安全狗
忠诚守护 值得信赖

云原生安全威胁分析报告

厦门服云信息科技有限公司

2022年7月

序

2006 年，亚马逊 AWS 把存储与计算能力通过 S3 和 EC2 以互联网的方式输送给广大用户，开启了云计算时代。云服务使得算力与数据资源像工业时代的电力资源一样，通过电网按需输送给千家万户。然而早期的云计算技术架构以虚拟机为主，资源损耗极大，就像由一大堆小型发电机堆砌的发电厂。云原生则是新一代云计算技术架构，采用了基于云计算特点的新理念与方法，包括容器、微服务、服务网格、DevOps 等技术，好比运转现代化大型发电机的发电厂，将云计算资源充分发挥出了优势。2013 年发布的 Docker 项目，2014 年发布的 Kubernetes 项目，2015 年成立的 CNCF 云原生基金会对云原生发展起到了重要作用。

然而，新兴威胁总是伴随着新兴技术而来，传统的 IT 与云安全解决方案对应新的云原生威胁捉襟见肘。CSA 在云计算顶级威胁白皮书指出了云原生技术的重大新威胁，在云安全指南 4.0 中提出过对应措施，在 CCM 云安全标准中制定出相应的安全要求，在 CNST 云原生安全标准与认证中设置了设计与测评基线，并成立全球容器与微服务工作组、无服务工作组、云密钥管理工作组和大中华区云原生安全工作组持续进行云原生技术安全研究与标准制定。

云安全联盟大中华区理事单位安全狗发布的《云原生安全威胁分析报告》是业内对云原生安全威胁全面的分析，对广大云厂商和上云企业提供了极具价值的实践洞见 , 不仅覆盖容器化基础设施、容器编排平台、云原生应用以及无服务等方面云原生安全威胁，还提出了云原生漏洞风险检测模型“CCICA”，基于 ATT&CK 容器矩阵的产品检测能力模型方法论“AKDA”，各类云原生威胁检测技术，这项报告对于云原生生态的安全发展具有重要的意义。

从发展趋势看，更多企业将会广泛应用云原生技术，云原生应该基于开源、开放的技术标准，云原生安全应该具有包括安全检测的内生安全能力，这样云原生将为企业带来快速业务创新的更大价值，相信数字领域的所有安全工作者都会从阅读本白皮书中受益。

——国际云安全联盟大中华区

目录

一、前言

随着云计算技术的蓬勃发展，传统上云实践中的应用升级缓慢、架构臃肿、无法快速迭代等“痛点”日益明显。能够有效解决这些“痛点”的云原生技术正蓬勃发展，成为赋能业务创新的重要推动力，并已经应用到企业核心业务。然而，云原生技术在创造效益的同时，却也面临着严峻的安全问题。

本报告基于安全狗“云原生安全团队”近几年的研究成果，以及近年来对云原生安全领域的观察编制而成。我们期望通过我们对以“云原生威胁检测”为主要视角的总结分析，能够为各个行业 / 企业 / 单位对云原生安全技术开展后续的云原生安全建设提供进一步的参考建议。

本报告的主要内容如下：

第一部分，从容器化基础设施、容器编排平台、云原生应用以及无服务等方面分析云原生安全威胁；

第二部分，从云原生安全事件风险、云原生安全漏洞风险等方面介绍近年来一些重要的云原生安全威胁；

第三部分，从容器镜像、容器运行时、容器网络、云原生可观测性、宿主机威胁检测以及容器 ATT&CK 矩阵等角度介绍云原生威胁检测技术；

第四部分，介绍基于我司实践经验提出的五层云原生漏洞风险检测模型“CCICA”；

第五部分，介绍基于我司实践经验提出的一种基于 ATT&CK 容器矩阵的可应用于提取检测规则，提升产品检测能力的模型方法论“AKDA”；

第六部分，介绍我司基于 CCICA、AKDA 的理论基础的云原生威胁检测实战场景、案例；

第七部分，从云原生未来发展趋势、云原生安全未来发展趋势以及云原生攻防对抗未来发展趋势三个角度进行云原生安全实践与技术展望。

二、云原生安全威胁分析

云原生技术逐渐成为云计算市场发展的新趋势之一。随着越来越多的云原生应用落地和使用，其相关的安全风险与威胁也不断涌现。云原生安全事件频频发生也直接影响了整个云原生系统的安全性。以下章节将围绕云原生环境中存在的威胁展开分析。

（一）容器化基础设施的威胁风险

在实现云原生的主要技术中，容器作为支撑应用运行的重要载体为应用的运行提供了隔离和封装，因此成为云原生应用的基础设施底座。云原生架构的安全风险包含容器化基础设施自身的安全风险，容器化部署则成为云原生计算环境风险的输入源。

1．容器全生命周期的威胁风险

容器的秒级启动或消失的特性以及持续频繁的动态变化，极大程度地缩短了生命周期，但也增加了安全保护的难度和挑战。容器的安全问题，在容器全生命周期（创建、分发、运行、停止）的各个阶段都存在相应的风险和威胁隐患，如图 2-1 所示。

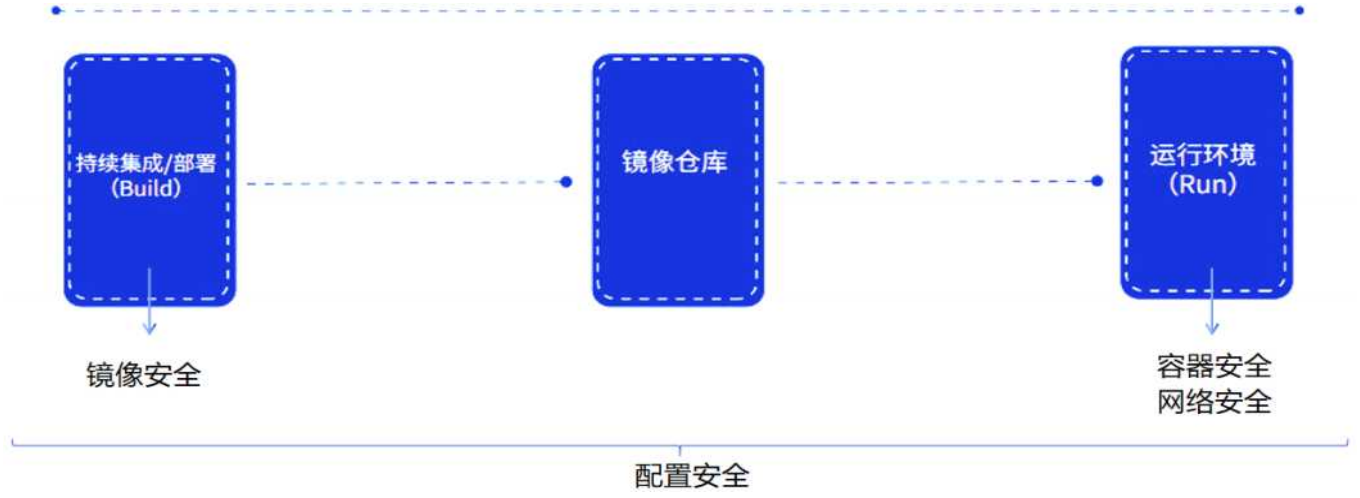


图 2-1 容器全生命周期的威胁风险

（1）容器镜像构建阶段存在的风险

随着容器技术的普及，容器镜像也成为了软件供应链中重要的一环。因此，当业务依赖的基础镜像存在安全漏洞或者包含恶意代码时，其潜在危害比黑客从外部发起攻击要严重得多。

① 镜像 CVE 漏洞利用

“镜像漏洞利用”指的是镜像本身存在漏洞时，依据镜像创建并运行的容器通常也会存在相同漏洞，镜像中存在的漏洞会被攻击者用以攻击容器的情况。这种行为往往会对容器造成严重影响。备受开发者青睐的 Alpine 镜像曾曝出过一个漏洞，编号为 CVE-2019-5021。在 3.3 到 3.9 版本的 Alpine 镜像中，root 用户密码被设置为空，攻击者在攻入容器后获得容器内部 root 权限的机会增大。

② 镜像投毒

镜像投毒是一个宽泛的话题。指的是攻击者通过某些方式，如上传镜像到公开仓库、入侵系统后上传镜像到受害者本地仓库以及修改镜像名称假冒正常镜像等，欺骗、诱导受害者使用攻击者指定的恶意镜像创建并运行容器，从而实现入侵或利用受害者的主机进行恶意活动的行为。
根据目的不同，常见的镜像投毒有三种类型：投放恶意挖矿镜像、投放恶意后门镜像和投放恶意 exploit 镜像。

投递恶意挖矿镜像。这种投毒行为主要是通过欺骗受害者在机器上部署容器的方式获得经济收益。2018 年 6 月，一份研究报告指出，一个名为 docker123321 的账号向 Docker Hub 上陆续上传了 17 个包含挖矿代码的恶意镜像。截至 Docker Hub 官方移除这些镜像时，它们已经累计被下载超过 500 万次。据统计，黑客借助这一行为获得了市值约 9 万美元的门罗币。

投放恶意后门镜像。这种投毒行为的主要目的是控制容器。通常，受害者在机器上部署容器后，攻击者会收到容器反弹过来的 Shell。在 2017 年 9 月，有用户在 Docker Hub 的反馈页面中反馈前述 docker123321 账号上传的 Tomcat 镜像包含了后门程序。结合该账号上传的其他恶意镜像的挖矿行为可推测出攻击者在连接上述后门后会部署挖矿程序以此获得经济收益的可能性。

投放恶意 exploit 镜像。这种投毒行为是为了在受害者部署容器后尝试寻找宿主机上的各种漏洞来实现容器逃逸，以实现对受害者机器更强控制。从攻防对抗的角度来看，恶意 exploit 镜像只不过是一种攻击载荷投递方式，其特点在于具有隐蔽性和可能产生巨大的影响范围。试想，如果 Docker Hub 上某一热门镜像包含了某个 1day 甚至 0day 漏洞的利用程序，理论上，攻击者将一下子获取上百万台计算机的控制权限。

（2）容器镜像分发阶段存在的风险

镜像的安全需要重点建设。镜像的安全性会直接影响容器安全，因为容器镜像在存储和使用的过程中，可能会被植入恶意程序或修改内容以此篡改。一旦使用被恶意篡改的镜像创建容器后，将会影响容器和应用程序的安全。

（3）容器运行时存在的风险

在容器运行时存在的安全问题中，“容器逃逸”是最具代表性的高风险安全问题。攻击者可通过利用漏洞“逃逸”出自身拥有的权限，实现对宿主机或者宿主机上其他容器的访问，同时带来进行时资源耗尽的风险。

① 容器逃逸

与其他虚拟化技术类似，逃逸是最为严重的安全风险。无论是 Docker 容器、还是 Kata 类安全容器，都暴露过各类逃逸漏洞。逃逸风险对于容器化的云原生场景是一个不可避免的风险面，因为其直接危害了底层宿主机和整个云计算系统的安全。根据层次的不同，可以进一步展开为：危险配置导致的容器逃逸、危险挂载导致的容器逃逸、相关程序漏洞导致的容器逃逸、内核漏洞导致的容器逃逸、安全容器逃逸风险。

危险配置导致的容器逃逸。用户可以通过修改容器环境配置或在运行容器时指定参数来缩小或扩大约束。如果用户为不完全受控的容器提供了某些危险的配置参数，就为攻击者提供了一定程度的逃逸可能性，主要包括未授权访问带来的容器逃逸风险，特权模式运行带来的容器逃逸风险。

危险挂载导致的容器逃逸。将宿主机上的敏感文件或目录挂载到容器内部，尤其是那些不完全受控的容器内部，往往会带来安全问题。在某些特定场景下，为了实现特定功能或方便操作，人们会选择将外部敏感卷挂载入容器。随着应用的逐渐深化，挂载操作变得愈加广泛，由此而来的安全问题也呈现上升趋势。例如：挂载 Docker Socket 引入的容器逃逸风险、挂载宿主机 procfs 引入的容器逃逸风险等。

相关程序漏洞导致的容器逃逸。所谓相关程序漏洞，指的是那些参与到容器生态中的服务端、客户端程序自身存在的漏洞。CVE-2019-5736 正是这样一个存在于 runC 的容器逃逸漏洞，它由波兰 CTF 战队 Dragon Sector 在 35C3 CTF 赛后基于赛中一道沙盒逃逸题目获得的启发，对 runC 进行漏洞挖掘，成功发现能够覆盖宿主机 runC 程序的容器逃逸漏洞。

内核漏洞导致的容器逃逸。Linux 内核漏洞的危害之大、影响范围之广，使得它在各种攻防话题下都占有一席之地。近年来，Linux 系统曝出过无数内核漏洞，其中能够用来提权的也不少，脏牛（CVE-2016-5195）大概是其中最有名气的漏洞之一。事实上，脏牛漏洞也能用来进行容器逃逸，一种利用方法的核心思路是向 vDSO 内写入 shellcode 并劫持正常函数的调用过程。在成功触发漏洞后，攻击者可以获得宿主机上反弹过来的 Shell，实现容器逃逸。

安全容器逃逸风险。无论是理论上，还是实践中，安全容器都具有非常高的安全性。然而，在 2020 年 Black Hat 北美会议上，Yuval Avrahami 分享了利用多个漏洞成功从 Kata containers 逃逸的议题，安全容器首次被发现存在逃逸可能性，他使用发现的三个漏洞（CVE-2020-2023、CVE-2020-2025 和 CVE-2020-2026）组成漏洞利用链先后进行容器逃逸和虚拟机逃逸，成功从容器内部逃逸到宿主机上。

② 资源耗尽

容器在运行时默认情况下并未对容器内进程在资源使用上做任何限制，以 Pod 为基本单位的容器编排管理系统也是类似的。Kubernetes 在默认情况下同样未对用户创建的 Pod 做任何 CPU、内存用限制。限制的缺失使得云原生环境面临资源耗尽型攻击风险。

（4）容器网络存在的风险

在网络安全方面，由于 K8S 默认不做网络隔离，传统物理网络的攻击手段依然有效（如 ARP 欺骗、DNS 劫持、广播风暴等）。同时，攻击者可利用失陷的容器进行针对 K8s 集群内部、内网主机的横向渗透；
容器网络环境也面临着“访问关系梳理难，控制难”的痛点。比如：东西向访问流量庞大，无法感知业务间访问关系；业务访问关系错综复杂，无法制定精细化的访问控制策略；静态策略无法跟随虚拟机自动迁移；容器平台体量巨大，上下线周期短，容器 IP 地址变化频繁，难以适配容器环境等等。

2．宿主机的威胁风险

容器与宿主机共享操作系统内核，因此宿主机的配置对容器运行的安全有着重要的影响，比如宿主机中安装有漏洞的软件可能会导致任意代码执行风险，端口无限制开放可能会导致任意用户访问风险，防火墙未正确配置会降低主机的安全性，没有按照密钥的认证方式登录可能会导致暴力破解并登录宿主机等。安全是一个整体，任何一个环节出问题，都会影响到其他环节。应用安全可影响到容器安全，容器逃逸可影响到虚拟主机及容器集群的安全，虚拟主机逃逸会影响到虚拟化基础设施的安全。

3．容器化开发测试过程的威胁风险

DevOps 所倡导的理念是追求敏捷、协作与快速迭代。开发人员往往为了追求便捷的开发环境与快速的迭代节奏选择将安全系统配置（如权限管理、访问控制等）的优先级降低，为效率与敏捷让步，最终可能使敏感数据权限管理不当而在容易发生泄露、系统资源无防护的内部开放。若研发环境采用传统安全防护措施，一旦边界防御措施被突破，内部数据与资源将会对外部攻击者完全开放，产生不可估量的损失与影响。

（二）容器编排平台的风险分析

在云原生环境中，编排系统无疑处于重中之重的地位。任何编排系统都会受到一些攻击，这些攻击会影响部署的整体安全性和运行时的持续安全性。而 Kubernetes 早已成为容器编排系统的“事实标准”，下面对容器编排平台的风险进行分析。

1．Kubernetes 组件不安全配置

Kubernetes 编排工具组件众多、各组件配置复杂，配置复杂度的提升增加了不安全配置的概率。不安全配置引起的风险不容小觑，比如 Kubernetes API Server 的未授权访问，Kubernetes Dashboard 的未授权访问，Kubelet 的未授权访问等，都可能会导致编排工具中帐户管理薄弱，或部分帐户在编排工具中享有很高特权，入侵这些账户可能会导致整个系统遭到入侵。

2．Kubernetes 提权

Kubernetes 非法提权，是指普通用户获得管理员权限或 Web 用户直接提权成管理员用户。编排工具可能存在多种漏洞导致此类攻击，Kubernetes 权限提升。以 CVE-2018-1002105 漏洞为例，这是一个 Kubernetes 的权限提升漏洞，允许攻击者在拥有集群内低权限的情况下提升至 Kubernetes API Server 权限；通过构造一个对 Kubernetes API Server 的特殊请求，攻击者能够借助其作为代理，建立一个到后端服务器的连接，攻击者就能以 Kubernetes API Server 的身份向后端服务器发送任意请求，实现权限提升。

3．Kubernetes 拒绝服务

传统虚拟化技术设定明确的 CPU、内存和磁盘资源使用阈值，而容器在默认状态下并未对容器内进程的资源使用阈值做限制，以 Pod 为单位的容器编排管理工具也是如此。资源使用限制的缺失使得云原生环境面临资源耗尽的攻击风险，攻击者可以通过在容器内运行恶意程序，或对容器服务发起拒绝服务攻击占用大量宿主机资源，从而影响宿主机和宿主机上其他容器的正常运行。

4．Kubernetes 中间人攻击

默认情况下，Kubernetes 集群中所有 pod 组成了一个小型的局域网络，那么就可能发生像中间人攻击这样针对局域网的攻击行为。假如攻击者借助 Web 渗透等方式攻破了某个 Pod，就有可能针对集群内的其他 Pod 发起中间人攻击，进而进行 DNS 劫持等。攻击者能够潜伏在集群中，不断对其他 Pod 的网络流量进行窃听，甚至可以悄无声息地劫持、篡改集群其他 Pod 的网络通信，危害极大。

（三）云原生应用的威胁风险

云原生化应用主要包括微服务、Serverless；同时云原生基础设施和云原生应用也会在原有云计算场景下显著扩大 API 的应用规模，新的安全风险也不容忽视。

1．微服务的威胁风险

首先，随着微服务的增多，暴露的端口数量也急剧增加，进而扩大了攻击面，且微服务间的网络流量多为东西向流量，网络安全防护维度发生了改变；其次，不同于单体应用只需解决用户或外部服务与应用的认证授权，微服务间的相互认证授权机制更为复杂，人为因素导致认证授权配置错误成为了一大未知风险。并且微服务通信依赖于 API，随着业

务规模的增大，微服务 API 数量激增，恶意的 API 操作可能会引发数据泄漏、越权访问、中间人攻击、注入攻击、拒绝服务等风险；最后，微服务治理框架采用了大量开源组件，会引入框架自身的漏洞以及开源治理的风险。

微服务入口点增加导致攻击面增大。单体应用的场景下，入口点只有一个，所有的请求都会从这个入口点进来。在这个入口点建立一组 Filter 或者 Interceptor，就可以控制所有的风险。微服务场景下，业务逻辑并不在一个单一的进程里，而是分散在很多进程里。每一个进程都有自己的入口点，导致需要防范的攻击面比原来更大，风险也会更高。

微服务调度复杂增加访问控制难度带来越权风险。在单体应用架构下，应用作为一个整体对用户进行授权；而在微服务场景下，所有服务均需对各自的访问进行授权，明确当前用户的访问控制权限。传统的单体应用访问来源相对单一，基本为浏览器，而微服务的访问来源还包括内部的其它微服务。因此，微服务授权除了服务对用户的授权，还增加了服务对服务的授权。默认情况下，容器网络间以白名单模式出现，如果不对 Pod 间访问进行显式授权，一旦某一 Pod 失陷，将极速扩展至整个集群的 Pod。

微服务治理框架漏洞引入应用风险。常用的微服务治理框架，例如 Spring Cloud、Dubbo 等都是基于社区的模式运作，虽然内置了许多安全机制，但默认值通常并不安全，常常引入不可预料的漏洞，例如用户鉴权混乱、请求来源校验不到位等。这些漏洞将会导致微服务业务层面的攻击变得更加容易，为微服务的开发和使用者带来安全隐患。比如 Spring Cloud config 存在 CVE-2019-3799、CVE-2020-5405 漏洞，有的社区对这些漏洞进行了及时修复，需要软件及时更新到新版本；但也有的漏洞长期缺乏修复，需要微服务开发厂商自行修复，这类漏洞通常修复比较复杂、修复成本较高。

2．API 的威胁风险

云原生带来 API 爆发式增长增加滥用风险。云原生化之后，从基础架构层、到上面的微服务业务层都会有很多标准或非标准的 API。因为 API 既充当外部与应用的访问入口，也充当应用内部服务间的访问入口，所以数量急剧增加、调用异常频繁。爆发式的增长导致 API 在身份认证、访问控制、通信加密、以及攻击防御等方面的问题更加明显，面临更多潜在的风险。与此同时，企业面对大量的 API 设计需求，其相应的 API 安全方案往往不够成熟，从而引起滥用的风险，API 的主要安全风险如表 2-1 所示。

风险类型	风险引入途径
未授权访问	API 管理不当
	API 设计存在缺陷
数据泄露	安全措施不足
DDoS 风险	资源和速率没有限制

表 2-1 API 主要安全风险

（四）无服务的威胁风险

Serverless（无服务器运算）又被称为函数即服务（Function-as-a-Service，缩写为 FaaS），是云计算的一种模型。以平台即服务（PaaS）为基础，无服务器运算提供一个微型的架构，终端客户不需要部署、配置或管理服务器服务，

代码运行所需要的服务器服务皆由云端平台来提供 ,Serverless 使得底层运维工作量进一步降低，业务上线后，也无需担忧服务器运维，而是全部交给了云平台或云厂商。。对应 Serverless 的安全风险，一方面包含应用本身固有的安全风险，另一方面包含 Serverless 模型以及平台的安全风险。

应用程序固有的安全风险，总体上类似传统应用程序的安全风险内容，包括：应用程序漏洞带来的安全风险、第三方依赖库漏洞引入的安全风险、权限控制缺陷带来的安全风险等。

erverless 模型以及平台的安全风险，主要包括以下几类：

（1）函数多源输入导致供应链安全风险资源权限管控不当带来函数使用风险

（2）设计使用不规范引入数据泄露风险

（3）FaaS 平台自身漏洞带来入侵风险

（4）平台设计机制缺陷引入账户拒绝服务风险

（5）缺乏 Serverless 专有安全解决方案导致风险应对能力不足

（五）服务网格的威胁风险

服务网格作为一种云原生应用的体系结构模式，应对了微服务架构在网络和管理上的挑战，也推动了技术堆栈分层架构的发展。从分布式负载均衡、防火墙的服务的可见性，服务网格通过在各个架构层提供通信层来避免服务碎片化，以安全隔离的方式解决了跨集群的工作负载问题，并超越了 Kubernetes 容器集群，拓展到运行在裸机上的服务。服务网格与微服务在云原生技术栈中是相辅相成的两部分，前者更关注应用的交付和运行时，后者更关注应用的设计与开发。若未在服务网格中采用双向 TLS 认证，服务间容易受到中间人攻击。若未做东西向、南北向的认证鉴权，服务间容易受到越权攻击。

三、云原生安全近年威胁

（一）近年云原生安全事件风险

1．近年来云原生安全问题分布

根据 StackRox 在 2021 年发布的《容器和 Kubernetes 安全态势报告》显示，针对云原生应用的威胁数量呈现上升趋势。在过去的 12 个月里，有 94%的组织在其容器环境中遇到过安全问题，其中 69%的组织检测到错误配置，27%的组织在运行时遇到安全事件，还有 24%的组织发现了严重的安全漏洞，如图 3-1 所示。

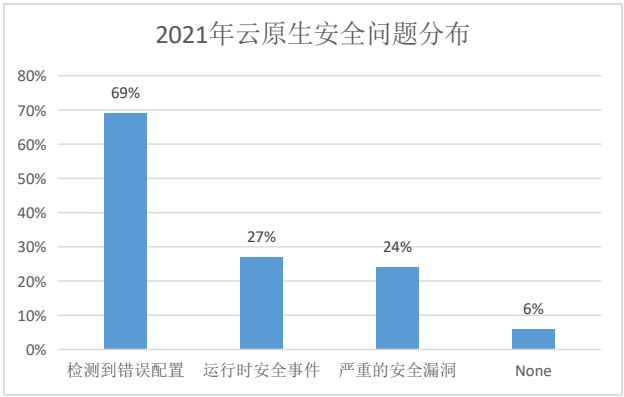


图 3-1 2021 年云原生安全问题分布

在看到惊人的数字之余，进一步考虑到安全事件带来的影响，以及可能会给组织带来灾难性的后果。在云原生环境，一旦没有实施全面有效的安全防护，黑客可利用漏洞攻击容器上的服务，或可进一步利用容器漏洞，直接访问容器云上的敏感信息，获取服务器特权，对容器云进行修改并最终完全控制服务器，造成恶劣影响和重大损失。所以必须综合考虑云原生环境下的 Docker、Kubernetes 等组件的整体安全性，分析研究它们的漏洞特征，进而才能构建一条完整防线。

2．典型的云原生安全事件

（1）攻击团伙 TeamTNT 入侵 KUBERNETES 集群植入挖矿木马

TeamTNT 是一个主要入侵在线容器并通过挖矿和 DDoS 进行牟利的攻击团伙。2021 年年初，该团伙被发现入侵了某 Kubernetes 集群，通过结合脚本和现有工具，最终在容器内植入挖矿木马。研究人员已经发现并确认将近 50000 个 IP 在 TeamTNT 对多个集群实施攻击中被攻击到。在 3 月至 5 月发生的事件期间，TeamTNT 反复使用了多个 IP。大多数被攻击的节点来自中国和美国。

（2）第一个已知的针对 Windows 容器的恶意软件

2021 年 3 月，研究人员发现了第一个已知的针对 Windows 容器的恶意软件，恶意软件命名为 Siloscape。其主要目标是逃离容器。通过 Windows 容器针对 Kubernetes 集群的严重混淆恶意软件。其主要目的是为配置不当的 Kubernetes 集群打开一个后门，以运行恶意容器。该恶意软件可以利用 Kubernetes 集群中的计算资源进行加密劫持，并可能从受感染集群中运行的数百个应用程序中窃取敏感数据。

（3）下载 2000 万次的 Docker Hub 镜像来自加密矿工

研究人员发现有基于云的加密货币挖矿攻击活动，其中矿工通过 Docker Hub 中的镜像进行部署。研究人员分析发现 Docker Hub 中存在用于加密货币挖矿攻击活动的恶意镜像文件，并发现了来自 10 个不同 Docker Hub 账户的 30 个镜像文件。多个容器映像文件被挖矿软件劫持了至少两年之久，且已被下载了超过 2000 万次。其中主要进行门罗币挖矿，保守估计加密货币挖矿活动获利 20 万美元。

（4）安全公司 Rapid7 遭到 Codecov 供应链攻击，源码泄露

网络安全公司 Rapid7 透露，2021 年 4 月 1 日，程序审计平台 Codecov 遭黑客攻击，该事件可能影响其 2.9 万名客户，并且引发大量公司连锁数据泄露，造成又一起“供应链”重大安全危机。Codecov 表示从 1 月 31 日开始，黑客就已经对 Codecov 发起了攻击，利用 Codecov 的 Docker 镜像创建过程中出现的错误，非法获得了其 Bash Uploader 脚本的访问权限并且进行了修改。这意味着攻击者很有可能导出存储在 Codecov 用户的持续集成（CI）环境中的信息，最后将信息发送到 Codecov 基础架构之外的第三方服务器。

（二）近年云原生安全漏洞风险

1．近年云原生安全漏洞统计

截至 2021 年 12 月末，我们通过对 CVND 漏洞数据统计分析了云原生基础设施的漏洞分布情况。首先，其中 Docker 相关的漏洞共 174 个，2021 年 38 个。与过去几年相比，在漏洞数量上并没有明显减少的趋势，一旦黑客突破 Docher，如容器逃逸，将会对主机造成巨大威胁；其次，Kubernetes 的安全是整个项目安全生产运行的重要保障。但是与 Kubernetes 相关的漏洞数量达到 106 个。2021 年则有 10 个，主要漏洞类型包括：拒绝服务、非法提权、容器逃逸等。云原生基础设施相关的漏洞数量变化趋势统计图如图 3-2 所示。

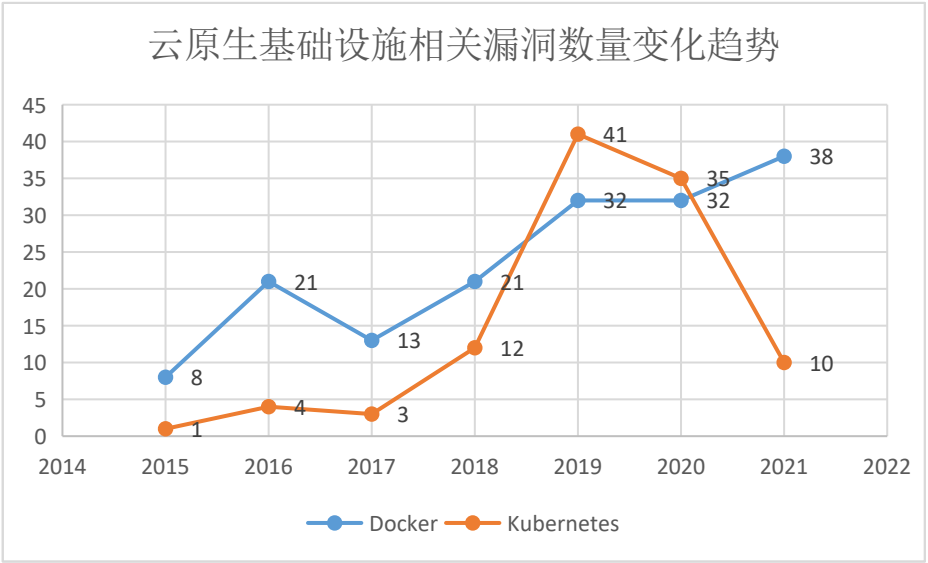


图 3-2 云原生基础设施相关漏洞数量变化趋势统计图

近年来，随着容器云广泛应用，容器云安全漏洞频繁被发现，漏洞影响越来越严重。如果没有采取及时、有效的防范措施，一旦攻击者利用这些漏洞，可轻易对云原生基础设施造成致命攻击。

2．典型的云原生安全漏洞

针对近年来出现的众多云原生安全漏洞，一些值得关注的漏洞如下。

（1）CVE-2021-30465

Open Containers 社区披露了 runC 相关的漏洞，攻击者通过创建恶意 Pod，利用符号链接以及条件竞争漏洞将宿主机目录挂载至容器中，最终可能导致容器逃逸问题。

当 Pod 中启动多个容器时，攻击者可以利用条件竞争，构造一个恶意 Pod 并挂载包含了软链接的目标路径，在特定条件下可以逃逸并获取容器 rootfs 之外的主机文件系统访问权限。

（2）CVE-2021-41103

Containerd 社区公布了安全漏洞 CVE-2021-41103，该漏洞源于 Containerd 中的缺陷。当容器根目录和一些系统插件中缺少必要的权限约束时，可使一个非特权的主机 Linux 用户拥有遍历容器文件系统并执行目标程序的权限。
在多租户场景下，如果集群节点中的容器包含扩展权限（例如 setuid），非特权的 Linux 用户可能发现并执行该程序。如果非特权的主机 Linux 用户 UID 碰撞到了容器中执行程序的所属用户或组，这个非特权的 Linux 用户可以读写该文件从而导致越权访问。

（3）CVE-2021-25735

Kubernetes 官方披露了 kube-apiserver 组件的安全漏洞，攻击者可以在某些场景下绕过 Validating Admission Webhook 的准入机制更新节点。
如果集群中存在使用 Validating Admission 机制进行节点更新准入校验的 Webhook，并且该 Webook 的准入依赖节点原状态的某些字段，则攻击者可以通过某些手段绕过准入校验完成对节点实例模型的修改。

（4）CVE-2021-25741

Kubernetes 社区公布了安全漏洞 CVE-2021-25741，该漏洞可使攻击者使用软链接的方式在容器中挂载指定 subPath 配置的目录逃逸到主机敏感目录。
在多租户场景下，拥有以 Root 用户启动容器权限的恶意攻击者可以利用该漏洞逃逸至主机文件系统，获取主机敏感目录的读写权限。

四、云原生威胁检测技术

（一）容器镜像安全检测技术

1．容器镜像安全现状

据 Sysdig 的《2022 云原生安全和使用报告》的数据显示：在运行于生产环境的镜像中，至少包含一个可修复漏洞的镜像占比高达 85%。此外，包含严重程度为“高”或“严重”的可修复漏洞占比高达 75%，如图 4-1 所示。

4-1 运行时环境的容器可修复漏洞分布情况

除此之外，攻击者还可以投放恶意镜像到 Docker Hub 中，如果这些恶意镜像被下载以后，就等同于引狼入室。总体而言，容器镜像安全现状不容乐观。

2．容器镜像安全检测

容器是从本地的镜像仓库中拉取来的，因此镜像的安全性直接关系到容器安全。并且，在实际环境中，本地构建的镜像还会引入第三方库，所以对引入的第三方库和本地构建的镜像进行安全检测就尤为重要。这里的镜像安全检测主要分为三个层级，第一级是静态检测，基于已知 CVE 漏洞进行安全扫描和检测；第二级是供应链检测，通过供应链的威胁情报作为数据支持，并通过自动化测试技术方法来检测镜像中的漏洞；第三级是动态检测，结合已知 CVE 和威胁情报，并在此基础之上，构建容器沙箱，进行实时动态检测。三个层级层层递进，目前最为基础的是“静态检测”。正在不断提升和优化的是“供应链检测”。未来，“动态检测”技术将可能得到广泛的应用。

（1）静态检测

静态检测的核心是检测镜像中是否存在 CVE 漏洞。通过扫描获取的镜像信息，与 CVE 规则库中的名称和版本进行比对，从而判断是否存在漏洞。目前，比较典型的镜像扫描引擎有 Clair、Trivy 等。

（2）供应链检测

现代的应用 78%-90% 都融入了开源的组件，平均每个应用包含 147 个开源组件，且 67% 的应用采用了带有已知漏洞的开源组件，供应链安全不容乐观。

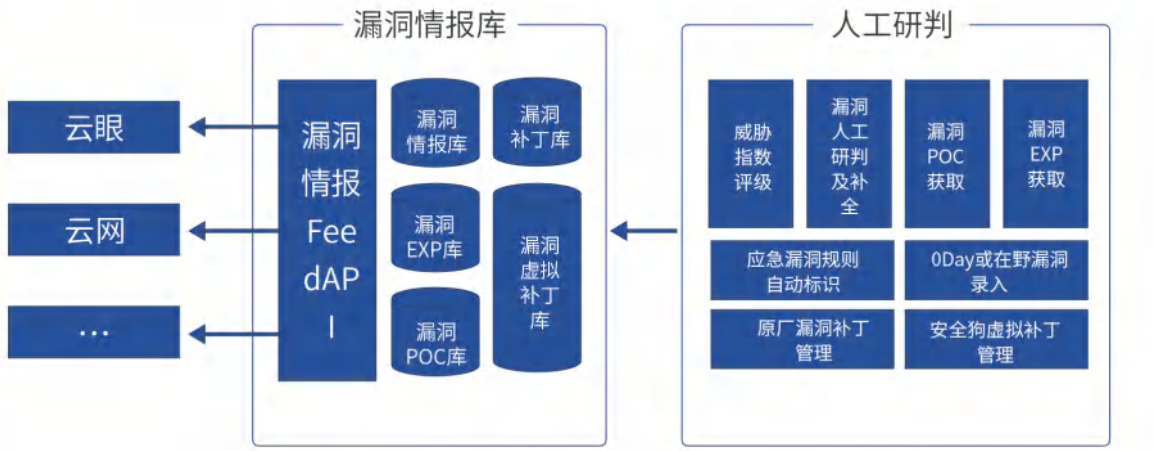
基于对上述两点的考量，业界普遍按照“安全左移”理念采用自动化解决方案帮助开发人员和安全人员左移测试，摒除耗时手动过程。但该方法也使得开发和安全团队应对的最明显挑战就是冗长的漏洞列表，这需要找到恰当的方法高效处置漏洞结果。

在安全运营工作中，想修复所有的漏洞几乎不可能，为了应对此类问题给镜像带来的安全风险，我们可以结合相关威胁情报，发现应用中用到的开源软件，并分析相关代码版本，将其与威胁情报比对，检测出真正需要修复的高风险漏洞隐患进行告警，安全狗漏洞运营团队关注的高风险漏洞如表 4-1 所示。

类型	评判依据
1	属于 0Day 或在野漏洞
2	云服务器关键应用，比如中间件、数据库、容器编排、软件成分等关注对象清单
3	具备以下特点：可远程利用、攻击复杂度低、权限要求低、影响范围广、具备 POC、EXP

表 4-1 安全狗漏洞运营团队关注的高风险漏洞

比如：安全狗采用基于漏洞运营情报的漏洞管理方法和流程，动态地将需要修复的漏洞进行优先级排序和流程优化，提高修复效率，以达到用最少的时间实现最好的效果。图 4-2 为安全狗漏洞运营团队的漏洞情报运营体系总体设计图的局部。安全狗漏洞运营团队对从 CVE、NVD、CNVD、CNNVD、安全社区、高质量头部安全厂商等渠道获取的漏洞情报进行研判、研究，输出高质量的高风险漏洞情报研究成果赋能产品，缓解客户“海量漏洞无从下手，无法聚焦重点的高危漏洞”的痛点。



4-2 安全狗漏洞运营团队的漏洞情报运营体系总体设计图局部

（3）动态检测

基于静态检测和威胁情报，增加基于 eBPF 内核构建容器沙箱。结合自动化渗透测试和模糊测试，使用 eBPF 内核技术构建容器沙箱，对针对镜像文件的可疑访问进行动态检测，对系统的可疑调用、容器之间的可疑互访、检测异常进程、对可疑行为进行取证。该技术是未来发展方向，目前市面上暂无成熟的产品，处于研究或试用阶段。

（二）容器运行时安全检测技术

1．基于规则的已知威胁检测

基于规则的异常检测方法一直是最为经典的方案，容器环境下也依然适用。与传统威胁检测不同的是，容器环境下出现了许多新的威胁场景，如容器逃逸、容器应用漏洞利用，相关的规则和检测引擎也发生了新的变化。

2．基于行为的未知威胁检测

大多数未知行为是无法通过规则匹配检测出来的，因而可通过行为分析来发现未知威胁。一般来说，不同应用的业务场景是不同的，但在生产运营时间内，业务依赖的程序和模块的行为是能确定、可预测的。

基于上述分析，收集内部业务正常运行期间的进程集合及进程行为、属性集合，采用自学习、无监督的方式，可自动构建出合理的镜像、容器进程行为基线。在自学习结束后，可以利用这些行为基线对容器内的未知威胁进行检测。

（三）容器网络安全检测技术

1．微隔离

微隔离是一种细粒度的网络隔离技术，其核心能力聚焦于东西向流量，对容器、主机的东西向流量进行隔离和控制，阻止当攻击者进入云原生网络后进行横向移动。

（1）基于 Network Policy 实现微隔离

Network Policy 是 Kubernetes 的一种资源，可用来说明一组 Pod 之间是如何被允许互相通信，以及如何与其他网络端点进行通信。Network Policy 使用标签选择 Pod，并定义选定 Pod 所允许的通信规则。每个 Pod 的网络流量包含流入（Ingress）和流出（Egress）两个方向。

在默认的情况下，所有的 Pod 之间都是非隔离的、完全可以互相通信，也就是采用了一种黑名单的通信模式。当为 Pod 定义了 Network Policy 之后，只有允许的流量才能与对应的 Pod 进行通信。在通常情况下，为了实现更有效、更精准的隔离效果，会将这种默认的黑名单机制更改为白名单机制，也就是在 Pod 初始化的时候，就将其 Network Policy 设置为 deny all，然后根据服务间通信的需求，制定细粒度的策略，精确地选择可以通信的网络流量。

（2）基于 Sidecar 实现微隔离

另外一种微隔离的实现方式就是采用 Service Mesh 架构中的 Sidecar 方式。Service Mesh（比如 Istio）的流量管理模型通常与 Sidecar 代理（比如 Envoy）一同部署，网格内服务发送和接收的所有流量都经由 Sidecar 代理，这让控制网格内的流量变得异常简单，而且不需要对服务做任何的更改，再配合网格外部的控制平面，可以很容易地实现微隔离。

（3）云原生网络入侵检测

由于云原生环境下，传统的 IDS 无法监控和检测到云原生环境的的入侵行为，所以云原生网络入侵检测机制需要实现对 Kubernetes 集群中每个节点上 Pod 相关的东西及南北向流量进行实时监控，并对命中规则的流量进行告警。告警能够定位到哪一个节点、哪一个命名空间内的哪一个 Pod。

针对云原生环境下的微隔离需求，一些厂商参考了 NetworkPolicy 以及 SideCar 方案等微隔离实施标准，结合其在主机微隔离上的技术积累。提出了同时支持主机工作负载与容器工作负载的微隔离解决方案。安全狗“云隙”微隔离产品就是这种技术路线的代表。具体来说，就是使用 Kubernetes 集群中已经提供的 Network Policy 基础设施进行白名单的网络管控，并使用产品 In Container Firewall 的功能对集群中 Pod 实施黑名单的网络管控。这样不仅有效利用了 Kubernetes 已有的成熟基础设施，同时也利用了产品在主机微隔离体系的成熟积累。

（四）云原生可观测性

在云原生时代，容器化的基础设施使得应用自身变得更快、更轻。一台主机上可以同时承载上百个容器的部署运行，主机上应用的部署密度以及变化频率，较传统环境，有着巨大的变化。因此，需要可观测性来清晰地发现和记录主机快速变化的应用行为。

正所谓“未知攻，焉知防”，面对云原生架构下的大规模集群以及海量灵活的微服务应用，只有知道集群中应用的具体操作、行为，才能够进行高效的安全防护。

全面的云原生可观测性，包括了系统日志、应用日志等一整套完善的日志审计，还包括了各类运行指标的高性能监控，以及进程行为追踪、服务调用链追踪等，使我们能够清晰地了解系统详细的运行状况，进而快速高效地进行威胁的检测、追踪与溯源。

1．日志

日志记录是应用程序执行过程中对操作系统、服务程序等软硬件设备在运行中所产生动作的描述，包括发生时间、动作发起者以及执行过程等详细信息。日志在服务调试、错误或异常定位和数据分析中的作用不言而喻。合理地使用日志可以帮助用户在定位问题和决策分析时做到有理可循、有据可依。随着安全技术的不断进步，日志分析在安全检测中的应用也日趋广泛，日志在容器安全领域发挥极大的作用。

（1）漏洞分析与系统加固

据已有研究显示，容器化漏洞隐藏在云原生应用的方方面面，如文件系统隔离、进程与通信隔离、设备管理与主机资源限制、网络隔离和镜像传输等。可以通过日志记录来回溯攻击的入口与方式、定位漏洞详情、判断业务影响面，进而及时地进行漏洞修复与针对性的系统加固。

（2）安全监测和趋势预判

日志审计是检测安全事件的不可或缺的重要手段之一。通过对业务内部的各项安全日志数据（如集群控制组件产生的攻击日志、应用操作日志等）进行关联分析，实现不同维度的容器安全风险检测和安全趋势的预判。

（3）制定应急响应机制

相比较传统虚拟技术，容器化应用的安全防御技术仍有待提高。一旦遭受恶意攻击，可以通过从日志中还原攻击者的攻击路径挽回已造成的业务损失。

2．监控

监控与日志有所不同，日志提供的是显式数据，而监控是通过对数据的聚合，对应用程序在特定时间内的行为进行衡量。监控数据是可累加的，他们具有原子性，每个都是一个逻辑计量单元，或者一个时间段内的柱状图。通过监控数据可以观察系统的状态和趋势。

传统的监控系统，在面对大规模复杂的业务和平台时，在设计和实现上，就会有重大的挑战。在云原生架构下，如何

实现高效、稳定的监控系统，其面临的挑战将会更严峻。

监控系统在设计和实现上，需要收集并分析大量的系统组件运行信息。除了对单个组件进行监控之外，还需要对系统整体进行多个维度的监测、分析和预警。

对传统业务系统进行监控，我们通常可以知道每个服务组件有多少个实例，每个组件部署在什么位置等信息。但是在云原生架构下，一方面，除了业务本身，还增加了集群、节点、命名空间、Service、Pod 等众多维度；另一方面，节点上承载微服务的容器密度较传统应用部署变的更大，而且容器生命周期变的极短，这就意味着短时间内有大量的容器 ID、Pod 名称、标签等信息在不断的变化。

3．基于 eBPF 实现容器运行时动态追踪

eBPF 机制通过在 Linux 内核事件的处理流程上，插入用户定义的 eBPF 程序，实现对内核的软件定义，极大地提高了内核行为分析与操作的灵活性、安全性和效率，降低了内核操作的技术门槛。

因此，对于云原生环境来讲，如果能够拿到内核所拥有的种种信息，必然对云原生应用的性能提升、可视化监控以及安全检测起重要的意义。

（1）eBPF 介绍

近几年，云原生领域飞速发展，K8s 成为公认的云操作系统。容器的高频率部署、短暂的生命周期、复杂的网络路由，都给内核安全带来了新的挑战。系统内核面对的复杂性在不断增长，在满足性能、可扩展性等新需求的同时，还需要保障系统稳定可用，这是极其困难的事情。此时，eBPF 出现，它以较小的子系统改动，保障了系统内核的稳定，还具备实时动态加载的特性，能将业务逻辑加载到内核，实现热更新的动态执行。

eBPF 由 BPF 发展而来，BPF 全称 Berkeley Packet Filter，1992 年由 Steven McCanne 和 Van Jacobson 提出，1997 年引入 Linux Kernel 2.1，3.0 中增加了即时编译器，应用在网络过滤领域。2014 年 Alexei Starovoitov 实现了 eBPF 并扩展到用户空间，威力更大。常用的 TCPDUMP&LIBPCAP 就是基于它。在 Linux Kernel 4.x 中，扩展了内核态函数、用户态函数、跟踪点、性能事件（perf_events）以及安全控制等事件类型。尤其是近几年云原生快速发展，也带动了 eBPF 的繁荣。

（2）eBPF 在实践中遇到的问题

我们对 eBPF 做了预研，基于 eBPF 技术能够实现更精准、更高效的容器安全防护，对主机和容器异常行为进行检测，对有问题的节点和容器进行自动隔离，保证了多租户容器平台容器运行时安全，但是，目前也存在一些环境方面的客观问题导致无法普及和推广：

① eBPF 内核版本要求高，比如 ecapture 一款基于 eBPF 技术实现的用户态数据捕获工具，需要最低的内核版本为 5.8，CentOS 8.2 以上才能支持。而目前企业级用户生产环境操作系统中 CentOS 7.x/Redhat7.x 占了绝大部分，其内置的 3.10 版本内核对 eBPF 的支持均在较大局限性。

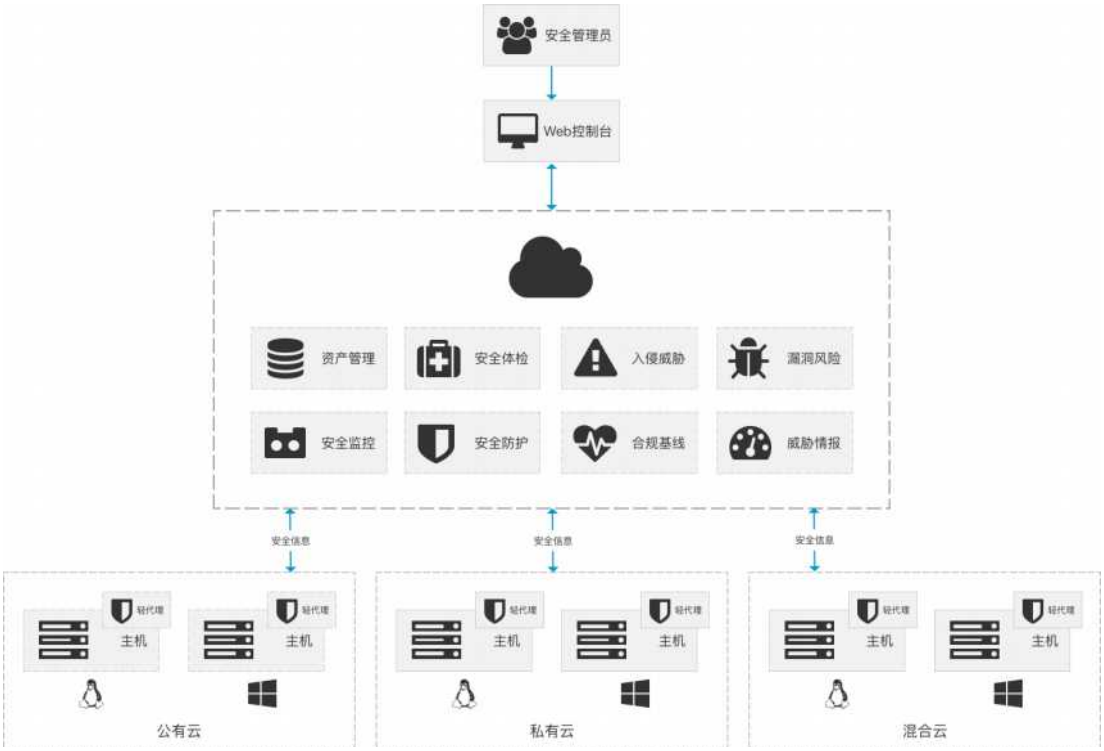
②在每个不同的内核版本对 eBPF 功能特性支持程度不尽相同，这对于企业级生产环境来说，就需要针对每个内核版本进行编译，并对其功能兼容性和稳定性进行测试验证。

总体而言，eBPF 技术是未来的发展方向，随着系统内核不断升级，未来 eBPF 技术有机会被广泛应用。

（五）宿主机威胁检测技术

1．总体技术架构

宿主机入侵监测及安全管理平台通常由端 + 云 + 控制台三部分构成，提供了包含安全体检、资产管理、漏洞风险管理、入侵威胁管理、安全监控、安全防护、合规基线、安全报表、安全告警等功能。其总体技术架构如图 4-3 所示。



4-3 宿主机入侵监测及安全管理平台总体技术架构图

端：主机客户端轻量 Agent

客户端 Agent 部署在服务器上（支持物理服务器、虚拟化服务器），提供多个层面的安全监测和安全防护，可快速识别及阻断黑客攻击。Agent 主要功能包含：采集主机资产，扫描系统和应用漏洞及弱口令等脆弱性风险，检测针对主机的各种攻击入侵行为，监控主机上账号、进程和文件的行为，负责将采集的资产、漏洞风险、入侵告警数据上报到云端安全引擎。

云：Server 端安全引擎

Server 端安全引擎作为云眼平台的中枢，持续接收各个 Agent 上报的信息，并进行解析、处理、分析和保存，根据经验库和用户自定义规则，发现漏洞、异常登录行为、异常网络连接行为、文件异常操作、账号异常操作、异常命令调用行为和进程异常行为，实时发现入侵行为。

控制台：Web 管理平台

以 Web 控制台的形式和用户交互，清晰展示各项安全检测和分析的结果，并对重大威胁进行实时告警，同时以人机交互的方式提供用户自定义各种安全策略，以及灵活的策略分发，提升精细化管理，帮助用户更好更快地处理安全问题。

2．关键技术介绍

现有安全检测体系主要采用基于已知威胁特征的传统检测技术，包括防火墙、IPS、杀毒、沙箱等被动技术。这类技术在以云主机、容器、微服务等为代表的云 / 云原生技术架构场景下，存在一系列难以克服的局限，比如：不能有效检测和防护新型云攻击威胁、云内部访问控制无法精细化隔离、安全代理软件自身运行占资源影响业务应用的正常运行等等。

针对云工作负载新型恶意攻击现有检测能力不足的问题，须结合基于机器学习和多维度运行时监控算法，研发多引擎查杀技术，以有效提升云工作负载新型恶意攻击检测能力。

（1）基于 AI 的云工作负载网页后门查杀技术

网页后门在业内通常称 Webshell，是以文本形式存在的恶意网页脚本文件。黑客可以通过执行 Webshell 的相关命令，达到入侵、控制服务器的目的。常见的 Webshell 可由 php、jsp、asp 等开发语言编写。传统 Webshell 检测方法基于特征匹配进行检测，但由于这些语言的语法灵活且易于混淆或者隐藏特征码，使得传统基于特征匹配的检测方法难以快速且准确地检出 Webshell 文件，对云工作负载安全构成重大威胁。

为弥补传统查杀引擎查杀率不高，易被混淆绕过的缺陷，业界经常采用机器学习技术来研发智能检测引擎以增强系统捕获 webshell 威胁的能力。使得模型在保持相对高效的同时准确率比传统引擎明显提高，并能捕获混淆、变形、加密的 Webshell，甚至可以轻松捕获无法准确提取特征的高级 Webshell。

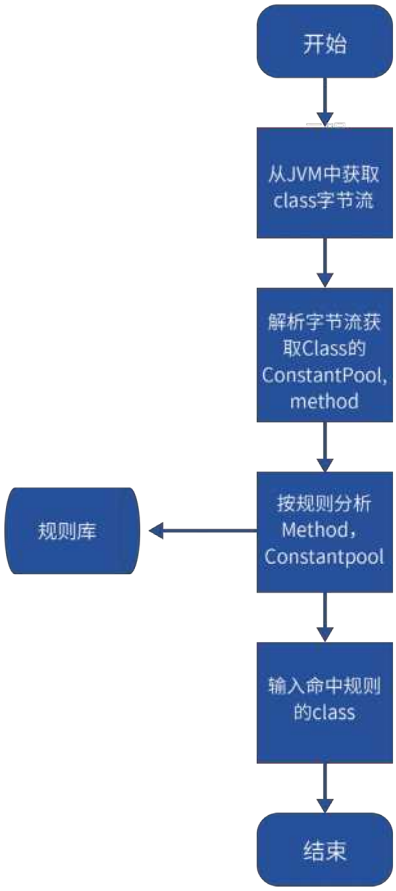
（2）基于多维度运行时监控算法的云工作负载新型攻击检测技术

无文件攻击是现阶段云工作负载面临的又一新型恶意攻击策略，是指没有持久驻留在系统磁盘中的恶意代码攻击，攻击技术包括：内存木马、WEB 远程代码执行、容器逃逸等。对此，目前市场上的产品大多基于内核探针、RASP 等侵入性较强的技术进行检测，此类技术可致系统资源紧缺时，业务连续性、稳定性受影响，且难以对无文件攻击做全面检测。

为解决以上问题，业界开始关注基于多维度运行时监控的“主机层 - 应用层联动”新型攻击检测技术。在应用层，基于多维度运行时监控算法及威胁特征库，通过动态注入防护 Agent 进行检测；在主机层通过进程行为分析，自动匹配预设检测规则。该类技术支持对目前已知的各类无文件攻击场景有较高的检出能力，并且，由于避免使用 dump 及反编译等技术，使得资源消耗低于传统检测方法，面对复杂用户业务场景时也具较强适应性。

针对不同攻击类型，该技术的具体应对策略如下：

① Java 内存马攻击检测技术，获取 JVM 运行时的字节流，提取其 class 字节码的 ConstantPool、Method 等特征，进而检测恶意特征，可高效查杀 Filter、Servlet、Agent 等多类内存木马，如图 4-4 所示。



4-4 Java 内存木马检测流程

② WEB 远程代码执行攻击检测技术，从进程信息中获取数据源，并跟据预设规则模板对实时数据进行分析与检测，可实时地针对 ATT&CK 攻击链各阶段的关键攻击手段做检测，能有效发现黑客利用漏洞执行命令的行为痕迹，并及时进行告警。

③容器逃逸攻击检测技术，将进程监控、文件监控、Linux 内核等方面的安全技术应用在容器新场景，可在攻击前、中、后各阶段开展检测，如表 4-2 所示。

攻击前	攻击时	攻击后
资产清点 镜像安全 基线合规 资产加固	镜像运行控制 容器安全 网络安全 K8S 安全机制	日志审计

表 4-2 基于容器全生命周期的容器逃逸检测

（3）基于云上安全防护组件的资源控制技术，实现多安全场景云工作负载稳定运行

企业级云或云原生安全产品对运行于主机上的云安全代理软件有着非常严苛的技术要求，即安全产品 Agent 自身运行要求非常稳定、性能消耗很低、即使在业务高峰期也不能影响业务应用的正常运行，同时还要防住各类高级威胁。而现有云上安全防护组件往往需要堆叠多个安全 Agent 软件，这在实际环境中将产生严重的性能消耗，影响业务应

用的正常运行，成为阻碍云原生安全防护产品应用部署的关键因素。

解决上述基础性问题的关键技术就是云上安全防护组件的资源控制技术，比如安全狗创新推出“N 合 1”轻代理开放式技术架构和云工作负载安全 Agent 资源控制自适应算法。

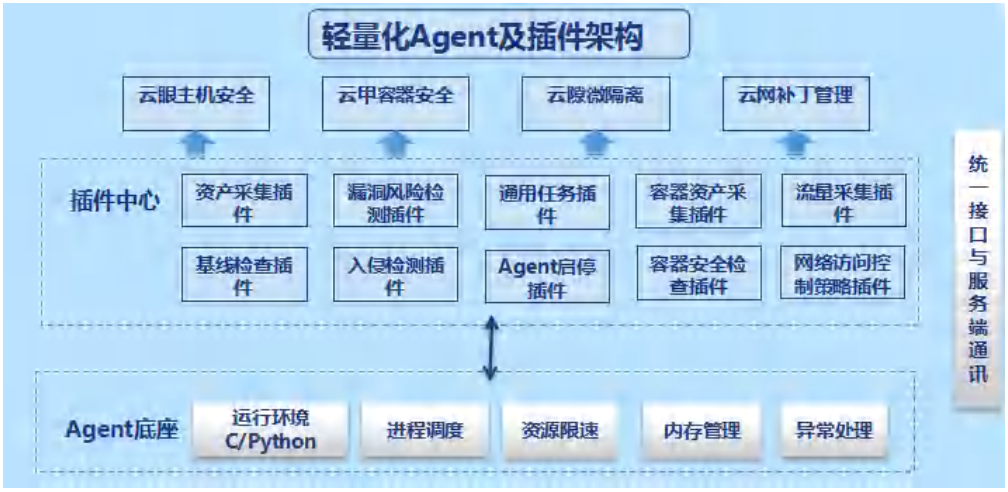


图 4-5 安全狗“N 合 1”轻代理技术架构逻辑图

基于“N 合 1”轻代理技术架构做到在一个工作负载上只需部署一个 Agent 底座，通过插件灵活扩展就能同时满足宿主机安全、容器安全、微隔离、漏洞补丁等多个场景，无需在一个主机上重复部署多个 Agent 软件。

Agent 资源自适应算法在传统的内核态进行资源控制和隔离的基础上提出在用户态应用进程 CPU 资源占用率控制装置 , 用户可根据自身需求设置控制策略，有效提高了 CPU 资源利用率，保障关键业务能分配到足够资源。基于该算法确保云上安全防护组件几乎涵盖所有传统安全防护功能，还能做到单核 CPU <1%, 峰值 <5%。

（六）容器 ATT&CK 矩阵

MITRE ATT&CK（Adversarial Tactics,Techniques,and Common Knowledge）是基于实战的、全球可访问的网络安全对抗战术、技术和常识知识库，也是近几年网络安全领域最热门的工具。在 ATT&CK 框架的帮助下，安全团队对攻击者的行为有了更广泛的了解，成为网络安全学科有用的工具，具有可用于红蓝对抗以及改进网络和系统威胁监测的能力。2021 年 4 月，MITRE 发布了 ATT&CK 第九版，其中 ATT&CK 容器技术矩阵的出现使得针对 K8s 容器环境的攻防对抗研究有了可遵循的基本框架，如图 4-6 所示。该框架创建了一个明确的、可衡量可量化的行业标准和通用语言，以评估其容器的安全风险和安全能力。

4-6 MITRE 的容器 ATT&CK 矩阵

ATT&CK 容器技术矩阵涵盖了编排层（例如 Kubernetes）和容器层（例如 Docker）的攻击行为，还包括了一系列与容器相关的恶意软件。ATT & CK 容器矩阵有助于我们了解与容器相关的安全问题，包括镜像安全、镜像仓库完整性问题、容器运行时安全问题、容器网络层安全以及配置安全问题。于是将 ATT&CK 容器技术矩阵的威胁监测能力与云原生安全技术相结合，以此提升云原生安全下的威胁监测水平，形成“两翼一统”的格局。这对将高度复杂的云原生安全攻防对抗和入侵检测从“玄学”变成“科学”，具有深远的意义。

五、云原生漏洞风险检测模型

云原生漏洞风险的隐患当前主要涉及两方面，其一是用户业务运行所依赖的系统组件中存在漏洞；其二是用户业务组件中存在漏洞。这两方面的漏洞，都会一定程度威胁到云原生整个系统的安全性，所以两方面都应该进行相应的漏洞检测，确保安全无死角。

而从云原生环境的架构角度来对这两个方面进行进一步的细化，又可以得到一个 5 层安全模型，简称“CCICA 安全模型”。

（一）CCICA 云原生安全模型介绍

为了进行云原生环境下的安全治理，须关注“Cloud”（云）、“Cluster”（集群）、“Image”（镜像）、“Container”（容器）以及“Application”（应用）等组成部分，安全狗将这 5 个云原生环境的关键组成部分概括成“CCICA 安全模型”，下面依次展开介绍。

1．Cloud

Cloud——云，一般指数据中心的基础设施。基础设施一般指运行着 Linux 操作系统的宿主机集群，并通过专业的数据交换机进行连接。常见的安全问题主要集中在操作系统本身、Web 中间件以及 rootfs 等方面的漏洞。例如，特定版本内核或者某些驱动模块本身包含有漏洞，CVE-2016-5195（“脏牛漏洞”）就是该类型漏洞的典型代表。该漏洞存在于特定版本的 Linux 内核，由于内核代码没有正确处理 copy-on-write(COW) 功能写入只读内存映射，导致本地攻击者可利用该漏洞获取权限。例如，一些基础组件像 bash、ssh 等软件自身的漏洞，近期爆出的 CVE-2021-4034（Linux Polkit 权限提升漏洞）就是该类型漏洞的典型代表，攻击者可利用该漏洞通过精心设计环境变量诱导 pkexec 程序执行任意代码；再如，一些 Web 中间件像 Tomcat、Weblogic 等软件自身的漏洞，近期爆出的 CVE-2020-2551（Weblogic IIOP 反序列化漏洞）就是该类型漏洞的典型代表，攻击者可利用 IIOP 协议执行远程代码。此外，在基础设施部署过程中，某些不安全配置的引入也可导致的漏洞等，例如对外暴露了某些不安全的端口。

2．Cluster

Cluster——集群，一般指承载云原生环境的编排引擎集群。当前云原生体系建设所使用的编排引擎主要是以 Kubernetes 为基础的各种发行版本。作为 Kubernetes 来说，其本身是由多个组件构成的集群系统。这些组件包括但不限于 kubelet、Docker、containerd、cri-o、etcd、kube-apiserver、kube-controller、kube-scheduler 以及

kube-proxy 等等。这些组件一般都是云原生安全领域重点研究的对象，并确实爆出过一些影响范围广以及威胁较高的漏洞。比较典型的漏洞是 CVE-2019-5736（Docker runC 容器逃逸漏洞），该漏洞源于程序没有正确地处理文件描述符，攻击者可利用该漏洞覆盖主机 runC 的二进制文件并以 root 权限执行命令。同时，由于集群部署的过程中涉及到大量的配置以及权限分配过程，难免会留下一些薄弱的配置选项，一些常见的安全风险包括但不限于 Kubernetes 组件或容器运行时组件未鉴权、允许非安全通道访问的的 Kubernertes Dashboard 服务。它们均可能是导致集群的被攻击面扩大的主要因素。

3．Image

Image——镜像，一般指容器运行的基础镜像。镜像安全问题要一分为二分析，分为静态容器镜像与活动容器镜像。当前云原生体系上的业务应用是以容器的方式进行部署，容器引擎服务支持使用不同的镜像启动相应的容器。为了提高容器镜像数据的复用度，容器镜像一般都采用分层文件系统的方式进行组织。而大部分被复用的数据主要来自于互联网或者某些未知的地方。一些攻击者可能会通过某些精心配置的上游镜像投放来实现对系统的渗透，这些方案包括植入某些可进行漏洞利用的工具或者动态库、Webshell、病毒木马，甚至是添加一些不安全的配置到上游镜像中; 同时，对私有仓库的入侵也可能会导致镜像被污染或者投毒。同时，开发者无心导致的应用漏洞也可能将安全风险带给容器。

4．Container

Container——容器，一般指容器镜像是以容器的形式运行起来后的状态。与传统的 IT 环境类似，容器环境下的业务代码本身也可能存在 Bug 甚至安全漏洞。无论是 SQL 注入、XSS 和文件上传漏洞，还是反序列化或缓冲区溢出漏洞，它们都有可能出现在容器化应用中。与此同时，容器中的 Web 应用容器若对外开放端口，则很有可能被黑客直接利用。容器虽然天然地与主机内核有着一定的隔离，这使得它们有着一定的安全性。但是攻击者可能轻易打破容器的隔离性。比如若某容器被配置了“-privileged”等不安全的配置选项，将不受 Seccomp 等安全机制的限制，容器内 root 权限将变得与宿主机上的 root 权限无异。此外“容器逃逸”问题仍然是运行时容器最为严重的安全风险。因相关程序漏洞导致的容器逃逸（比如 CVE-2019-5136）, 或内核漏洞导致的容器逃逸（比如 CVE-2016-5195）等漏洞风险仍然是需要重点关注的问题。

容器逃逸攻击的实施往往并非一蹴而就，往往是一系列以“权限提升”为目的的攻击步骤的组合，并且达到容器逃逸目的可能的途径也是多样化的。

例如，据 SysdigSecure《2021 容器安全和使用报告》中的“默认启用的 Falco 安全策略触发的报警”统计，“在 /etc 下执行写操作”“启动特权容器”以及“在 root 下执行写操作”是最常见的违规事件，它们均可能是容器逃逸攻击的一环，如图 5-1 所示。

违规项	违规项的描述	为什么它是一个违规项
在 etc 下执行写操作	尝试在/etc 目录下写入文件。	在/etc 中添加或修改文件，可能会改变应用程序的行为。
启动特权容器	开启特权容器	特权容器可以与主机系统级服务进行交互，会对主机操作系统造成危害，它也可以访问其他容器。
在 root 下执行写操作	尝试在 / 或 /root 目录下写入文件。	修改这些目录中的数据可能会在没有授权的情况下在容器上安装软件。
文件系统的可疑活动	文件系统的可疑活动可能会修改系统中敏感或重要的文件。	攻击者可能会尝试访问敏感数据。
启动挂载敏感资源的容器	启动一个挂载敏感目录的容器	容器表明可以访问可能包含敏感文件的数据卷。
可疑的容器活动	鉴别可疑容器相关活动（execs 进如容器等）	可能是容器系统内有危险的迹象
使用 shell 终端进入容器	使用 shell 终端进入容器	那可能会使攻击者能操控系统，下载恶意软件，或进行其他恶意活动。

图 5-1 Sysdig《2021 容器安全和使用报告》中的高频运行时违规事件

5．Application

Application——应用，一般指各类微服务应用。由于研发人员在开发的过程中，会不可避免地使用一些开源项目的代码或者组件，这些代码和组件可能存在漏洞；同时，研发人员在代码研发的过程中若使用不安全的编码方式，可能给微服务应用引入漏洞，进而造成微服务应用对外暴露漏洞，被黑客远程利用。
这些安全问题都应该得到重视，并在正式发布之前进行安全加固，践行“安全左移”。

（二）CCICA 云原生安全模型漏洞检测方法概述

基于云原生安全的五大关键组成部分，总结出“CCICA”云原生漏洞风险检测模型。该模型以云原生架构为基础，由外到内将云原生环境分为 5 个层次，如图 5-2 所示。

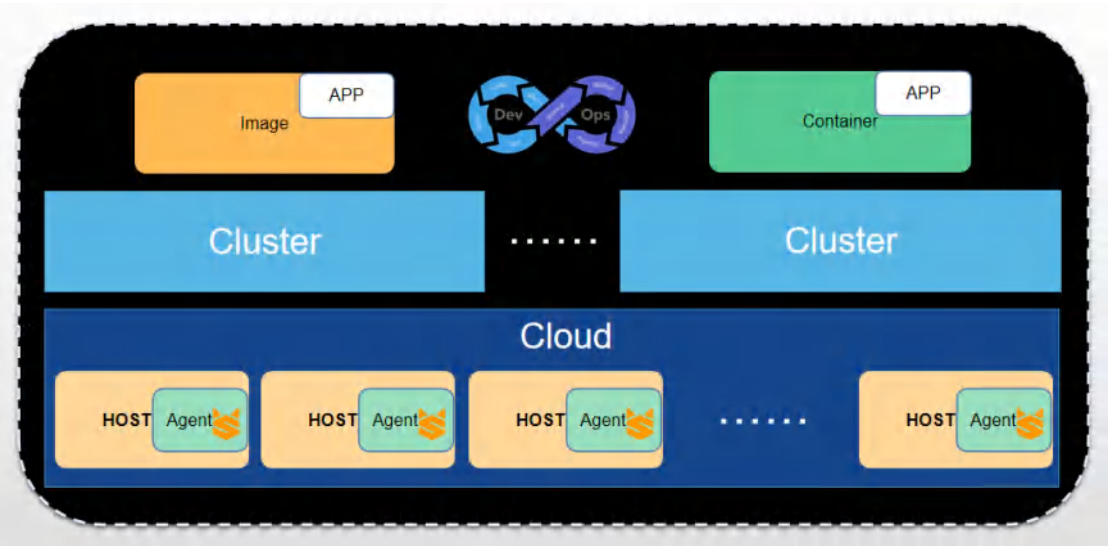


图 5-2 CCICA 模型

在云原生环境下，漏洞风险检测应具备以下两个特点：

（1）采用内嵌方式，而无需外挂部署；

（2）要充分利用云平台原生的资源和数据优势，解决云计算面临特有的安全问题。

安全狗的基于 Agent 的“轻量化、插件化”架构具备上述两个特点，可针对“CCICA”这一云原生安全五层模型进行漏洞扫描，下面依次展开介绍。

1．Cloud

这一层级主要对应于数据中心的基础设施，属于主机安全范畴。目的是帮助客户检测基础设施的安全风险，作为实施主机安全加固的依据。主要应用方案为云安全领域的主机漏洞扫描方案。（案例：云眼的应急漏洞、系统漏洞、网站漏洞、弱口令、风险账号、配置缺陷扫描）。图 5-3 为云眼的“应急漏洞”功能截图。

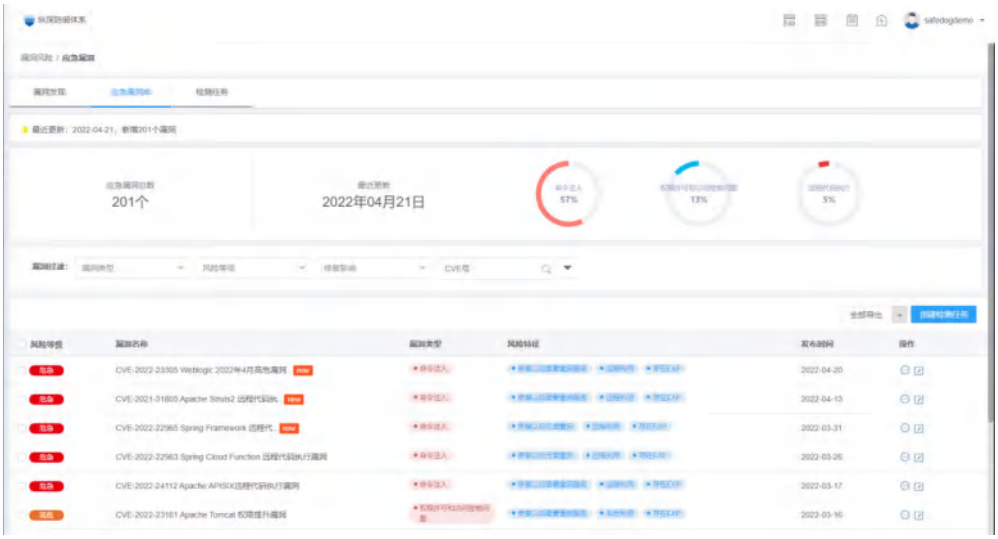


图 5-3 云眼“应急漏洞”功能截图

2．Cluster

这一层级主要对应云原生编排平台，主要组成部分是云原生环境的基础平台组件（例子：Kubernetes 的控制层平面）。目的为帮助客户获知集群自身的安全风险基本信息，为后续加固方案制定提供依据。主要应用方案为编排平台漏洞扫描以及 CIS 安全合规基线检查。（案例：云甲的 Kubernetes 集群漏洞扫描，基于端口扫描、PoC 测试，检查集群环境可能存在的漏洞；云甲基于 CIS 的 Docker、Kubernetes、OpenShift 安全基线，可进行一键自动化检测并提供可视化基线检查结果和代码级的修复建议帮助用户及时发现并修复脆弱点。）图 5-4 为云甲的“基础设施安全 - 组件漏洞”功能截图。

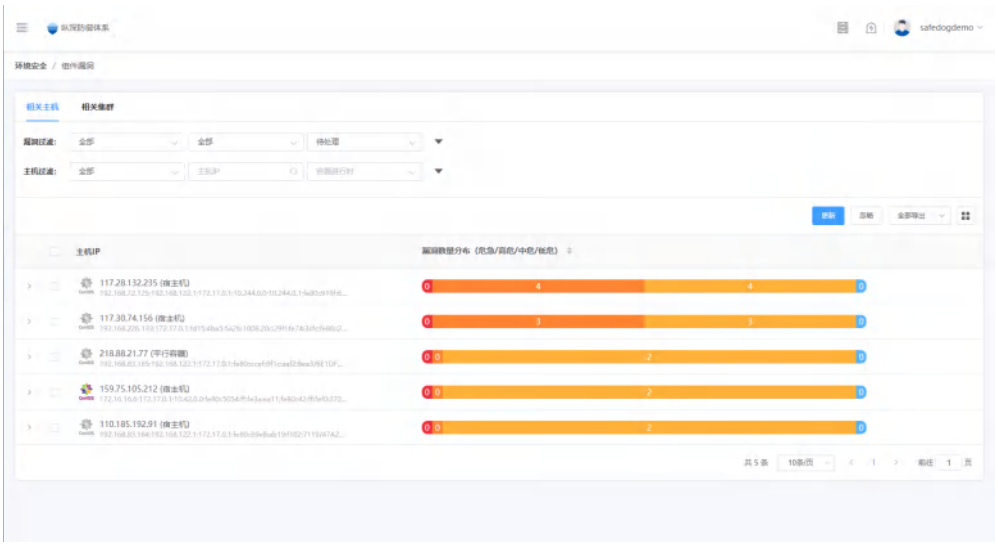


图 5-4 云甲“基础设施安全 - 组件漏洞”功能截图

3．Image

这一层级主要对应云原生环境中的各种镜像，包括来自镜像仓库以及宿主机的各种容器镜像。目的是解决云原生环境中镜像安全问题。主要应用方案为容器镜像漏洞扫描、敏感信息扫描、病毒木马扫描、网页后门扫描。（案例：云甲镜像安全扫描），图 5-5 为云甲的“镜像漏洞扫描”功能截图。

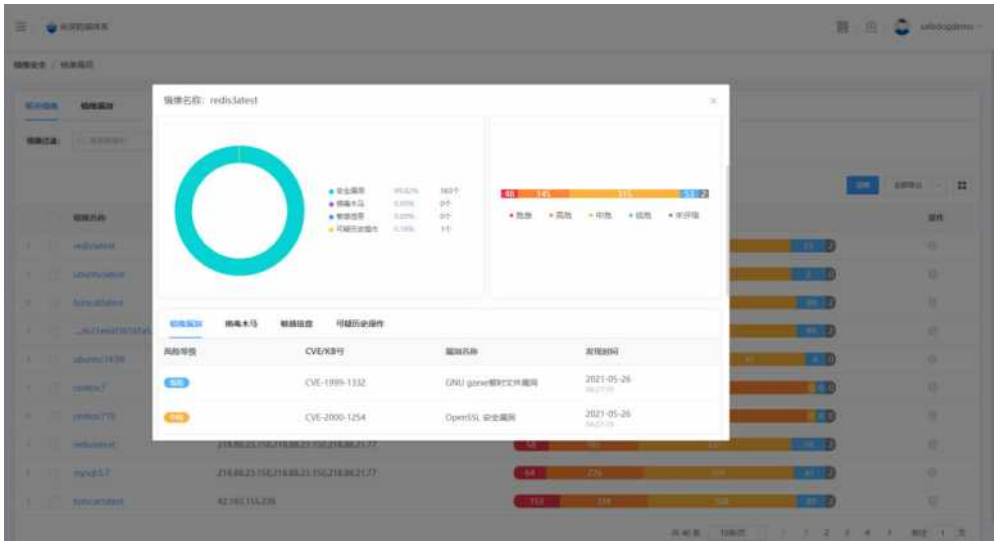


图 5-5 云甲“镜像漏洞扫描”功能截图

4．Container

这一层级主要对应云原生环境中运行的各种容器，目的是解决云原生环境中容器运行时安全问题。主要应用方案为“运行时威胁检测”；该应用方案可以分为“恶意文件查杀”“基于规则的已知威胁检测”以及“基于行为模型的未知威胁检测”三方面来看。在“恶意文件查杀”方面，基于“静态扫描”和“动态实时防护”进行恶意代码查杀，并对它们进行信任、隔离、下载、删除等；在“基于规则的已知威胁检测”方面，深入研究漏洞底层原理，进行威胁分析、特征提取（如进程的异常文件行为、高危系统调用、等方面的特征）和规则构建，并利用构建好的规则进行检测实战。同时，深入研究攻击者的渗透手法，进行容器内的异常命令和文件异常操作威胁检测；在“基于行为模型的未知威胁检测”方面，通过对容器的进程事件、文件事件、网络事件、进行一定周期的学习，构建容器行为基线，以期对基线外的行为产生告警，检测 0day 漏洞攻击。（案例：云甲运行时安全功能）。为了进行容器运行时安全的威胁检测，需要层层监控、层层设防，如图 5-6 所示。

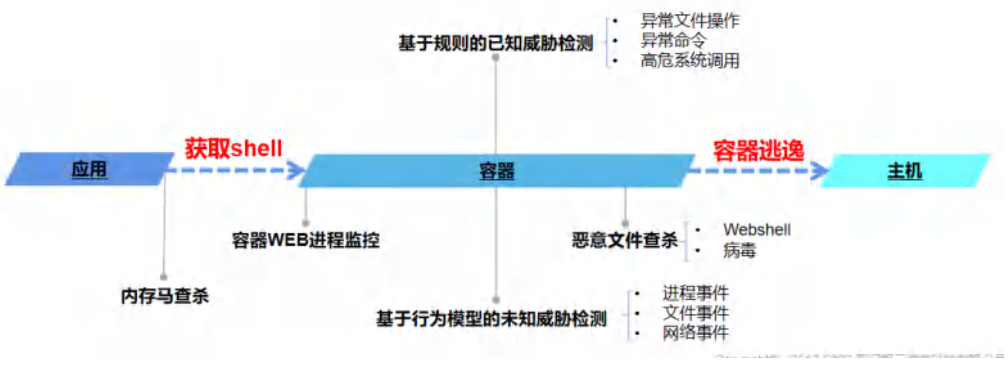


图 5-6 云原生运行时安全检测思路

5．Application

这一层级主要对应容器镜像中用户业务相关的运行实体，主要是安装于镜像之中的各种业务模块，比如 WEB 应用。目的是防止客户业务自身引入不安全的配置、组件和代码到系统之中，从而影响整体的系统安全。主要应用方案为软件成分分析以及 WEB 漏洞扫描。（案例：云甲镜像扫描，针对 CI/CD 构建阶段以及宿主机及仓库中的镜像进行第三方依赖检查；针对上线的容器应用开展软件成分分析、WEB 漏洞扫描，发现 WEB 安全风险），如图 5-7、5-8 所示。

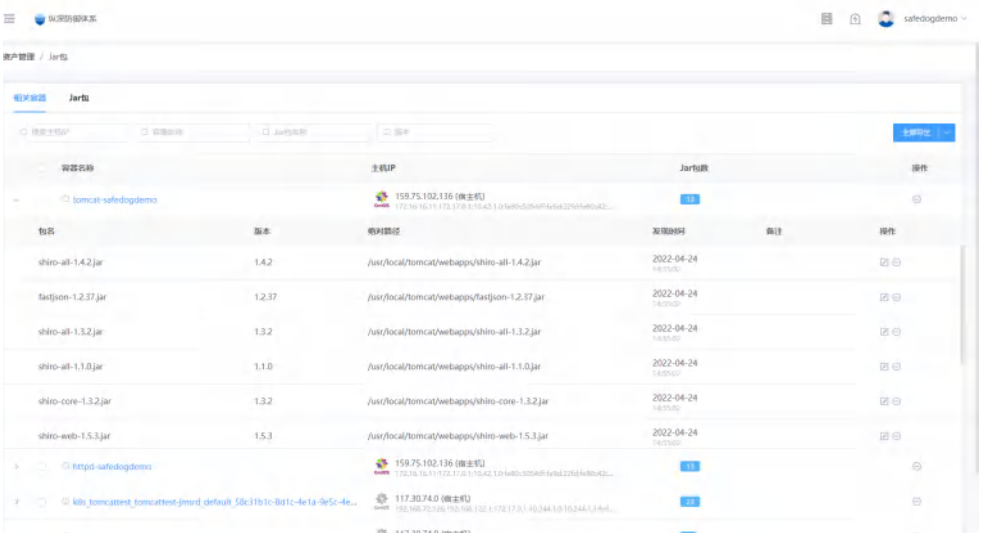


图 5-7 使用云甲进行软件成分分析

图 5-8 使用云甲进行微服务漏洞扫描

六、云原生入侵风险检测模型

当前，虽然业界对基于 ATT&CK 评估、强化产品安全能力有着广泛的研究，但是却没有针对 K8s 容器环境的，以提高“安全产品检测能力”为目标的框架模型，这无疑会阻滞容器安全产品检测能力的持续提升。在这一问题背景下，安全狗提出了一种基于 ATT&CK 容器矩阵的可应用于提取检测规则，提升产品检测能力的 AKDA 模型。

（一）AKDA 模型

1．模型架构

ATT&CK 的出现为终端安全产品的检测能力提供了一个明确的，可衡量，可落地的标准。为了能够更好地基于 ATT&CK 评估、强化安全产品的能力上，我们提出基于 ATT&CK 的 AKDA 模型，该模型的示意图如图 6-1 所示。

图 6-1 AKDA 模型示意图

AKDA 是 Attack、Knowledge、Data Sources 以及 Algorithm 这 4 个单词的首字母，即为此模型的关键四步，故名为“AKDA 四步法模型”。其中“A”代表“Attack”，其含义是：模拟红队攻击；“K”代表“Knowledge”，其含义是：构建攻防知识图谱；“D”代表“Datasource”，其含义是：确定数据源；“A”代表“Algorithm”，其含义是：总结威胁检测算法。该模型的该模型的具体使用步骤如下：

（1）模拟红队攻击

基于 ATT&CK 框架中的某个技战术或子技战术构建攻防对抗场景，红队模拟威胁行为并执行相关技术；

（2）构建攻防知识图谱

事后复盘，通过将整个模拟攻击过程提炼成一张攻防对抗图谱，图谱内的信息包括红队使用的攻击技战术、威胁 IoC、产生的安全事件以及数据源。通过攻防对抗图谱展现红蓝双方的博弈；

（3）确定数据源

模拟攻击结束后通过查看系统产生的特征或行为确定数据源（ATT&CK 也在知识库中定义了数据源、数据组件等字段）；

（4）提炼威胁检测算法

结合模拟实验确定的数据源，思考防御规则的设置，从而确定该技战术的威胁检测算法。

（二）模拟红队攻击

“模拟红队攻击”是利用 AKDA 模型的第一步。为了进行模拟红队攻击，首先应根据需求构设实验环境，传统的渗透测试侧重于突出攻击者可能在某个时间段会利用不同类型系统上的哪些漏洞。而模拟实验则不同，为了能有效提升产品的检测能力，需要不断调整红蓝对抗场景，研究不同场景下的攻击技术。由于在真实的场景下，大多数真正的攻击者都有特定的目标（例如，获取对敏感信息的访问权限）；所以，在模拟对抗时，往往会为红队指定特殊的目标，让蓝队能够针对特定的攻击技战术进行研究和测试。

在 ATT&CK 框架中包含了攻击组织、软件、技术 / 子技术、战术、缓解措施这五大对象，各对象之间的关系可以通过图 6-2 直观地看到。我们可以用之作为指导框架设计实验。

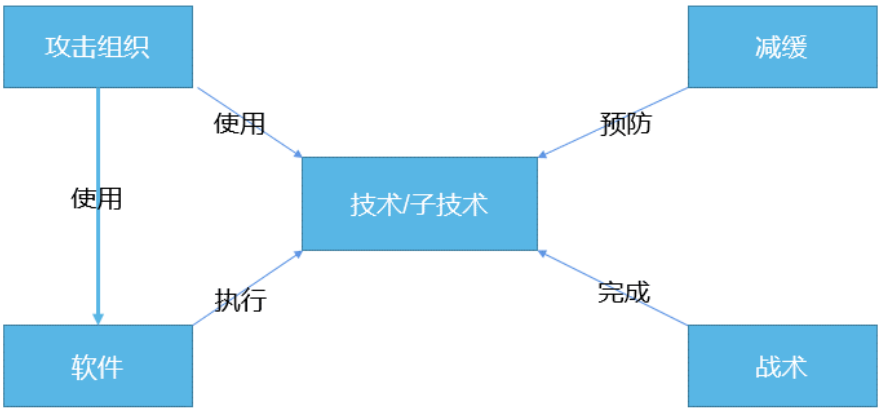


图 6-2 ATT&CK 五大对象间关系

图 6-3 展示了在设计相关的技战术实验环境后，红队人员利用“容器内应用漏洞入侵”拿下集群内一个 WEB 服务的容器，继而通过集群的内网扫描发现集群的节点和可利用端口，并逃逸至宿主机，以实现进一步的集群内的横向渗透的过程。在该过程中，攻击者通过获取到了一个简单的容器 shell 后逃逸宿主机，进而进一步横向渗透扩大战果影响到整个集群，并且利用到多种安全工具，实现宿主机 getshell 后创建后门容器以实现第二次的攻击。

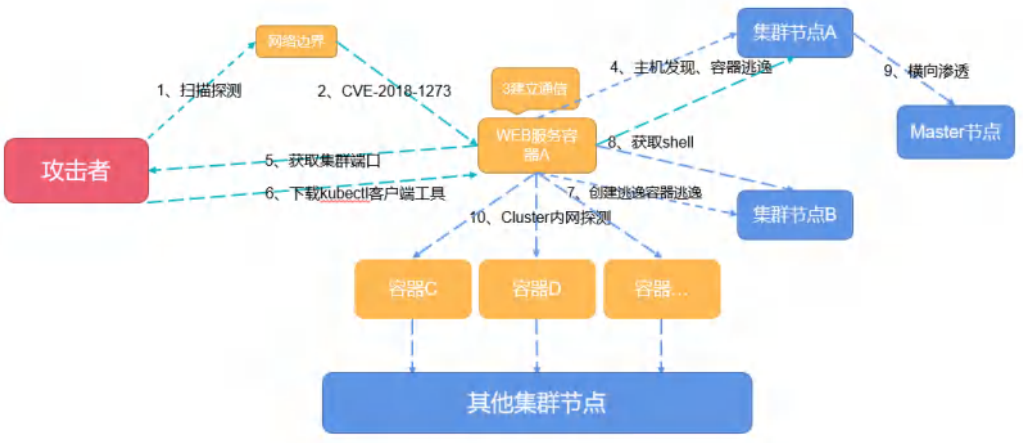


图 6-3 渗透 K8s 集群实验的流程

在蓝队人员在此次红队的技战术实验后，进行安全事件分析、入侵溯源所创建的热力图。由图 6-4 可知，红队的技战术可对应 ATT&CK 矩阵中的多项技战术。ATT&CK 矩阵可作为红蓝对抗的“战术板”。

Initial Access 初始访问	Execution 执行	Persistence 持久化	Privilege Escalation 权限提升	Defense Evasion 防御逃逸	Credential Access 窃取凭证	Discovery 探测	Lateral Movement 横向移动	Impact 影响
云账号 AK 泄露	通过 kubectl 进入容器	部署远控容器	利用特权容器逃逸	容器及宿主机日志清理	K8S Secret 泄露	访问 K8S API Server	窃取凭证攻击云服务	破坏系统及数据
使用恶意镜像	创建后门容器	通过挂载目录向宿主机写文件	K8S Rolebinding 添加用户权限	K8S Audit 日志清理	云产品 AK 泄露	访问 Kubelet API	窃取凭证攻击其他应用	劫持资源
K8S API Server 未授权访问	通过 K8S 控制器部署后门容器	K8S cronjob 持久化	利用挂载目录逃逸	利用系统 Pod 伪装	K8S Service Account 凭据泄露	Cluster 内网扫描	通过 Service Account 访问 K8S API	Dos
K8S configfile 泄露	利用 Service Account 链接 API Server 执行指令	在私有镜像库的镜像中植入后门	通过 Linux 内核漏洞逃逸	通过代理或匿名网络访问 K8S API Server	应用层 API 凭据泄露	访问 K8S Dashboard 所在 Pod	Cluster 内网渗透	加密勒索
docker daemon 公网暴露	带有 SSH 服务的容器	修改核心组件访问权限	通过 Docker 漏洞逃逸	清理安全产品 Agent	利用 K8S 准入控制器窃取信息	访问私有镜像库	通过挂载目录逃逸到宿主机	
容器内应用漏洞入侵	通过云厂商 CloudShell 下发指令		通过 K8S 漏洞进行提权	创建影子 API Server		访问云厂商服务接口	访问 K8S Dashboard	
Master 节点 SSH 登录凭据泄露			容器内访问 docker.sock 逃逸	创建超长 annotations 使 K8S Audit 日志解析失败		通过 NodePort 访问 Service	攻击第三方 K8S 插件	
私有镜像库泄露			利用 Linux Capabilities 逃逸					

图 6-4 渗透 K8s 集群实验的 ATT&CK 矩阵热力图

（三）攻防知识图谱

基于“模拟红队攻击”，可以进行 ATT&CK 的攻防知识图谱的构建，如下图 6-5 所示。“攻防知识图谱”的依据主要在于“红队在实施攻击的时候，将不可避免地产生失陷信标 IoC 以及攻击信标 IoA，因而蓝队在识别攻击的时候是有迹可循的”。由攻防对抗图谱可以洞悉红蓝双方的博弈。在该图中，蓝队的目标是“监控”，需要从左往右看，而红队的目标是“攻击”，需要从从右往左看。蓝队可通过此图思考防御规则的设置，而红队可通过此图思考防御规则的绕过。

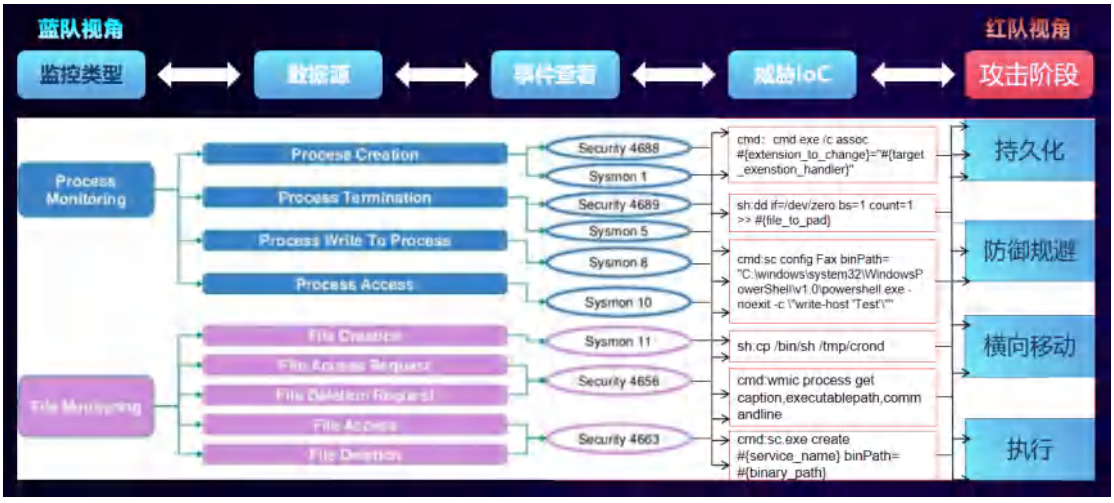


图 6-5 攻防知识图谱示意图

基于 ATT&CK 框架进行检测分析，这种方式与传统的检测方式有所不同。基于 ATT&CK 检测分析并不是识别已知的恶意行为然后进行阻断，而是收集在系统上发生的事件数据，并基于这些数据来识别是否为 ATT&CK 知识库中描述的可疑行为。

（四）数据源

在 AKDA 模型中，确定数据源是重要的一环。数据源的确定是总结该安全场景下的威胁检测算法的前提。在 ATT&CK 框架下的每项技术描述中都有对应于该技术的数据源信息。它告诉我们可以从哪些类型的数据中找到攻击技术实施后所留下的痕迹。例如对于“命令及脚本执行”的攻击战术，数据源可以是命令执行、模块加载、进程创建以及脚本执行。数据源往往是模拟攻击结束后蓝方“打扫战场”发现的特征或行为。并且在实际的实战过程中，针对某个技战术往往要联合多个数据源进行报警，使报警的准确率更高。笔者经过对多个容器安全产品进行研究中发现，备受关注的前三分别是进程监控、进程命令行参数以及文件监控。通过异常进程、异常文件操作及异常命令等角度，可覆盖 K8s 运行时安全的高频攻击场景。这和 ATT&CK 官方统计的数据是相吻合的。

（五）威胁检测算法

威胁检测算法是“AKDA 模型”的最后一环，也是提升产品检测能力的关键。“红队模拟攻击”是以红队的视角思考攻击策略，模拟黑客攻击组织机构，而“威胁检测算法”则是思考针对此类攻击的“防御”。

我们可以基于进程行为、文件行为、网络行为以及注册表行为等等关键行为，通过威胁检测算法判断出“异常行为”对 ATT&CK 技战术的命中情况，如图 6-6 所示。

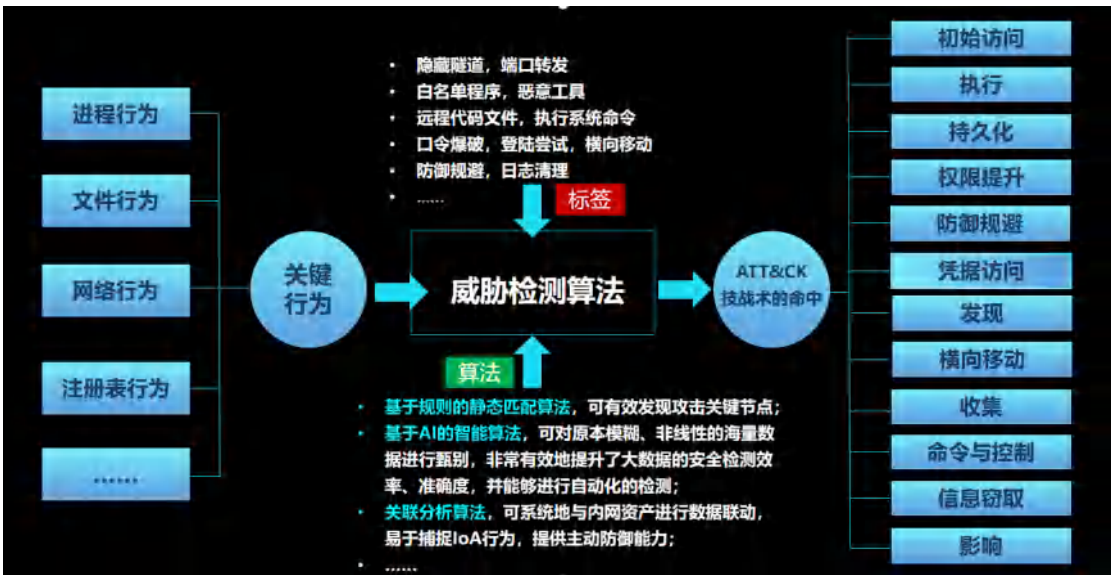


图 6-6 威胁检测算法图构建过程

威胁检测算法的构建往往需要不同专业背景的人员进行协作。例如，由安全研究人员梳理具有专业背景的知识库、标签，并与编程人员、算法研究人员一道设计并优化基于规则的静态匹配算法、基于 AI 的智能算法，或者基于关联分析的算法。

（六）AKDA 模型与 ATT&CK 之间的关系

在云原生环境中，容器编排系统处于重中之重的地位。据 Sysdig 的《2021 容器安全及使用现状报告》统计，K8s（Kubernetes）在容器编排市场中的占比高达 90%，已经成为容器编排的“事实标准”。然而，传统的安全防护方式对解决 K8s 容器安全问题有着“威胁监测能力不足”的局限性。因此针对 K8s 容器环境的威胁检测技术亟待发展，该项研究意义重大。

ATT&CK 容器矩阵的出现使得针对 K8s 容器环境的攻防对抗研究有了可遵循的基本框架。该知识库不仅为针对 K8s 集群的渗透测试和红蓝对抗提供了理论基础，还成为了一种攻防双方都认可的一种通用语言，并已被广泛应用于渗透测试活动、防御检测评估、安全运营中心成熟度评估以及网络威胁情报收集等方面。

通过模拟红队攻击、构建攻防知识图谱、确定数据源以及威胁检测算法四个阶段阐述云原生下攻防实践以提升产品能力的方法。让相对抽象的容器攻击技战术变得有迹可循，以更高效、更准群的方式评估、提升安全产品的检测能力。

未来的安全产品的检测能力将随着攻击技术的发展变得更全面，“AKDA 四步法模型”也应该变得更完善，弥补基于容器 ATT&CK 在安全检测能力方面的短板。

七、云原生威胁检测实战场景

在真实的云原生威胁入侵的场景中，安全威胁往往会涵盖到镜像、应用、容器、网络以及主机等多个维度。旦有一个脆弱点被利用，那么产生的安全威胁就会像“多米诺骨牌倒塌”一样迅速扩大。

安全是一个整体，任何一个环节出问题，都会影响到其他环节，比如容器逃逸会影响到虚拟主机，虚拟主机逃逸会影响到虚拟化基础设施，如图 7-1 所示。

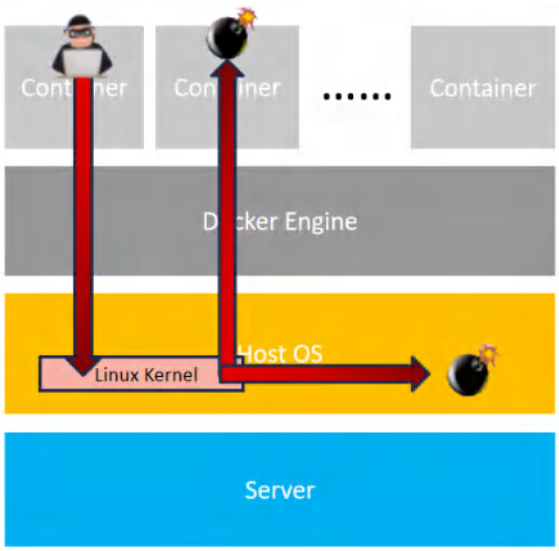


图 7-1 容器逃逸攻击示意图

K8s 集群环境下的攻击模式图如下图所示，当黑客入侵初始脆弱点的“起点容器”，可进一步攻击利用其他存在的脆弱点，扩大自己的战果，如图 7-2 所示。

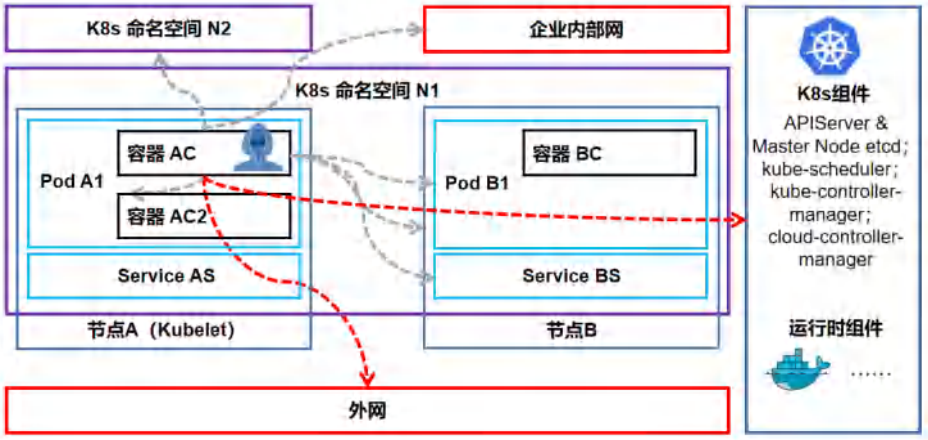


图 7-2 传统或新型的 K8s 攻击利用方式

由于传统的安全方式无法完整、有效保护容器和 K8s 这些重要的基础设施，因而它们容易成为攻击者的攻击目标。对此，不少安全团队推出了可用于剖析容器和 Kubernetes 环境攻击面的 ATT&CK，如下图 7-3 所示。

Initial Access 初始访问	Execution 执行	Persistence 持久化	Privilege Escalation 权限提升	Defense Evasion 防御逃逸	Credential Access 窃取凭证	Discovery 探测	Lateral Movement 横向移动	Impact 影响
云账号 AK 泄露	通过 kubectl 进入容器	部署远控容器	利用特权容器逃逸	容器及宿主机日志清理	K8S Secret 泄露	访问 K8S API Server	窃取凭证攻击云服务	破坏系统及数据
使用恶意镜像	创建后门容器	通过挂载目录向宿主机写文件	K8S Rolebinding 添加用户权限	K8S Audit 日志清理	云产品 AK 泄露	访问 Kubelet API	窃取凭证攻击其他应用	劫持资源
K8S API Server 未授权访问	通过 K8S 控制器部署后门容器	K8S cronjob 持久化	利用挂载目录逃逸	利用系统 Pod 伪装	K8S Service Account 凭据泄露	Cluster 内网扫描	通过 Service Account 访问 K8S API	Dos
K8S configfile 泄露	利用 Service Account 链接 API Server 执行指令	在私有镜像库的镜像中植入后门	通过 Linux 内核漏洞逃逸	通过代理或匿名网络访问 K8S API Server	应用层 API 凭据泄露	访问 K8S Dashboard 所在 Pod	Cluster 内网渗透	加密勒索
docker daemon 公网暴露	带有 SSH 服务的容器	修改核心组件访问权限	通过 Docker 漏洞逃逸	清理安全产品 Agent	利用 K8S 准入控制器窃取信息	访问私有镜像库	通过挂载目录逃逸到宿主机	
容器内应用漏洞入侵	通过云厂商 CloudShell 下发指令		通过 K8S 漏洞进行提权	创建影子 API Server		访问云厂商服务接口	访问 K8S Dashboard	
Master 节点 SSH 登录凭据泄露			容器内访问 docker.sock 逃逸	创建超长 annotations 使 K8S Audit 日志解析失败		通过 NodePort 访问 Service	攻击第三方 K8S 插件	
私有镜像库泄露			利用 Linux Capabilities 逃逸					

图 7-3 云上容器 ATT&CK 攻防矩阵

从容器 ATT&CK 的攻击技术可以得知，其攻击手段覆盖“镜像安全”“应用安全”“容器安全”“主机安全”等维度的云原生威胁实战场景，下面就这些维度展开介绍。

（一）镜像安全

镜像是容器运行的基础。容器引擎服务可使用不同的镜像启动相应的容器。在容器出错后，它能迅速通过删除容器、启动新的容器来恢复服务，这都需要以容器镜像作为支撑技术。所以在云原生安全领域，容器镜像的安全首当其冲。

随着容器技术的成熟和流行，大部分的开源软件都提供了容器镜像及其构建脚本（如 Dockerfile）。在现实的容器化应用开发过程中，人们很少从零开始构建自己的业务镜像，通常将公开镜像作为基础镜像，在此基础上增加自己的代

码或程序，然后打包成最终的业务镜像并上线运行。

毫无疑问，这种积累和复用减少了造轮子的次数，大大提高了开发效率和软件质量，推动了现代软件工程的发展。如今，一个较为普遍的情况是，用户自己的代码依赖若干开源组件，最后打包的业务镜像中好包含有众多冗余的开源组件。包含的组件数量与存在的安全风险程度成正比，大量引入第三方组件的结果也 增加了漏洞被利用的风险。并且，黑客可能将病毒木马放到镜像中大肆传播，造成大量使用该镜像的用户受到威胁，如图 7-4 所示。

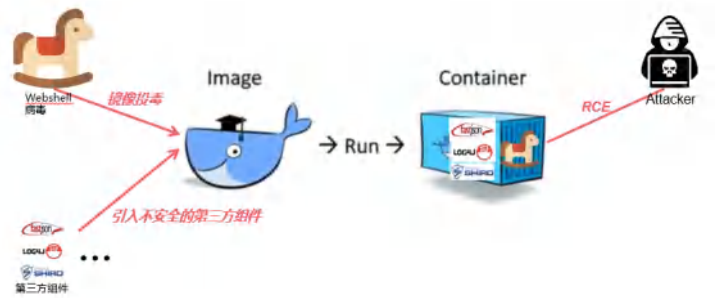


图 7-4 不安全镜像可引发容器 RCE 攻击

对于木马病毒或网页后门文件，依照处理病毒镜像的方式方法以及表现的行为特征可以总结如表 7-1 所示的攻防知识图谱。

监控	数据源	事件查看	IoC/IoA	阶段
镜像 安全 扫描	镜像	植入病毒木马 植入 Webshell	病毒木马特征 Webshell 特征	初始访问、 持久化

表 7-1 “镜像投毒”攻防知识图谱

对总结的攻防知识图谱反馈的信息可以得知，基于镜像安全扫描功能的防病毒引擎以及恶意代码特征库检测出被植入病毒、网页后门的镜像。

被投毒镜像的检测结果如图 7-5 所示。

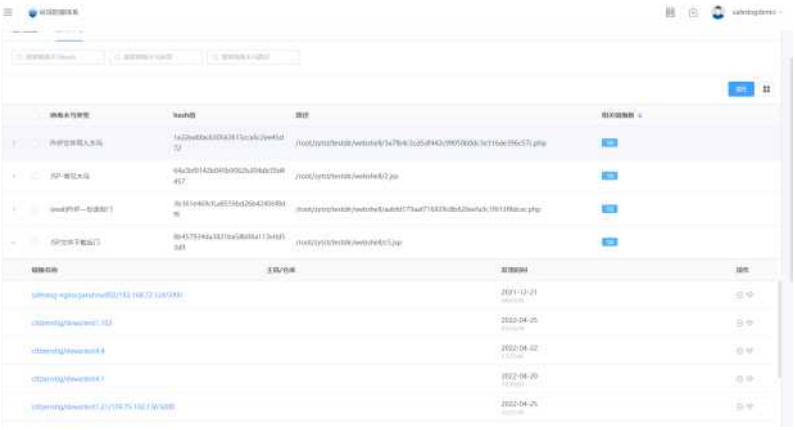


图 7-5 检出投毒镜像

又如，近日如平地惊雷般席卷整个安全圈的 Log4J2 RCE 漏洞（CVE-2021-44228），由于 Log4J2 组件在镜像中被广泛引用，其覆盖面之广使许多用户都难逃一劫。若用户的 WEB 应用镜像里使用了受到该漏洞的影响的 Log4J2 组件，则很可能遭受黑客攻击。由此可见，对于镜像的安全检测就显得尤为重要。

对于引入了不安全的第三方组件的检测，如含有 Log4J2 RCE 漏洞的 Log4J2 组件，可通过检索、扫描镜像中的 jar 包，并匹配相应的特征，进而判定镜像是否受到漏洞影响。

镜像中有诸多安全风险需要检测，其核心也包括已知系统的 CVE 检测。扫描器获取到镜像后，将它分离成相应的层和软件包，接着针对软件包进行漏洞扫描，从而判定是否存在漏洞。

对于镜像安全检测，可以将之归纳为如图 7-6 所示的流程图。

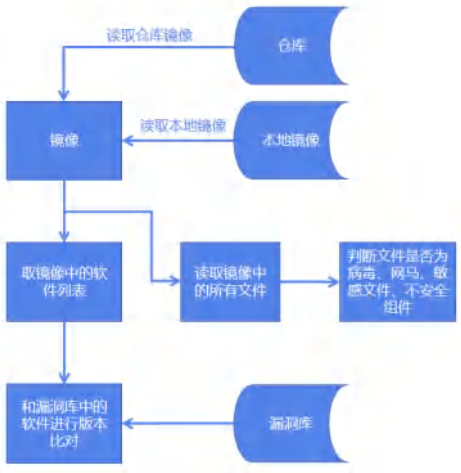


图 7-6 镜像安全检测流程图

（二）应用安全

编排系统的成熟极大促进了微服务架构的云原生应用的落地实践，然而这些新型的微服务体系也同样存在各种安全风险。

云原生应用源于传统应用，因而云原生应用也就继承了传统应用的风险。因此，在云原生环境下，传统的 WEB 攻击手段依然有效。例如，黑客若探测到某云原生应用存在前文所述的 Log4J2 组件漏洞，可以进一步利用该漏洞实施类似传统环境下的攻击，如下图 7-7 所示。

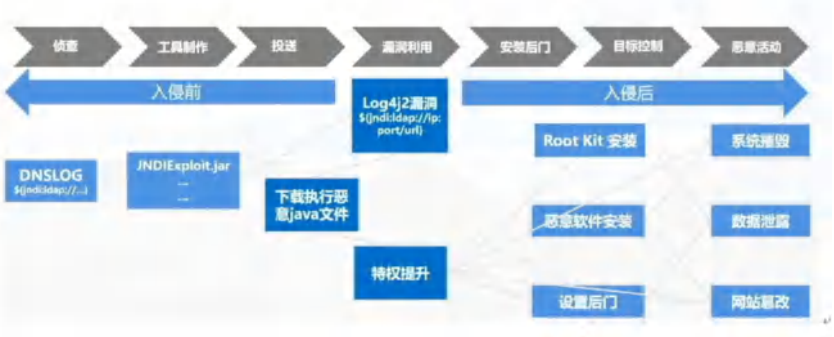


图 7-7 Log4J2 RCE 攻击场景

由于云原生应用架构的变化进而导致应用 API 交互的增多，可以说云原生应用中大部分交互模式已从 WEB 请求 / 响应转向各类 API 请求 / 响应，因而 API 风险也进一步提升。不仅传统的常见 WEB 攻击风险同样存在与 API 之中。除此之外，API 也面临着许多业务层面的安全威胁，例如调用参数异常，调用频率异常，调用逻辑异常等。

此外，新型的微服务体系（如无服务、服务网格）也催生出更新型的攻击手段。例如，攻击者可通过编写一段无服务器的代码获得运行无服务程序容器的 shell 权限，进而对容器网络进行渗透。

对于应用安全的检测，可基于“软件成分分析”“SAST”“DAST”“IAST”与“RASP”等技术手段进行威胁检测。

对于存在有 Log4J2 RCE 类似的安全问题的容器，客户可通过软件成分分析规避风险。根据样本组件特征来匹配被测程序中的特征来判断应用程序是否引用该组件，因此支持组件的数量越多，其检测率也就越高，支持的组件数量越少，则越容易导致检测遗漏；另外检测算法和特征设计是否合理也直接影响到分析的准确性和分析效率。我司产品针对 Log4J2 漏洞的检测效果如图 7-8、7-9 所示。

图 7-8 检出 Log4J2 RCE 漏洞

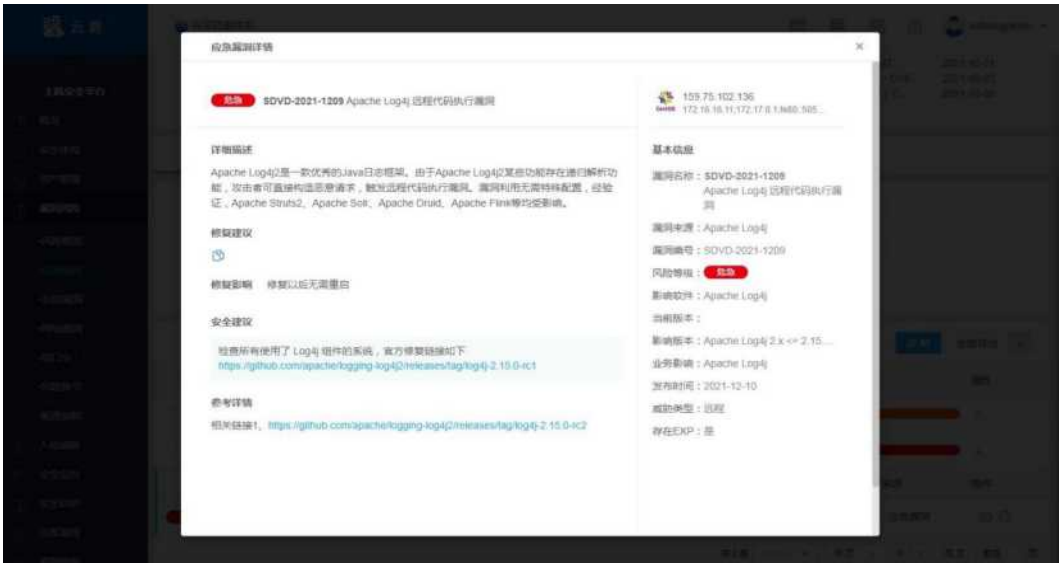


图 7-9 检出 Log4J2 RCE 漏洞

而“SAST”“DAST”“IAST”等技术用于上线前的安全检查，“RASP”技术则用于上线后的应用运行时保护。

（三）容器安全

在容器安全问题中，容器逃逸是最被重视的部分之一。原则上，容器环境与主机环境隔离。然而，容器的便捷性和容器与宿主机共用内核息息相关，容器技术天然地存在内核隔离性不足的问题。因此，攻击者有机会突破容器以访问底层主机。这可以允许攻击者从主机级别或主机本身访问其他容器化资源。

黑客在获取到容器的反弹 shell 后，会想方设法从容器 shell 逃逸到宿主机。他们可以通过多种方式逃逸到宿主机环境中，可使用的攻击利用方式包括但不限于：“脏牛”等主机内核漏洞、不安全的容器配置（特权容器、不安全挂载）、“Docker runC”等编排组件漏洞，如图 7-10 所示。

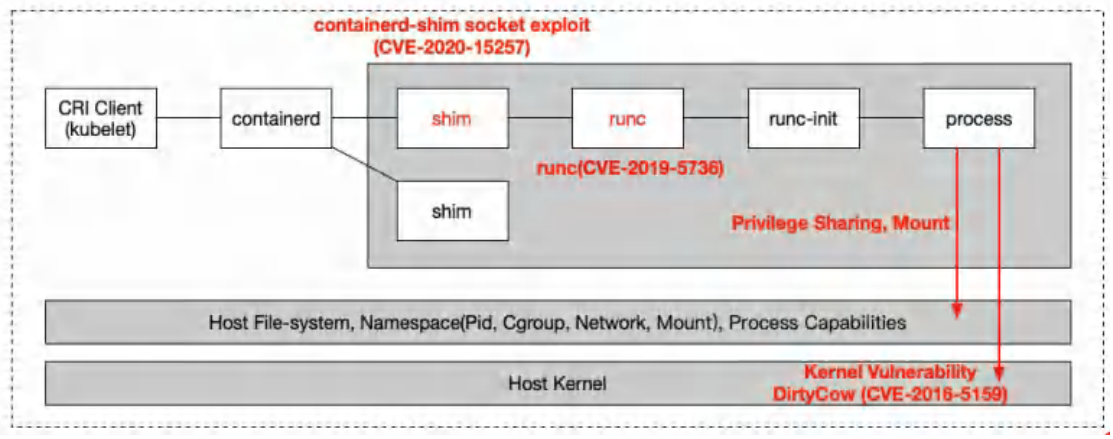


图 7-10 部分容器逃逸漏洞的原理图

在容器逃逸成功后，黑客还可为自己提供实现后续目标的机会（例如植入后门、在环境中横向移动或在主机上建立命令和控制通道）。

此处介绍一个利用容器应用漏洞以及容器不安全配置的“容器逃逸”实例，攻击者按以下两个步骤进行容器逃逸：

（1）利用“Log4J RCE”应用漏洞对 WEB 应用进行攻击，获取到容器的远程命令执行权限；

（2）利用“挂载宿主机敏感目录的容器”对宿主机进行“容器逃逸”攻击，获取到宿主机的命令执行权限，如图 7-11 所示。

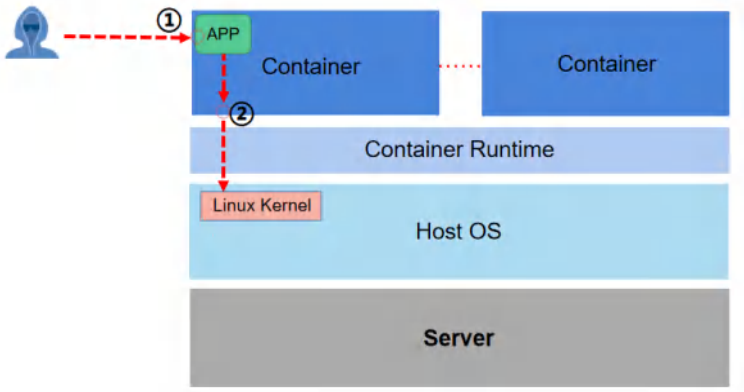


图 7-11 容器逃逸实例

对于步骤（1）的检测，由于应用安全与容器安全有着关联性，对外暴露的 WEB 应用安全对攻击者而言时高价值资产，容器的本质是资源受到隔离和限制的进程，因此在攻击者通过应用漏洞获取容器 Shell 权限的阶段，可基于“容器 WEB 进程监控”进行威胁检测。从容器进程信息中获取数据源，并跟据预设规则模板对实时数据进行分析与检测，可实时地发现容器内 Web 中间件的异常进程行为，其检测效果如图 7-12 所示。

图 7-12 容器 WEB 进程监控检测效果

对于步骤（2）的检测，可采用“基于规则”以及“基于行为模型”的方式。

其中“基于规则”的异常检测方法一直是最为经典的威胁检测实现方法。这种检测方法是直观的，步骤是清晰的，结果是可解释的。在容器环境下，这种检测方法仍然适用于针对已知威胁的检测。

比如，对于已知的云原生漏洞来说，触发漏洞的过程会产生相应的特征，如果要准确检测某漏洞的利用行为，就要获取入侵行为产生的特征，比如在此过程中产生的异常进程、异常文件操作等。从安全研究人员对 CVE-2019-5736、CVE-2020-15257 等多个容器逃逸漏洞的研究可以总结诸如表 7-2 所示的攻防知识图谱。

监控	数据源	事件查看	IoC/IoA	阶段
进程监控	进程访问	进程名称 子进程名称	异常命令行	执行、探测
文件监控	文件写入	文件操作	异常文件操作行为	执行、探测

表 7-2 “容器逃逸”攻防知识图谱

基于实验分析，从它所反馈的特征来看，可以基于对容器的目录开启“文件完整性监控”或“系统进程监控”进行威胁检测。在该攻击过程中，如果在容器内出现新增文件或者宿主机出现可疑的进程，则该 K8s 环境可能正面临威胁。

例如，容器安全产品实现对攻击者“利用用户的‘不安全挂载’进行容器逃逸”的攻击利用方式作出检测，如图 7-13 所示。

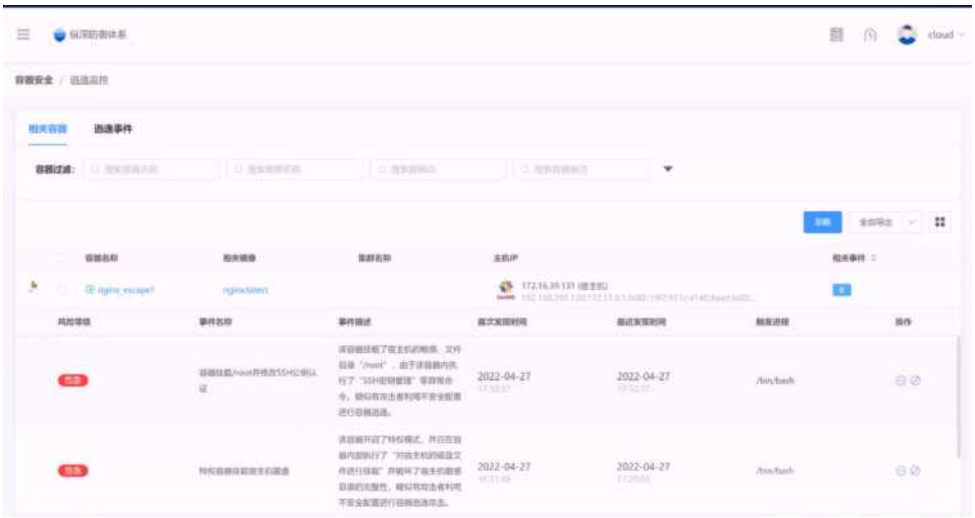


图 7-13 检出容器异常命令

基于规则的检测方式虽然能够快速、精确地检测出已知威胁，但往往无法针对未知威胁做出检测。此时，则可借助构建行为模型的方法进行检测，通过白名单自学习的方式，可以有效检测出因未知威胁产生的进程、文件、网络等方面的异常行为，可有效检出 0day 攻击。图 7-14 展示了容器行为模型的训练过程，图 7-15 为相应的检出效果。

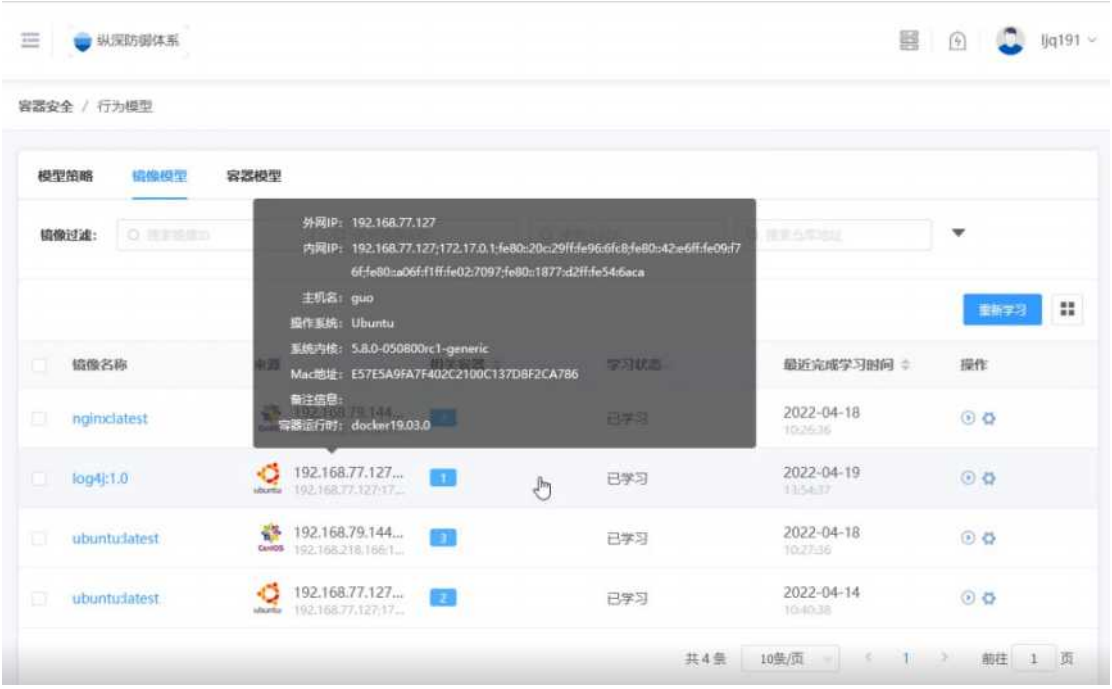


图 7-14 容器行为模型训练过程

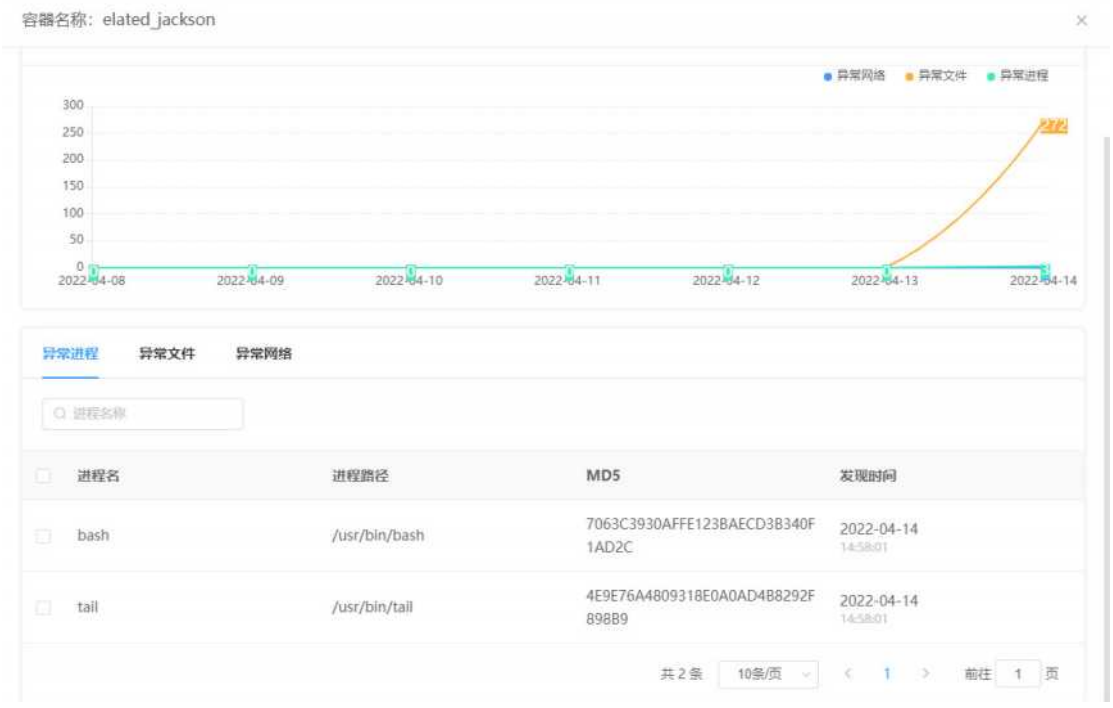


图 7-15 容器行为模型训练过程

此处介绍另一个攻击场景，攻击者首先通过 Webshell 获取容器权限，并在容器中释放了挖矿病毒，企图牟利，如图 7-16 所示。

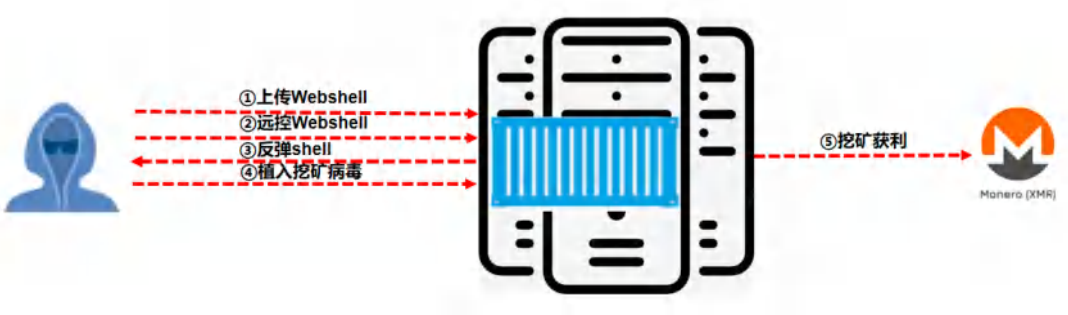


图 7-16 容器环境挖矿案例

面对这一攻击场景，云甲支持基于“静态扫描”和“动态实时防护”对进行 Webshell 以及挖矿木马等恶意代码进行查杀，并对它们进行信任、隔离、下载、删除等，图 7-17、图 7-18 分别展示了 Webshell 和挖矿木马的查杀效果。

图 7-17 容器 Webshell 查杀

图 7-18 容器病毒查杀

（四）主机安全

横向渗透，就是在已经攻占部分内网主机的前提下，利用既有的资源尝试获取更多的凭据、更高的权限，进而达到控制整个内网、获取最高权限，甚至发动 APT 的目的。

在横向渗透中，最先得到的主机，以及之后新得到的主机，会成为突破口、跳板。如同一个不断扩大的圆形，获得的主机越多，圆能触及之处越大，让其周遭的“横向”部分由未知成为已知，如图 7-19 所示。

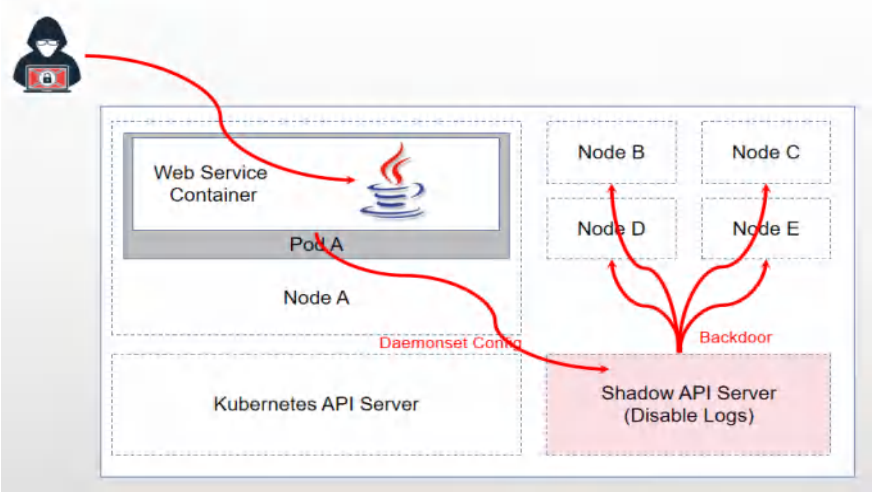


图 7-19 K8s 后渗透模式图

在一定程度上，主机安全与容器安全之间的联系是不可割裂的。主机安全影响着容器安全，容器安全影响着主机的安全。在云原生的环境下，在发生容器逃逸以后，K8s 可能会成为黑客实施后渗透攻击的工具。黑客会在保证自己安全的前提下将已攻占的主机最大化地利用。根据对利用方式的分类，往往可以分为数据挖掘、后门和攻击三大类。而 K8s 环境的存在使得复杂的内网环境多了一个攻击面。例如，若逃逸的宿主机是 K8s 的 Master 节点，则黑客还可利用 K8s 的平台资源进行横向渗透。

针对主机系统服务或应用架构的多样性，攻击者可以对其实现“无文件落地”攻击。例如，攻击者可以利用公网服务器上的恶意代码对靶机进行远程代码执行漏洞利用，通过“远程代码执行漏洞 + 系统白名单程序 + 无文件落地组合”攻击实现更加隐蔽的渗透。对此，相应的威胁检测方法也需不断提升，做层层剖析。对于场景“主机中的 Java 中间件应用漏洞被利用，或已被植入 Java 内存木马，且有调用系统白名单进程”，则在主机上会启动相对应的异常进程、子进程以及异常的命令行，对此可总结出如表 7-3 所示的攻防实战图谱。

监控	数据源	事件查看	IoC/IoA	阶段
进程提权	进程访问	进程名称 子进程名称	异常命令行 子进程权限	执行攻击
内存木马	JVM	Class 类加载	恶意加载类	持久化

表 7-3 WEB 中间件攻防知识图谱

基于以上所反馈的行为特征，可以对主机开启“进程行为监控”和“内存木马”，识别主机上的“进程”行为和“Java 内存”行为。通过监控得到关键的“异常进程”和“恶意加载类”解析所命中的容器 ATT&CK 中对应的技战术和子技战术，从而能确定所遭受的安全威胁。

例如，主机安全产品的 ATT&CK 视图威胁分析效果如图 7-20 所示。

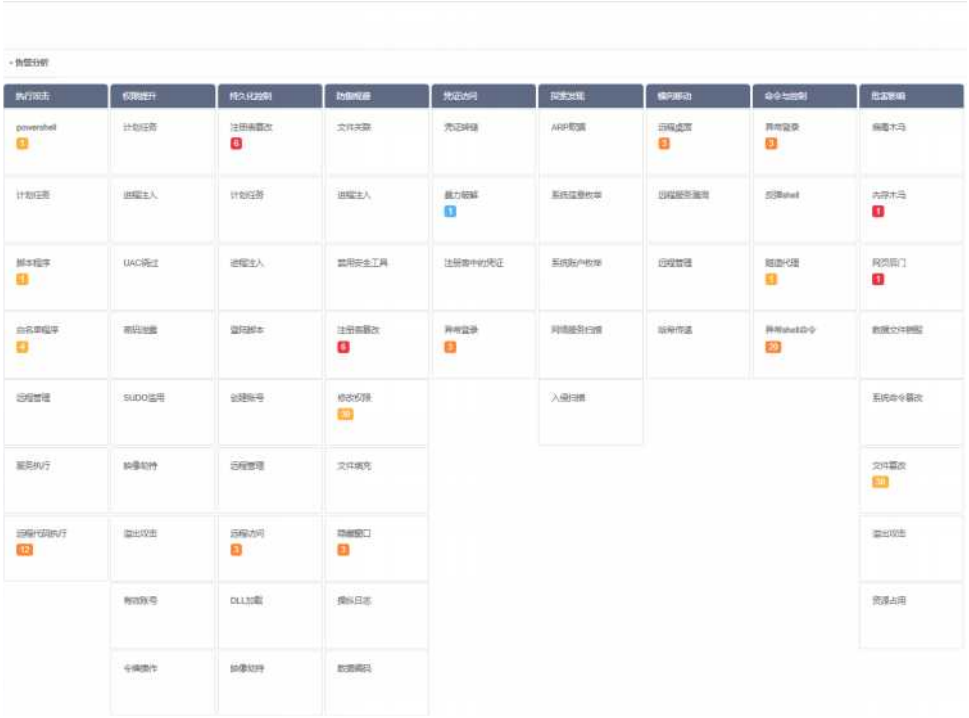


图 7-20 主机 ATT&CK 视图威胁分析

通过获取 JVM 运行时的字节流，并提取其中 class 字节码包含的攻击特征，并对 Filter、Servlet、Agent 等多类 Java 内存木马进行检测的效果如图 7-21 所示。

图 7-21 内存马检测效果

八、云原生安全实践与技术展望

（一）云原生安全实践

作为致力于提供云安全领域相关产品、服务及解决方案的云安全厂商，这里以我们为例，介绍其在产品和解决方案方面的一些实践，抛砖引玉。

安全狗容器安全产品云甲采用主机安全 Agent 和安全容器相结合的技术，既能做到对容器的全面保护又能灵活地跟容器编排体系相结合。在整个容器的安全生命周期中，采用自动检测、自动分析、自动处理的方式来保障容器在构建、部署和运行整个生命周期的安全。在容器镜像、容器编排、容器运行时几个方面，通过扫描、检测、防篡改等手段进行风险和发现；在容器内应用安全方面，使用智能检测、机器学习与威胁预测等先进的方法来确保容器及容器内应用安全，如图 8-1 所示。

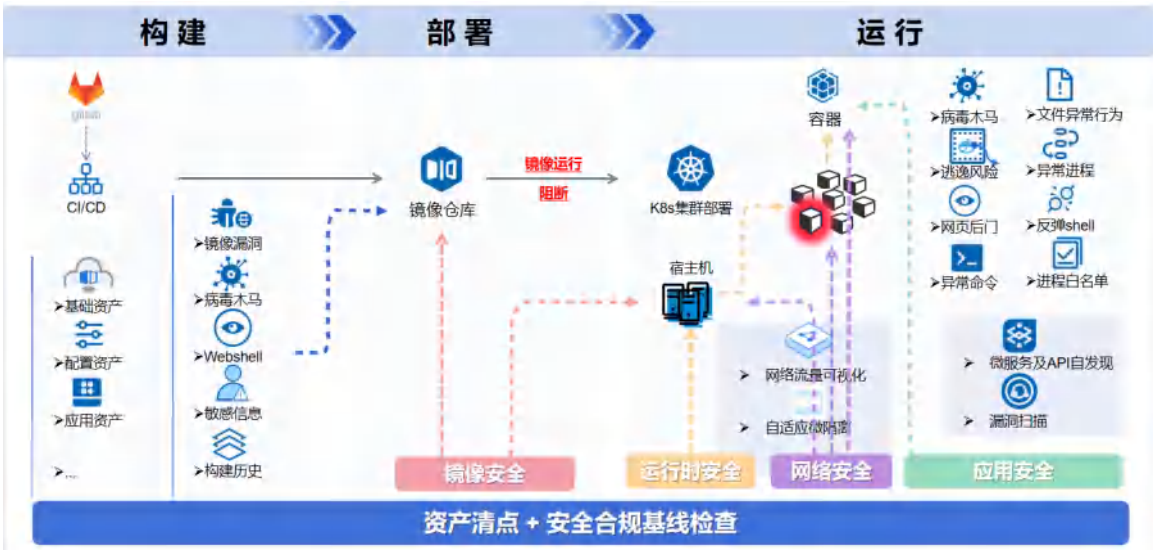


图 8-1 安全狗容器安全产品“云甲”平台架构图

1．构建阶段

在构建阶段，云甲提供镜像、设施基础检查。针对镜像检测提供镜像在构建中构建后镜像检查服务，针对在 Jenkins 等工具构建生成的镜像进行镜像风险检测，通过镜像准入控制，把控镜像来源，确保安全；提供在仓库镜像的镜像风险检查，主机中的镜像风险检查。包括但不限于漏洞风险、软件包、证书信息合规、敏感密钥信息、可疑历史操作、病毒木马文件等。

2．部署阶段

在部署阶段，云甲提供镜像阻断控制、仓库访问控制、添加基础可信镜像等安全能力。

镜像阻断控制：采用自定义镜像阻断规则，对存在 root 用户启动、病毒木马、网页后门、特定漏洞或非信任镜像等规则下的镜像阻断其运行。

仓库访问控制：支持通过身份认证、访问权限控制，避免用户提权访问其他用户的镜像资源。

添加基础可信镜像：通过支持添加基础镜像、可信镜像，在运行、管理镜像中提升运行镜像安全，减少维护时间。

3．运行阶段

在运行阶段，云甲提供运行时安全、容器网络安全、云原生应用安全、基础设施检查。

运行时安全：提供行为基准检测，通过建立容器正常运行时的行为基准模型，圈定正常行为范围，将运行时行为与基准模型进行实时监控比对，从而发现模型外的异常行为，进一步发现未知漏洞攻击等行为。

提供异常风险监控，包括但不限于逃逸风险、反弹连接、进程行为监控、病毒木马、网页后门、执行 ssh 命令、非法挂载设备等；提供风险事件阻止隔离，包括隔离 POD、停止容器、阻止程序、隔离文件等，并根据严重性发出不同等级的告警通知。

容器网络安全：支持集群内 POD、容器、服务等资源访问关系、流量关系拓扑。帮助用户在容器运行阶段业务流量可视化，制定策略；支持容器微隔离，基于自然语言模型进行策略管理，根据端口协议添加策略，添加多种级别的策略规则，并且支持策略验证与策略回滚。

云原生应用安全：支持自动检测发现 K8s 集群中的微服务并做可视化；支持对发现的微服务进行漏洞风险扫描。

基础设施检查：提供主机环境及编排平台环境的漏洞检测，检测运行组件风险；提供针对主机容器 Docker、K8S 的基线合规检查，减少由于配置不当、挂载不当等带来的风险。

（二）云原生未来展望

安全狗云原生安全威胁分析报告已到尾声，但云原生产业在中，云原生安全刚刚拉开序幕，云原生安全将开启多元主导、深度融合、轻量设计的趋势。

1．云原生未来发展趋势

（1）云原生应用将成为新基建的重要抓手

2020 年 3 月 4 日，中共中央政治局常务委员会会议强调，要加快推进国家规划已明确的重大工程和基础设施建设，其中要加快 5G 网络、数据中心等新型基础设施建设进度；随后，国家发改委也进一步明确了“新基建”的范围，包括以云计算等为代表的新技术基础设施和以数据中心为代表的算力基础设施。

落实国家战略部署，落实新基建是当务之急，从目前的发展现状来看，传统行业企业基本实现了业务的云迁移，其应用系统的弹性、自动化数据管理能力得到一定的改善，但与类似“双 11”、“618“等互联网支撑系统相比，仍然存在明显差距，其原因在于业务应用软件云原生化改造和创新尚未跟上。可以说，云原生应用已经成为新基建的重要技术抓手。

（2）运维下沉，服务网格将成为主流，Serverless 逐步推广

云计算的一个发展方向就是运维下沉，将和业务无关的管理功能和运维工作尽量下沉到基础设施中，应用可以聚焦在业务能力的开发和运营。这个趋势演化的过程，影响了云计算的发展方向。从一开始的虚拟化，到 IaaS，到 PaaS 都是将应用系统的部分运维职责交给平台运维的过程，如图 8-2 所示。

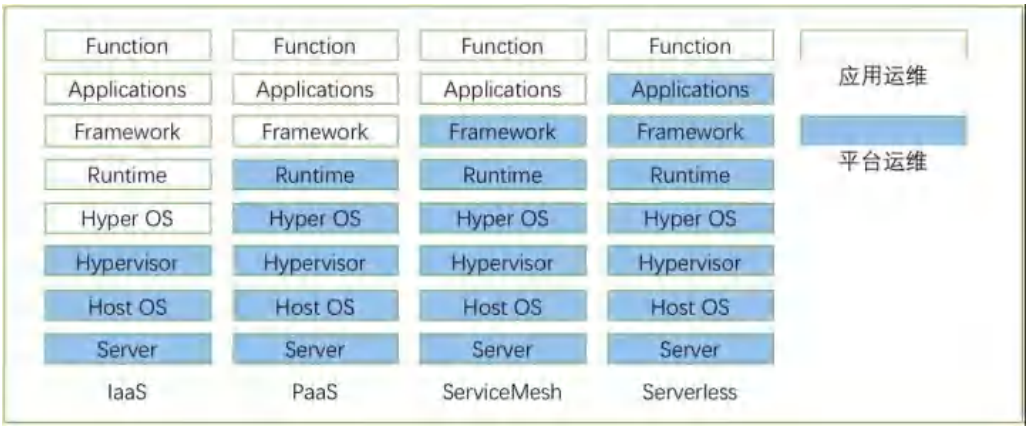


图 8-2 各技术架构的应用运维、平台运维分层

PaaS 为云应用提供了运行容器，解决了应用部署的问题和运行时管理的问题，但是应用仍然有大量的运维工作，特别是对于微服务应用，需要解决诸多的问题，如服务的发布和感知，多实例应用的负载均衡，服务故障检测和隔离，已经应用灰度发布的策略等。这些在 PaaS 层面是无法解决的，通常是由开发框架解决，就是我们前面提到的微服务治理框架。

因为业务功能的提供才是业务开发团队的价值体现，业务开发团队应该聚焦于业务功能的实现，非功能的需求应该交给平台处理。基于这个诉求服务网格出现了，微服务治理的问题可以有服务网格统一运维管理，业务应用只需关注业务能力的实现。

服务网格出现后，业务应用本身的生命周期还是需要应用来运维保障。这就逐步演化出了 Serverless 的概念，Serverless 并非没有 Server，而是对于开发团队来说根本不在意 Server 是什么样的。开发团队只需要提交业务代码，就可以得到需要的运行实例，对应用开发团队来说，Server 是不存在的。

从目前业界的技术趋势来看，ServiceMesh 的概念已经被大部分的大型云上企业接受，ServiceMesh 被诟病的性能问题也在被逐步解决中，可以预测今年将有更多的微服务应用采用这一基础能力。Serverless 目前发展还比较初期，包括了全托管的服务和 FaaS（函数即服务），全托管服务在公有云已经逐步成熟，随着混合云的普及，全托管服务会逐步发展。FaaS 由于涉及开发模式的转变，目前要取代现有的开发模式还需要时日，不过有些合适的应用场景应该会有越来越多的应用。

（3）软硬结合，解决虚拟化性能问题的利器

随着云计算的发展，虚拟化技术越来越多的被使用，从计算虚拟化到存储虚拟化到网络虚拟化。虚拟化技术带来了很多的好处，虚拟化是基础设施服务化的基础，通过虚拟化，可以实现基础设施即代码，大大提升了资源的可管理性和自动化程度。但是虚拟化带来了另外一个问题，就是性能的损耗和软件进程之间的相互影响问题。

对于性能损耗，会导致需要的资源比实际业务耗费的资源更多，提升了服务器资源的成本；进程之间的相互影响则

会导致云平台的整体性能问题，网络虚拟化和存储虚拟化都需要通过软件进程的方式，来处理网络流量和 IO。为了实现分布式高可用和减少数据包转发，基础的 SDN，SDS 的进程通常是和应用进程部署在同一套集群上。这就导致了有可能部分的 SDN 和 SDS 的管理进程所在服务由于各种原因，CPU 或是内存占用过大，导致无法及时处理网络和 IO 请求，导致云平台整体性能下降。

为了解决这两个问题，目前一个解决思路就是软硬结合，讲云平台的管理进程，如调度管理，网络的虚拟交换机，存储的虚拟存储网关从操作系统进程中剥离出来，让这些进程跑在专门设计的服务器板卡上，这些板卡专门设计的，通常含有定制化的芯片（FPGA），可以进行编程，从而可以保持虚拟化话的优势的同时，使的管理进程和业务进程隔离，避免相互影响；同时由于通过定制芯片（如 FPGA）来处理，性能会有很大提升，大大降低了虚拟化的损耗。

2．云原生安全未来发展趋势

（1）云工作负载安全成为“新基建”安全的基础

没有云工作负载安全就没有云安全，保护云工作负载安全将成为今后云安全防护的关键。随着新基建的快速推进，云计算和云原生技术正在影响各行各业的 IT 基础设施、平台和应用系统，并随着渗透到工业互联网、5G、车联网、边缘计算等新型基础设施中，云工作负载安全防护将进一步扩展应用到工业网、车联网、IT 和 OT 融合的 5G 网络、边缘计算等新型场景中，为设计、部署和运营这些云工作负载基础设施和应用系统企事业单位提供云原生技术相关的风险、威胁和安全防护手段进行融合的最佳实践。

（2）安全产品将具有云原生的特性

云原生架构将赋予传统安全弹性、可编排、微隔离的云原生能力，传统安全将与云原生深度融合，构建云原生安全架构，形成对云原生基础设施的防护、检测和响应的能力。

（3）云原生安全落地方案走向敏捷化、精细化

采用容器部署的运行实例具有生命周期短的特点，云原生安全必然要求能够迅速敏捷、及时发现容器的异常行为，并快速响应。随着微服务、无服务的应用，服务粒度越来越细，相应的云原生安全粒度也越来越细，从过去的容器粒度，到目前的函数粒度，未来可能是更精细化的粒度。

（4）托管式安全运营服务将成为主流

云原生安全服务商在云端建立统一的安全运营中心，并将分散到各个用户业务场景数据接入进来，进行统一的安全防护，安全专家基于这些数据进行安全监测和快速处置，大大降低了应对云原生攻击的难度。

（5）云安全建设责任划分更加清晰

目前，在云安全建设当中，对安全责任划分依然比较模糊，租户认为业务系统迁移到云平台之后，所有的安全都由云平台负责。其实不然，网信办早在 2014 年就已经发文（中网办发文 [2014]14 号），明确指出“四不”原则，“安全管理责任不变、安全管理标准不变、数据归属关系不变、敏感信息不出镜”，即云安全建设有明显的责任划分，云平台服务商负责基础设施的安全建设，云租户负责其自己的软件平台、应用平台的安全建设，由双方共同承担虚拟化资源层的安全建设。未来，随着云原生技术的广泛应用，相关标准不断完善，云安全建设的责任划分将更加清晰。

3．云原生攻防对抗未来发展趋势

（1）云原生技术会融合攻防技术

云原生技术既然能够构建敏捷、弹性扩展的应用，那么攻击者也会使用云原生技术来构建自己的武器库，使用云原生技术进行的攻防对抗将成为常态。

（2）黑产集团已经具备自动化攻击手段

2019 年，UNIT42 安全团队发现第一款新型挖矿蠕虫，命名为 Graboid 大地虫，被发现时该新型挖矿蠕虫已传播到 2000 多个不安全的 Docker 主机。该病毒可以定期从 C&C 中提取新脚本，因此不仅仅是传播挖矿病毒，而且可以用于勒索软件或任何恶意软件，可见在云原生技术更新迭代的同时黑产的攻击技术也紧跟其后。

（3）红蓝对抗基于云原生技术增多

红蓝对抗中，利用 ATT&CK 容器矩阵，来覆盖了更多维度的攻防流程和对象，通过初始访问、执行攻击、实现持久防御作战，再到防御绕过、横向移动，最终窃取数据造，来推动整个云生态的安全稳定。

参考资料

[1] 云原生产业联盟 . 云原生架构安全白皮书 [R], 2021.
[2] 刘文懋 , 江国龙 , 浦明 , 阮博男 , 叶晓虎 . 云原生安全 攻防实践与体系构建 [M].1. 机械工业出版社 , 2021.
[3] 绿盟科技 . 云原生安全技术报告（2020）[R], 2021.
[4]CSA. 云原生安全技术规范 [R], 2022.
[5]StackRox.Kubernetes adoption, security, and market trends report 2021[EB/OL].[2022-5-6].https://www.stackrox.com/kubernetes-adoption-and-security-trends-and-market-share-for-containers/.
[6]Sysdig.Sysdig 2021 container security and usage report: Shifting left is not enough[EB/OL].[2022-5-6].https://www.stackrox.com/kubernetes-adoption-and-security-trends-and-market-share-for-containers/.
[7] 阿里云安全 . 国内首个云上容器 ATT&CK 攻防矩阵发布，阿里云助力企业容器化安全落地 [EB/OL].[2022-5-6].https://developer.aliyun.com/article/765449.
[8] 奇安信 CERT.2021 年度漏洞态势观察报告 .[2022-02-28].https://www.secrss.com/articles/39757.
[9]https://attack.mitre.org/matrices/enterprise/containers/