Containers

RunC

Proc File System

/proc/<pid>/exe

This is a symbolic link that points back to the running process. However, unlike a regular symbolic link, /proc/<pid>exe points directly to the process' data in kernel memory.

For example, if we deleted a process' binary but the process is still running, then its binary is still loaded in kernel memory. We can read the binary from /proc/<pid>/exe even though the binary itself is deleted.

/proc/self

This is a symbolic link that can be seen by every process, and resolves to the process' /proc/<pid> directory.

CVE-2019-5736

When a container engine (like Docker, Kubernetes, etc) creates a container or attaches to a container (ex: Docker exec) it runs RunC, which then does an "execv()" on the process we give it.

The RunC vulnerability exploits this and the /proc/self/exe link to write to the host's RunC binary from within the container. By telling RunC to run /proc/self/exe, an attacker can create a /proc directory for RunC within the container which contains the /<pid>/exe link that points directly tp the RunC binary in memory.

note: this exploit requires a root use inside the container (for write access to RunC inside the container), and that the victim running the malicious container has root access as well (to execv RunC).

Exploit

Step 1: setup

We create a container with a virtual linux file system (this file system links to the system's files, like /bin/sh and /bin/bash, but we can access these).

We can overwrite the /bin/sh link to return /proc/self/exe, so that a user attempting to connect to our container (ex: using docker exec) will cause RunC to run itself.

We then run a process which will loop through each /proc folder looking for a /proc/<pid>/exe that points to RunC.

Step 2: trigger

When a user connects to the container, they may want to open a shell in order to examine the files and processes inside. When they do this (for example, if they try to run /bin/sh where we've overwritten) RunC will receive the /proc/self/exe link and attempt to run itself.

The new instance of RunC will be spawned from inside the container, creating its entry in the /proc file system.

The malicious program within the container will find this folder and save its file descriptors (/proc/<pid>/fd) and start trying to open the file for writing. It won't work yet (since we can't write to the binary while RunC is still running) but the program will keep trying until it does.

When RunC terminates (we didn't give it appropriate arguments since we were trying to launch /bin/sh) it will simply close with a usage message and "docker exec" will report that it couldn't run /bin/sh. Now the malicious process can write a payload to the RunC binary.

Step 3: infected RunC

Now that a payload has been written to RunC, whenever a superuser attempts to create a container they will run an instance of RunC with root privileges, which will run the attacker's code.

Mitigation