VSRC2017城市沙龙

Docker安全：从入门到实战

tuhao

VSRC 2017-07-29

唯品会安全应急响应中心

目录

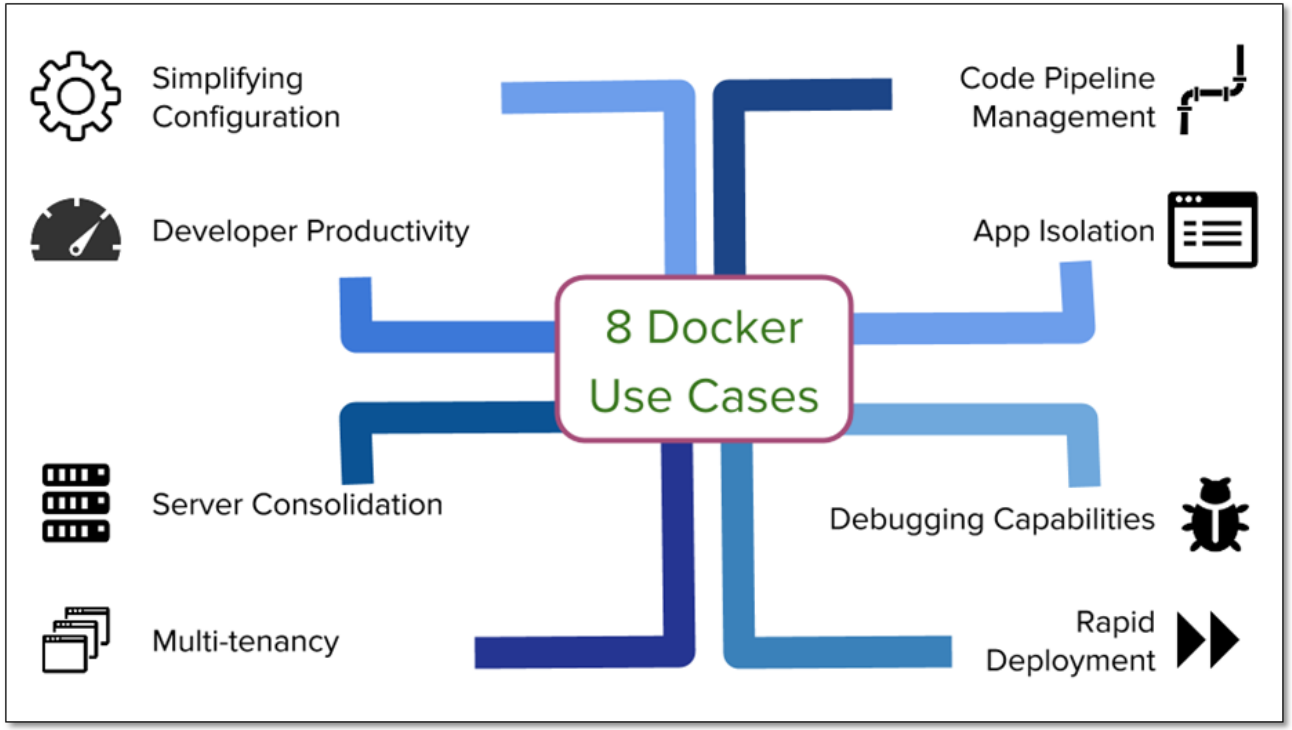
CONTENTS

01 Docker安全概述

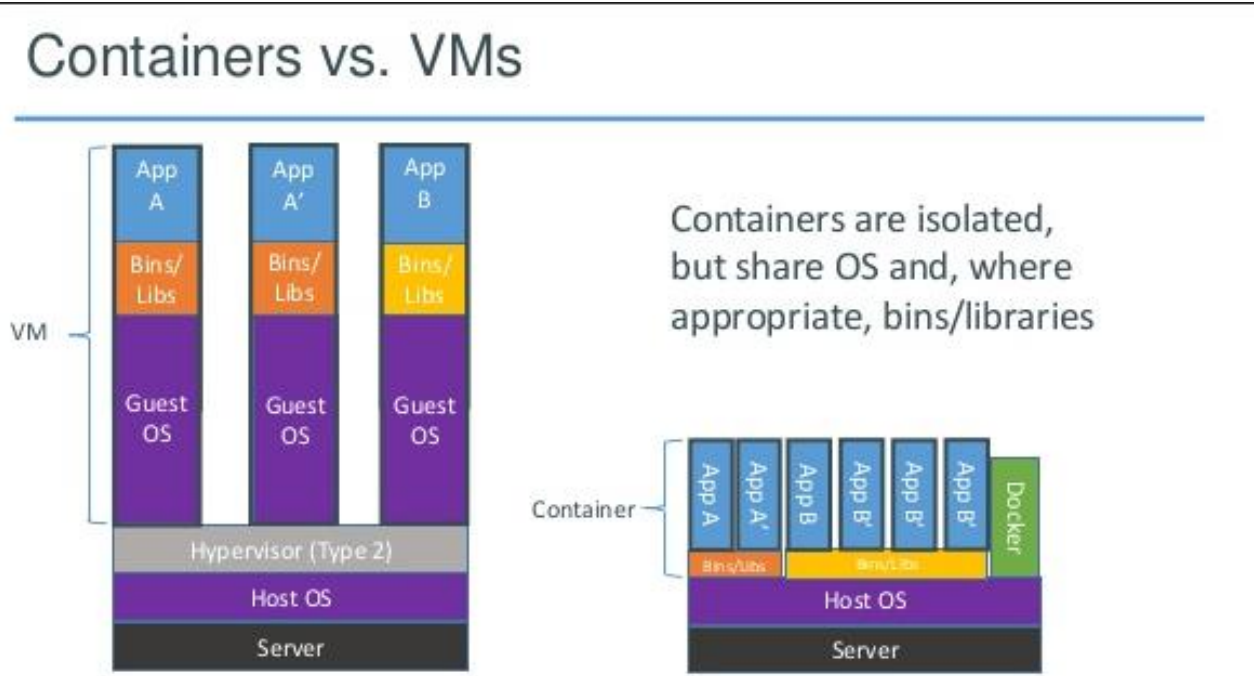
Docker安全概述

- Docker存在的意义
- Docker与VM对比
- Docker安全问题

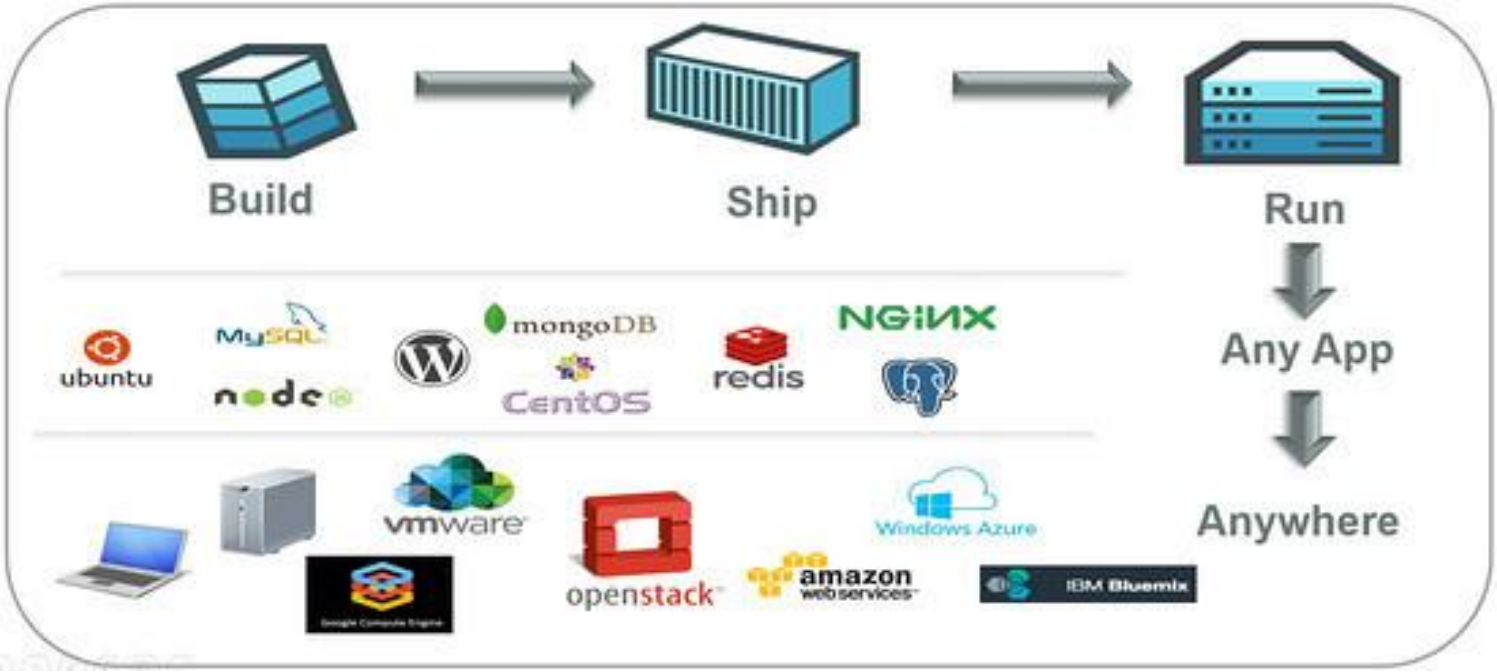
Docker存在的意义



Docker与VM对比



Docker安全问题

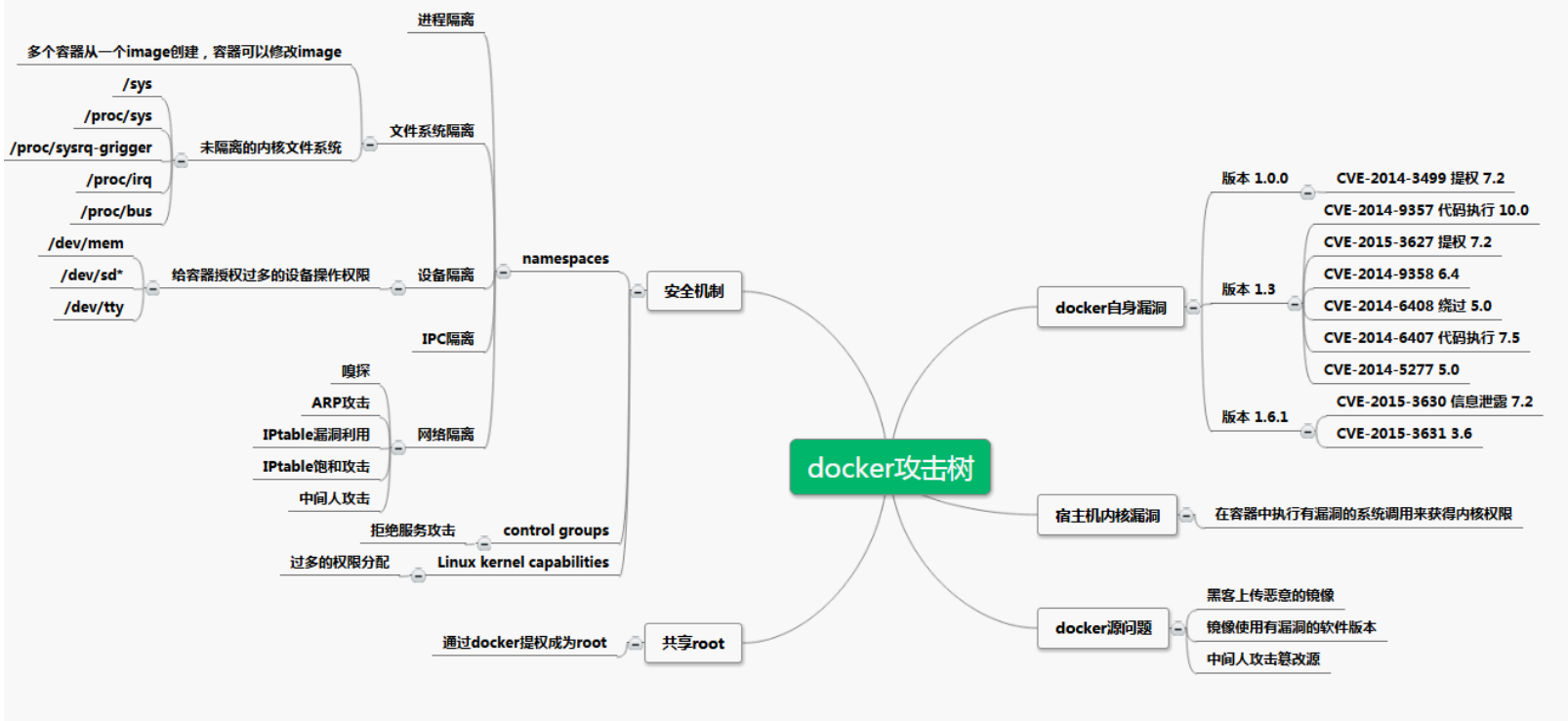

Build
Ship
Run
Any App
Anywhere
ubuntu
MySQL
node
mongoDB
CentOS
redis
NGINX
vmware
openstack
Windows Azure
amazon web services
IBM Bluemix

02 Docker漏洞与利用

Docker漏洞与利用

- Docker攻击树
- Docker渗透案例

Docker攻击树


docker攻击树
安全机制
namespaces
进程隔离
文件系统隔离
多个容器从一个image创建，容器可以修改image
未隔离的内核文件系统
/sys
/proc/sys
/proc/sysrq-grigger
/proc/irq
/proc/bus
设备隔离
给容器授权过多的设备操作权限
/dev/mem
/dev/sd*
/dev/tty
IPC隔离
网络隔离
嗅探
ARP攻击
IPtable漏洞利用
IPtable饱和攻击
中间人攻击
control groups
拒绝服务攻击
Linux kernel capabilities
过多的权限分配
共享root
通过docker提权成为root
docker自身漏洞
版本 1.0.0
CVE-2014-3499 提权 7.2
版本 1.3
CVE-2014-9357 代码执行 10.0
CVE-2015-3627 提权 7.2
CVE-2014-9358 6.4
CVE-2014-6408 绕过 5.0
CVE-2014-6407 代码执行 7.5
CVE-2014-5277 5.0
版本 1.6.1
CVE-2015-3630 信息泄露 7.2
CVE-2015-3631 3.6
宿主机内核漏洞
在容器中执行有漏洞的系统调用来获得内核权限
docker源问题
黑客上传恶意的镜像
镜像使用有漏洞的软件版本
中间人攻击篡改源

Docker渗透案例

Docker remote api漏洞利用

利用条件：
1.Docker remote api无认证且对外开放
2.使用root用户启动docker
3.允许对宿主任意磁盘/目录进行读写
利用方法：
启动一个容器，挂载宿主机的/root/目录，之后将攻击者的ssh公钥~/.ssh/id_rsa.pub的内容写到入宿主机的/root/.ssh/authorized_keys文件中，之后就可以用root账户直接登录了
启动一个容器，挂载宿主机的/etc/目录，之后将反弹shell的脚本写入到/etc/crontab中，攻击者会得到一个反弹的shell。第2种利用方法也可以挂载/var/spool/cron/目录，将反弹shell的脚本写入到/var/spool/cron/root（centos系统）或/var/spool/cron/crontabs/root(ubuntu系统)

Docker渗透案例

kubernets api未授权漏洞导致node宿主沦陷

```
// 获得所有节点
> kubectl -s http://1.2.3.4:8080/ get nodes
// 获得所有容器
> kubectl -s http://1.2.3.4:8080/ get pods --all-namespaces=true
// 在 myapp 容器获得一个交互式 shell
> kubectl -s http://1.2.3.4:8080/ --namespace=default exec -it myapp bash
```

如果可以控制容器的运行，我们也可以尝试获取宿主机（即 nodes）的权限。 参考 Docker Remote API 未授权访问漏洞利用，流程大体为创建新的容器 -> 挂载宿主机目录 -> 写 /etc/crontab 定时任务反弹 shell。


```
{
  "paths": [
    "/api",
    "/api/v1",
    "/apis",
    "/apis/extensions",
    "/apis/extensions/v1beta1",
    "/healthz",
    "/healthz/ping",
    "/logs/",
    "/metrics",
    "/resetMetrics",
    "/swagger-ui/",
    "/swaggerapi/",
    "/ui/",
    "/version"
  ]
}
```

03 Docker安全基线

Docker安全基线

- 内核级别
- 主机级别
- 网络级别
- 镜像级别
- 容器级别
- 其他

内核级别

· 及时更新内核
· User NameSpace（容器内的root权限在容器之外处于非高权限状态）
· Cgroups（对资源的配额和度量）
· SELiux/AppArmor/GRSEC（控制文件访问权限）
· Capability（权限划分）
· Seccomp（限定系统调用）
· 禁止将容器的命名空间与宿主机进程命名空间共享

主机级别

· 为容器创建独立分区
· 仅运行必要的服务
· 禁止将宿主机上敏感目录映射到容器
· 对Docker守护进程、相关文件和目录进行审计
· 设置适当的默认文件描述符数
· 用户权限为root的Docker相关文件的访问权限应该为644或者更低权限
· 周期性检查每个主机的容器清单，并清理不必要的容器

网络级别

· 通过iptables设定规则实现禁止或允许容器之间网络流量
· 允许Dokcer修改iptables
· 禁止将Docker绑定到其他IP/Port或者Unix Socket
· 禁止在容器上映射特权端口
· 容器上只开放所需要的端口
· 禁止在容器上使用主机网络模式
· 若宿主机有多个网卡，将容器进入流量绑定到特定的主机网卡上

镜像级别

- 创建本地镜像仓库服务器
- 镜像中软件都为最新版本
- 使用可信镜像文件，并通过安全通道下载
- 重新构建镜像而非对容器和镜像打补丁
- 合理管理镜像标签，及时移除不再使用的镜像
- 使用镜像扫描
- 使用镜像签名

容器级别

- 容器最小化，操作系统镜像最小集
- 容器以单一主进程的方式运行
- 禁止privileged标记使用特权容器
- 禁止在容器上运行ssh服务
- 以只读的方式挂载容器的根目录系统
- 明确定义属于容器的数据盘符
- 通过设置on-failure限制容器尝试重启的次数
- 限制在容器中可用的进程树，以防止fork bomb

其他设置

· 定期对宿主机系统及容器进行安全审计
· 使用最少资源和最低权限运行容器
· 避免在同一宿主机上部署大量容器，维持在一个能够管理的数量
· 监控Docker容器的使用，性能以及其他各项指标
· 增加实时威胁检测和事件响应功能
· 使用中心和远程日志收集服务

04 Docker安全规则

Docker安全规则

- Docker remote api访问控制
- 限制流量流向
- 使用普通用户启动Docker服务
- 文件系统限制
- Docker client端与 Docker Daemon的通信安全
- 资源限制
- 镜像安全
- 漏洞扫描
- 安装安全加固
- 能力限制
- suid/guid限制

Docker remote api访问控制

Docker的远程调用API接口存在未授权访问漏洞，至少应限制外网访问。
如果可以，建议还是使用socket方式访问。
建议监听内网ip或者localhost，docker daemon启动方式：
docker -d -H uninx:///var/run/docker.sock -H
tcp://10.10.10.10:2375#或者在docker默认配置文件指定
other_args=" -H unix:///var/run/docker.sock -H
tcp://10.10.10.10:2375"
然后在宿主iptables上做访问控制
*filter:
HOST_ALLOW1 - [0:0]
-A HOST_ALLOW1 -s 10.10.10.1/32 -j ACCEPT
-A HOST_ALLOW1 -j DROP
-A INPUT -p tcp -m tcp -d 10.10.10.10 --port 2375 -j HOST_ALLOW1

限制流量流向

使用以下iptables 限制Docker容器的源IP地址与外界通讯

```
iptables -A FORWARD -s <source_ip_range> -j REJECT --reject-with icmp-admin-prohibited
iptables -A FORWARD -i docker0 -o eth0 -j DROP
iptables -A FORWARD -i docker0 -o eth0 -m state –state ESTABLISHED -j ACCEPT
```

使用普通用户启动Docker服务

截至Docker 1.10用户命名空间由docker守护程序直接支持。此功能允许容器中的root用户映射到容器外部的非uid-0用户，这可以帮助减轻容器中断的风险。此功能可用，但默认情况下不启用。

1.使用用户映射

对于Docker，这意味着将其作为-lxc-conf参数添加到docker run：

```
docker run -lxc-conf ="lxc.id_map = u 0 100000 65536" -
lxc-conf ="lxc.id_map = g 0 100000 65536"
```

2.启动容器时不带--privileged参数

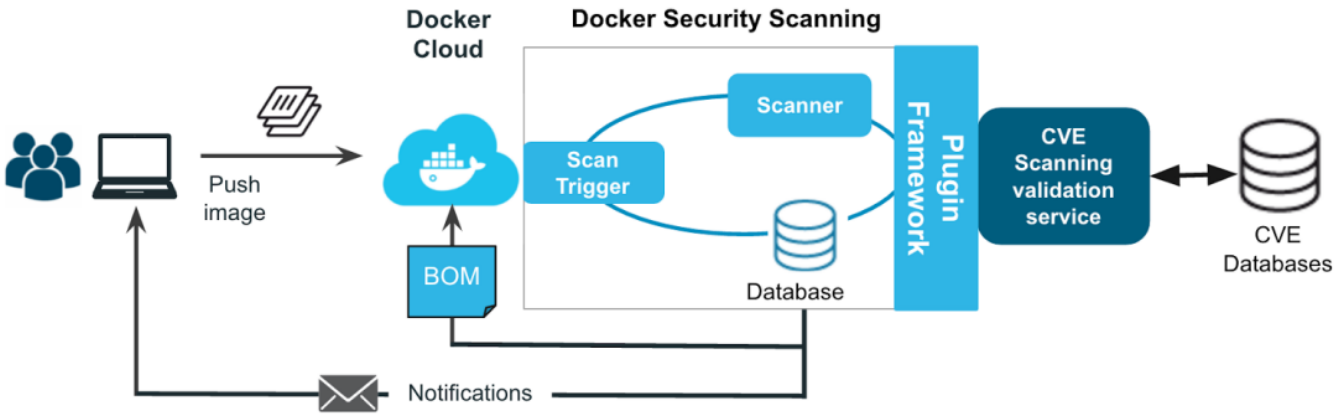
```
docker run -it debian8:standard /bin/bash
```

文件系统限制

挂载的容器根目录绝对只读，而且不同容器对应的文件目录权限分离，最好是每个容器在宿主上有自己单独分区。

```
su con1
docker run -v dev:/home/mc_server/con1 -it
debian8:standard /bin/bash
su con2
docker run -v dev:/home/mc_server/con2 -it
debian8:standard /bin/bash
```

镜像安全



镜像仓库客户端使用证书认证，打包的镜像需要合法签名。对下载的镜像进行检查，通过与CVE数据库同步扫描镜像，一旦发现漏洞则通知用户处理，或者直接阻止镜像继续构建。

镜像安全

Clair：
静态分析容器（目前包括appc和docker）中的漏洞，适合跟Registry结合在一起，在镜像构建过程进行漏洞扫描，支持os广泛，提供api，能提供构建阻断和报警

Docker Bench for Security：
根据30+的Docker安全最佳实践经验检测宿主镜像与容器安全状态

Dagda：适合对服务/镜像/容器（实时）检测，提供api，但只能跑在宿主上

Lynis：
dockerfile安全扫描以及docker宿主环境检测

Docker client端与 Docker Daemon的通信安全

为了防止链路劫持、会话劫持等问题导致docker通信时被中间人攻击，c/s两端应该通过加密方式通讯。

docker –tlsverify –tlscacert=ca.pem –tlscert=server-cert.pem –tlskey=server-key.pem -H=0.0.0.0:2376

资源限制

限制容器资源使用，最好支持动态扩容，这样既可以尽可能降低安全风险，也不影响业务。下面是使用样例，限制cpu使用第2核、分配2048：

docker run -tid –name ec2 –cpuset-cpus 3 –cpu-shares 2048 -memory 2048m –rm –blkio-weight 100 --pids--limit 512

安装安全加固

如果可能，使用安全的Linux内核、内核补丁。如SELinux，AppArmor，GRSEC等，都是Docker官方推荐安装的安全加固组件。
如果先前已经安装并配置过SELinux，那么可以在容器使用setenforce 1来启用它。Docker守护进程的SELinux功能默认是禁用的，需要使用--selinux-enabled来启用。容器的标签限制可使用新增的—-security-opt加载SELinux或者AppArmor的策略进行配置，该功能在Docker版本1.3引入。例如：

```
docker run --security-opt=secdriver:name:value -i -t centos bash
```

suid和guid限制

SUID和GUID程序在受攻击导致任意代码执行（如缓冲区溢出）时将非常危险。
docker run -it --rm --cap-drop SETUID --cap-drop SETGID
还有种做法，可以考虑在挂载文件系统时使用nosuid属性来移除掉SUID能力。
最后一种做法是，删除系统中不需要的SUID和GUID程序。这类程序可在Linux系统中运行以下命令而找到：
find / -perm -4000 -exec ls -l {} \; 2>/dev/null
find / -perm -2000 -exec ls -l {} \; 2>/dev/null
然后，可以使用类似于下面的命令将移除SUID和GUID文件权限：
sudo chmod u-s filename sudo chmod -R g-s directory

能力限制

尽可能降低Linux能力。
Docker默认的能力包括：chown、dac_override、fowner、kill、setgid、setuid、setpcap、net_bind_service、net_raw、sys_chroot、mknod、setfcap、和audit_write。
在命令行启动容器时，可以通过--cap-add=[]或--cap-drop=[]进行控制。例如：
docker run --cap-drop setuid --cap-drop setgid -ti <container_name> /bin/sh
此功能在Docker 1.2版本引入。

漏洞扫描

前面的镜像安全，跟这里的漏洞扫描关联很密切，可以使用相同的工具去实现安全扫描，不过漏洞扫描更倾向于外部检测，镜像安全则需要镜像仓库和CI系统联动

开源方案：openvas/w3af/arachni/巡风

05 Docker安全防御体系构建

Docker安全基础设施建设实践

- Docker安全标准化
- Docker安全扫描

Docker安全标准化

- 为Docker集群管理平台提供安全服务
- 宿主标准初始化
- Docker安全基线推广

Docker安全扫描

· Docker镜像安全扫描系统与镜像仓库联动

. 漏洞扫描系统自动化扫描（宿主、容器）

. Dockerfile安全扫描服务化

感谢聆听

—

THANKS!

SRC
唯品会安全应急响应中心