

■ 多数企业数字化进程处于第二阶段向第三阶段转型过程中

- 云原生成为企业重点投入方向：29.06%用户云原生支出占云建设总费用30%-50%，19.32%的用户云原生支出占云建设总费用50%以上
- 云原生的广泛应用：50.71%用户将容器技术应用于核心生产环境，27.5%的用户将容器技术应用与次核心生产环境
- 2021年云原生市场规模已经达到875亿元，预计2025年将接近6000亿，2022-2026期间复合增长率为62%



国内云原生与海外差异对比

应用场景差异

- 国内: 超过 60% 企业使用 混合云或私有云。 (Gartner)
- 海外: 70% 企业全面采用公有云。 (Gartner)

技术选型

- 国内: 传统企业数字化转型多聚焦混合云和多云管理的场景
- 海外: 对云原生技术接受较早且成熟, 针对复杂常见的微服务和工具化链已经比较完善

需求发展

- 国内: 从虚拟化向容器化过度; 企业政府对自主可控需求明显, 有大量的国产替代需求
- 海外: 许多企业已经完全云化, 在利用云原生提高研发和运维效率。

安全合规

- 国内: 强调数据主权, 多采用私有云和混合云, 非全面的公有云
- 海外: 强调数据隐私、安全合规, 在 GDPR 等严格法律下确保数据保护符合标准, 公有云+云原生应用广泛

开源生态

- 国内: 生态逐渐发展, 与海外还有明显差距, 贡献比例不到 5%, 主要集中在大厂 (如阿里云、腾讯云), 企业多基于开源项目定制开发。
- 海外: 大部分云原生项目贡献来自美国公司, CNCF 推动云原生技术标准, 在工具集成和生态构建方面发挥了重要作用。



用户分析

用户占比Top1

互联网与金融用户占比

传统行业用户占比

新兴行业用户占比

典型应用案例

美国用户使用云原生技术情况

软件/技术公司、金融、电商、电信、医疗

技术驱动型

广泛采用公有云基础设施和云原生技术应用于业务创新中

高度采用云原生技术, 广泛采用容器化技术, 广泛采用微服务架构, 广泛采用 DevOps 实践, 广泛采用云原生应用开发框架。



美国国家实验室与CERN等科研机构, 高度采用云原生技术, 广泛采用容器化技术, 广泛采用微服务架构, 广泛采用 DevOps 实践, 广泛采用云原生应用开发框架。

中国用户使用云原生技术情况

互联网和电信服务企业、金融、电商、医疗、电信

技术驱动型

部分企业开始采用, 但整体采用率较低, 广泛采用容器化技术, 广泛采用微服务架构, 广泛采用 DevOps 实践, 广泛采用云原生应用开发框架。

部分企业开始采用, 但整体采用率较低, 广泛采用容器化技术, 广泛采用微服务架构, 广泛采用 DevOps 实践, 广泛采用云原生应用开发框架。



部分企业开始采用, 但整体采用率较低, 广泛采用容器化技术, 广泛采用微服务架构, 广泛采用 DevOps 实践, 广泛采用云原生应用开发框架。

托管 Kubernetes 服务 PaaS

- 华为云 CCE
- 阿里云 ACK
- 腾讯云 TKE

部署工具本地部署

- 官方工具 kubeadm
- 轻量级k8s发行版 k3s/k0s
- 单节点集群 minikube

集群管理平台部署

- Rancher 开源的k8s管理平台
- KubeSphere 青云开源 k8s 管理平台-KubeKye
- 灵雀云

控制平面组件:

- kube-apiserver
- etcd
- kube-scheduler
- kube-controller-manager

工作节点组件:

- kubelet
- kube-proxy
- Container Runtime (如 Docker, containerd, CRI-O)

附加组件:

- CoreDNS
- Ingress Controller
- Dashboard
- ...

容器自身:

- 容器镜像存在的风险:
 - 不安全的第三方组件
 - 不安全的镜像
 - 敏感信息泄露
- 活动中的容器存在的风险:
 - 不安全的容器应用
 - 不收限制的资源共享
 - 不安全的配置与挂载
- 容器管理程序接口风险:
 - unix socket
 - tcp socket
- 其他风险:
 - 宿主机操作系统风险
 - ...

容器管理平台k8s:

- 组件接口存在的风险:
 - API Server
 - Kubelet
 - Dashboard
 - etcd
- 集群网络存在的风险:
 - 集群内部内网横向
 - 集群向外横向
- 访问控制机制存在的风险:
 - 利用混乱的访问控制机制进行提权操作
 - ...

运维管控

- 集群监控与日志管理
 - 使用 Prometheus 和 Grafana 进行资源使用监控和可视化。
 - 利用 Kubernetes Dashboard 或集群管理平台提供的可视化工具来监控集群状态。
 - 利用skywalking进行网络性能监控
- 自动化运维与配置管理
 - 搭建私有 Helm与Harbor 进行配置与镜像管理
 - 利用集群管理平台提供的可视化工具进行自动化管理

安全管控

- 身份验证和访问控制
 - 配置 RBAC (基于角色的访问控制), 限制用户和服务账户的权限。
 - 实施网络策略控制不同 Pod 之间的网络访问权限。
- 容器安全和镜像管理
 - 使用容器安全工具扫描镜像漏洞和配置错误
 - 配置镜像签名和镜像准入控制, 确保只有经过验证的镜像能够运行在集群中。
 - 容器运行时安全监测
- 数据加密
 - 启用 etcd 数据加密, 确保敏感数据在存储时被加密。
 - 使用Kubernetes Secrets管理敏感数据如密码、密钥等。

理想的安全管控

□ 容器基线扫描

- 结合容器最佳实践对容器基线进行检查
- 针对镜像遭受供应链攻击、病毒植入等风险进行全流程管控，从入口处降低风险镜像的引入建立完善的容器基线标准和管理机制

□ 容器准入

- 构建进程、网络、文件的白名单，实时确保准入机制有效性
- 替代传统入侵检测机制，有效监控容器异常状态

□ 容器威胁检测与响应

- 检测计算资源消耗和业务稳定性的影响
- eBPF监控容器的系统调用和文件操作，防止容器逃逸

□ 容器白名单

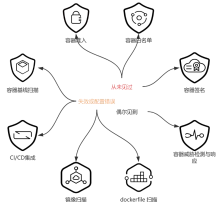
□ 容器签名

□ Dockerfile扫描

□ 镜像扫描

□ CI/CD集成

现实的安全管控



从攻击者视角，在集群外部可以看到什么？内网且不专门做访问控制的情况下，攻击者观测到的集群

观测到运行在集群上的业务

- 对外开放的业务
 - 业务访问页面和api接口【通过负载均衡、通过ingress开放、通过nodeport开放】
- 业务支撑组件：
 - 数据库：mysql, redis
 - 配置中心：nacos, apollo
- CI/CD组件：
 - 代码仓库：gitlab
 - 代码编译：jenkins
 - maven私服：Nexus
 - 镜像仓库：Harbor

观测到集群本身

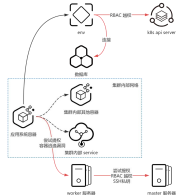
- API Server
 - Kubernetes API Server 默认端口6443在默认情况下可以从集群外部直接访问，可以用于确认master节点
 - kubelet 端口10250
- 监控组件
 - Prometheus
 - Grafana
 - Alertmanager
 - Node Exporter【DaemonSet】
- 集群节点本身开放的端口
 - 例如用于主机运维的ssh端口22

攻击过程中关注的重点/信息收集

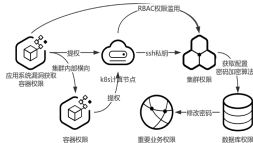
攻击进入集群上的业务—底层技术视角攻击

- 对外开放的业务【处于容器中的shell】
 - 从env获取大量配置信息
 - 业务配置 (nacos地址、数据库地址、各种密码)
 - 本质上是配置到该容器的configmap
- 获取当前容器的Service Account
 - 尝试RBAC权限滥用
- 获取当前容器本身的配置
 - 尝试通过配置错误进行提权
 - 尝试通过内核漏洞进行提权
- 可访问集群内部网络
 - 尝试横向攻击集群内部其他业务pod
 - 连接数据库获取更多信息

攻击进入集群内部后，可以观测到什么？
获取到什么？该如何做？



• 底层技术视角提权

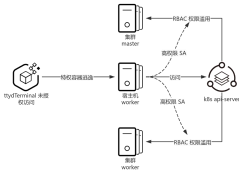


■ RBAC 权限滥用：
只要存在 access token 的位置均可能存在权限滥用

■ 常见于：
Pod中
Node中运行的容器中
Secrets中
镜像中【硬编码】

■ ssh 私钥：
为了方便运维与管理，整个集群通用一个私钥的情况非常常见

• RBAC权限滥用提权案例



■ RBAC 权限滥用：
只要存在 access token 的位置均可能存在权限滥用

■ 常见于：
Pod中
Node中运行的容器中
Secrets中
镜像中【硬编码】

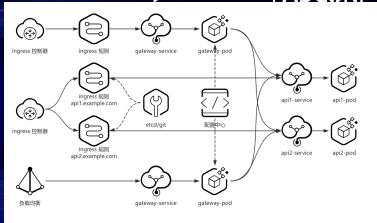
■ ssh 私钥：
为了方便运维与管理，整个集群通用一个私钥的情况非常常见

攻击过程中关注的重点/业务权限

如何真正获取业务权限？获取后台密码加密算法，数据库替换密码

跟着流量走--理清业务逻辑

- k8s的三种服务暴露方式
 - NodePort
 - LoadBalance
 - Ingress
- 与微服务结合时常见的路由方案
 - Ingress + Spring Cloud Gateway
 - Ingress + 直通 Service 路由
 - 外部负载均衡 + Gateway 作为直接入口
- 本质在于service和service之间的调用
 - 需要理清service之间的调用

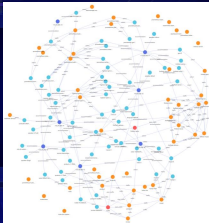


攻击过程中关注的重点/业务权限

如何真正获取业务权限?
获取后台密码加密算法,
数据库替换密码

service之间的调用--理清业务逻辑

- 自动化的部分
 - 获取配置中心的配置文件 (Nacos/Spring Cloud Config...)
 - 获取k8s本身的配置 (Etcd)
 - 进行绘图
 - 过滤service
 - 自动标注入口开始的调用链路(指定若干级)
 - 获取可能存在加密算法的service
- 人工的部分
 - 从对应pod中下载代码
 - 反编译并从代码中找到密码加密算法



或许你没有集群权限？
加上一点点运气！

攻击进入集群上的业务--业务视角攻击

- 业务支撑组件【成功登录】：
 - 数据库：mysql, redis
 - 获取大量数据，尝试理解业务内容并寻找业务后台管理员表
 - 配置中心：nacos, apollo
 - 获取配置信息，尝试从配置名称推测业务架构
 - 尝试利用配置信息横向
- CI/CD组件【成功登录】：
 - 代码仓库：gitlab
 - 获取代码进行审计，修改数据库中管理员的密码接管业务
 - 代码编译：jenkins
 - 获取git账号密码横向，获取编译服务器密码横向，获取推送目标服务器密码横向，获取jenkins自身shell横向
 - 镜像仓库：Harbor
 - 镜像分析获取敏感信息

如果你对业务足够熟悉--就像开发者那般熟悉

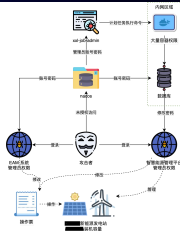
- 盲猜加解密代码所在的微服务/chat-gpt辅助
 - 根据经验盲猜
 - prod/dev-是否生产环境
 - login、uc、manager等关键词
- 盲猜密码算法/chat-gpt辅助
 - 照着数据库直接猜密码加密算法
- 根据常见业务逻辑对数据库进行修改
 - 在数据库中将已知帐户的uid修改为管理员uid
 - 在数据库中将已知密码的帐户的密文替换给管理员
 - 在数据库中将已知帐户的权限修改为管理员权限
 - 在数据库中寻找默认口令替换给管理员
 - 或许是前端加密？

案例

从nacos 未授权访问开始的冒险--业务视角攻击

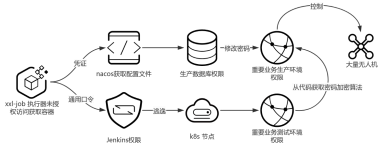
- nacos:
互联网暴露nacos, 严重错误
- xxl-job:
在nacos中看到了xxl-job相关配置, 猜测xxl-job管理端可能存在于互联网上
- 进入内网:
xxl-job管理端控制内网执行器反弹shell
- 登录业务系统:
利用已爆出的弱口令的密码密文替换管理员的密码密文

文



CONF

Jenkins到权限陷落— 业务视角攻击

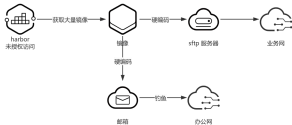


■ jenkins:
CI/CD 的核心组件, 负责代码
的编译与推送

■ xxl-job执行器:
xxl-job 分为执行器与管理器,
执行器在微服务架构中一般是一个容器

■ 这里 nacos 中同时存储了
生产环境与测试环境的数据
库账号密码

Docker镜像仓库未授权访问导致的内网突破



- harbor未授权访问:
运维人员为了图方便，将大量镜像设置为公开导致可以直接下载获取
- 硬编码:
开发人员在代码中硬编码了大量敏感信息

云原生环境中的驻留如何实现？一种优雅隐蔽的持久化手段

■ 常见的驻留手段：

- 直接在pod里做驻留
- 部署Cronjob 【恶意定时任务】
- 部署ShadowApiServier 【开启匿名访问的ApiServier】
- 部署Deployment 【恶意配置&恶意容器】
- 部署Deamonset 【恶意配置&恶意容器】

■ 动态容器注入：

● 探针类型

- 启动探针 (Startup Probe)
- 活动探针 (Liveness Probe)
- 就绪探针 (Readiness Probe)

● 探针参数

- initialDelaySeconds定义容器启动后要等待多长时间
- periodSeconds定义执行探针检查的频率（秒）
- timeoutSeconds探针等待响应的超时时间
- failureThreshold指定在认为探针检查失败之前，允许探针检查失败的次数
int32 最大2147483647

● 探针检查模式：

- GET请求
- TCP Socket连接
- Exec执行命令

ADCONF

主场·见真章

THANKS