664

- Symlink exchange race attack

```
docker cp /host_file ctr:/somedir/file
```

1. `/somedir/file`
2. `/var/lib/.../$ctrid/merged + /somedir/file`

`somedir -> /critical/path`

3. `cp /host_file /var/lib/.../merged/somedir/file`

`/critical/path/file`

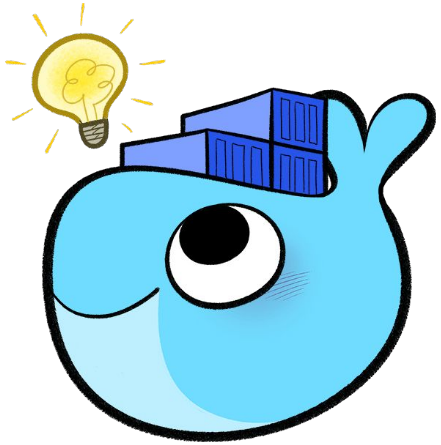
Dealing with symlinks

- **Sort of partially enter the container!**
  - Fork and run helper binary
  - Partially enter container (chroot)
  - Do all steps that can have symlink issues

* Symlinks are resolved under the accessing process root

Docker - Copying Out

- Daemon forks and runs `docker-tar`
  - Chroot to container
  - Tar the requested files
  - Pass back tar to docker daemon
- No symlink issues!



So What Could Go Wrong?

- You're partially entering the container…
  - Creating a bridge between the container and the host

Case #3 - Docker CVE-2019-14271

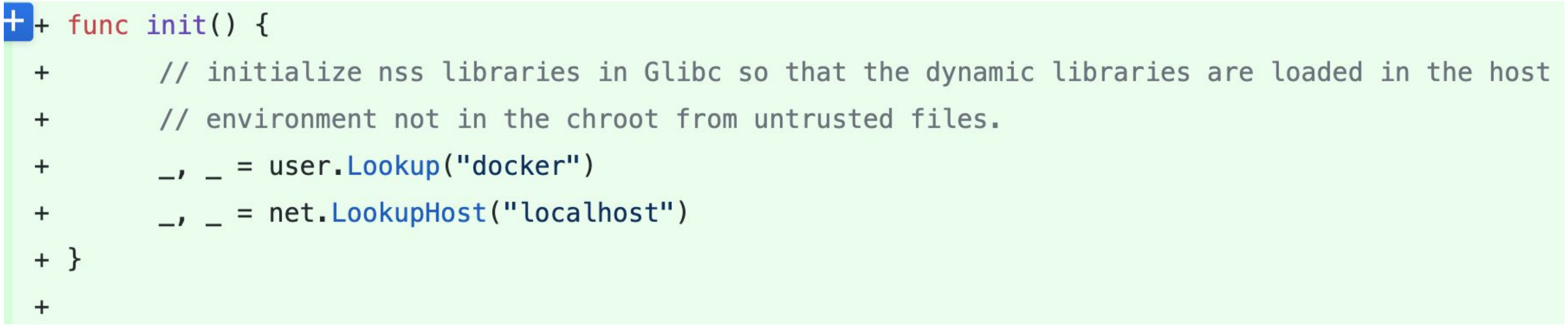
- Full host compromise upon copying out
- `docker-tar` chroots to the container
    - Golang v1.11 feature/bug - some packages (`net`, `os/user`) with cgo (embedded C code) dynamically load shared libraries at run time
- `docker-tar` dynamically loads `libnss_*.so` libraries from the container!

Case #3 - Docker CVE-2019-14271

- Attack scenarios
  - Malicious image with bad `libnss_files.so`
  - Attacker compromised a container and switched `libnss_files.so`
- `PoC`

Case #3 - Docker CVE-2019-14271

- Fix - Force lib loading before chroot



```
+ func init() {
+       // initialize nss libraries in Glibc so that the dynamic libraries are loaded in the host
+       // environment not in the chroot from untrusted files.
+       _, _ = user.Lookup("docker")
+       _, _ = net.LookupHost("localhost")
+ }
+
```

Fully Entering the Container

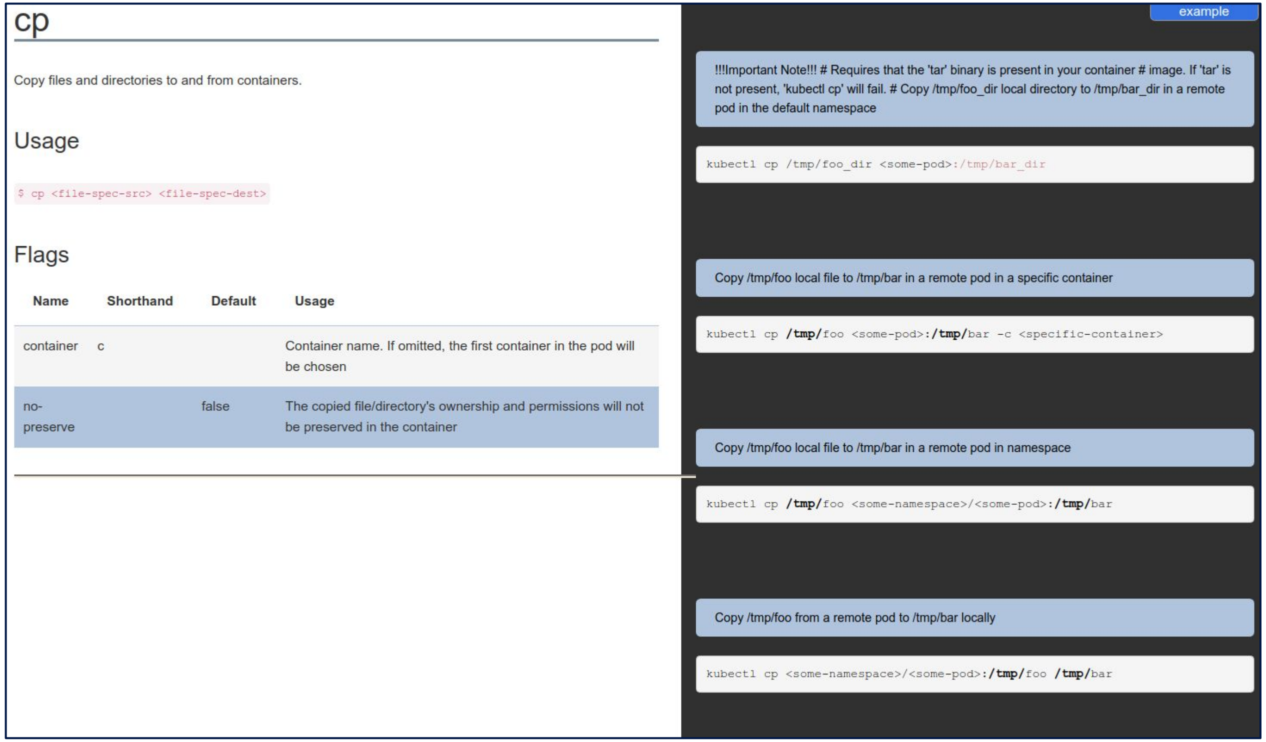
- Helper binary runs inside the container
  - Fully containerized process (`docker exec`)
  - Helper process can’t directly access host

What Could Go Wrong

- Your helper binary is exposed to attackers in the container

Kubernetes Implementation

- `kubectl cp` doc



cp

Copy files and directories to and from containers.

Usage

```
$ cp <file-spec-src> <file-spec-dest>
```

Flags

Name	Shorthand	Default	Usage
container	c		Container name. If omitted, the first container in the pod will be chosen
no-preserve		false	The copied file/directory's ownership and permissions will not be preserved in the container

example

!!!Important Note!!! # Requires that the 'tar' binary is present in your container # image. If 'tar' is not present, 'kubectl cp' will fail. # Copy /tmp/foo_dir local directory to /tmp/bar_dir in a remote pod in the default namespace

```
kubectl cp /tmp/foo_dir <some-pod>:/tmp/bar_dir
```

Copy /tmp/foo local file to /tmp/bar in a remote pod in a specific container

```
kubectl cp /tmp/foo <some-pod>:/tmp/bar -c <specific-container>
```

Copy /tmp/foo local file to /tmp/bar in a remote pod in namespace

```
kubectl cp /tmp/foo <some-namespace>/<some-pod>:/tmp/bar
```

Copy /tmp/foo from a remote pod to /tmp/bar locally

```
kubectl cp <some-namespace>/<some-pod>:/tmp/foo /tmp/bar
```

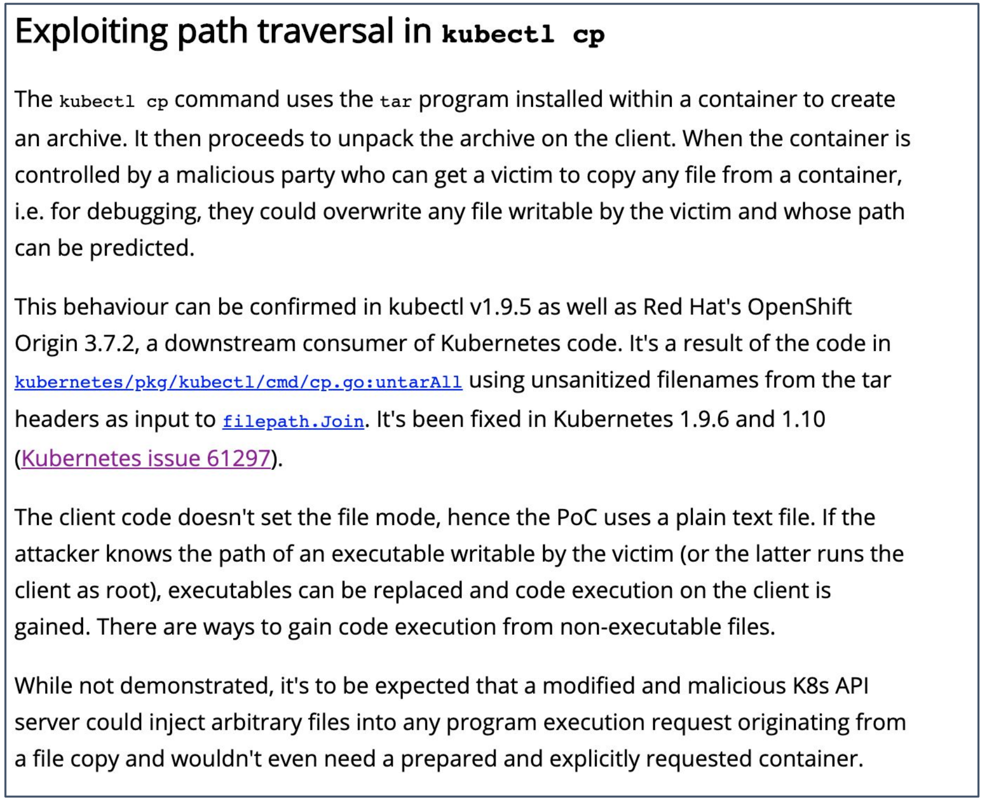
Kubernetes Implementation

- To copy files from a container
  - Kubectl uses the **container's** `tar` binary to archive requested files, unpacks at host
- What if an attacker replaces `tar` binary?

Case #4 - Kubernetes CVE-2018-1002100

March 2018

Michael Hanselmann



Exploiting path traversal in `kubectl cp`

The `kubectl cp` command uses the `tar` program installed within a container to create an archive. It then proceeds to unpack the archive on the client. When the container is controlled by a malicious party who can get a victim to copy any file from a container, i.e. for debugging, they could overwrite any file writable by the victim and whose path can be predicted.

This behaviour can be confirmed in kubectl v1.9.5 as well as Red Hat's OpenShift Origin 3.7.2, a downstream consumer of Kubernetes code. It's a result of the code in `kubernetes/pkg/kubectl/cmd/cp.go:untarAll` using unsanitized filenames from the tar headers as input to `filepath.Join`. It's been fixed in Kubernetes 1.9.6 and 1.10 (Kubernetes issue 61297).

The client code doesn't set the file mode, hence the PoC uses a plain text file. If the attacker knows the path of an executable writable by the victim (or the latter runs the client as root), executables can be replaced and code execution on the client is gained. There are ways to gain code execution from non-executable files.

While not demonstrated, it's to be expected that a modified and malicious K8s API server could inject arbitrary files into any program execution request originating from a file copy and wouldn't even need a prepared and explicitly requested container.

Case #4 - Kubernetes CVE-2018-1002100

- Classic directory traversal
- Tar file includes path with ../ and can escape target directory
  - `/some/remote/dir/../../../../tmp/foo`
  - Writes to `/tmp/foo`
- Fixed by sanitizing path

Case #4.5 - Kubernetes CVE-2019-1002101

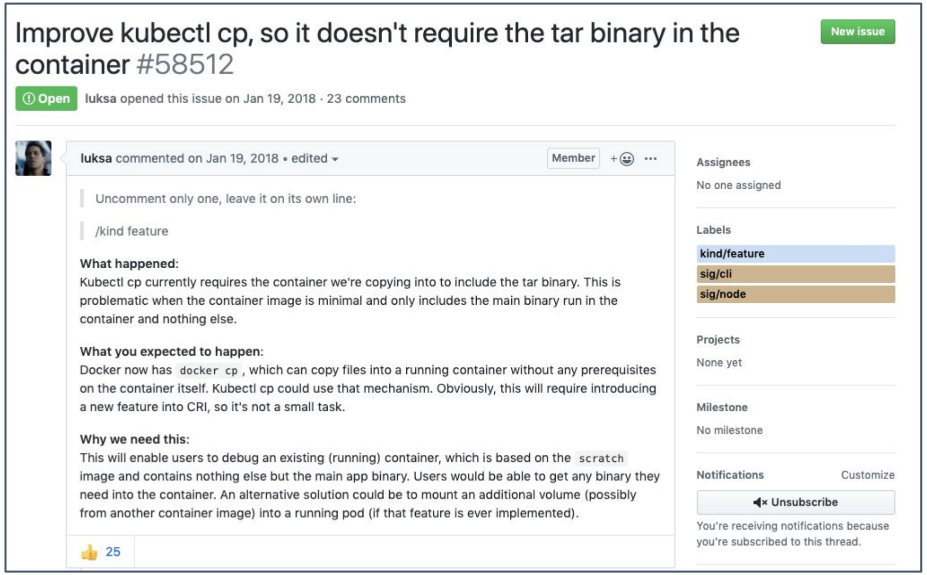
- **Symlinks!**
- Tar format supports files, directories and **symlinks**
- So what?

Case #4.5 - Kubernetes CVE-2019-1002101

- Create a malicious tar that has a header with symlink to an outside directory
    - `/sym -> /critical/path`
    - `/sym/malicous_file`
- Surprise!
    - **`/critical/path/malicious_file`**
    - Kubectl copies last file to the symlink target

Case #4.5 - Kubernetes CVE-2019-1002101

- Disclosed to the Kubernetes and Openshift security teams, patch was issued
- Redesign suggested



Improve kubectl cp, so it doesn't require the tar binary in the container #58512

New issue

Open luksa opened this issue on Jan 19, 2018 · 23 comments

luksa commented on Jan 19, 2018 • edited Member

> Uncomment only one, leave it on its own line:
>
> /kind feature

What happened:
Kubectl cp currently requires the container we're copying into to include the tar binary. This is problematic when the container image is minimal and only includes the main binary run in the container and nothing else.

What you expected to happen:
Docker now has `docker cp`, which can copy files into a running container without any prerequisites on the container itself. Kubectl cp could use that mechanism. Obviously, this will require introducing a new feature into CRI, so it's not a small task.

Why we need this:
This will enable users to debug an existing (running) container, which is based on the `scratch` image and contains nothing else but the main app binary. Users would be able to get any binary they need into the container. An alternative solution could be to mount an additional volume (possibly from another container image) into a running pod (if that feature is ever implemented).

👍 25

Assignees
No one assigned

Labels
kind/feature
sig/cli
sig/node

Projects
None yet

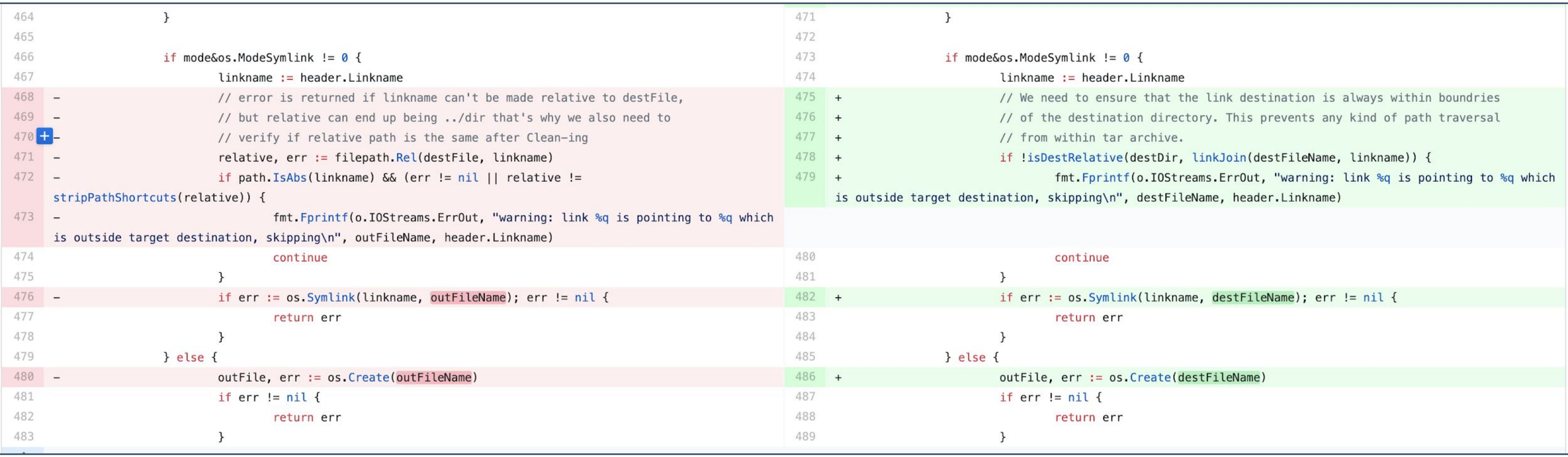
Milestone
No milestone

Notifications Customize
Unsubscribe
You're receiving notifications because you're subscribed to this thread.

OPENSHIFT by Red Hat

Case #4.5 - Kubernetes CVE-2019-11246

- CNCF Security Audit later revealed the fix was insufficient

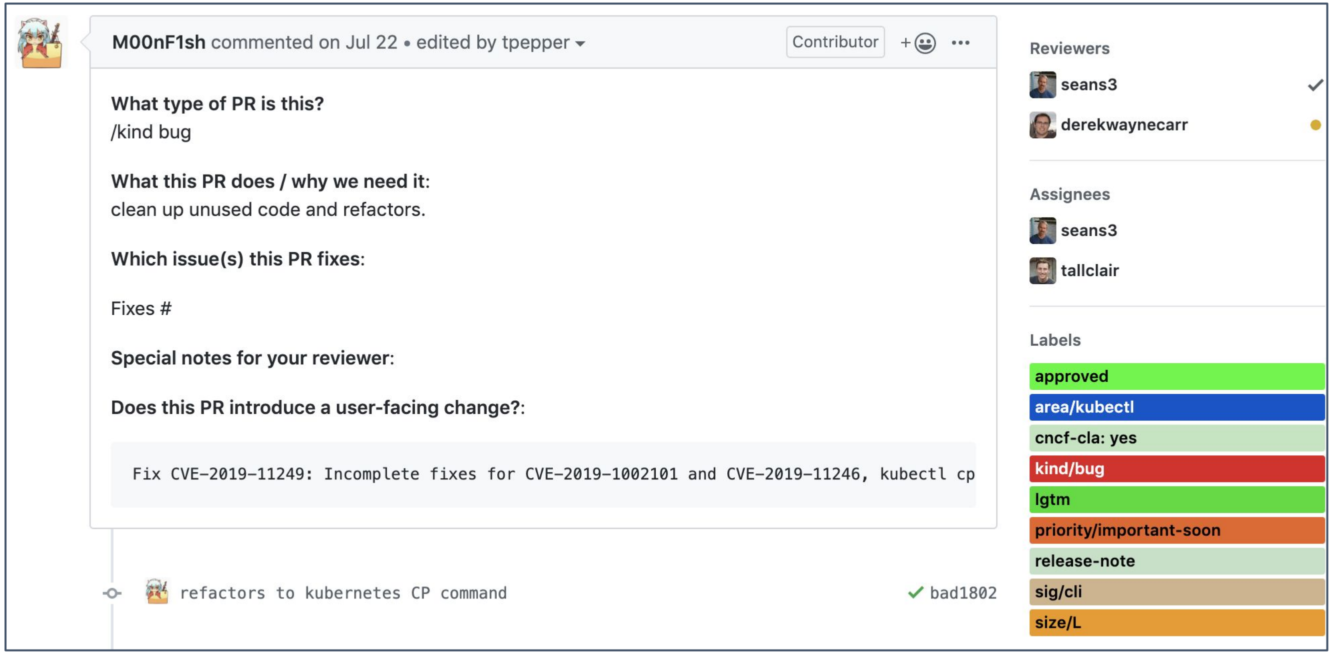


```diff
        }

        if mode&os.ModeSymlink != 0 {
            linkname := header.Linkname
-           // error is returned if linkname can't be made relative to destFile,
-           // but relative can end up being ../dir that's why we also need to
-           // verify if relative path is the same after Clean-ing
-           relative, err := filepath.Rel(destFile, linkname)
-           if path.IsAbs(linkname) && (err != nil || relative != stripPathShortcuts(relative)) {
-               fmt.Fprintf(o.IOStreams.ErrOut, "warning: link %q is pointing to %q which is outside target destination, skipping\n", outFileName, header.Linkname)
                continue
            }
-           if err := os.Symlink(linkname, outFileName); err != nil {
                return err
            }
        } else {
-           outFile, err := os.Create(outFileName)
            if err != nil {
                return err
            }
```

```diff
        }

        if mode&os.ModeSymlink != 0 {
            linkname := header.Linkname
+           // We need to ensure that the link destination is always within boundries
+           // of the destination directory. This prevents any kind of path traversal
+           // from within tar archive.
+           if !isDestRelative(destDir, linkJoin(destFileName, linkname)) {
+               fmt.Fprintf(o.IOStreams.ErrOut, "warning: link %q is pointing to %q which is outside target destination, skipping\n", destFileName, header.Linkname)
                continue
            }
+           if err := os.Symlink(linkname, destFileName); err != nil {
                return err
            }
        } else {
+           outFile, err := os.Create(destFileName)
            if err != nil {
                return err
            }
```

Case #4.5 - Kubernetes CVE-2019-11249

- Symlink restriction is (still) not easy



M00nF1sh commented on Jul 22 • edited by tpepper
Contributor

What type of PR is this?
/kind bug

What this PR does / why we need it:
clean up unused code and refactors.

Which issue(s) this PR fixes:

Fixes #

Special notes for your reviewer:

Does this PR introduce a user-facing change?:

```
Fix CVE-2019-11249: Incomplete fixes for CVE-2019-1002101 and CVE-2019-11246, kubectl cp
```

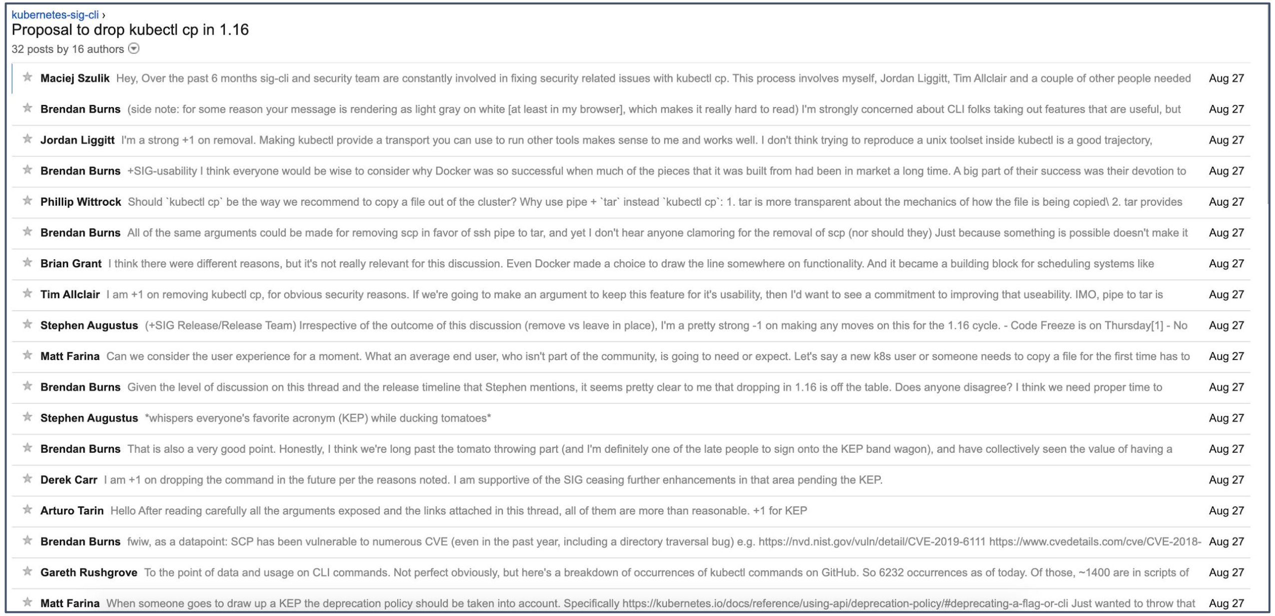
refactors to kubernetes CP command ✓ bad1802

Reviewers
seans3 ✓
derekwaynecarr ●

Assignees
seans3
tallclair

Labels
approved
area/kubectl
cncf-cla: yes
kind/bug
lgtm
priority/important-soon
release-note
sig/cli
size/L

Kubernetes Future

- KEP future-of-kubectl-cp
- 

kubernetes-sig-cli ›
Proposal to drop kubectl cp in 1.16
32 posts by 16 authors

Maciej Szulik Hey, Over the past 6 months sig-cli and security team are constantly involved in fixing security related issues with kubectl cp. This process involves myself, Jordan Liggitt, Tim Allclair and a couple of other people needed Aug 27

Brendan Burns (side note: for some reason your message is rendering as light gray on white [at least in my browser], which makes it really hard to read) I'm strongly concerned about CLI folks taking out features that are useful, but Aug 27

Jordan Liggitt I'm a strong +1 on removal. Making kubectl provide a transport you can use to run other tools makes sense to me and works well. I don't think trying to reproduce a unix toolset inside kubectl is a good trajectory, Aug 27

Brendan Burns +SIG-usability I think everyone would be wise to consider why Docker was so successful when much of the pieces that it was built from had been in market a long time. A big part of their success was their devotion to Aug 27

Phillip Wittrock Should `kubectl cp` be the way we recommend to copy a file out of the cluster? Why use pipe + `tar` instead `kubectl cp`: 1. tar is more transparent about the mechanics of how the file is being copied\ 2. tar provides Aug 27

Brendan Burns All of the same arguments could be made for removing scp in favor of ssh pipe to tar, and yet I don't hear anyone clamoring for the removal of scp (nor should they) Just because something is possible doesn't make it Aug 27

Brian Grant I think there were different reasons, but it's not really relevant for this discussion. Even Docker made a choice to draw the line somewhere on functionality. And it became a building block for scheduling systems like Aug 27

Tim Allclair I am +1 on removing kubectl cp, for obvious security reasons. If we're going to make an argument to keep this feature for it's usability, then I'd want to see a commitment to improving that useability. IMO, pipe to tar is Aug 27

Stephen Augustus (+SIG Release/Release Team) Irrespective of the outcome of this discussion (remove vs leave in place), I'm a pretty strong -1 on making any moves on this for the 1.16 cycle. - Code Freeze is on Thursday[1] - No Aug 27

Matt Farina Can we consider the user experience for a moment. What an average end user, who isn't part of the community, is going to need or expect. Let's say a new k8s user or someone needs to copy a file for the first time has to Aug 27

Brendan Burns Given the level of discussion on this thread and the release timeline that Stephen mentions, it seems pretty clear to me that dropping in 1.16 is off the table. Does anyone disagree? I think we need proper time to Aug 27

Stephen Augustus *whispers everyone's favorite acronym (KEP) while ducking tomatoes* Aug 27

Brendan Burns That is also a very good point. Honestly, I think we're long past the tomato throwing part (and I'm definitely one of the late people to sign onto the KEP band wagon), and have collectively seen the value of having a Aug 27

Derek Carr I am +1 on dropping the command in the future per the reasons noted. I am supportive of the SIG ceasing further enhancements in that area pending the KEP. Aug 27

Arturo Tarin Hello After reading carefully all the arguments exposed and the links attached in this thread, all of them are more than reasonable. +1 for KEP Aug 27

Brendan Burns fwiw, as a datapoint: SCP has been vulnerable to numerous CVE (even in the past year, including a directory traversal bug) e.g. https://nvd.nist.gov/vuln/detail/CVE-2019-6111 https://www.cvedetails.com/cve/CVE-2018- Aug 27

Gareth Rushgrove To the point of data and usage on CLI commands. Not perfect obviously, but here's a breakdown of occurrences of kubectl commands on GitHub. So 6232 occurrences as of today. Of those, ~1400 are in scripts of Aug 27

Matt Farina When someone goes to draw up a KEP the deprecation policy should be taken into account. Specifically https://kubernetes.io/docs/reference/using-api/deprecation-policy/#deprecating-a-flag-or-cli Just wanted to throw that Aug 27

Design Suggestion

- Freeze with freezer cgroup
  - Avoid races
- Enter with caution
  - Mount ns and chroot (LXD)
  - Do not use anything from inside the container
  - Statically linked helper binaries

The Future

- New syscall!
- `openat2()` - restrict path resolution
  - **LOOKUP_BENEATH**
  - **LOOKUP_IN_ROOT**
  - LOOKUP_NO_SYMLINKS
  - LOOKUP_NO_MAGICLINKS
  - LOOKUP_NO_XDEV

KubeCon | CloudNativeCon

North America 2019

Thank you

Ariel Zelivansky | azelivansky@paloaltonetworks.com

Yuval Avrahami | yavrahami@paloaltonetworks.com

Unit42.paloaltonetworks.com

Appendix - Copy vulnerabilities

- Docker moby#5720, moby#6000, CVE-2018-15664, CVE-2019-14271
- Kubernetes CVE-2018-1002100, CVE-2019-1002101, CVE-2019-11246, CVE-2019-11249
- Podman CVE-2019-10152