Deconstructing Voidlink: Why New AI and Cloud-Native Threats Require a New Class of Defense

By Isovalent

Archived: 2026-04-05 16:10:07 UTC

VoidLink's emergence serves as a stark reminder that AI and automation aren't just for defenders. The new malware toolkit blends rapid innovation with deep technical modularity, pushing the boundaries of what we expect from modern threats.

VoidLink is one of the newest, more effective malware frameworks designed and generated largely with the assistance of AI tooling. Research shows the toolkit's rapid development cycle with functional code in less than a week, corroborated by technical analysis of the attack stages that shows a sophisticated framework designed to bypass standard cloud security perimeters.

We use Tetragon, the leading eBPF runtime security visibility tool, to investigate the behavior of VoidLink and then mitigate the toolkits startup sequence using runtime security controls. If you are looking for a walkthrough of the detection policy, please navigate to the policy example and walkthrough below.

Quick FAQ's

- **What is VoidLink:** A Linux malware toolkit characterized by its modularity, technical sophistication, and development process. VoidLink combines custom loaders, implants, kernel rootkits, and more than 30 plug-in modules, allowing attackers to maintain persistent access across a wide range of Linux environments, including cloud, containerized, and bare-metal Linux systems.
- **When was VoidLink discovered:** First discovered and publicized in December 2025. It appears to have been built in late Fall 2025, over the course of ~1 week.
- **Who developed VoidLink:** Attributed to a Chinese affiliation (unknown exact organization), leveraging AI models and coding agents (e.g. TRAE SOLO) to automate and accelerate development.
- **What does VoidLink target:** Built to target Linux systems, specifically for cloud workloads, containers, and Kubernetes environments.
- **What makes VoidLink unique:** VoidLink's AI-driven development, modular plugin architecture, and adaptive runtime capabilities targeting cloud, container, and Kubernetes workloads. It includes adaptive techniques to map out target environments, understand their security tooling, and take dynamic steps to avoid detection.
- **How to update or patch:** There is no single patch to apply, instead Security teams need controls in place to detect and block Voidlink's attempts to hide activities within standard system processes.
- **How does Isovalent detect and mitigate:** Tetragon acts as the low-level sensor, providing eBPF-powered kernel introspection. The Tetragon sensor monitors all file access and process execution with incredibly low overhead, allowing you to see exactly what VoidLink is attempting in real-time without slowing down the business.

- **Who is using VoidLink:** For more information on known ATP groups using VoidLink, check out the Talos threat research post ↗.

Solving the visibility challenge

While VoidLink is highly sophisticated, runtime monitoring identifies the characteristic syscall and file creation events it relies on to establish a foothold in cloud workloads.

Voidlink operates with intent to evade, disguising its processes by mimicking legitimate system names and behaviors (as we see in policy examples below covering stage [0] early name change syscalls) and dynamically evading legacy EDR tools. By leveraging process name changes and embedding itself within standard system tasks, it blends malicious activity into routine operations, making detection through conventional monitoring or application-level logging extremely difficult.

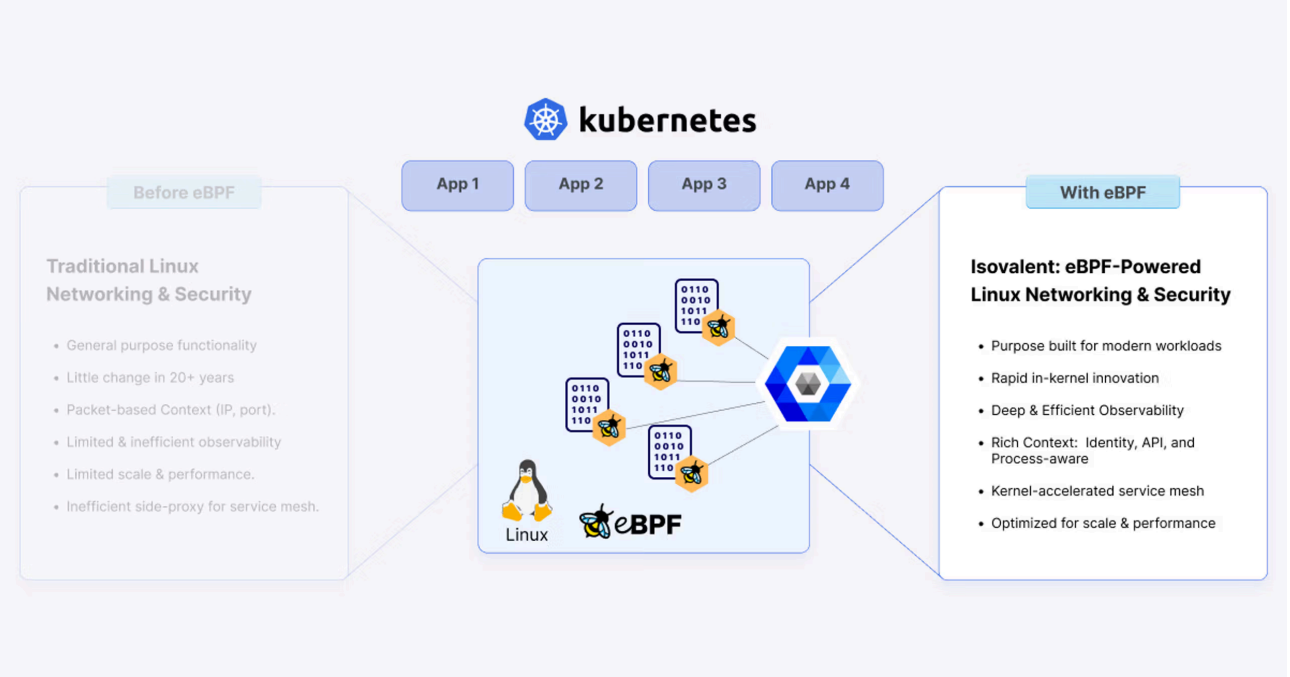
Tetragon exposes this kernel-level activity:

Tetragon detects when a process binary calls the kernel function `prctl` by hooking into `security_task_prctl`. In the snippet from the Tetra CLI above, Tetragon detected arguments for the binary [`/usr/local/bin/voidlink/stage0`] to change its process attributes to resemble a legitimate system worker `kworker/u16:0`.

Detecting sophisticated threats requires observing system behavior beyond the application layer. Tetragon addresses this gap by implementing deep kernel-level visibility, monitoring actions as they pass through the kernel in real time, ↗ not just as they are reported by user space.

eBPF policies ↗ continuously monitor file access, process execution, and sensitive syscalls with minimal system overhead. By living in the kernel, these policies are the optimal control point to observe and act on suspicious behaviors.

Tetragon both alerts on and blocks events directly within the kernel. Enforcement mechanisms include overriding the syscall's return value (preventing the action's execution), and issuing a signal to immediately terminate the offending process, halting the malware execution chain.



Neutralizing the threat: blocking and mitigation

It's not enough to just detect Voidlink; teams have to stop the attack chain before it implants the system. Tetragon supports two main enforcement actions: `override` (returning an error, e.g., `-1`) and process termination (`SIGKILL`).

Override blocks the immediate malicious action (as with the name change event we see further below) by disallowing the syscall to complete successfully. However, the malware chain may remain active after receiving the null value (so, even if it is expecting the name change to return successfully, the chain may continue to the next step even with a `-1`) and still attempt further actions.

This is where a layered policy halts the process. `SIGKILL` immediately terminates the process; this signal cannot be caught or ignored by user space code, halting the malware and stopping any subsequent stages (such as loading plugins or establishing connection to the C2).

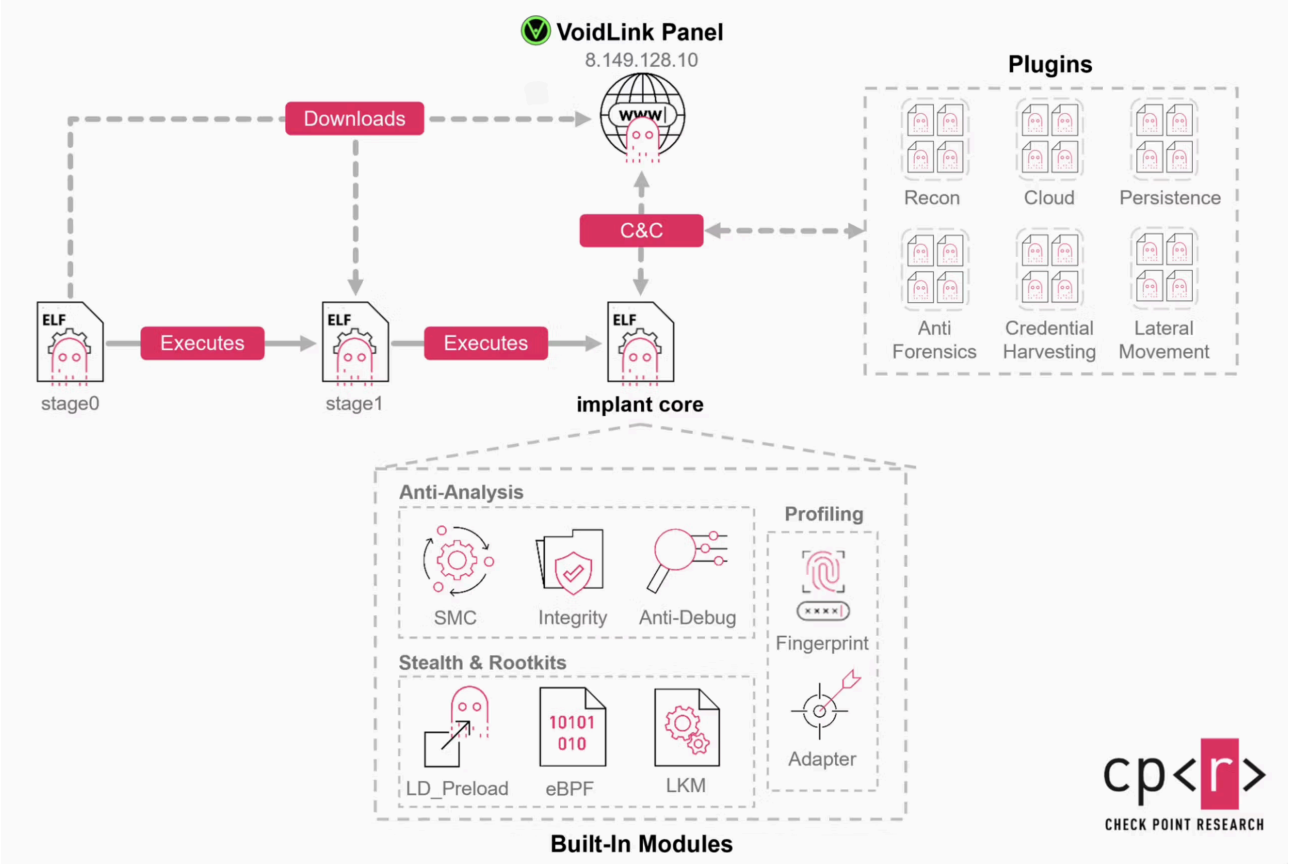
In practice, using both actions together, `override` to block the specific action and `SIGKILL` to stop the process, provides the strongest guarantee of interruption and containment. We use this layered approach in the policy below for a hardened policy.

VoidLink attack framework and sequence

Voidlink is best understood as a modular, AI-developed malware framework engineered to target modern cloud and containerized workloads. It establishes a connection to the target system, and then provides the attacker a variety of tools to complete their intended outcome (steal data, change files, access logs or a specific system).

The malware leverages custom loaders, kernel and user-mode rootkits, and a dynamic plugin system that allows for real-time adaptation to cloud-native targets, including Kubernetes-managed containers and short-lived workloads.

Technically, Voidlink is designed to bypass standard cloud security perimeters, embedding itself at multiple layers to maintain persistence and facilitate lateral movement. The modular design supports covert data exfiltration and on-demand capability loading, making many legacy defenses insufficient to handle uncovering the initial attack sequence and the dynamic core plugins. In highly distributed cloud and containerized environments, layered runtime enforcement and continuous kernel-level monitoring are now operational requirements for proper defense in depth.



This attack flow diagram from Check Point Research ↗ shows the attack pattern of the malware toolkit, with early stages being used for obfuscation and implanting before moving into more targeted attack patterns to exfiltrate data or reach sensitive workloads. Stages [0] and [1] implant the malware onto the workload and establish the command-and-control (C2) server to relay specific actions and plugins (e.g. establish SSH tunnel, container escape, scan local logs, and more).

This overview is complemented by Sysdig's research, which details the specific syscall sequences observed ↗ during the early stages of the attack. If you are curious about the encoding of each sequence and attack stage, we encourage you to read that blog here. ↗ In the next section, we take the known attack pattern and use Tetragon policies to hook into the sequence and block VoidLink from executing.

Detect and mitigate VoidLink with Tetragon

By living in the kernel Tetragon is particularly effective in capturing and acting on low-level syscall-level events, particularly those associated with the process manipulation and C2 activity that pass through the kernel and are core to VoidLink's known attack stages.

Stage [0] detection and mitigation

VoidLink's initial attack stage mask the process names, creates new memory files, and establishes C2 connections using syscalls `prctl` (for changing process names), `connect` (for network operations), `fork` (create new child processes), `memfd_create` (create memory file), and more. Let's break down detecting and blocking during this early attack phase.

In the attack chain, VoidLink takes a routine sequence during stage [0] that involves invoking the `prctl` syscall (short for process control)and changing its name to something benign (`kworker/` , `watchdog/` , etc). This re-naming attempts to keep the process relatively unflagged for anomalous behavior. However, this syscall sequence gives an ideal early hook point for the policy to detect and kill the attack chain.

We've modified the policy to obfuscate the full list of args and index values, let's dive into the high-level important bits.

This policy detects and prevents obfuscation techniques used by Voidlink, which attempts to evade detection in stage [0] by changing its name to mimic legitimate system processes. This is not a standard action and is notable in production environments.

The policy attaches to the `security_task_prctl` kernel hook , called whenever a process attempts to use the `prctl` syscall. `prctl` has a number of use cases in the kernel, and here specifically, the option `PR_SET_NAME` (`values = 15`) changes the process name.

The eBPF policy sits in the kernel, observing all `prctl` events to check if they are attempting to set a new name for the process, and if so, do they match the listed values (`"kworker/"` , `"watchdog/"`) in the policy.

When, the policy detects a process attempting to rename itself with a prefix matching the listed values, it triggers two enforcement actions: 1) `SIGKILL` to terminate the process immediately and 2) `Override` the syscall to prevent the name change from executing successfully. This blocks the malicious activity at the kernel level and stops the offending process before it can continue its execution.

These policies can be modified simply to generate alerts instead of enforce or increase the scope of values covered (here we have only listed a sample).

Additionally, this same policy framework addresses the other syscalls in the startup sequence. In the example, we targeted `prctl` , but each syscall (`connect` , `recv_from` , `memfd_create`) provides the ideal hook point to layer in other pre-built policies that block VoidLink's startup sequence.

Control channel detection and mitigation

If the attacker successfully implants after stages [0] + [1], the C2 server communicates which plugins to execute and on what targets. The toolkit relies on a "magic number" (unused, non-standard value) to signal the kernel rootkit to listen for subsequent instructions. This technique is typical in advanced malware, using covert values in syscalls to bypass standard detection and trigger custom behaviors.

How it works is the rootkit uses a kernel module to hook into `prctl` events and listen for the magic number (0x564C). If the toolkit's hook observes the magic number, then it uses arguments 2 and 3 passed in the same event to take a specific action. [Ie. If `arg2 = 12`, then activate the container escape malware.] This simplifies the detection workflow, as we only have to hunt for the magic number.

We've also modified this policy to obfuscate certain fields, let's break down the important bits.

The eBPF policy sits in the kernel, observing when a `prctl` call is made with the VoidLink-specific magic number (`0x564C` , `decimal 22092`) as the first argument, an indicator the rootkit is receiving a command from the C2 infrastructure. This number has no other common use case, and is only relevant to the VoidLink execution chain.

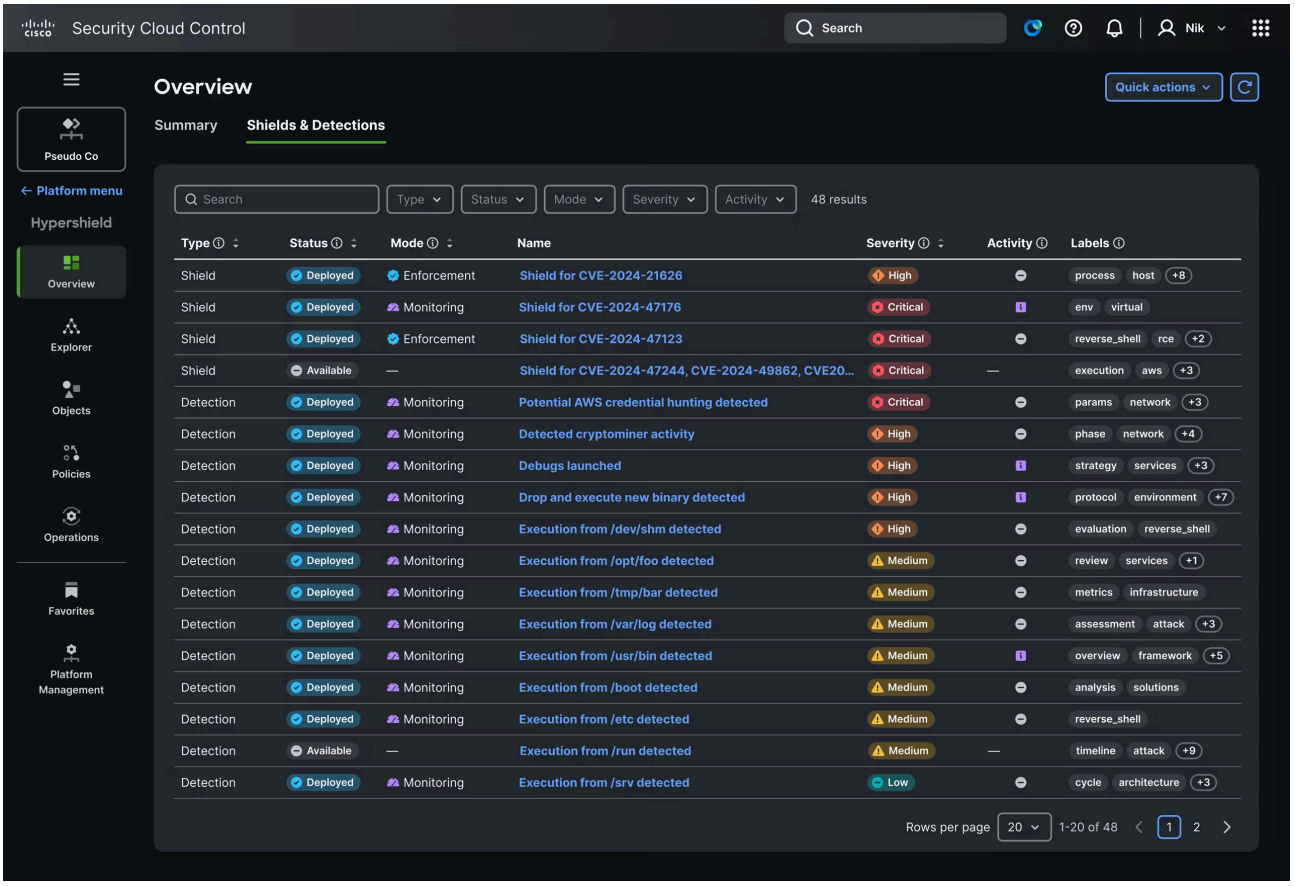
On detecting the magic number, the policy immediately issues a `SIGKILL` to terminate the process and overrides the syscall, blocking command execution at the kernel level. Same as the actions in the first policy, this targeted approach gives teams reliable detection and mitigation of VoidLink's attack chain, actively stopping an ongoing attack and shutting down escalation through other follow-on techniques.

Scaling the shield across modern cloud

Protecting one cluster is easy; protecting a global cloud footprint is hard. Variations in cluster configurations, ephemeral container workloads, and multi-cloud topologies require unified and scalable approaches to policy management and enforcement.

Cisco Security Cloud gives security and platform teams the single point of control to define, deploy, and update runtime policies across all environments from a single interface. This brings consistent coverage, rapid policy propagation, and real-time insight into workload security posture at scale.

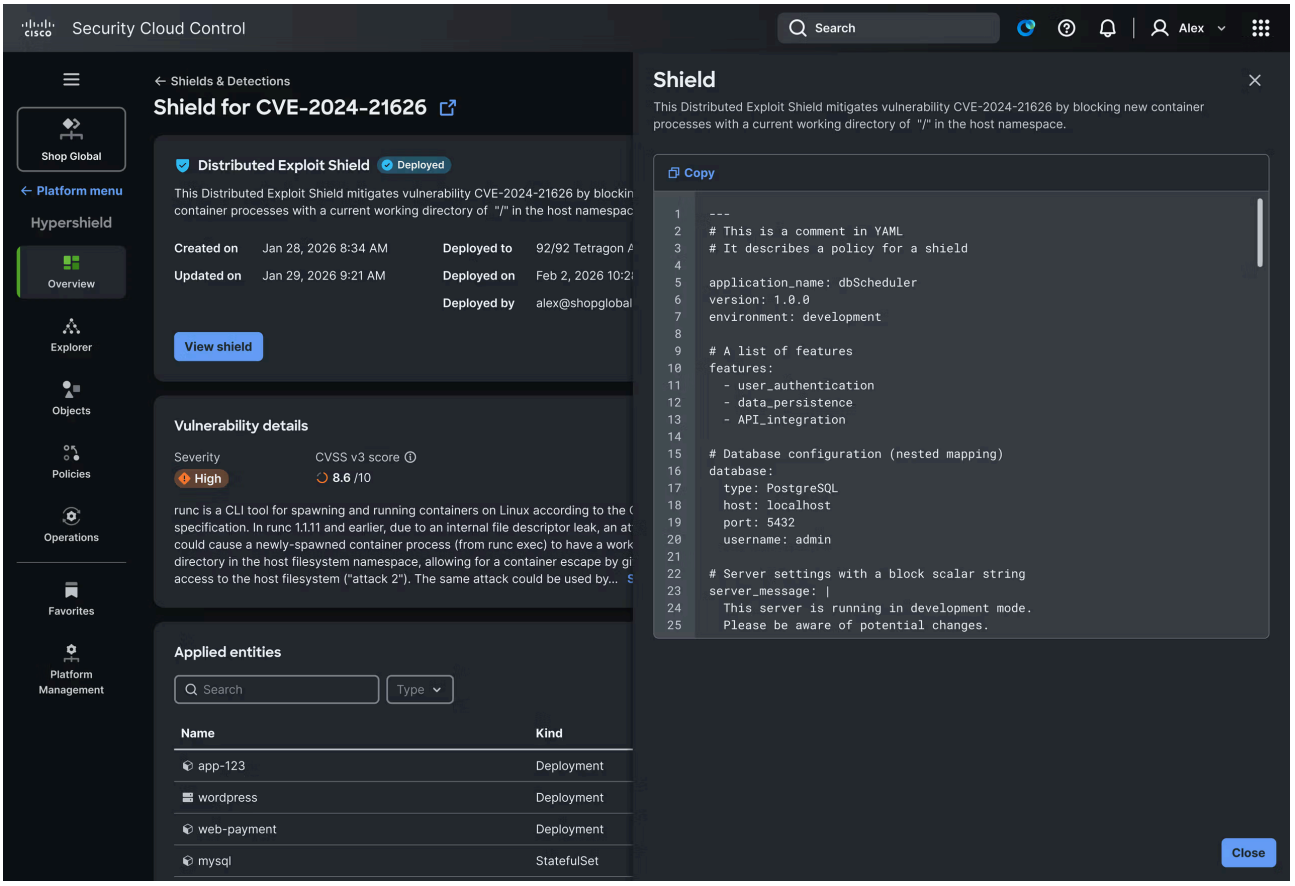
Through the central Security Cloud dashboard, Security teams seamlessly operationalize and audit VoidLink protections across their entire infrastructure, reducing manual overhead and hardening security. As threats evolve, this model allows defenses to adapt fleet-wide in minutes not days or weeks.



In this first dashboard, a security admin sees a list of shields (compensating controls) that protect against known CVE's. On the second line from the top, we see a shield for the Critical severity CVE-2024-47176 ↗. This policy is actively deployed in monitoring mode. With labels, SecOps teams quickly apply tags to each event to filter for further incident response or create downstream automation logic when an alert / enforcement event is triggered. Testing

In just a few clicks, this is the central hub for monitoring and embedding protection policies across the fleet.

In the next dashboard below, we see that the security team has selected the shield for CVE-2024-21626, ↗ a runc ↗ vulnerability for container environments. Here the SecOps team easily sees metadata around the policy itself (creation date, updated on, deployed where, and by whom), details on the specific workloads where it has been applied, CVE details and context, and full transparency of the actual policy (obfuscated with an example policy).



No more security black boxes; teams see and modify security policies across their fleet, having full control over what to observe and where to act.

Learn more and see Tetragon in action

VoidLink provides a glimpse into the evolving threat landscape facing cloud-native environments, where adversaries leverage AI and advanced modular techniques to outpace conventional defenses.

With Tetragon, Cisco delivers real-time, kernel-level security enforcement that powers security teams to detect and stop these threats before they take hold. To learn how Cisco Security Cloud and Isovalent help safeguard your global infrastructure, connect with us!

Do you have any questions?

References

- Overview of VoidLink background and architecture
- Background and development of VoidLink
- Additional background and development
- Technical break down of compiled kernel rootkits
- Targeting of cloud and container environments
- Use of AI to create malware toolkit
- APT and UAT groups using VoidLink
