最初理解赛题

增强现有Linux容器技术的文件系统隔离能力，抵御非法文件操作相关容器逃逸漏洞

其次在完成上述情况下减小性能开销

学习和问题

1，学习

（1） Linux的安装和基础使用

（2） Docker的安装和基础了解

（3） 要抵御漏洞，了解容器安全抵御机制

（4） 复现漏洞理解漏洞攻击原理，寻求方法如何解决该漏洞

2，问题

（1）现在是通过复现漏洞来从相关方面加强，目前我能了解的加强是通过主机配置和相关权限限制来控制，比如设置文件所属权限，开启实时监控，文件异常行为检测

（2）如何编写一些检测程序来加强？怎么优化？

（3）最后是通过攻击来检测容器安全，我们如何相同的进行测试（不了解如何评判，大概通过漏洞复现可以比较慢的检测？还是说需要了解漏洞挖掘相关知识）

这个现在解决了

（4）复现结束后解决方案-安装最新版本docker？？？

二，再次思考

完全不清楚前进方向

主要目的是对现有轻量安全虚拟化技术进行优化或者提出新的方法

缺乏实践

漏洞测试相关

关于回放中提到的CVE漏洞。
1、回放中的CVE漏洞不是必须使用的，也可以使用其他的漏洞进行测试。
2、使用CVE漏洞进行防御测试是为了更好地量化防御效果，如果大家有更好的验证方法也可以提出来，比如自己编写攻击脚本模拟攻击。
这里也给大家分享一个链接，里面容器逃逸相关的漏洞比较全面，如果需要可以参考一下：https://www.container-security.site/attackers/container_breakout_vulnerabilities.html

不同版本容器下载 https://download.docker.com/

漏洞复现操作手册 https://cvexploits.io/search

Docker 安全性解决实践: https://docs.docker.com/engine/security/

漏洞逃逸复现1 docker逃逸漏洞复现（CVE-2019-5736） - FreeBuf网络安全行业门户

NVD - CVE-2019-5736 (nist.gov)

主要漏洞

1. CVE-2019-5736
2. CVE-2019-14271
3. CVE-2021-25741
4. CVE-2022-0492
5. CVE-2017-1002101
6. CVE-2018-15664
7. CVE-2019-10152
8. CVE-2019-18466
9. CVE-2022-1227
10. CVE-2023-0778
11. CVE-2021-30465
12. CVE-2022-23648
13. CVE-202423652
14. CVE2024-21626

进一步思考

在实现一个漏洞复现后，发现对赛题的理解似乎有偏差

漏洞在最新版本的docker上进行一些安全配置就能解决

docker安全配置相关

相关参考

https://www.bilibili.com/read/cv15553799/?spm_id_from=333.999.collection.opus.click

容器安全并不是简单在某一环做出相应的安全配置就可以的，我们需要考虑四个主要方面，一是内核的内在安全性及其对命名空间和cgroup的支持，二是Docker 守护进程本身的攻击面，三是容器配置文件中的漏洞，四是内核的“强化”安全特性以及它们如何与容器交互。所以我们必须从基础操作系统环境、容器服务、容器镜像以及业务与运维开发人员分别入手，才能提高容器的安全性，减少被攻击的可能性。

Docker 服务为了防止黑客在控制容器后能够对宿主机进行攻击，提供了三个主要的隔离机制，其分别是 Namespace 机制、Capabilities 机制和 CGroups 机制。如果攻击对宿主机产生了影响，就说明入侵者已经突破了Docker服务的保护，这就是容器安全中常说的Docker容器逃逸。

默认情况下，Docker 启动的容器被严格限制只允许使用内核的一部分能力。并且Docker采用白名单机制，禁用必需功能之外的其它权限

为了加强安全，容器可以禁用一些没必要的权限。

完全禁止任何 mount 操作；

禁止直接访问本地主机的套接字；

禁止访问一些文件系统的操作，比如创建新的设备、修改文件属性等；

禁止模块加载。

这样就算攻击者在容器中取得了 root 权限，也不能获得本地主机的较高权限，能进行的破坏也有限

从3大机制方面理解下docker基础安全防御

主要从漏洞复现去思考怎么解决

给定14个漏洞都是非法文件操作，所以从文件操作方面加强

以下为一些可进行操作的环节

参考https://www.bilibili.com/read/cv15554240/?spm_id_from=333.999.collection.opus.click

1.主机安全配置

1.1 更新docker到最新版本

1.2 为容器创建一个单独的分区

1.3 只有受信任的用户才能控制docker守护进程

1.4 审计docker守护进程

1.5 审计docker相关的文件和目录

2.docker守护进程配置

2.1 限制默认网桥上容器之间的网络流量

2.2 设置日志级别为info

2.3 允许 docker 更改iptables

2.4 不使用不安全的镜像仓库

2.5 建议不使用aufs存储驱动程序

2.6 docker守护进程配置TLS身份认证

2.7 配置合适的 ulimit 资源控制

2.8 启用用户命名空间

2.9 使用默认cgroup

2.10 启用docker客户端命令的授权

2.11 配置集中和远程日志记录

2.12 禁用docker resitry v1版本支持

2.13 启用实时恢复

2.14 禁用 userland 代理

2.15 限制容器获取新的权限

3.docker 守护进程文件配置

3.1 设置 docker.service 文件所属和权限

3.2 设置docker.socket文件所属和权限

3.3 设置/etc/docker目录所有权为root:root

3.4 设置仓库证书文件所有权为root：root

3.5 设置TLS CA证书文件所有权为root:root

3.6 设置docker服务器证书文件所有权为root:root

3.7 设置docker服务器证书密钥文件所有权为root：root

3.8 设置/var/run/docker.sock文件所有权为root:docker

3.9 设置daemon.json文件所有权为root：root

3.10 设置 /etc/default/docker 文件所有权为 root:root

4.容器镜像和构建文件

4.1 创建容器的用户

4.2 容器使用可信的基础镜像

4.3 容器中不安装没有必要的软件包

4.4 扫描镜像漏洞并且构建包含安全补丁的镜像

4.5 启用docker内容信任

4.6 将HEALTHCHECK说明添加到容器镜像

4.7 不在dockerfile中单独使用更新命令

4.8 镜像中删除setuid和setgid权限

4.9 在dockerfile中使用copy而不是add

4.10 涉密信息不存储在dockerfile

4.11 仅安装已经验证的软件包

4.12 容器内部项目指定运行用户

5.容器运行时保护

5.1 设置SElinux安全选项

5.2 linux内核特性在容器内受限

5.3 不使用特权容器

5.4 敏感的主机系统目录不要挂载在容器上

5.5 SSH 不在容器中运行

5.6 特权端口禁止映射到容器内

5.7 只映射必要的端口

5.8 不共享主机的网络命名空间

5.9 确保容器的内存使用合理

5.10 正确设置容器上的CPU优先级

5.11 确保进入容器的流量绑定到特定的网卡

5.12 容器重启策略on-failure设置为5

5.13 确保主机的进程命名空间不共享

5.14 主机的IPC命令空间不共享

5.15 主机设备不直接共享给容器

5.16 设置默认的ulimit配置（在需要时）

5.17 设置主机的UTS命令空间不共享

5.18 不要使用docker的默认网桥docker0

寻求老师帮助，重新认识赛题

去询问赛题的技术老师，对赛题进一步理解

本项目的主要目标是采用轻量级、高效的设计理念，实现更强大的Linux文件系统隔离，同时最小化对性能的影响。可对现有轻量安全虚拟化技术进行优化，或提出新的轻量安全虚拟化方法并实现。

我们这个目标，其实目的是优化现有安全虚拟化技术或者提出一种更好的安全虚拟化技术。使用cve漏洞进行防御测试，只是为了更好的验证方案的安全性，而不是目的

目前主流的安全容器有kata和gvisor

kata是把容器跑在轻量虚拟机中，gvisor是用户态模拟内核

所以是从这些安全容器下手，了解并优化，在满足防御漏洞逃逸的情况下减少消耗性能，在指导老师的回放里讲过这两个性能开销一个20%一个10%

最新版的kata会更低

优化的话，从cpu、内存、硬盘io，这几个是对系统方案的开销。容器本身一般会测试下容器的启动时间

我决定从kata入手，因为kata近年来发展迅速，相对完善一些

kata的具体性能开销

Kata Containers的性能开销取决于多种因素，包括硬件配置、虚拟化技术的选择以及工作负载的特性。Kata Containers是一种轻量级虚拟机，旨在与Docker和Kubernetes等容器编排软件无缝集成，提供额外的隔离性，尤其适用于运行不受信任的工作负载。以下是具体介绍：

1. **启动时间**：相比于传统容器，Kata容器的启动时间略长，但这种差异在现代硬件上已经变得不那么显著。
2. **磁盘I/O性能**：在某些情况下，Kata容器的性能可能低于裸机或原生容器，特别是在处理大量随机I/O操作时。然而，对于顺序写入，Kata容器的性能可能会优于裸机解决方案。
3. **内存开销**：Kata Containers 2.0通过使用Rust重写，显著降低了容器运行时的内存开销，使得容器更加轻量级。
4. **网络性能**：Kata容器支持高效的网络接口，如vhost-user，这有助于提高网络I/O性能。
5. **CPU性能**：Kata容器在不同的CPU架构上表现不同，Westmere及之后的处理器架构通常能提供更好的性能。
6. **存储优化**：利用virtio-fs等高效的文件系统共享方式可以提升存储性能，尤其是在支持嵌套虚拟化的环境下。
7. **资源管理**：合理的资源分配和限制可以避免资源过度竞争，从而减少性能损失。
8. **安全与性能的平衡**：虽然Kata容器提供了额外的安全性，但这可能会带来一定的性能开销。用户需要根据实际的安全需求和性能目标来做出权衡。

此外，为了进一步提升Kata Containers的性能，可以考虑以下几点：

1. **硬件加速**：确保底层系统支持相关的硬件虚拟化技术，以获得最佳性能。
2. **裁剪内核**：使用裁剪过的内核以减少不必要的设备支持，减轻虚拟化的负担。
3. **监控与调试**：利用Kata Containers提供的日志管理和监控工具，及时排查性能瓶颈并进行调优。

总的来说，Kata Containers的性能开销与多种因素相关，包括硬件配置、工作负载类型以及虚拟化技术的选择。通过合理的配置和优化，可以在保证安全性的同时，尽可能地减少性能损失。

优化

针对Kata Containers的输入输出优化

1. **使用virtio-fs**：virtio-fs是一种高效的文件系统共享方式，它可以在宿主机和容器之间提供快速的文件系统访问。通过使用virtio-fs，可以减少文件系统操作的延迟，提高I/O性能。
2. **配置vhost-user**：vhost-user是一种高性能的网络接口，它允许在用户空间中实现网络协议栈的处理，从而减少数据包在内核空间和用户空间之间的拷贝次数。通过配置vhost-user，可以提高网络I/O的性能。
3. **利用DAX技术**：DAX（Direct Access）技术允许容器直接访问宿主机的内存，从而减少数据在宿主机和容器之间的拷贝次数。通过利用DAX技术，可以提高I/O性能，尤其是在需要大量内存访问的场景下。
4. **精简镜像大小**：精简容器镜像大小，移除不必要的文件和依赖，以加快传输和加载速度。这可以通过使用更小的基础镜像、移除不必要的文件和依赖以及使用更有效的打包工具来实现。
5. **优化启动流程**：优化容器的启动流程，减少初始化所需时间。这可以通过减少不必要的服务启动、优化配置文件的位置和格式以及使用更快的启动脚本来实现。
6. **监控与调试**：利用Kata Containers提供的日志管理和监控工具，及时排查性能瓶颈并进行调优。这可以帮助开发者和运维人员更好地了解I/O性能的状况，并采取相应的优化措施。
7. **硬件虚拟化支持**：确保底层系统支持相关虚拟化技术，并通过`kata-runtime kata-check`命令进行检查。这可以确保Kata Containers能够充分利用硬件虚拟化技术的优势，提高I/O性能。
8. **定制配置**：通过configuration.toml文件可调整各种设置，以适应不同的系统需求。例如，可以调整网络和存储相关的配置，以提高I/O性能。

此外，在进行I/O优化时，还可以考虑以下几点：

1. 根据实际应用场景选择合适的虚拟化技术，如Qemu或Firecracker，以及对应的配置选项。
2. 考虑使用云原生技术栈中的其他工具，如Kubernetes和containerd，与Kata Containers协同工作，以获得更好的性能和管理能力。
3. 定期检查并应用Kata Containers的更新和补丁，以利用最新的性能改进和安全修复。

virtio-fs在kata上配置

irtio-fs是一种高效的文件系统共享方式，它利用了FUSE协议在宿主机和虚拟机之间进行通信，通过virtio作为传输层来承载FUSE协议，而不是使用传统的/dev/fuse设备。这种设计使得virtio-fs能够在不同的虚拟机guest之间以快速、一致且安全的方式共享同一个宿主机目录树结构，同时具有较好的性能和与本地文件系统（如ext4）相同的语义。以下是配置virtio-fs的步骤：

1. **安装必要软件**：确保你的系统中已经安装了Kata Containers和Docker，并且Docker配置为可以使用Kata作为其容器运行时环境。

2. **启动容器**：使用以下命令启动一个新的容器，该命令会指定使用virtio-fs作为文件系统的共享方式。这里使用的是qemu作为hypervisor，你也可以根据需要选择其他的hypervisor

```
docker run --runtime=kata-qemu-virtiofs -it busybox
```

3. **验证配置**：为了验证新容器是否正在运行并使用了qemu hypervisor以及virtiofsd，可以查找宿主机上的hypervisor路径和virtiofs守护进程。

```
ps -aux | grep virtiofs
```

你应该能看到输出，显示了qemu hypervisor和virtiofsd的进程信息

此外，还可以尝试使用cloud-hypervisor VMM来体验virtio-fs。使用以下命令启动一个新的容器实例：

```
docker run --runtime=kata-clh -it busybox
```

配置vhost-user

参考 [在arm64上基于qemu的vhost user blk设备hotplug_qemu hotplug-CSDN博客]

在Kata Containers上配置vhost-user，主要是为了提升网络I/O的性能。以下是具体步骤：

1. 首先，确保已经安装了Docker和Kata Containers。如果没有安装，可以访问以下链接进行安装：
   - Docker: https://docs.docker.com/get-docker/
   - Kata Containers: https://github.com/kata-containers/documentation/blob/master/install.md
2. 克隆Kata Containers的GitHub仓库：

```
git clone https://github.com/kata-containers/kata-containers.git
```

3. 进入Kata Containers的目录：

```
cd kata-containers
```

4. 创建一个名为`vhost_user`的目录，用于存放配置文件：

```
mkdir -p /etc/kata-containers/configuration/vhost_user
```

5. 在`vhost_user`目录下创建一个名为`config.json`的文件，并添加以下内容：

```
{
  "name": "vhost-user",
  "description": "Virtual Host User Configuration",
  "type": "guest",
  "vm": {
    "runtime": "qemu",
    "kernel": "/path/to/your/kernel",
    "cmdline": "console=ttyS0 console=tty0",
    "rootfs": "/path/to/your/rootfs"
  },
```

```
  "network": {
    "type": "vhost-user",
    "tap": true,
    "port": "tap0"
  }
}
```

注意：请将 `/path/to/your/kernel` 和 `/path/to/your/rootfs` 替换为实际的内核和根文件系统路径。

6. 修改 `/etc/kata-containers/configuration/kata-containers.conf` 文件，添加以下内容：

```
[vhost_user]
path = /etc/kata-containers/configuration/vhost_user
```

7. 重启Kata Containers服务：

```
systemctl restart kata-agent kata-proxy kata-runtime kata-shim
```

现在，Kata Containers已经配置了vhost-user。可以使用以下命令创建一个新的容器，并指定使用vhost-user配置：

```
docker run --rm -it --privileged --net=none --name vhost-user-container kata-runtime-vhost-user
```

下面是一个优化启动的脚本

```
`#!/bin/bash`

`#设置环境变量`

`export KATA_HYPERVISOR=qemu`
`export KATA_VM_PATH=/var/run/kata-containers/shared/sandboxes/default`
`export KATA_IMAGE_PATH=/var/lib/kata-containers/images`
`export KATA_SHARED_MEM_SIZE=2G`
`export KATA_SHM_SIZE=67108864`
`export KATA_VHOST_USER_SCSI=true`
`export KATA_EXTRA_ARGS="--network=vhost-user --vsock-id=3"`

`#创建共享目录`

`mkdir -p $KATA_VM_PATH`

``chmod 777 $KATA_VM_PATH`

`#创建镜像存储目录`

`mkdir -p $KATA_IMAGE_PATH`
`chmod 777 $KATA_IMAGE_PATH`

`#启动Kata Containers`
```

```
`kata-runtime kata-start --mem 512M --cpu 2 --kernel /path/to/kernel --initrd
/path/to/initrd --image /path/to/image --append "console=ttyS0 root=/dev/vda rw"
--share-dir $KATA_VM_PATH --share-dir $KATA_IMAGE_PATH --hypervisor
$KATA_HYPERVISOR --extra-args "$KATA_EXTRA_ARGS"`
```

当然，使用和验证时确保已经下载了Kata Containers，并且为脚本开放权限

致谢，总结与反思

首先感谢赛题指导老师们和李航，黄伯虎老师的指导和帮助

在探索赛题途中，由于队伍成员之间发生矛盾，赛题被搁置甚至于放弃

这里特别感谢指导老师的支持和帮助，让我决定把赛题做下去

由于中间的耽搁，时间其实是不够的，gitlab上也没有记录进程，加上学校实验全都开始，一个人的力量有限，做的程度终究是不够完善的

许多优化方案没来得及测试，许多代码来不及写

但是我也已经相当满意了

学习的意义不在于结果，而是旅途的收获，不是么?

我加深了和熟练了对linux的使用，学会了配置和使用docker（以前没实践用过），了解了许多机制和原理

学会写shell脚本，在了解漏洞复现和容器安全时又接触了云原生等前沿技术

看了很多论文和资料，又学会了很多方便快捷的小工具

我收获满满，不会去怨谁，不会惋惜是否能继续做下去

如果您（各位）看到这里，也非常感谢您能抽出时间认真读我写的文档

毕竟自己努力的结果，哪怕不尽如人意，也是希望被认真对待的

就这样，请收下吧，这是我最后的波纹啦

感谢！祝各位事业有成，万事如意