



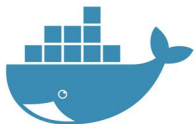
CVE-2019-5736

RunC Vulnerability

Jay, Anish

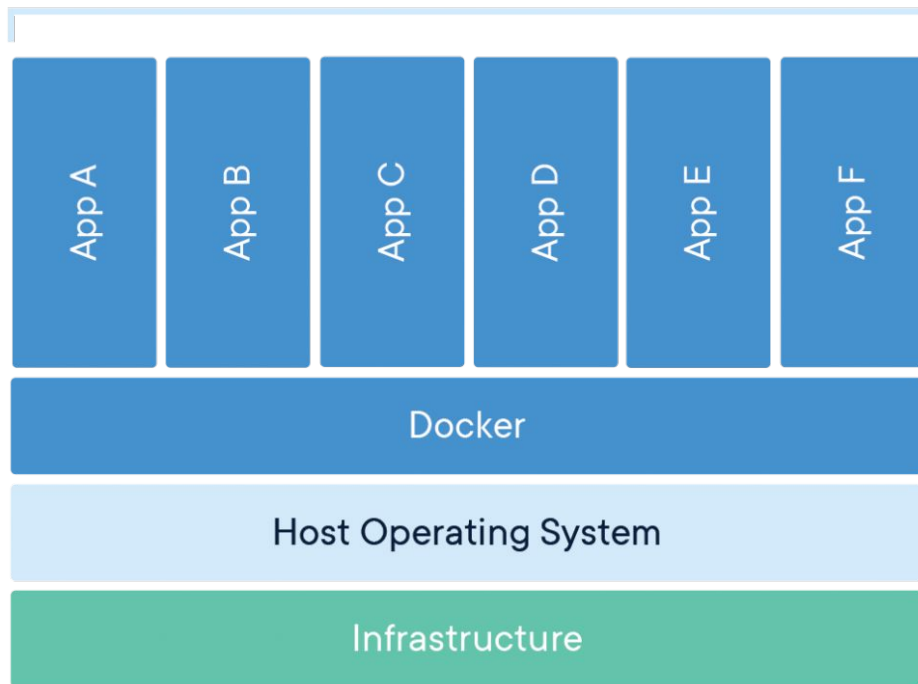
Containers

- Package of software that has all code and dependencies
- Standalone, lightweight, has everything that it needs to run
- Containers run on “Engines”
 - As long as there is an engine running on the host machine, a container will run on the engine



Containers

Containerized Applications



Containers - More Detail

- Namespaces
 - Pid namespaces: You become PID 1, and children are other processes, separate from host
(Demo)
- Cgroups
 - Limit processes in how much memory of CPU the process can run
- Seccomp-bpg
 - Limit what system calls a process can run



What is runC?


- runC is a tool which contains all the system features container engines use
- Used for spawning and running containers
- Mostly used by container software like Docker, but can be used as a standalone tool - Docker Demo



Proc File System (Procfs)

- Virtual filesystem that presents information about processes
- Every process has a folder in this file system, labelled using its PID
- Interface to system data that the kernel exposes as a filesystem
- Lets see this

Note: a container can have its own /proc filesystem. A process within the container cannot see the /proc folders outside this container.



Proc File System (Procfs) - /proc/self

- Symbolic link to the /proc/<pid> directory of the currently running process
 - /proc/self/exe - symbolic link to executable file the process is running - Demo
 - /proc/self/fd - directory containing the file descriptors open by the process
- /proc/self/exe points to the process' binary data in memory
- When accessed, the kernel hands over the file data directly from kernel memory instead of resolving the link.

Demo



CVE-2019-5736

Vulnerability discovered Feb. 11th 2019 by some of the maintainers of RunC (Adam Iwaniuk and Borys Popławski).

Manipulates the RunC binary and the way we create containers.

End goal: re-write the runC binary

Allows an attacker to run arbitrary code with root access.



CVE-2019-5736

Exploits 2 important properties:

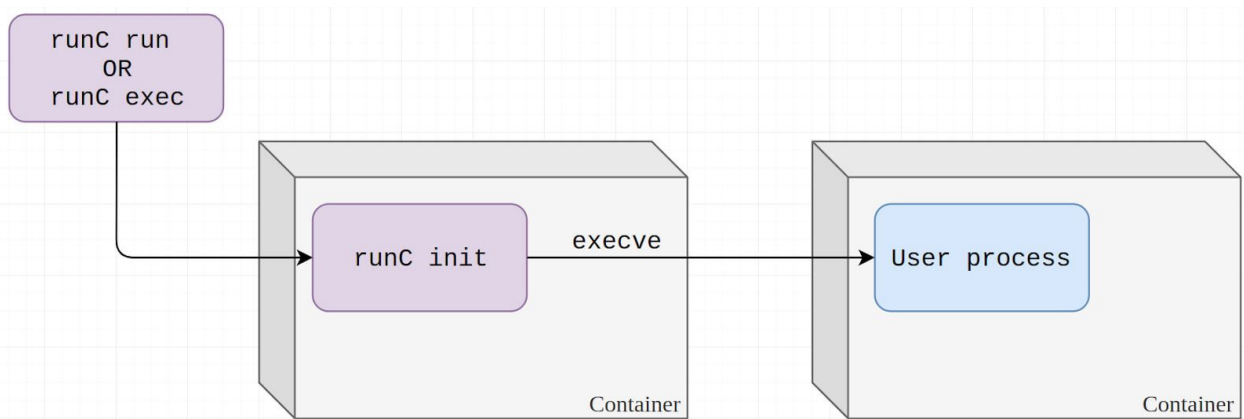
- The symbolic link at `/proc/self/exe`
 - Does not follow the usual semantics for symbolic links
 - The link points to the open file in memory (the currently running process). When accessed, the kernel gives the open file directly.
- The privileged container
 - Containers which map the root host
 - Docker containers are privileged by default



Spawning a Container

When RunC creates a container, it places itself in a privileged container and begins placing restrictions from inside.

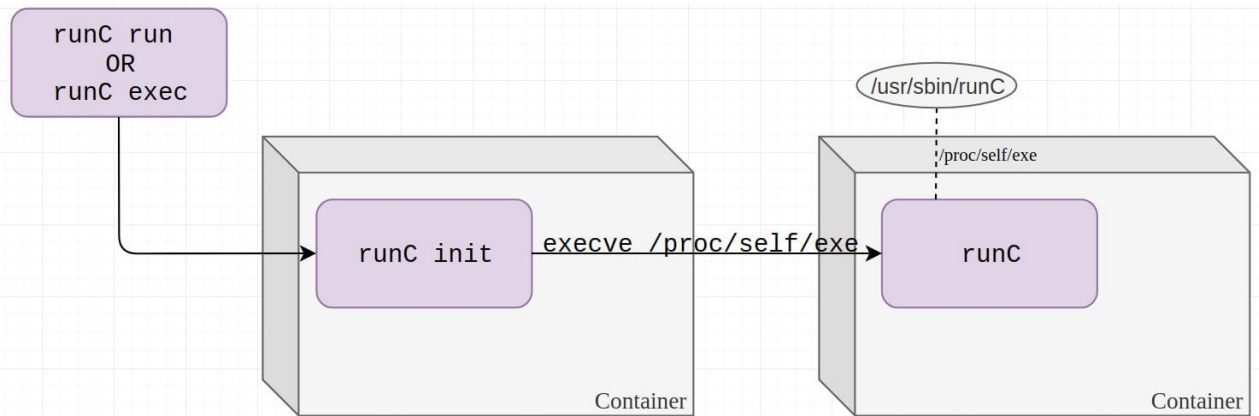
It then runs the target process and terminates itself.



Tricking RunC

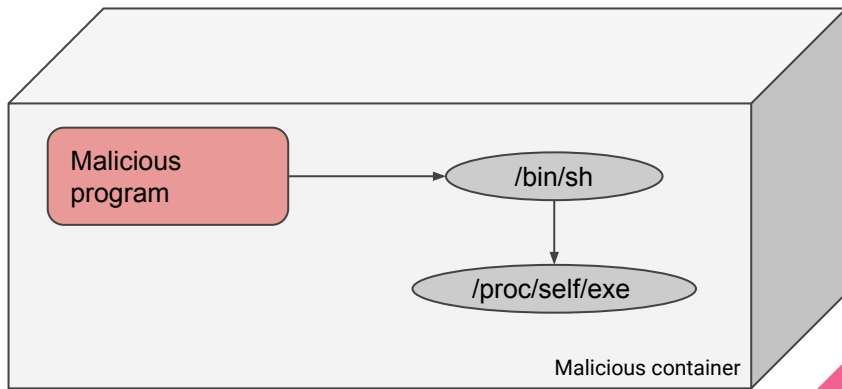
It is possible to make runC run itself.

This will create file descriptors for itself within the container, which we can use to write to the runC binary.



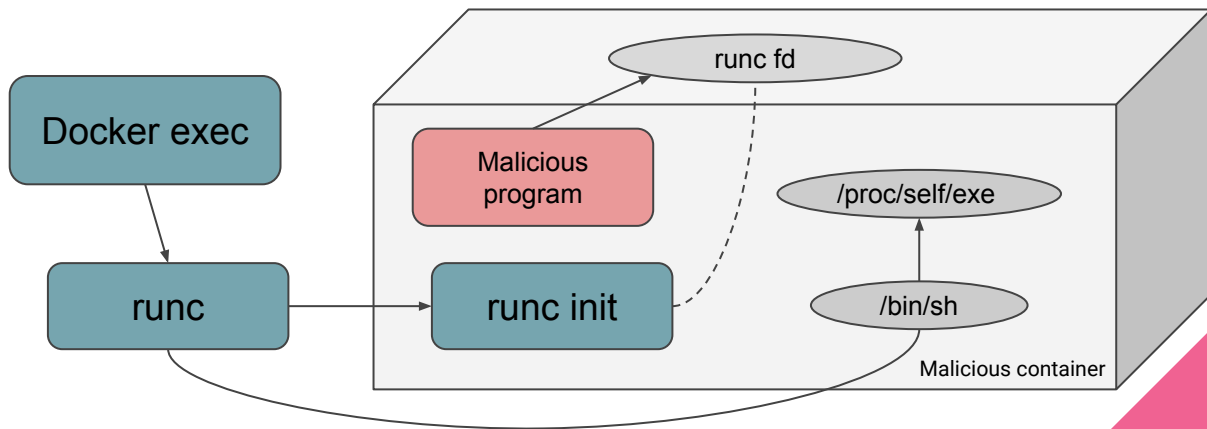
Docker exec

- We create a malicious program and run it inside a new container
 - The program will overwrite `/bin/sh` with `/proc/self/exe` and start checking for a process that points to runC
- Calling `Docker exec [args] /bin/sh` would cause Docker's runC to run itself



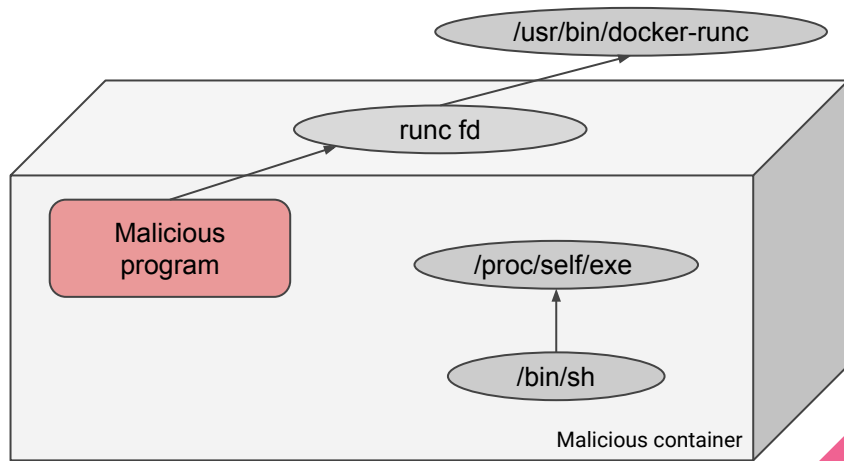
Docker exec

- We create a malicious program and run it inside a new container
 - The program will overwrite `/bin/sh` with `/proc/self/exe` and start checking for a process that points to `runC`
- Calling `Docker exec [args] /bin/sh` would cause Docker's `runC` to run itself



Docker exec

- Once runc terminates, we can open the binary for writing and add our exploit code.
- Whenever someone tries to run Docker again, it'll run our code with root privilege

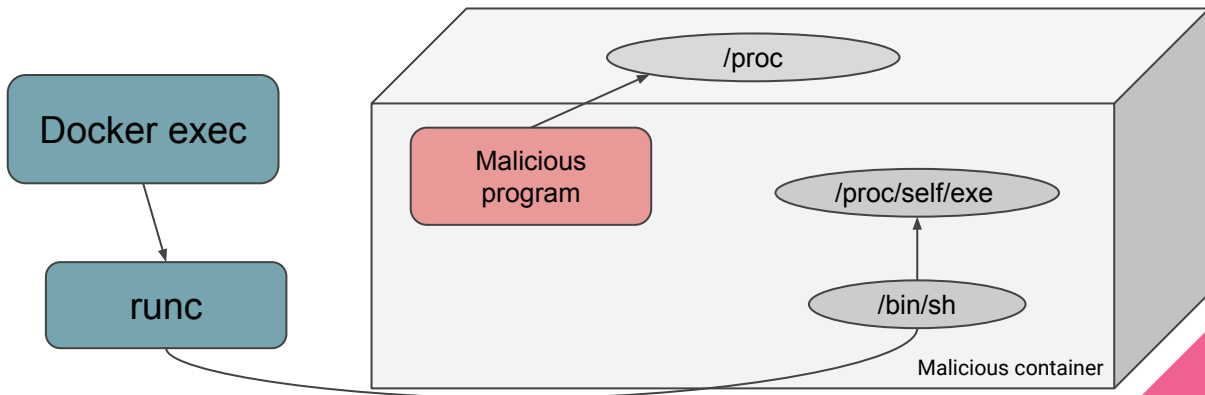


Breakdown - what's going on here? (1)

We've built our own container that contains its own filesystem, include its own /proc directories, and overwritten the /bin/sh symlink.

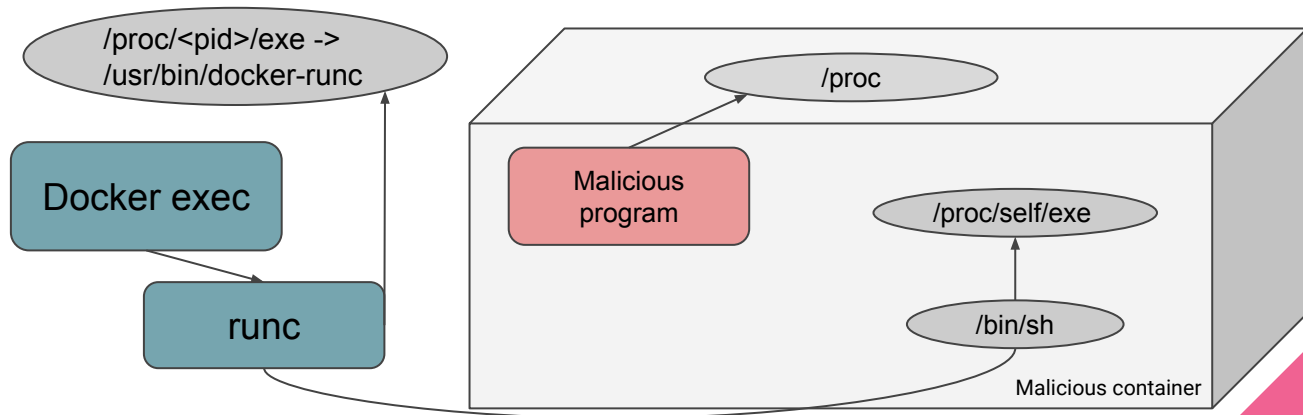
Docker exec launches RunC and looks for “/bin/sh”, the process we requested to run, inside the container.

Meanwhile, our malicious program is scanning the /proc filesystem for a /proc/<pid>/exe that points to RunC



Breakdown - what's going on here? (2)

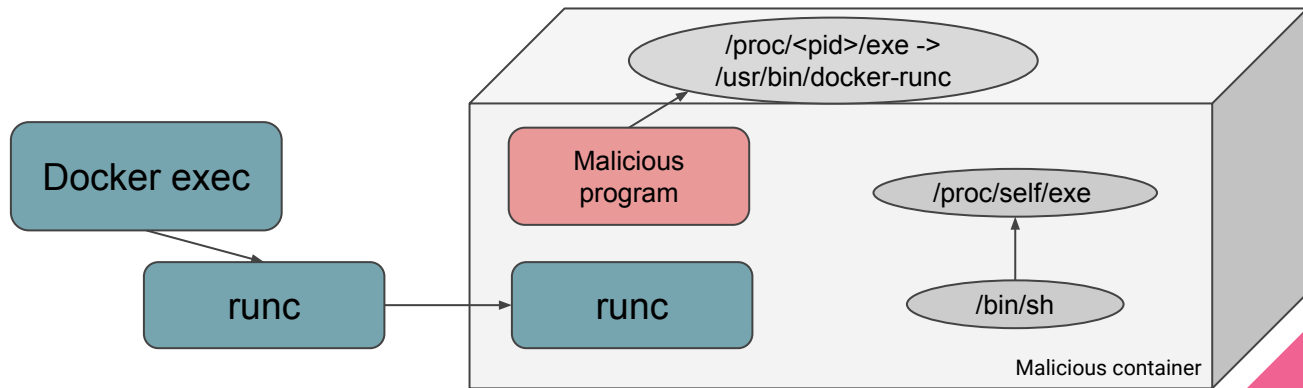
`/bin/sh` returns `/proc/self/exe`. RunC goes through its own `/proc` directory (on the host system, since this process belongs to the host) to find the `exe` symlink, and gets pointed to the `runc` binary.



Breakdown - what's going on here? (3)

So RunC runs an instance of RunC within the container.

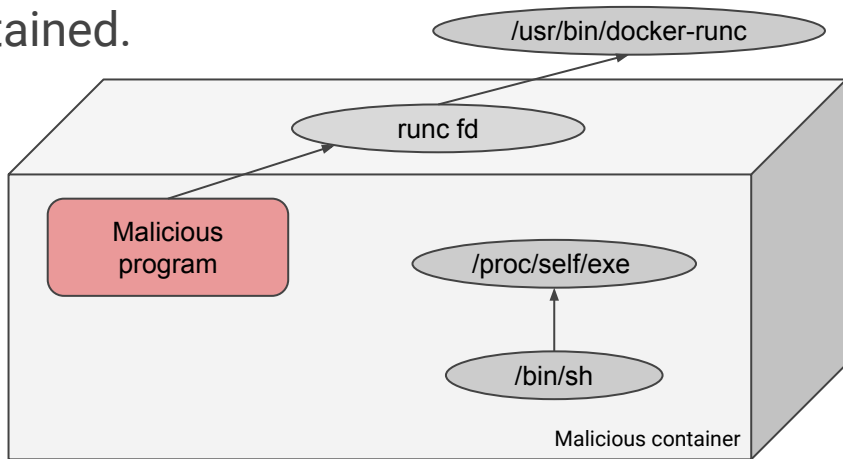
A directory now exists within our container's `/proc/` filesystem that points to RunC. We can't write to the binary yet, since the process is still running (the Kernel will deny write requests), but we can save its file descriptors.



Breakdown - what's going on here? (4)

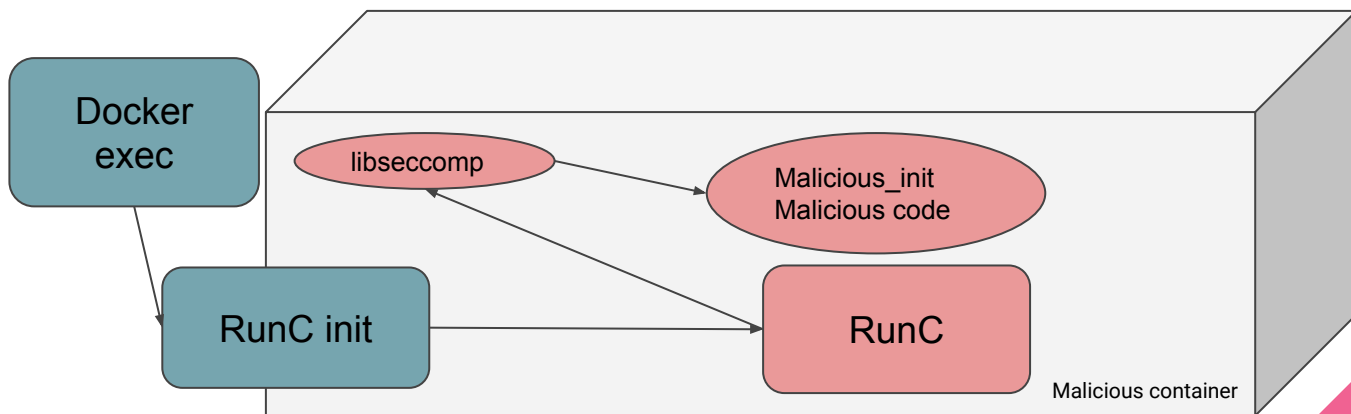
We gave no arguments to RunC (since we were initially just running `/bin/sh`) so RunC terminates without launching another process. Docker exec reports that it couldn't run `sh` (since it still thinks it was attempting to run `/bin/sh`).

Now we can write to the RunC binary, which is still in memory, using the file descriptors we obtained.



Shared Library Approach

- Create our own versions of one of RunC's shared libraries, and save it in the container.
- The new RunC will find our local version of the library before looking for the host system's copy.



Mitigation

- Patch your system!
- Don't give root permissions to software you don't trust.
- Set restrictions on your containers. The exploit does not work without root uid

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: non-root
spec:
  privileged: false
  allowPrivilegeEscalation: false
  runAsUser:
    # Require the container to run without root privileges.
    rule: 'MustRunAsNonRoot'
```


The Patch - Docker and LXC

Docker and LXC mitigate this vulnerability by copying itself into a new, sealed (read only) file in memory.

This new file is then run inside the new container, rather than placing itself directly in the container.

This way, write operations from within the privileged container to the host binary will instead go to the temporary copy file. This file gets deleted once the container terminates.

-> Memory Constraints? We make a new runC binary whenever
we make a new container



Sources

<https://jvns.ca/blog/2016/10/10/what-even-is-a-container/>

<https://seclists.org/oss-sec/2019/q1/119>

<https://kubernetes.io/blog/2019/02/11/runc-and-cve-2019-5736/>

<https://www.twistlock.com/labs-blog/breaking-docker-via-runc-explaining-cve-2019-5736/>

<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-5736>

