云原生架构安全白皮书

（2021 年）

云原生产业联盟

Cloud Native Industry Alliance, CNIA

2021 年 5 月

版权声明

编写说明

牵头编写单位：

中国信息通信研究院

参与编写单位：

阿里云计算有限公司、华为技术有限公司、腾讯云计算（北京）有限公司、北京百度网讯科技有限公司、北京升鑫网络科技有限公司（青藤云）、北京小佑科技有限公司、北京神州绿盟信息安全科技股份有限公司、奇安信科技集团股份有限公司、中国移动信息技术中心、启明星辰信息技术集团股份有限公司、北京云思畅想科技有限公司、杭州安恒信息技术股份有限公司、北京奇虎科技有限公司、苏州博纳讯动软件有限公司、烽火通信科技股份有限公司、浪潮电子信息产业股份有限公司、深信服科技股份有限公司

编写组成员：

中国信息通信研究院：郑剑锋、刘如明、陈屹力、杜岚、周丹颖、闫丹、郑立

阿里云计算有限公司：张大江、汪圣平、匡大虎、朱松

华为技术有限公司：张宇、卢宏旺、刘亚东、莫若

腾讯云计算（北京）有限公司：乐元、李滨、张祖优、姬生利

北京百度网讯科技有限公司：罗启汉、房双德

北京小佑科技有限公司：袁曙光、刘斌

北京升鑫网络科技有限公司（青藤云）：胡俊、李漫

北京神州绿盟信息安全科技股份有限公司：江国龙

奇安信科技集团股份有限公司：王亮

中国移动信息技术中心：王庆栋

前　言

云计算是信息技术发展和服务模式创新的集中体现，是信息化发展的重要变革和必然趋势。随着“新基建”加速布局，以及企业数字化转型的逐步深入，如何深化用云进一步提升云计算使用效能成为现阶段云计算发展的重点。云原生以其高效稳定、快速响应的特点极大地释放了云计算效能，成为企业数字业务应用创新的原动力，有效推动了国民经济的高质量发展。

云原生技术架构充分利用了云计算弹性、敏捷、资源池和服务化特性，在改变云端应用的设计、开发、部署和运行模式的同时，也带来了新的安全需求和挑战。传统基于边界的防护模型已不能完全满足云原生的安全需求，需要设计全新的云原生安全防护模式，基于动态工作负载，实现云原生技术架构和规模应用的全面防护。

本白皮书将从容器化部署、微服务应用模式、研发运营一体化等云原生特有的技术架构和应用模式出发，全面剖析云原生系统面临的安全威胁，针对性提出云原生安全边界、原则和防护模型，系统阐述涵盖基础设施安全、云原生计算环境安全、云原生应用安全、云原生研发运营安全、云原生数据安全及安全管理的云原生安全防护体系，列举了国内外云原生安全典型产品与方案，最后对云原生安全的发展趋势进行了展望。

目 录

图 目 录

表 目 录

一、技术变革、市场推动，呈现云原生安全新需求

（一）云原生技术在生产环境采纳率急速升高

云原生的理念经过几年发展，不断丰富、落地、实践，云原生已经渡过了概念普及阶段，进入了快速发展期。云原生技术以其高效稳定、快速响应的特点驱动引领企业的业务发展，成为企业数字业务应用创新的原动力。过去几年中，以容器、微服务、DevOps、Serverless 为代表的云原生技术正在被广泛采纳，2020 年 43.9%的国内用户已在生产环境中采纳容器技术，超过七成的国内用户已经或计划使用微服务架构进行业务开发部署等[1]，这使得用户对云原生技术的认知和使用进入新的阶段，技术生态也在快速的更迭。

（二）颠覆性技术架构变革带来全新安全隐患

云原生技术架构充分利用了云计算弹性、敏捷、资源池和服务化特性，在改变云端应用的设计、开发、部署和运行模式的同时，也带来了新的安全要求和挑战。以容器、Serverless 为载体的云原生应用实例极大地缩短了应用生命周期；微服务化拆分带来应用间交互式端口的指数级增长以及组件间组合设计复杂度的陡升；多服务实例共享操作系统带来了单个漏洞的集群化扩散；独立的研发运营流程增加了软件全生命周期各个环节的潜在风险。云原生的特有属性带来了架构

[1] 数据来源：中国信息通信研究院《中国云原生用户调查报告 2020》

防护、访问控制、供应链、研发运营等领域全新的安全隐患和安全防护需求。

服务实例应用周期变短增加监控和溯源难度。以容器、Serverless 为载体的云原生应用实例的生命周期极大缩短，容器秒级启动或消失的特性以及持续频繁的动态变化，增加了安全监控和保护的难度，准确捕捉容器间的网络流量和异常行为成为新挑战。

组件爆发式增长为应用防护能力提出更高要求。微服务将单体架构拆分成多个服务，应用间的交互端口呈指数级增长、攻击面大幅增加，相较于单体应用架构集中在单个端口防护的简单易行，微服务化应用在端口防护、访问权限、授权机制等方面难度陡增。

容器共享操作系统的进程级隔离环境增加逃逸风险。传统软件架构下，应用之间通过物理机或虚拟机进行隔离，可以将安全事件的影响限制在可控的范围内。在云原生环境下，多个服务实例共享操作系统，一个存在漏洞服务被攻陷可能会导致运行在同主机上其他服务受到影响，逃逸风险大大提高。

独立研发运营对软件流转的全链条安全提出新要求。每个微服务应用都涉及相对独立的开发和测试流程，并通过 CI/CD（持续集成/持续交付）流水线将应用部署运行到开发测试和生产环境中。在业务应用全生命周期中，为各个环节的自动化安全防护能力提出了全新要求，不仅要避免各个环节的潜在风险，而且要提高应用的安全交付效率。

（三）传统的边界防护模型难以应对云原生安全风险

谈及云原生安全，不少人还停留在传统安全意识和观念，关注 Web 攻防和系统攻防，关注密码暴力破解和反弹 Shell。然而，安全总是具有“短板效应”，有时，一个简单的端口暴露、未授权访问没及时处理就为攻击者提供了不费吹灰之力长驱直入的机会。此外，云原生技术架构带来的风险，在未来数年内，会成为攻击者关注和利用的重点，进而发动针对云原生系统的攻击。

传统基于边界的防护模型已不能完全满足云原生的安全需求，云原生关注快速开发和部署，这种特性要求进行防护模式的转变，从基于边界的防护模式迁移到更接近基于资源属性和元数据的动态工作负载的防护模式，从而有效识别并保护工作负载，以满足云原生技术架构的独特属性和应用程序的规模需求，同时适应不断变化的新型安全风险。

安全防护模型的转变要求在应用程序生命周期中采用更高的自动化程度，并通过架构设计（例如零信任架构）来确保安全方案实施落地；在云原生系统建设初期，需要进行安全左移设计，将安全投资更多地放到开发安全，包括安全编码、供应链（软件库、开源软件等）安全、镜像及镜像仓库安全等。

回顾安全发展史，安全始终是跟随 IT 基础设施和业务来发展的，即安全跟随其服务的对象而演进。进入云原生时代，物理安全边界逐

渐模糊并变成了无处不在的云原生安全体系，从外到内，无论可视化、运维和安全管理，还是南北向和东西向网络、容器基础架构、微服务应用模式、身份安全、数据安全等，都给安全市场带来了更丰富的产品维度，衍生出更多元的发展机遇。

二、深入架构、应用驱动，剖析云原生安全新型风险

云原生架构的安全风险包含云原生基础设施自身的安全风险，以及上层应用云原生化改造后新增和扩大的安全风险。云原生基础设施主要包括云原生计算环境（容器、镜像及镜像仓库、网络、编排系统等）、DevOps 工具链；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。本章将重点从云原生计算环境、DevOps、微服务、Serverless、API 这几个具体的方面展开分析云原生架构新增和扩大的安全风险。

（一）容器化部署成为云原生计算环境风险输入源

表 1 云原生计算环境主要安全风险

风险类型	风险引入途径
云原生网络安全风险	访问控制粒度过粗
	网络分离管控不合理
云原生编排及组件安全风险	编排工具自身漏洞
	编排组件不安全配置
	不同安全级容器混合部署
	编排组件资源使用不设限
	编排节点访问控制策略配置不当
镜像安全风险	镜像含软件漏洞

	镜像配置缺陷
	镜像来源不可信
镜像仓库安全风险	镜像仓库自身漏洞及管理问题
	镜像获取通道不安全
容器逃逸攻击	容器危险配置
	容器危险挂载
	相关程序漏洞
	宿主机内核漏洞
	安全容器逃逸风险

1. 网络的细粒度划分增加了访问控制和分离管控难度

访问控制粒度过粗引入了权限放大的风险导致越权攻击。云原生环境下，服务实现了细粒度拆分，业务依赖关系复杂。如果容器网络层不能依据业务关系实现细粒度的访问控制策略，就会带来网络权限放大的风险，例如：无需被外网访问的业务被默认设置了外网访问权限，容器网络可无限制访问宿主节点的下层网络等。攻击者将利用这些漏洞，获取权限外甚至核心系统的访问控制权限，最终实现越权甚至提权攻击。

网络分离管控不合理增加了横向攻击的风险导致威胁扩展。云原生网络既有南北向流量，也有东西向流量，服务之间的数据流动极其频繁；并且在云原生环境下，多个服务实例共享操作系统，一个存在漏洞服务被攻陷将会导致运行在同一主机上的其他服务受到影响。如果各服务单元对应的容器组之间、容器网络与物理网络之间没有做好网络权限的分离管控，外网攻击者就能从高风险实例逃逸，伴随东西向流量在集群网络内部的实例之间进行横向攻击，致使威胁在集群内

大范围扩散。另外，有些云原生平台采用 underlay 模式的网络插件，使得容器 IP 与节点 IP 共享同一网络平面，在业务 IP 暴露给最终用户的同时，管理控制台会面临被入侵的风险。

2.云原生编排组件存在漏洞及管控风险增加入侵概率

编排工具自身漏洞导致非法提权和逃逸攻击。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，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。**逃逸攻击，**是指攻击者通过劫持容器内某种权限下的命令执行能力，借助一些手段进一步获得该容器所在宿主机上某种权限下的命令执行能力。以 CVE-2017-1002101 漏洞为例，这是一个 Kubernetes 的文件系统逃逸漏洞，允许攻击者使用 subPath 卷挂载来访问卷空间外的文件或目录，这个漏洞本质上是 Linux 符号链接特性与 Kubernetes 自身代码逻辑两部分结合的产物。符号链接引起的问题并不新鲜，这里它与虚拟化隔离技术产生了逃逸问题，以前还曾有过在传统主机安全领域与 suID 概念碰撞产生的权限提升等问题。

编排组件不安全配置引起账户管理问题导致系统入侵。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，而不安全配置引起的风险不容小觑，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。

不同安全级容器混合部署导致高安全级容器面临入侵风险。编排工具侧重工作负载规模和密度的管理。默认情况下，编排工具可以将不同安全级别的工作负载部署在同一主机上，导致高安全级工作负载面临入侵风险。例如，将运行面向公众 Web 服务器的容器与运行处理敏感财务数据的容器置于同一主机上，在 Web 服务器有严重漏洞的情况下，就会显著提高处理敏感财务数据的容器面临的入侵风险。

编排组件资源使用不设限加大资源耗尽风险导致拒绝服务攻击。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。典型漏洞包括 CVE-2019-11253、CVE-2019-9512、CVE-2019-9514 等。

编排节点访问策略配置不当导致未授权主机非法访问。维护编排环境中各节点之间的信任关系时需要充分考虑依赖和调用关系，编排工具访问策略配置不当可能让编排工具和其上运行的容器技术组件面临极大风险，例如：主机之间通信未加密或未认证，导致未经授权的主机加入集群并运行容器；用于认证的密钥在所有节点中共享编排工具，导致集群中单个主机被入侵后扩散至整个集群被入侵等。

3.镜像构建部署过程不规范引入自身风险

镜像更新不及时导致软件漏洞。在传统模式中，部署的软件在其运行的主机上“现场”更新；与此不同，容器则必须在上游的镜像中进行更新，然后重新部署。因此，容器化环境的常见风险之一就是用于创建容器的镜像版本存在漏洞，从而导致所部署的容器存在漏洞。

镜像配置缺陷导致应用风险增加。镜像配置不当可能会让系统面临攻击危险，例如，镜像未使用特定用户账号进行配置导致运行时拥有的权限过高；镜像含 SSH 守护进程导致容器面临不必要的网络风险等。

镜像来源不可信注入恶意镜像。镜像及容器技术一个主要的特点就是方便移植和部署，云原生用户可以将符合 OCI 标准的第三方镜像轻松部署到自己的生产环境中。因此，攻击者可将含有恶意程序的镜像上传至公共镜像库，诱导用户下载并在生产环境中部署运行，从而实现其攻击目的。根据目的的不同，常见的恶意镜像有三种类型：一

是恶意挖矿镜像，欺骗受害者在机器上部署容器从而获得经济收益；二是恶意后门镜像，受害者在机器上部署容器后，攻击者收到容器反弹过来的 shell，实现对容器的控制；三是恶意 exploit 镜像，在受害者部署容器后尝试寻找宿主机上的各种漏洞实现容器逃逸，实现对受害者机器的控制。

4.镜像仓库模式增加云原生软件供应链风险来源

镜像仓库漏洞和管理问题带来自身安全风险。镜像仓库安全风险主要涉及仓库账号及其权限的安全管理、镜像存储备份、传输加密、仓库访问记录与审计等，这些方面如果存在加固或配置策略不足的问题，都可能导致镜像仓库面临镜像泄露、篡改、破坏等风险。例如，垂直越权漏洞，因注册模块对参数校验不严格，导致攻击者可以通过注册管理员账号来接管 Harbor 镜像仓库，从而写入恶意镜像。实际使用中，用户往往会将镜像仓库作为有效且获得批准的软件源，因此，镜像仓库遭到入侵将极大增加下游容器和主机的入侵风险。

镜像获取通道不安全引入中间人攻击。保证容器镜像从镜像仓库到用户端的完整性是镜像仓库面临的一个重要安全问题。如果用户以明文形式拉取镜像，在与镜像仓库交互的过程中极易遭遇中间人攻击，导致拉取的镜像在传输过程中被篡改或被冒名发布恶意镜像，从而造成镜像仓库和用户双方的安全风险。

5.容器特性增加云原生运行时逃逸风险和威胁范围

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，特别是在多业务系统、多租户环境下，风险更高，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，容器逃逸的类型可以分为以下三类:

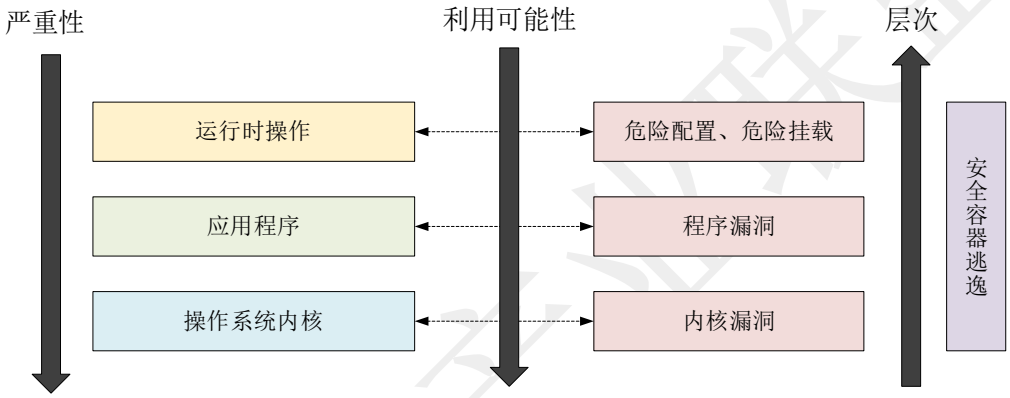

图 1 不同层次的容器逃逸问题

上图可以进一步展开为:

1) 危险配置导致的容器逃逸;

2) 危险挂载导致的容器逃逸;

3) 相关程序漏洞导致的容器逃逸;

4) 内核漏洞导致的容器逃逸;

5) 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。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。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。本节涉及到的相关漏洞均分布在容器及容器集群环境的程序组件中。例如 CVE-2019-5736 正是这样一个存在于 runC 的容器逃逸漏洞。

宿主机内核漏洞导致的容器逃逸。对内核漏洞的利用往往都是从用户空间非法进入内核空间开始，到内核空间赋予当前或其他进程高权限后回到用户空间结束。Linux 内核漏洞危害极大、影响范围极广，是各种攻防话题下不可忽视的一环。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，典型的就是脏牛漏洞（Dirty COW CVE-2016-5195）。

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。

（二）微服务细粒度切分增加云原生规模化应用风险

表 2 微服务主要安全风险

风险类型	风险引入途径
攻击端口和攻击面增加	微服务入口点增加
访问控制风险	访问控制策略复杂
治理框架风险	微服务治理框架漏洞

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，在这个入口点去建立一组 Filter 或者 Interceptor，就可以控制所有的风险。微服务场景

下，业务逻辑不是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，导致防范的攻击面比原来大得多，风险也会高得多。

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，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，这些漏洞有的社区进行了及时修复，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。

（三）Serverless 灵活性带来模型和平台管控风险

表 3 Serverless 主要安全风险

风险类型	风险引入途径
应用程序固有风险	应用程序漏洞、第三方依赖库漏洞、消息明文传递等传统应用程序风险
Serverless 计算模型风险	函数多源输入
	资源权限管控不当
	数据源增加
	环境变量替代配置文件存储
	跨函数调用数据清理不及时
Serverless 平台风险	平台自身漏洞
	平台账户拒绝服务（DoW）
	平台强依赖性

Serverless 是云原生架构下，针对应用程序构建和部署运行的模型，其目标是使开发人员专注于应用程序的构建和部署运行，而无需管理服务器等资源。Serverless 模型包含了开发者开发、部署、运行的应用程序，以及云计算供应商提供的 Serverless 支撑平台。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。

Serverless 模型以及平台的安全风险，主要包括以下几类：

函数多源输入导致供应链安全风险。Serverless 模式下函数调用由事件源触发，这种输入来源的不确定性增加了安全风险。如果函

数未对外界输入数据进行有效的过滤或安全校验，就可能存在 SQL 注入、系统命令执行等类型的攻击风险。

资源权限管控不当带来函数使用风险。Serverless 应用通常会由诸多函数组成，函数间的访问权限、函数与资源的权限映射会非常多，如何高效地进行角色和权限的管控是一个复杂的问题。许多开发者可能暴力地为所有函数配置单一的角色和权限，这一行为将极大地增加函数使用风险。

设计使用不规范引入数据泄露风险。一是数据源增加导致数据攻击面增加。传统应用只是从单一服务器上获取敏感数据，而 Serverless 架构中攻击者可针对各种数据源进行攻击，攻击面更广。**二是环境变量替代配置文件引入泄露风险。**Serverless 应用由众多函数组成，无法像传统应用程序使用单个集中式配置文件存储的方式，因此开发人员多使用环境变量替代，虽然存储更为简单，但使用环境变量本就是一个不安全的行为，一旦攻击者进入运行环境，可以轻易获取环境变量信息，引起信息泄露。**三是跨函数调用数据清理不及时引入数据泄露风险。**Serverless 函数基于触发机制实现函数调用与运行，在无状态函数的运行过程中，倘若函数调用结束后没有及时清除暂存的配置参数、账号信息或其他业务敏感信息，后续使用基础设施资源的函数可能会访问这些数据，存在机密数据泄露的风险。

FaaS 平台自身漏洞带来入侵风险。近年来，私有云 FaaS 平台由于其自身代码含有漏洞进而导致函数遭受攻击的案例越来越多，这些攻击行为主要以攻击者利用平台漏洞进行挖矿、数据窃取为主。例如，2018 年 6 月 Apache OpenWhisk 平台被曝出的 CVE-2018-11756 漏洞。

平台设计机制缺陷引入账户拒绝服务风险。针对平台账户的攻击主要为 DoW 攻击（拒绝钱包攻击），这是拒绝服务攻击的变种，其目的是耗尽账户的账单金额。Serverless 平台通常支持自动化的弹性扩展，开发人员按函数调用次数付费，这一特性产生的费用通常没有特定限制。如果攻击者掌握了事件触发器，在未受保护的情况下，函数调用可能会极速扩展，随之产生的费用也将爆发式增长，最终会导致用户账户受到重大损失。例如，2018 年 2 月 NodeJS “aws-lambda-multipart-parser”库被曝出的 ReDoS 漏洞（CVE-2018-7560）。

缺乏 Serverless 专有安全解决方案导致风险应对能力不足。广泛使用的安全检测体系如 WAF、IPS/IDS、DAST、SAST、日志审计工具等，均是基于云场景中虚机或容器底座、基于 Web 应用或传统应用架构进行设计、开发和应用的；而 Serverless 无论机制、架构还是应用场景均存在一定的特殊性，现阶段成熟有效的安全检测工具与机制无法直接匹配或移植，使得 serverless 服务的使用者和运营者难以有效应对网络中层出不穷的新攻击方法与手段，在方法机制与工具支撑方面存在较大的差距与风险。

（四）DevOps 提升研运流程和安全管理的防范难度

表 4 DevOps 主要安全风险

风险类型	风险引入途径
设计风险	安全理念不够重视程度不足
流程风险	安全团队与安全流程缺失
管理风险	人员权限扩大
工具风险	开源工具和开源组件大量使用

安全理念和重视程度不足带来一系列设计安全问题。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内部泄露、系统资源无防护而在内部开放。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。

安全团队与安全流程缺失增加流程中的风险引入。经典 DevOps 理念中，没有强调安全团队的参与和安全流程的控制，开发测试环境与生产环境的隔离也有悖于 DevOps 宣传的代码提交后部署到生产环境的流程（例如，代码中可能存在明文密钥、使用弱密码算法或不安全的传输协议等），这些都将引入全新风险。

人员权限扩大化导致敏感数据泄露带来管控风险。按照 DevOps 的理念，开发与运维的权限划分会有一部分模糊或权限的扩大化，导致开发人员有可能访问到生产环境里的客户敏感数据，或者运维能访问到研发内部的企业敏感数据，这将导致敏感数据的泄露风险。

开源组件和开源工具的大量使用导致权限混乱和漏洞注入。DevOps 的实践中，多数企业会基于类似 Jenkins 的开源工具来整合工具链，而每个不同工具或组件都可能有各自的账号体系，如果使用超级管理员账号进行对接则存在巨大的安全隐患；丰富的工具或组件也引入了更多安全漏洞需要持续关注和运维。

（五）API 爆发式增长催化分离管控和权限滥用风险

表 5 API 主要安全风险

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

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，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，API 的数量急剧增加、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。Gartner 预测，到 2022 年，API 滥用将成为导致企业 Web 应用数据泄露最频繁的攻击载体。

三、原则引领、架构融合，构建云原生安全体系模型

（一）云原生安全防护范围及责任划分

云原生安全不同于传统安全及云安全，它更关注容器和容器运行时，针对微服务和无服务的应用新形态，也需要重新考虑其安全隐患和防护措施。

云原生安全所保护的对象，是指以容器技术为基础底座，以 DevOps、面向服务等云原生理念进行开发并以微服务等云原生架构构建的业务系统共同组成的信息系统。容器服务、Service Mesh 与 Serverless 等均属于云原生范畴，是以容器技术为核心基础的不同交付/服务模式，不同的服务模式责任模型有所不同，如下图所示。

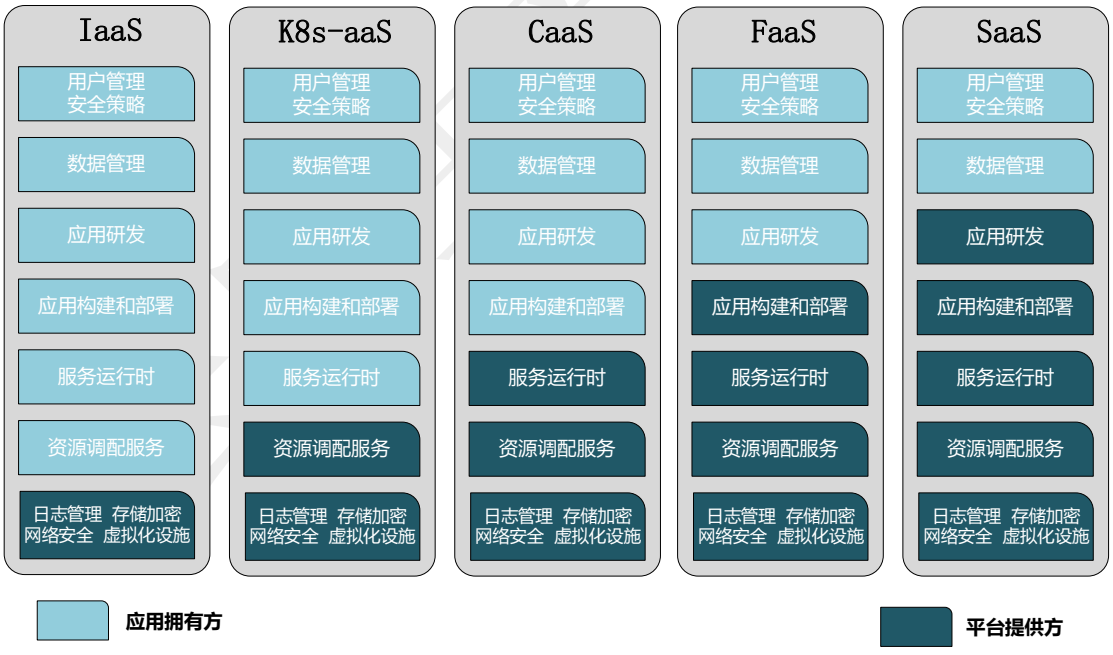

图 2 云原生安全防护责任共担模型

（二）云原生安全遵循的设计原则

1.零信任架构

NIST 在 2020 年 8 月发布了最新的零信任架构，在零信任安全模型中，会假设环境中随时存在攻击者，不能存在任何的隐形信任，必须不断的分析和评估其资产、网络环境、业务功能等的安全风险，然后制定相应的防护措施来缓解这些风险。在零信任中，这些防护措施通常要保证尽可能减少对资源(比如数据、计算资源、应用和服务等）的访问，只允许那些被确定为需要访问的用户和资产进行访问，并且对每个访问的身份和安全态势进行持续的认证和授权。

零信任架构通过细粒度拆分构建微边界的架构模型，并通过执行策略限制消除数据、资产、应用程序和服务的隐式信任，从而减轻了网络威胁横向扩散的可能性。零信任架构的最常见实现方法是依赖于加密概念的。首先要用硬件或者令牌来保护特定的密钥信息，并且能够用安全的方式和平台进行通信。

零信任架构通常由几个部分组成：每个实体都能创建自己的标识，每个实体都能独立地认证其它实体（例如用公钥体系），实体之间的通信是加密且不可篡改的。

最小权限原则也非常重要，甚至被认为是云原生架构中最重要的内容之一，云原生技术栈的所有层面在进行认证授权的设计实现过程中，都需要考虑这个原则。

2.安全左移

在云原生安全建设中，容器实例生命周期短、业务复杂，而且存在与操作系统虚拟化环境中现有的物理或虚拟化的安全设备无法有效工作的情况，此时，增加运行时的安全投入无助于提高整体的安全水平。因而国内外在过去两年提出了安全左移(Shift Left)的思路，即在云原生安全建设初期将安全投资更多地放到开发安全，包括安全编码、供应链（软件库、开源软件）安全、镜像及镜像仓库安全等。这些方面资源大多是白盒的，相应的安全投资相对较少；并且这些资源生命周期较长，如果能保证安全性，攻击者在攻击运行时实例得手后将更难持久化。

3.持续监控和响应

随着云原生时代的来临，业务变的越来越开放和复杂，固定的防御边界已不复存在，而攻击者的手段却越来越多样。Gartner 自适应安全架构指出，大多数企业在安全保护方面会优先集中在拦截防御、以及基于策略的防御上（例如防火墙），以将危险拦截在外。然而防御体系总是难以应对所有威胁，高级定向攻击总能轻而易举地绕过各种防御手段，安全威胁已防不胜防。

攻击者的攻击行为是不间断的，渗透系统的行为不可能被完全拦截，系统应假设自己时刻处于被动攻击中。为解决传统防御手段的被动处境，需要为系统添加强大的实时监控和响应能力，以帮助企业高

效预测风险、精准感知威胁、全面提升响应效率。企业监控应转被动为主动，持续监控尽可能多的云原生环境，如网络活动层、端点层、系统交互层等；同时应建立持续响应的防护机制，对攻击进行迅速分析和处理，并建立数据收集池进行溯源追踪，发现系统中的安全缺陷，逐步提升企业整体的安全防护水平。实现事前快速布局、事中精准响应、事后全面回溯。

4. 工作负载可观测

云原生的最终目标是通过自动化手段，实现敏捷的松耦合系统。因此，云原生安全也要符合这种自动化目标，自动化的安全检测就需要有详细准确的运行状态数据作为支撑，为自动化的云原生安全提供充足的决策依据。工作负载的可观测性完美提供了这样的能力。

云原生架构下大规模的集群和海量灵活的微服务应用为工作负载可观测性提出了全新的需求和挑战。容器化的基础设施使得应用本身变得更快、更轻，主机上应用的部署密度和变化频率较传统环境都有巨大的变化，需要可视化工具来发现和记录主机快速变化的应用行为；应用架构的微服务化，产生了包括服务和中间件在内的众多调用关系，通过可视化清晰地观察到这种调用关系对于应用性能提升、故障定位、安全监测都有重要意义。

工作负载可观测包括资源可观测和应用可观测。**从资源层面来看，**一方面，要包含传统的主机监控指标，例如，对计算、存储网络等主

机资源的监控，以及进程、磁盘 IO、网络流量等系统指标的监控等。另一方面，要涵盖云原生化资源监控，在资源监控上，要实现 CPU、内存等在容器、Pod、Service、Tenant 等不同层的识别和映射关系；在进程监控上，要能够精准识别到容器，甚至细粒度到进程的系统调用、内核功能调用；在网络监控上，除了主机物理网络之外，还要包括 Pod 之间的虚拟化网络，甚至是应用之间的 Mesh 网络流量的观察。**从应用层面来看，**微服务架构下的应用，使得主机上的应用变的异常复杂，可观测监控既包括应用本身的平均延时、应用间的 API 调用链、调用参数等；还包括应用所承载的业务信息，例如业务调用逻辑、参数、订单数量、商品价格等信息。

（三）云原生安全防护模型

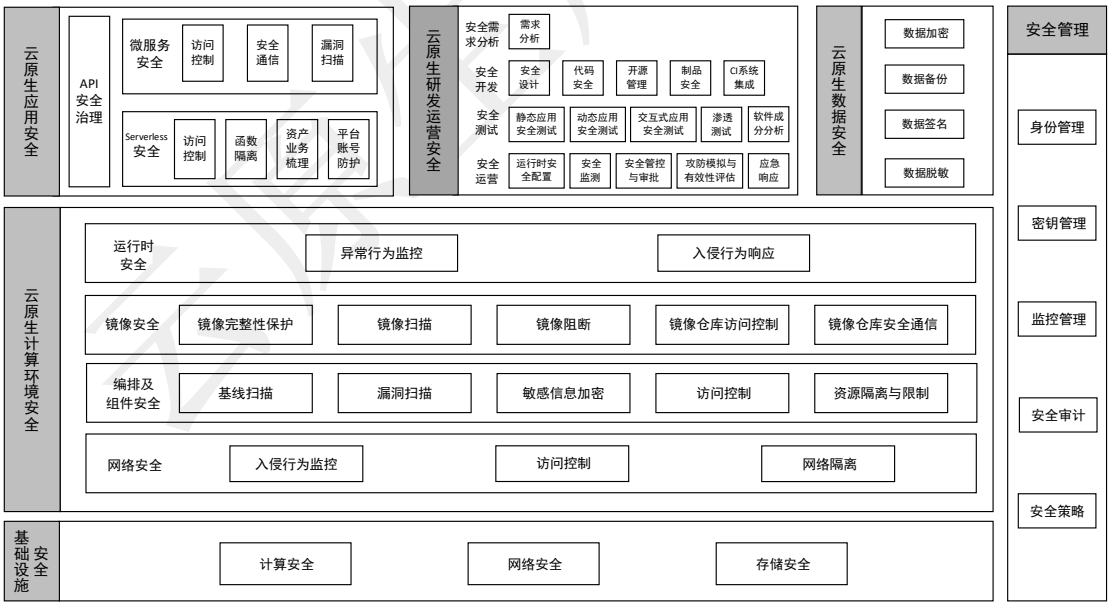

图 3 云原生架构安全防护模型架构图

四、分层构建、全景覆盖，打造云原生安全防护体系

（一）融合应用云安全防护模型夯实基础设施安全

1.计算安全

主机访问控制。首先是主机的账号需要按可动态定义的组织架构进行闭环安全管理的能力，包括明确的服务器负责人、运维、研发角色账号的明确，这些账号的离职转岗权限联动管理等；其次是主机的访问上需要具备双因素甚至多因素的安全认证能力、并具备符合安全追溯与合规要求的操作审计能力。

系统基础安全加固与配置。首先是尽可能确保主机在操作系统镜像的统一化、标准化，这是避免自底向上基础架构安全能力建设、安全风险治理成本指数化增长的基本举措；其次是针对统一标准化的操作系统镜像明确并落地基线安全能力的检测与管控，包括系统基础服务的安全配置、系统核心安全能力（比如 HIDS、安全日志采集访问等）的部署等。

主机入侵检测。在主机层面临入侵的场景，必须要具备主机层面的入侵检测能力，通常在产品上以 HIDS 来落实这一需求，HIDS 一般基于主机层日志、文件、进程、流量等信息来实现多维度互补的入侵检测能力。

2. 网络安全

网络设备安全。在基础网络（物理网络/云主机网络）构建过程中，网络设备（包括虚拟网络设备）自身的安全性是其中需要重点考虑的部分。首先，网络设备的管理需要有独立隔离的网络环境；其次，网络设备的访问同样需要有明确的授权流程/机制、以及安全的双因素/多因素认证机制，以确保访问入口的安全性；另外，设备的资产信息、通用策略配置等需要有统一的平台维护，并确保平台的认证、授权、审计符合安全要求。

网络访问控制。基础网络在边界应具备访问控制能力与策略的流程化管控制。基础网络边界包括物理网络与云主机 VPC 网络边界、以及这些网络与公网的边界，这些边界做到最小化的网络访问权限控制是安全纵深防御上的最基础防线；同时，这类边界网络访问控制权限策略也通常是动态变化的，那么对于每次变动操作都需要有平台化的审核、授权流程，以及操作的审计记录。

网络攻击检测与防御。在基础网络攻防的层面，核心需要考虑的就是 DDoS 防御，DDoS 依旧是网络服务面临的最大威胁之一，无论是针对传统服务还是云原生化的服务都需要有一定的 DDoS 防御机制（流量清洗、流量牵引等）。另外，为了在应用层面形成有效的纵深防御，通常在网络边界上还需要具备应用层防攻击的能力，例如部署 Web 应用防火墙。

3. 存储安全

认证与访问控制。储存产品及系统自身安全性的第一道防线就是访问入口的安全认证、访问空间和访问目录的权限控制，以及读写操作的记录与审计，这类基础安全机制在各类云厂商的产品上都有类似的落地实现。

备份管理与数据恢复。储存产品及系统需要具备充分的备份机制，以确保所有必要的信息和软件能在灾难或介质故障后进行恢复，比如：数据在多个集群及多个数据中心进行复制存储，并依据业务要求设置相应的备份程序、频率、保存周期等；异地存储；备份介质和恢复程序定期进行健康检查和测试等。

数据传输与存储加密。储存产品及系统需要具备满足数据加密传输、储存的功能需求，这是防御数据泄露的最基本安全能力；同时密钥相关的安全管理需要做到位。

（二）关注容器全要素全生命周期防护打造云原生计算环境安全

1. 云原生网络安全

云原生网络环境建议与物理网络或云主机网络进行区分，这样云原生网络层就可以独立为单独的容器网络层，为该层网络更为细粒度的访问控制与安全防御提供了更好的条件。

（1）入侵行为监控

基于服务粒度的入侵行为检测。基于云原生网络层的流量管理、可观察能力，可以以服务为单位进行流量的采集，并基于这些数据建立针对性的威胁感知和检测能力，例如针对外部入侵行为的检测，以及内部违规行为的检测等。

基于 Pod 的流量检测方法。在每个节点上部署一个流量控制单元和策略引擎，流量控制单元负责将特定 Pod 的流量牵引或镜像到策略引擎中，策略引擎对接入流量进行异常检测，并向日志系统发送恶意流量告警，根据告警定位到节点、命名空间、Pod。

（2）访问控制

云原生网络的访问控制一般基于类似 ACL（访问控制列表）的策略进行实现，允许或禁止两个实体间通信，实体可以是集群、节点、租户、命名空间、容器或者 IP 地址段。云原生平台应支持细粒度的控制，阻断南北向和东西向攻击，具体实现方式包括 IP tables、Network Policy 或其他方式，根据实际需求，可以选择白名单模式或黑名单模式。

（3）网络隔离

云原生系统根据划分细粒程度的不同划分为网络隔离和微隔离。**网络隔离**在云原生系统中多指为二层子网隔离、以租户划分的隔离，

其划分粒度较粗。**微隔离**主要针对的是东西向流量的隔离，重点是为隔离分区提供基于业务流向的视角，用于阻止攻击者进入网络内部后的东西向移动，能够有效阻止容器逃逸，平台应支持但不限于：基于命名空间的隔离，基于容器间的隔离，基于容器和节点间的隔离。具体的微隔离方案包括但不限于：基于 Network Policy 实现微隔离，基于 Sidecar 实现微隔离。

2.编排及组件安全

（1）基线扫描

云原生平台应支持对集群编排工具和 runtime 组件进行扫描，发现合规基线方面的不满足项并加以整改。合规项通常是面向容器集群如何使用进行设置，所以关注点更多集中在配置层面，例如，禁止匿名访问、启用 TLS 和证书、禁止不安全端口，以及弱口令、账号权限、身份鉴别、密码策略、访问控制、安全审计和入侵防范等安全配置的核查。

（2）漏洞扫描

集群编排组件或 runtime 组件的漏洞危害往往大于用户某一个业务镜像的漏洞危害，一旦出现问题，会影响到集群平台。云原生平台应支持对编排组件和 runtime 组件进行漏洞检测。基于 CNNVD、CVE 等漏洞信息库，发现已知的漏洞组件风险并整改，避免在集群控制层

被渗透和控制。与其他漏洞扫描不同，此处扫描的对象是编排组件，例如 K8S、OpenShift 等，而容器 runtime 组件则不限于 docker、containerd、kata-container 等。

（3）敏感信息加密

基于配置与代码分离的原则，配置信息不应包含在容器镜像及编排环境中，应在编排时对配置进行设定。对于敏感的配置信息，应采用密文方式存储数据，对其进行集中管理，防止未授权访问。

（4）访问控制

云原生计算环境中，需要提供对编排组件和 runtime 组件的访问控制能力。通过基于角色的权限控制，区分平台管理员、集群管理员与命名空间使用者（有时也称为项目用户或租户管理员），以及其他自定义角色。针对不同角色，应基于业务管理需求设置合理的权限，禁止未授权的访问行为；同时，需要注意角色划分的颗粒度控制，避免颗粒过粗导致角色权限过高，避免颗粒度过细导致角色爆炸，避免访问控制混乱和复杂导致有效性下降；另外，访问控制需要做到可扩展性和动态性，以应对云原生系统特性。

（5）资源隔离与限制

提供以安全域/物理域划分资源区域的能力，采用创建 pod 时限制资源的方式来限制容器的资源使用，提供集群层面的资源扩展能力。

通过 cgroup 应用不同的对象级别和资源请求及限制，有助于防止节点和集群资源的耗尽。

3.镜像安全

（1）镜像完整性保护

云原生系统应为用户提供用于保障镜像完整性的机制或功能，并通过一定的控制手段阻止无法通过完整性校验的镜像部署到容器集群中。可通过签名技术实现镜像完整性保护，并通过与镜像构建的 CI/CD 流水线工具进行整合，实现镜像构建过程控制，从构建、测试、扫描到完整性检测全流程管控镜像内容安全可靠。例如，用户构建一个新的镜像，使用镜像签名与 CI/CD 流水线结合的机制进行镜像安全控制，流水线中每一个环节（如漏洞扫描、测试等）完成后均会基于镜像 Hash 值完成一次签名；最终在容器平台侧需要验证待部署镜像具备 CI/CD 流水线每一个环节的合法签名，容器平台才能够基于该镜像进行容器启动；缺失任何环节的非法镜像或被篡改镜像均无法通过最终的验证环节，从而有效地保障了镜像的全流程控制与内容完整性。业界已有类似 Google 开源的 Kritis 组件以及 Docker 提供的 DCT（Docker Content Trust）机制用于镜像完整性检测以及基于完整性进行安全控制。

（2）镜像扫描

安全基线监测。针对镜像中配置文件可能存在敏感信息泄露等进行检测。镜像安全基线包括但不限于以下内容: 创建容器内的用户且只保留必要用户; 容器使用可信的基础镜像; 容器中不安装没有必要的软件包; 扫描镜像漏洞并且构建包含安全补丁的镜像; 启用 docker 内容信任机制; 将 HealthCheck 说明添加到容器镜像; 不在 dockerfile 中单独使用更新命令; 在 dockerfile 中使用 copy 而不是 add; 镜像中删除 setuid 和 setgid 权限; 涉密信息不存储在 dockerfile; 仅安装已经验证的软件包等。

漏洞检测。云原生系统应为用户提供功能全面的镜像扫描工具，一个功能全面的镜像扫描工具应能够对镜像仓库中的镜像和工作节点中运行容器的镜像进行检测扫描。检测扫描的内容应包括但不限于: 基于权威漏洞库信息（如 CVE、CNNVD、RHSA 等）的镜像内组件安全漏洞情况、镜像不安全配置信息、镜像是否含有恶意代码、镜像是否存在密钥等机密信息的硬编码情况等。除了具备安全扫描功能外，还应提供必要的漏洞管理能力，帮助用户清晰明了的了解每一个镜像的安全漏洞情况，给出关于漏洞的修复建议。

恶意镜像检测。通过多种病毒库对镜像进行检测，检测内容包括但不限于远控木马、蠕虫病毒、僵尸网络等木马病毒程序，防止恶意镜像的启动和运行。

（3）镜像阻断

云原生平台应对镜像的实例化运行进行防护，针对不安全镜像进行告警或阻断，包括含有严重漏洞或含有特定漏洞规则、发现木马病毒或风险文件、未授权的软件许可，以及其他自定义规则。由于节点镜像未必来自于镜像仓库，所以镜像阻断有必要在集群节点上启用并防止被绕过。（备注：节点镜像如果不是来自于受信的镜像仓库，也可以作为被阻断或告警的规则之一。）

（4）镜像仓库访问控制

镜像仓库需要实现用户的身份认证、访问权限控制，避免用户提权访问其他用户的镜像资源。建议仓库使用双向认证，对镜像变更或者提交代码都需要进行认证。

（5）镜像仓库安全通信

通过专门的认证和权限模型访问镜像仓库（如：TLS），实现加密安全通信，防止信息泄露。

4.运行时安全

（1）异常行为监控

不安全启动监控。需要对容器异常启动进行监控，异常行为包括但不限于：关闭安全防护机制（如关闭 seccomp 系统隔离机制），使

用过高权限（如使用特权容器），挂载系统敏感目录（如挂载宿主机/etc 目录到容器内）等，使用存在风险的镜像启动等。

容器内核心文件完整性监控。需要具备对核心文件如 system.d、crontab 等的监控能力，防止恶意程序修改文件，保证重要系统文件的一致性。

恶意访问容器监控。在常规业务运维场景下，客户是不需要进入容器内部进行运维操作的。因此，当发生宿主系统尝试进入容器或者远程登录容器等行为时，可以定义其为可疑行为。**一方面，**要对宿主系统尝试使用 docker exec、kubectl exec 等方式进入容器的行为进行监控，当此类行为伴随着容器内的可疑操作行为、敏感文件读写行为发生时，则认为宿主系统是不可信的，例如，具有高权限的管理员可以私自窥探或窃取容器内的数据，主机系统被攻陷后可能会进一步攻击容器；**另一方面，**对于暴露到公网的容器，攻击者可以通过远程登录的方式暴力登录至容器内，例如，安装了 SSH 的容器可被攻击者暴力破解登录，系统需要监控远程登录容器的行为，实时监控用户是否通过远程 SSH 等方式暴力登录容器，避免攻击者登录容器后读取容器内的敏感数据信息或执行恶意操作。因此，容器安全需构建恶意访问容器的实时监控防护能力，防止容器内的敏感信息泄露，避免对容器客户造成损失。

容器内恶意文件识别。攻击者可能通过应用程序漏洞、SQL 注入等方式上传恶意文件，或者篡改镜像从而注入一些恶意文件，这些恶意文件既会有 Webshell 文件，也会有一些可执行的恶意脚本文件。**针对 Webshell 文件**，系统应自动化识别监控容器内的 web 文件上传，通过检测引擎识别 web 文件内容来发现其中存在的 webshell 文件；**针对可执行脚本文件**，系统需要对程序执行的可疑脚本进行识别，实时监控应用程序的脚本执行行为，对执行的脚本内容通过威胁情报库进行检测，避免正常的应用程序执行了恶意脚本文件导致容器失陷。

容器内恶意程序识别。监控容器内正在运行的程序，通过多种病毒库进行协同检测。检测内容包括但不限于远控木马、蠕虫病毒、僵尸网络等木马病毒程序，防止恶意程序的启动和运行。

容器内攻击行为监控。容器被入侵之后，攻击者持续渗透的过程中会进行一系列的恶意行为，比如读取容器内的密钥文件等。平台应提供对容器内恶意行为的监控，监控的恶意行为包括但不限于修改系统配置文件、清空日志文件、使用特定网络工具、修改默认二进制文件、挂载 docker.sock 文件、端口扫描行为等。

容器反向连接行为监控。能够监控容器内应用、程序的反向连接行为。反向连接将具有容器控制权的 shell 发送给攻击者，反向连接成功后会让攻击者接管容器管理权限，从而进入容器执行恶意操作、读取敏感信息，并能够执行后续的容器逃逸等攻击行为。

容器网络连接监控。监控容器程序的网络连接行为，通过威胁情报库检测外联的目的 IP、域名信息是否为矿池地址或其他可疑网址，防止容器被攻击者攻破后用于挖矿或 DDoS 攻击等恶意行为。

容器无文件攻击行为监控。无文件攻击是一种攻击的高级利用手法，其特征表现在恶意代码、文件直接运行在程序的内存空间中而不落地到物理磁盘上。对于此种不落盘的攻击方式，系统需要能够实时监控程序内存空间的信息变化，对内存空间中的代码段和数据段进行检测，通过规则引擎匹配其恶意特征，防止攻击者的攻击手段常驻于容器内存空间，持续地实施恶意行为。

逃逸攻击行为监控。容器安全应监控所有正在运行的容器，发现容器中的异常，包括基于漏洞的逃逸攻击、基于异常提权的逃逸攻击等，并给出解决方案。具体来讲：**一是本地提权监控，**监控容器进程链中用户权限提升为 root 的能力，防止攻击者攻陷容器后通过提权的方式来获取更高的用户权限，进而能够进一步执行逃逸宿主系统行为。**二是逃逸攻击监控，**实时监控容器内的行为，防止攻击者攻陷容器后进一步扩散至容器的宿主系统，利用宿主机的内核漏洞提权获得宿主机用户权限。常见的逃逸行为包括 ： runc 漏洞逃逸(CVE-2019-5736)、docker cp 漏洞逃逸(CVE-2019-14271)、alpine docker 镜像漏洞(CVE-2019-5021)、宿主系统脏牛漏洞(CVE-2016-5195)等。

容器运行行为基准检查。单个容器运行时对应的应用相对单一，其容器内的进程、网络等运行模式相对固定（例如一个数据库容器内可能只会运行 mysql 进程、对外暴露 3306 端口），并且进程也只会调用固定的系统调用、访问固定的文件、连接固定的网络、对集群内 API 的调用和操作内容也比较固定。因此，可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。

（2）入侵行为响应

启动时入侵行为阻断。所有镜像即使安全镜像，如果是以恶意启动方式，也可能会被利用来进行破坏活动。因此云原生平台应能够阻断恶意启动容器的方式，例如：恶意镜像，特权容器，以及对宿主机敏感目录的挂载等。

实时入侵行为响应。在确认容器遭受入侵行为攻击后，平台及系统需要具备对容器内入侵事件的快速响应能力，一方面对失陷容器进行快速处理，另一方面让威胁不会渗透影响到其他容器。响应处置方式包括但不仅限于：停止容器运行；容器网络隔离；终止容器内的恶意进程；隔离/删除容器内的恶意文件。

（三）跟进微服务、Serverless 应用新模式持续推出云原生应用安全

1.微服务安全

微服务是一种架构风格，将关联的业务逻辑及数据放在一起形成独立的边界，目的是在不影响其他组件应用（微服务）的情况下更快地交付并推出市场。而容器是一种新的软件交付方式，它把应用以一个标准镜像的格式打包，能保证其运行环境的统一。容器技术的出现更好地解决了微服务治理的问题，使微服务的设计理念得以更好的落地实现，使业务能实现更松的解耦和更快速的迭代。

微服务安全的建设，**一是**需要与 DevOps 进行集成，针对微服务的漏洞进行发现；**二是**需要关注由于微服务的拆分，带来的更频繁的东西向访问流量；**三是**需要做好微服务之间的访问控制和通信安全，提高应用的健壮性，以避免不安全的、未授权的恶意访问。

微服务安全是云原生应用安全中的重要组成。云原生基础架构应基于零信任理念，默认微服务之间没有信任，所有的微服务均需要有身份和合理的权限配置，所有的互访均需要认证鉴权。

（1）访问控制

根据业务需求，提供细粒度的内部服务间认证与外部服务认证，支持单点登录(一次登录，全部访问)，支持第三方授权登录等功能。

为了获得授权，需要一个集中的架构来提供和执行管理所有微服务的访问策略；微服务需具备网关对服务请求的控制能力；具备应用端对服务访问的控制能力，应用间服务调用通常需要传递用户上下文。如果实施了 service mesh，也可以通过 sidecar 实现应用层的微隔离。

（2）安全通信

用户与服务（南北向）和服务与服务（东西向）之间的安全通信对于基于微服务的应用程序的操作至关重要。某些安全服务策略（例如身份验证或建立安全连接）可能在单个微服务节点上进行处理。因此，需要使用 SSL/TLS 建立安全的连接。

（3）漏洞扫描

微服务漏洞扫描是对镜像漏扫的查漏补缺，是针对运行中的代码以及服务或接口的动态扫描。由于云原生平台中的微服务数量庞大，在扫描前应支持微服务的自动发现、自动管理，减少人工维护的成本；另外，工具通常不能发现应用程序中与业务逻辑相关的问题，必要时也可考虑手工测试来发现微服务的漏洞。

2. Serverless 安全

对应 Serverless 安全威胁，Serverless 的安全防护也可以从两个层面展开，**一方面**是针对 Serverless 应用程序本身的安全防护，**另一方面**是针对 Serverless 计算模型以及平台进行安全防护。

针对 Serverless 应用程序本身的安全防护，可参考传统应用程序的安全防护方法，比如：确保修复已知的漏洞问题、确保依赖库无漏洞问题、通信加密、权限控制等。

针对 Serverless 模型以及平台的安全防护，主要包含：

（1）最小权限原则实施权限管控

每个用户只能访问指定资源，粒度越细，攻击面暴露的就越少。在 Serverless 中，运行单元为一个个函数，Serverless 中最小特权原则通过事先定义一组具有访问权限的角色，并赋予函数不同的角色从而实现函数层面的访问控制。每个函数都应当将任何事件输入视为不受信任的源，并同时对输入进行安全校验。

（2）基于 FaaS 平台实现函数隔离

FaaS 平台应提供基于不同租户、不同应用的函数隔离机制，例如 AWS Lambda 采用 Amazon 弹性计算云（Elastic Compute Cloud EC2）模型和安全容器 Firecracker 模型的机制实现函数隔离。

（3）Serverless 资产业务梳理

为防止攻击者利用敏感数据、不安全的 API 发起攻击。开发者需要在应用的设计阶段对资产业务进行详细梳理，包括但不限于：确认应用中函数间的逻辑关系，如果函数间的应用逻辑关系发生了变化，

要进行相应的安全审查；确认应用的数据类型及数据的敏感性；评估 Serverless 数据的价值；评估可访问数据 API 的安全性。

（4）平台账号安全防护

为了应对 DoW 攻击，云服务商可通过提供账单告警机制对用户进行账号消费告警，如 serverless 使用者可通过设定调用频率与调用费用门限值进行及时的 DoW 攻击响应；除此之外，还可以基于资源限额进行控制，即函数到达一定副本数就不再进行扩展，从而降低 DoW 攻击带来的影响。

（四）全流程安全左移打造云原生研发运营安全

1.安全需求分析

该阶段需要考虑客户的安全需求，包含软件本身的安全功能需求，安全合规的需求等。需求分析阶段通常需要重点考虑的安全需求点包括：身份识别、认证、访问控制和授权、日志审计、敏感数据保护、第三方开源软件选型等关键点。在相对成熟的 DevSecOps 落地实践中，可以做到大部分典型需求场景的关键安全需求自动分析，并自动给出经过实践验证的可落地实施方案。

2.安全开发

（1）安全设计

进行攻击面分析，对系统的安全风险点进行判断，并入开发需求，并构建安全设计的知识库，构建安全设计原则和规范。

（2）代码安全

开发人员遵循安全编码规范，第三方组件安全使用规范，在 CI 中自动集成一些静态安全检查工具（SAST），发现代码中的安全问题；在 CI 中集成镜像扫描工具，检测构建完成的镜像中的安全问题；对代码进行分布式管理或者使用分布式架构的代码管理工具（如 git）进行开发，避免源头丢失导致的问题；对代码进行定期扫描，检查漏洞和 Bug，并及时修复；严格把控代码 Review 环节，减少恶意代码的引入。

（3）开源管理

管理开源及第三方组件安全风险，对第三方组件根据风险级别，制定明确的优选、可用、禁用机制；针对第三方组件安全风险，推荐安全解决方案；进行开源及第三方组件确认，采用工具与人工核验的方式确认第三方组件的安全性、一致性。

（4）制品安全

对制品进行质量检测，比如漏洞扫描等，检查缺陷和漏洞，及时修改和提供替换方案；定期更新漏洞库并进行扫描，及时升级或改用替换方案。建立完备的仓库操作审计，实行镜像签名；实行基础镜像最小化，不要在构建过程中引入攻击者可以利用的工具包。规范化镜像构建，不要泄露敏感信息（证书密钥、密码、认证 token 等）；使用可信的容器镜像，不从不可信的公开仓库中下载镜像。

（5）CI 系统集成

CI 系统应与现有安全系统打通，将安全扫描工具嵌入到实际流程中，在镜像开发、编译等过程中增加监测点，在镜像流转的每一步进行检测控制，保证镜像安全。

3. 安全测试

在测试环节中，除 DevOps 中常规的质量测试外，安全上需要尽可能构建多样化互补的安全自动化测试能力，让应用中存在的安全漏洞、不安全方案尽可能在上线前暴露出来并及时修复，以显著降低应用研发迭代过程中的安全问题修复成本以及线上安全风险。

（1）静态应用安全测试（SAST）

通过分析或者检查源程序的语法、结构、过程、接口等来检查程序的正确性，是从白盒层面对应用源代码进行安全分析发现漏洞。

（2）动态应用安全测试（DAST）

通过分析或检查程序运行时的动态状态，模拟攻击者行为，发现程序的安全风险，是从黑盒层面对应用进行安全漏洞的扫描检测，这类技术主要基于请求响应对于大多数典型漏洞可检测。

（3）交互式应用安全测试（IAST）

通过 IAST 在实时数据，并在运行状态下的应用程序中找到、修复安全漏洞，是结合黑盒检测与白盒检测信息来进行高精度的漏洞检测。对于单纯黑盒或单纯白盒不易发现的漏洞有不错的检出效果。

（4）渗透测试

通过人工的渗透测试，发现工具无法发现的安全问题。

（5）软件成分分析

软件成分分析是一种对二进制软件的组成部分进行识别、分析和追踪的技术。在开源软件日益盛行的今天，开源安全威胁成为企业组织无法回避的话题，与此同时，交付规模和频率正在快速增长，软件成分分析对于安全合规风险管控和权限态势感知都是不可或缺的能力。

4.安全运营

（1）运行时安全配置

主要包括语言、Web Server、数据库等中间件的安全配置以及应用自身的相关安全配置，确保在启动权限、日志记录与保存标准、Web Server 配置标准、数据备份标准等方面符合企业自身及安全合规的具体要求。为确保实际运行时环境中的这些安全配置符合预期的基线标准，安全上需要构建运行时基线安全的自动化检查能力，并联动安全运营平台流程化实现问题及时发现、并及时高效推动治理。

（2）安全监测

线上服务的安全漏洞发现，外部漏洞情报的获取分析，高危漏洞和事件应急响应工作是安全运营环节的重点。在线上安全漏洞发现方面通常有两个主要途径：一种是主动的例行扫描，另一种是建立 SRC 从外部安全社区收集企业服务相关漏洞。

（3）安全管控与审批

运营过程建立对应安全管理流程，变更管理(如外网开放)，进行安全审核等机制建立。提供平台工具和接入面，主要包括安全策略/账号与权限变更、安全备案等的管理、审批。要确保集中化高效的可见可管，需要将安全策略设置功能、账号权限管理功能等内生构建到各云产品中（比如网络产品、云原生 PaaS 产品、访问控制产品等）、

然后统一基于标准化 API 形成与集中化安全管理平台形成对接，让同一管理平台对策略变动、权限变动、人员变动等能够自动感知并根据设定的基线进行判断是否需要走到人工审核流程。

（4）攻防模拟与有效性评估

云原生安全建设应不止于从无到有，还应在安全运营中持续评估安全架构与防护产品的有效性。评估方式可以采用人工方式，也可以考虑将 BAS（入侵和攻击模拟）产品加入到计划之中，利用无害化 POC 来模拟对业务的攻击和对防护的绕过。从而能够为企业或机构提供持续性的防御态势评估，帮助安全团队更有效地识别安全态势缺口并更高效地确定安全举措的优先级别。但攻防模拟活动中务必需要注意，不应对正常业务活动造成干扰，影响到平台或业务应用的可用性。

（5）应急响应

针对安全事件进行响应与处置；进行运营反馈；建立安全运营度量机制，收集问题并优化安全运营流程。在 0 Day 漏洞、安全事件的应急响应方面，同样需要建立类似于常规漏洞处置的流程与标准，但时效性上的要求需要更高。整体上，安全运营要达成最终预期的实际效果，核心是要具备基础的安全能力、流程与标准，流程与标准的平台化实现、运转是确保实际效果达成的关键，这不但可以让安全问题处置效果可控、可衡量，也使得整体安全风险可见、可管理。

（五）优化实施成本提升运营效率实现云原生数据安全

数据安全从业界的实践来看，对应的防护主体主要包括资产类数据（代码、算法、模型等）、办公/业务数据（方案文档、客户资料、合同资料等）、公司运营数据（财务数据、人力资源数据等）、用户相关数据等，而数据安全体系化防护的思路也相对比较成熟，从数据的采集、传输、存储、分析应用、交换、销毁等全生命周期来针对不同类数据在不同场景下区别化实施安全防护，从相关法律法规的数据安全要求、到乙方及各大云厂商数据安全产品/云产品数据安全能力都给云计算时代数据安全防护做了不错的指引。

而在云计算向云原生架构演进升级的过程中，数据安全面临的威胁以及防护思路本质上没有明显的变化，不过随着云原生化对业务迭代及运维效率的提升，势必也对数据安全防护的实施成本与运营效率有了进一步提升的要求。因此，云原生架构下要进一步显著释放云计算的效能，也需要在数据安全防护所需各个环节的安全能力上与云原生架构结合做升级，比如容器安全登录鉴权与租户企业组织信息映射、密码/凭据安全托管能力 built-in 到对应云产品/DevOps 基础设施上、基于 sidecar 模式做细粒度网络访问控制/api 调用异常监测等，以确保数据安全防护方案与业务层更加解耦、方案应用操作上对上层业务更加透明。

（六）全链融合一体纳管构建云原生安全管理策略

1.身份管理

认证和访问管理（IAM）具备面向云原生架构全面的身份管理、用户、及服务认证能力，在私有云和混合云下，能够对用户、设备及服务身份进行管理。对于多云环境下，需要具备和考虑应用和工作负载的身份统一和关联（代理）管理。通过支持临时安全令牌(STS)的方式实现应用和工作负载的访问和鉴权，以满足云原生架构对于身份凭证的短暂性和控制需求。所有工作负载和服务，包括控制通道和数据通道的访问，不管是人还是非人。都需要进行认证，并根据控制策略。为了简化认证过程，可以和企业身份认证服务进行对接。

为了使客户端和服务器能够通过密码双向验证身份，所有工作负载建议利用双向身份验证。认证授权建议独立确定（决策点），并在环境内和跨环境强制执行（执行点）。理想情况下，应实时确认所有工作负载的安全操作，并在可能的情况下验证更新的访问控制和文件权限，因为缓存可能允许未经授权的访问（如果访问被吊销且从未经过验证）。根据已分配工作负载的属性和角色/权限来进行工作负载授权。强烈建议组织同时使用基于属性的访问控制（ABAC）和基于角色的访问控制（RBAC）在所有环境以及整个工作负载生命周期中提供精细的授权实施。

必须对所有人员和非人员集群，以及工作负载操作员进行身份验证，并且必须根据访问控制策略评估其所有操作，访问控制策略将评估每个请求的上下文、目的和输出。为了简化身份验证过程，可以将身份配置为允许使用企业功能，例如多因素身份验证。然后必须使用本节中提到的访问控制机制来进行授权。

2.密钥管理

密钥应该由云平台硬件安全模块（HSM）生成。加密 Secret 有效期应该基于最短可用原则，在过期后即宣告失效，因此密钥的生成机制应该是高可用、高易用的；如果使用了长期有效的 Secret，则应建立流程和指导，定期或在意外泄露的情况下进行轮换或撤销。所有密钥都必须通过安全的通信方式进行分发，并应得到与其保护的访问或数据水平相称的保护。在任何情况下，Secret 都应该在工作负载运行时通过非持久性的机制进行注入，这些内容不应在任何日志、监控审计、或者系统转储中。

3.监控管理

云原生环境中系统行为方式正在发生变化，系统需要满足的要求也在发生变化，导致系统需要提供的保证也发生了变化。为了成功应对这些挑战，不仅需要改变软件构建和运营的方式，还要更好地了解全流程服务状态。

把可观察性引入云原生架构，能够提高安全方面的观察力，有助于解决和缓解异常情况；这个领域的工具能够收集信息并进行可视化展示。如果说监控是定位于报告系统的整体运行状况，那么可观察性就是在提供对系统行为高度细化的见解以及丰富的上下文，非常适合提供隐式故障模式所需的信息和调试所需的信息。可观察系统不仅仅是数据的收集，获得数据后还可以将数据转化为有用的监控信息。可观察性意味着更全面的内容，包括监控、应用程序代码检测、即时调试时的主动检测以及对系统各个组件更透彻理解的文化。监控既要以故障为中心，主动监控系统变化；又要以人为中心，确保运维测试人员能尽快收到警报。可观察性的三大支柱是日志（log）、指标（metric）和跟踪（trace）。

4.安全审计

审计日志分析是识别和关联系统入侵、滥用或配置不当的最成熟方法之一。持续地、自动化地对审计日志进行分析和关联，对于安全团队来说至关重要。和传统应用对比，云原生架构能够为工作负载生成更精细的审计配置，更方便进行过滤。对日志的过滤能力，能够避免下游处理机制的过载。与传统的日志分析一样，将日志中的数据关联、上下文转化为信息，生成可操作的审计事件是重中之重，以此为基础来触发决策、进行事件响应。

为了能够对使用集群的实体的行为进行审计，启用 API 审计并对特定 API 进行过滤是很重要的。这些 API 是安全团队、集群管理员或其他研究领域的团队的工作重心之一。对外服务的 API 走统一的 API Gateway 保障 4A 安全能力；对内服务的 API 应加入安全准入管控审核，包括是否接入统一的认证方案（基于统一管理的证书/aksk/token 等）、以及涉及高敏感数据的服务 API 是否开启加密传输配置等。服务 API 访问审计管理需要针对重要服务 API 的访问日志设定异常审计策略。

攻击者可能会通过禁用日志或删除其活动日志来掩盖踪迹，为了制止这种行为，应尽快将日志转发到通过集群级凭证无法访问的位置。处理警报的系统应定期对假阳性报告进行调整，以避免警报泛滥、疲劳，并防止假阴性情况的出现。

5.安全策略

（1）云原生计算环境安全策略

云原生计算环境安全需要与其配套的管理能力，以确保不同场景生效不同的安全防护能力，比如通过云原生 PaaS 平台全局统一确定容器实例的启动权限、控制容器镜像的可信来源等。镜像、运行时环境的基线安全、入侵检测能力等也同样需要有全局统一的策略管理能力，包括从问题检测到问题处置的流程化对接与管理能力、按服务标签等区别性管理安全检测/防护策略的能力等。

（2）云原生研发过程安全策略

云原生场景下要做到高效且执行到位的应用代码安全质量保障，就需要在研发运营安全的核心技术能力基础上，具备灵活、高效的安全质量管理管控能力，这里的管理能力主要包括以下三点：一是支持云原生应用设计、开发、测试阶段的安全评审流程管理能力：可以按应用模块属性（重要性、来自第三方等等）设定安全评审策略。二是安全自动化检测策略管理能力：检测策略管理上是否可以按应用属性/业务线灵活配置安全检测的介入环节与类型等。三是安全问题跟踪处置管理能力：具备处置时效性管理/自动联动复测接单管理等。

（3）云原生访问控制安全策略

在运维场景访问控制方面，针对不同角色需要有不同粒度、不同有效期的访问控制管理能力，尽可能防控运维层面引入大范围误操作、恶意操作等风险。运维权限授权粒度上可配置到集群粒度、服务粒度、实例粒度等；运维管理策略应该有分级，例如，集群粒度只允许通过评审与测试的固化运维任务执行、纯手动运维工作仅限于实例粒度等；在运维授权时限上针对不同角色也应该有对应的管理策略。

在云原生应用运行时访问控制方面，从管理上要能感知、管控应用层的认证访问控制策略、以及网络层的访问控制策略，即在安全策略上不仅对访问控制有明确的要求、还应该在实际的云原生 PaaS 平台等环境上具备落地性的管控能力，确保访问控制策略执行生效。

五、关注中外、聚焦产品，描绘云原生安全生态图景

（一）放眼世界，体系化了解云原生安全开源项目与组件

CNCF landscape 中关于安全的项目、工具如图所示。本文将对其中部分项目、工具进行简要介绍。

图 4 CNCF landscape 安全项目及工具示意图

1.Clair

Clair 的目标是能够从一个更加透明的维度去看待基于容器化的基础框架的安全性。通过对镜像的分层文件系统进行扫描，发现漏洞并进行预警，使用数据是基于 Common Vulnerabilities and Exposures 数据库(简称 CVE)，各 Linux 发行版一般都有自己的 CVE 源，而 Clair 则是与其进行匹配以判断漏洞的存在与否。

2.dex

dex 是一个基于 OpenID Connect 的身份服务组件，用来进行用户认证和授权，并提供基于多种标准的身份服务和认证解决方案。dex

的设计采用了安全和加密的最佳实践来最小化攻击者获得系统访问权限的风险，dex 的架构划分也可以减轻任何单个攻击可能带来的损害。例如，dex 缺省使用 token 生命周期管理，并自动轮换签名密钥。由于密钥本身是加密的，攻击者需要在短时间内同时侵入数据库和一个 dex 工作节点才能得到 token。

3.kube-bench

kube-bench 是针对 Kubernetes 的安全检测工具，根据 CIS 的安全性最佳实践检查 Kubernetes 是否安全。kube-bench 可以运行在本地或 Kubernetes 集群环境中，根据执行的测试结果推荐一些可用于 master 或 worker 节点的安全配置。kube-bench 通过运行符合 CIS Benchmark 的测试来实现，并获得总结性的信息以及相关的修正建议。例如，如果 kube-apiserver 上关闭了身份认证，给出的建议中会解释如何启用身份认证，进行了相关修正操作之后，可以再次运行这个工具进行检测，直到完全符合安全标准。

4.Falco

Falco 是一个云原生运行时安全系统，可与容器和原始 Linux 主机一起使用。它由 Sysdig 开发，是 CNCF 的一个沙箱项目。Falco 的工作方式是查看文件更改、网络活动、进程表和其他数据是否存在可疑行为，然后通过可插拔后端服务发送警报。通过内核模块或扩展

的 BPF 探测器在主机的系统调用级别检查事件。Falco 包含一组丰富的规则，编辑这些规则以标记特定的异常行为，并为正常的计算机操作创建白名单规则。

5.Open Policy Agent

在应用开发中，应用程序往往需要跟据特定策略的决策结果来判断后续执行何种操作。比如，权限校验就是策略决策的一种典型场景，它需要判断哪些用户对哪些资源能够执行哪些操作。这些策略可能随着时间需要不断的动态更新。当前策略决策的逻辑往往硬编码实现在软件的业务逻辑中，当需要更新策略规则集时，还需要修改应用代码、重新部署应用，非常不灵活。Open Policy Agent，简称 OPA，为这类策略决策需求提供了统一的框架与服务，将策略决策从软件业务逻辑中解耦剥离，将策略定义、决策过程抽象为通用模型，实现为一个通用策略引擎，以便适用于广泛的业务场景。

（二）立足本土，系统化梳理云原生安全解决方案与服务

1.华为云·容器平台安全方案

华为云容器安全方案作为专门为容器技术设计的安全产品方案，给予企业容器虚拟环境从镜像开发构建到部署到仓库，再到生产环境运行的整个周期的安全性，减少 IT 安全部门与开发部门的协调沟通

成本，并对容器内部的应用程序活动具有高可见性，允许组织检测和防范可疑活动和实时攻击。

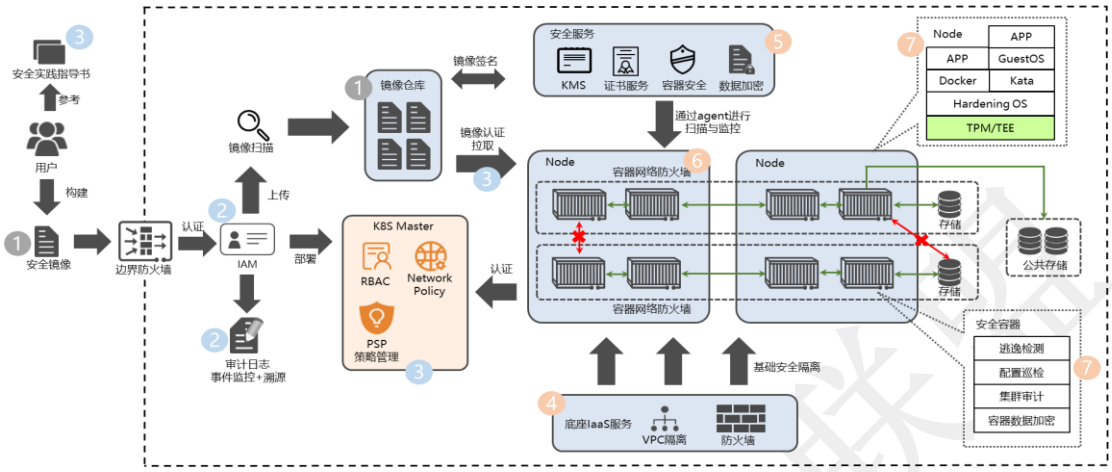

图 5 华为云容器安全方案架构图

华为云容器安全整体方案以容器安全服务 CGS 的能力为核心，以云容器引擎 CCE，云镜像仓库服务 SWR 提供的原生安全能力为辅助，具体七大核心能力详细介绍如下。

镜像安全。华为云容器安全方案通过软件开发平台 DevCloud 的代码托管、代码检查、可信测试、可信构建等能力，极大程度提升容器安全镜像构建的效率与安全性，同时华为云容器安全服务 CGS 提供 Docker 官方镜像的定时漏洞扫描能力，帮助用户构建安全基础镜像。

账号认证审计。华为云容器安全的账号认证审计包括 IAM 的全部认证与审计机制和安全特性，以及细粒度授权等功能。在此基础上华为云容器引擎 CCE 提供集群审计日志，以及集群事件监控与溯源能力。

容器部署安全。华为云容器安全方案的容器部署安全主要包括两个方面的内容，一是容器管理面的基础设施安全包括镜像仓库自身、

以及 K8S 管理面的配置安全; 二是容器镜像在仓库中的签名与认证机制，确保集容器群在从仓库拉取镜像过程中的镜像完整性与安全性。

IaaS 底座安全。用户可以通过配置华为云的 VPC 服务以及网络 ACL 与安全组特性，对容器底层的节点基础设施进行安全隔离与访问控制。同时用户还能通过在容器节点（ECS 或 BMS）上安装与使用华为云企业主机安全服务 HSS,统一管理所有容器底层节点的安全状态。

数据安全加密。华为云容器安全方案通过 SSL 证书和 https 实现镜像上传和下载过程中的全路径加密，同时对容器保存在 OBS 或数据库中的数据进行加密，确保镜像在传输和存储过程中的机密性。

容器网络防火墙。华为云容器安全方案基于 Kubernetes 的网络策略功能进行了加强，通过配置网络策略，允许在同个集群内实现网络的隔离，即可以在某些实例（Pod）之间架起防火墙。

容器运行时安全。华为云容器安全服务 CGS 在提供传统安全威胁检测（恶意程序，网站后门等）基础上，同时提供基于机器学习算法的容器逃逸行为检测，能够解决容器逃逸这个容器安全面临的最大安全威胁。

2.阿里云·容器平台安全方案

阿里云 ACK 容器服务面向广大的企业级客户，构建了完整的容器安全体系，提供了端到端的应用安全能力。在今年 Forrester IaaS

安全评测中，阿里云容器安全能力获得满分。下图为阿里云容器服务的安全体系架构图:

图 6 阿里云 ACK 容器服务安全体系架构图

整个容器安全体系依托于阿里云强大的平台安全能力，包括物理/硬件/虚拟化以及云产品安全能力，构建夯实平台安全底座。

在云平台安全层之上是容器基础设施安全层，容器基础设施承载了企业容器应用的管控能力，其默认安全性是应用能够稳定运行的重要基础。首先面向集群 host 节点 OS 镜像本身阿里云操作系统团队做了很多安全加固相关工作，Alibaba Cloud Linux 2 不仅是阿里云官方操作系统镜像，也是 ACK 的首选默认系统镜像。Alibaba Cloud Linux 2 在 2019 年 8 月 16 日正式通过了 CIS 组织的全部认证流程。ACK 正在支持对基于 Alibaba Cloud Linux 操作系统的集群进行 CIS 安全加固来满足简单、快捷、稳定、安全的使用需求。

在容器管控侧，阿里云容器服务基于 CIS Kubernetes 等业界安全标准基线对容器管控面组件配置进行默认的安全加固，同时遵循权限最小化原则收敛管控面系统组件和集群节点的默认权限，最小化攻击面。阿里云容器服务提交的 CIS Kubernetes benchmark for ACK 正式通过 CIS 社区组织的认证审核，成为国内首家发布 CIS Kubernetes 国际安全标准基线的云服务商。

ACK 管控侧和阿里云 RAM 账号系统打通，提供了基于统一身份模型和集群证书访问凭证的自动化运维体系，同时面对用户凭证泄露的风险，创新的提出了用户凭证吊销的方案，帮助企业安全管理人员及时吊销可能泄露的集群访问凭证，避免越权访问攻击事件。

针对密钥管理、访问控制、日志审计等企业应用交互访问链路上关键的安全要素，ACK 容器服务提供了对应的平台侧安全能力。

面向容器应用层在供应链和运行时的安全挑战，阿里云从容器应用的构建、部署到运行全生命周期，提供全方位覆盖安全能力：

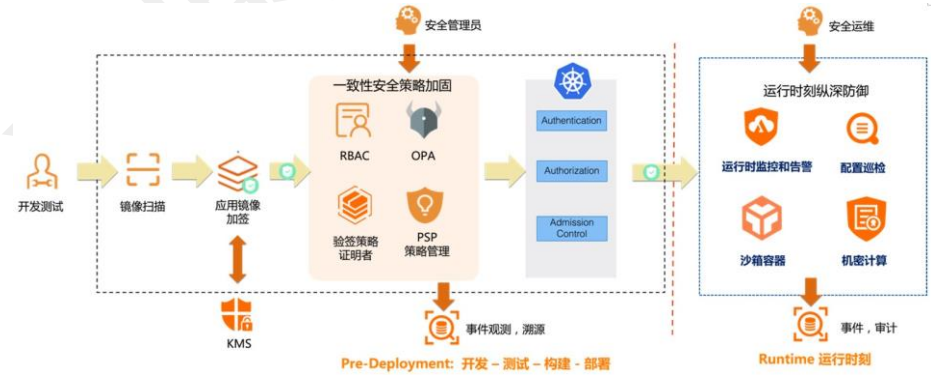

图 7 阿里云 ACK 容器服务应用全生命周期安全能力

3.腾讯云 • 容器平台安全方案

针对容器环境下的安全问题，腾讯云容器安全服务为企业提供镜像安全扫描、运行时安全检测（容器逃逸、反弹 shell、文件查杀等）、高级防御（进程检测、访问控制、高危系统调用等）、安全基线检测、资产管理等安全防护功能，帮助客户及时发现安全风险并提供防护解决方案。定期提供最新安全报告和建议，让企业及时掌握业务安全动态。

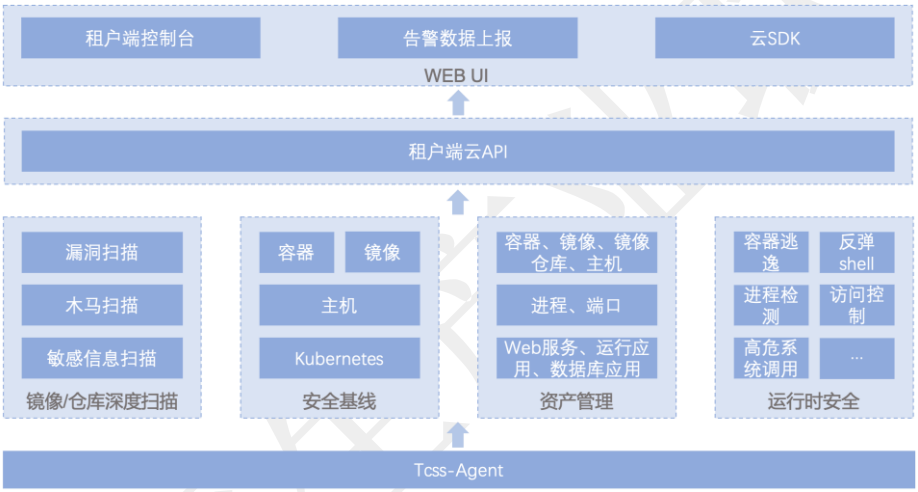

图 8 腾讯云容器安全服务（TCSS）架构图

集中管理。容器安全服务为用户提供资产管理、镜像安全扫描、攻击者入侵检测、合规基线检查等一体化安全服务，资产信息、安全事件等在统一的控制台进行集中管理。

基于可视化界面，提供资产集中管理:

- 资产分组管理
- 安全事件可视化呈现，一目了然
- 镜像/镜像仓库安全扫描结果实时更新

- 安全策略分组维护

云端安全威胁基础数据库。借助腾讯云的市场优势，云镜 Agent 已经覆盖超 200 万台云主机，云镜的云端防护系统主动接收来自全网海量 Agent 上报的各类安全事件数据，组成了国内最大的的云端安全威胁基础数据库，经过云镜后台系统的运营和挖掘，实时计算出大量的攻击行为模型，基于这些攻击模型，可以帮用户快速感知到容器环境安全状态。

实时告警。7*24 小时安全服务；实时检测，实时免费通知异常情况；对异常行为实时告警，实时感知攻击者入侵行为，帮助用户及时处理风险，减少损失。基于腾讯的安全运营能力，全面捕获和分析当前互联网的最新安全威胁，验证云上用户面临的安全风险，所有规则由安全专家多级审核，确保低漏报、误报。

操作简便。无需购置专业安全设备，无需具备专业安全知识，自动防护大部分安全攻击，购买专业版服务即可享用。使用容器安全服务后，容器业务环境面临的各种攻击行为会第一时间得到感知。

4.青藤云·蜂巢容器安全产品

蜂巢提供覆盖容器全生命周期的一站式容器安全解决方案，实现了容器安全预测、防御、检测和响应的安全闭环。能有效解决云原生架构安全防护体系中的基础设施安全、云原生计算环境安全、云原生应用安全的问题。

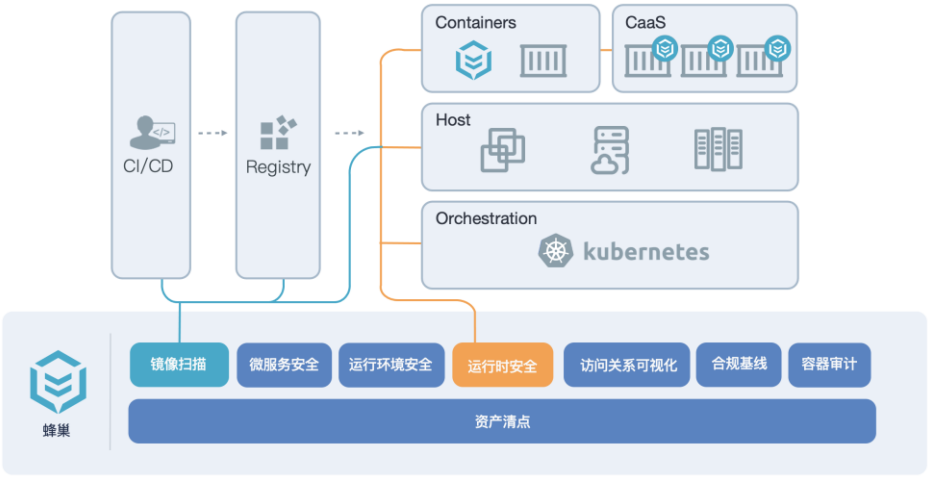

图 9 青藤云蜂巢容器安全产品架构图

实现工作负载可观测性。全自动化、细粒度的对工作负载进行分析，并可视化工作负载之间的网络访问行为。

网络层面：可视化工作负载间的访问关系，进一步了解业务之间的调用关系

运行时层面：梳理云原生环境工作负载，帮助安全人员了解运行的容器、容器内运行的 web 应用、数据库应用等。

全生命周期安全。覆盖 Build、Ship、Run 的全生命周期，助力企业完成 DevSecOps 安全实践落地，实现业务上线即安全。

镜像安全：从容器的构建、分发、运行角度出发，在各个阶段检测镜像中是否存在应用漏洞、是否使用基础镜像、是否是受信任镜像、是否存在恶意文件、敏感信息等。

集群安全：检查集群的配置安全性和潜在的漏洞风险。

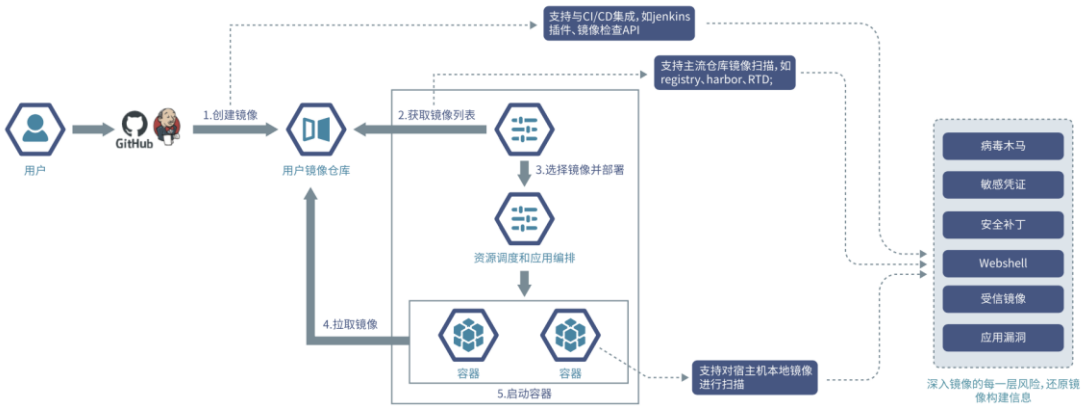

图 10 青藤云蜂巢容器安全产品全流程监测响应示意图

持续性的容器运行时监测&快速响应。提供多锚点的基于行为的检测能力，能够实时、准确地感知入侵事件，并快速进行安全响应处理。能精准检测容器特有的逃逸行为、基于 k8s 的攻击行为、容器内的挖矿、横向渗透等行为，同时进一步提供隔离容器、杀文件等方式进行处理。

安全原生，为云设计。Agent 轻量级的部署方式，能够很好集成到云原生复杂多变的环境中，如 containerd、Kubernetes、paas 云平台、openshift、 Jenkins、Harbor、Jfrog 等等。

5.小佑科技·镜界容器安全防护平台

镜界容器防护平台基于安全理念，从容器视角为云原生的全生命提供整体安全防护的解决方案。

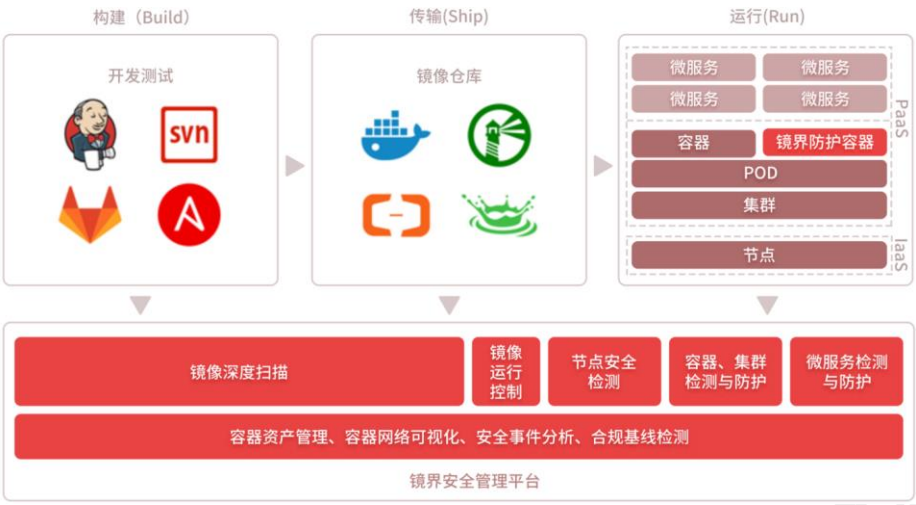

图 11 小佑科技镜界容器安全防护平台架构图

平台采用自动检测、自动分析、自动处理的方式来防御整个容器生命周期中所遇到的安全威胁。在防护技术上使用智能测试、机器学习与威胁预测的方式来确保容器及容器内的应用安全。产品功能包括但不限于:

表 6 镜界容器防护平台产品功能列表

功能模块	功能简介
资产管理	镜像资产管理、容器资产管理、服务资产管理、风险态势管理
镜像安全	镜像运行自动保护、快速的镜像扫描、 支持一键生成镜像报告、镜像内文件检测、安全溯源
容器安全	反弹 shell 检测、逃逸攻击检测、POD 隔离、容器的学习模型、容器文件保护、调查取证
基线合规	Docker 合规、kubernetes 合规、OS 合规、自定义合规
微隔离	网络拓扑展示、租户间隔离、租户内隔离、 支持自定义网络策略

镜界容器防护平台使用场景涵盖:

开发测试环境中的容器云平台。以安全左移的建设思路，与企业的 CICD 流程结合,对构建容器镜像的 dockerfile,构建的容器镜像，以及编排部署的 yaml 文件进行检测分析，从而构建安全的镜像仓库。

生产环境中的容器云平台的安全运营。包括基础设施合规基线、镜像安全分析、容器运行时的入侵检测与防护。通过漏洞扫描有效收缩攻击面，通过规则与机器学习相结合，对异常行为告警和阻断。

攻防演练、重保与各类合规场景。基于小佑产品，可以满足各类攻防演练、重保和各类合规政策对容器平台的安全要求，在产品之外还可以提供云原生安全相关渗透与风险评估服务。聚焦云原生领域，为云原生应用保驾护航。

六、多元主导、深度融合，洞悉云原生安全发展趋势

（一）安全技术的主导力量从单边走向多元

云原生是云计算的下半场，云原生安全将是未来几年云安全的主要发展方向。随着新基建的快速推进，云原生技术与信息基础设施的融合成为一个明显趋势，服务提供商在设计、部署和运营这些基础设施和应用系统时应充分考虑与云原生技术相关的风险、威胁和安全防护手段进行融合。同时，网络安全环境日益严峻，云原生安全所涉及的范围越来越广，构成越来越复杂，涵盖了从云基础设施到云原生技术架构，从云原生应用到研发运营管理的全方位安防体系，仅凭一家厂商的技术与管理能力，云原生安全建设难以面面俱到。未来，云服务商与安全厂商势必将加强深度合作，结合双方在技术研究、人才储备、产品应用等方面的积累和经验建设一个更加完善、开放的云原生

安全技术环境；与此同时，在云原生安全的不同赛道将衍生出更加专注于细分领域的安全服务商，进一步丰富和完善云原生安全生态。

（二）安全设计理念从以人为中心转向以服务为中心

云原生的架构模式注重以服务为基础的细粒度拆分，以容器和容器编排为代表的云原生基础设施，成为支撑云原生应用的重要载体；以微服务和 Serverless 为代表的云原生应用环境，显著增加了端口和服务的数量，打造了以服务为基础的新型应用模式；以 DevOps 为代表的研发运营体系，专注于敏捷迭代。传统安全侧重以人为主的防护策略，已经不能满足云原生实例频繁启停的生命周期变化以及海量的东西向流量交互，以服务为中心构建的容器安全防护措施、持续监控响应模型和可视化平台，将成为云原生安全防护的主流方案。

（三）安全产品形态从粗暴上云转向与平台/应用深度融合

随着云原生技术加速落地应用，在不久的将来，其必然会与各种信息系统、各类计算场景相融合，弹性敏捷、按需编排都会成为系统原生具备的能力。云原生安全也不得不与云原生平台、应用深度融合，提供新型云原生信息基础设施的防护、检测和响应能力，并将云原生技术赋能于这些安全产品、应用和解决方案；而云原生技术引入的全新安全威胁也要求安全产品不得不与深入云原生运行实例和应用内

部，实现进程级防护能力、微隔离访问控制、全流程实时监控响应，实现安全方案的内生配置和深度融合。

（四）安全落地方案从重型化走向轻量化、敏捷化、精细化

随着容器越来越广泛的应用，容器部署的环境也越来越复杂多样，包括一些对资源比较敏感的嵌入式领域，此时在对系统环境、业务应用轻量化的同时，要求安全方案也必须足够轻量化，避免因安全方案的引入占用过多资源，进而影响业务的正常运行也变得尤为重要。另外，据调查统计，容器的生存时间已经越来越短，生存时间小于 10s 的容器已经占了大部分，此时就要求安全方案的反应必须迅速、及时，及时发现容器启动，密切跟踪容器行为，并在发现异常时迅速反映。同时，云原生提供的服务粒度越来越细，相应的安全方案的防护粒度也需越来越细，从过去的容器粒度，到目前的函数粒度，未来可能是语句粒度、变量粒度，安全方案的防护粒度需小于提供的服务粒度。

中国信息通信研究院

地址：北京市海淀区花园北路 52 号

邮政编码：100191

联系电话：010-62300072

传真：010-62304980

网址：www.caict.ac.cn