eBPF Security Monitoring Buyer's Guide

SPYDERBAT

Practical Steps for Evaluating eBPF Security Solutions for Cloud Native Protection

www.spyderbat.com

Table of Contents

Business Case

Why invest in eBPF Security Monitoring?

eBPF fundamentally changes the approach to cloud native networking, observability, and security by enabling in-depth monitoring and analysis of network traffic and system events at the kernel level. This powerful new technology provides unparalleled visibility into the runtime activities of Linux systems and containers.

Gartner estimates that by 2027, “more than 90% of global organizations will be running containerized applications in production, which is a significant increase from fewer than 40% in 2021.” With containerized cloud native environments, applications are distributed across multiple containers and orchestrated by platforms like Kubernetes. This fundamental shift in how applications run impairs security monitoring tool’s ability to capture a stateful view of system and container activities. eBPF, on the other hand, enables real-time, low-overhead instrumentation at the Linux kernel, allowing deep insights into the network stack, system calls, and other critical events. This level of visibility empowers security teams to detect and respond to potential threats more effectively, identifying a wide range of supply chain compromises, insider threats, and external attacks.

Investing in an eBPF Security Monitoring program for runtime visibility gives organizations the ability to proactively identify and mitigate security risks, effectively monitor their environments, and ensure the integrity and availability of their applications and data. With the rapid growth of cloud native architectures, eBPF has emerged as a vital technology for runtime security monitoring in modern enterprises.

What is eBPF?

eBPF is a technology built into modern Linux kernels that allows sandboxed programs to run in a privileged context. In effect, eBPF puts a listening device on another running program to observe its behavior. eBPF programs are verified at load time to ensure that they run to completion in a limited time, and that all the memory access is safe, to avoid Kernel panics and memory crashes.

Since eBPF operates at the kernel level, it provides the ability to see incredible amounts of application execution behavior, making eBPF uniquely suited for solving observability problems that are difficult or often impossible to do by collecting, storing and aggregating various disparate logs from unrelated application systems.

Comparing eBPF to auditd (the traditional Linux auditing framework), shows that while both share some common capabilities such as monitoring for system calls, file access, and other configurable events, auditd falls far short of eBPF for system-level visibility into modern cloud and multi-cloud environments. In particular, auditd:

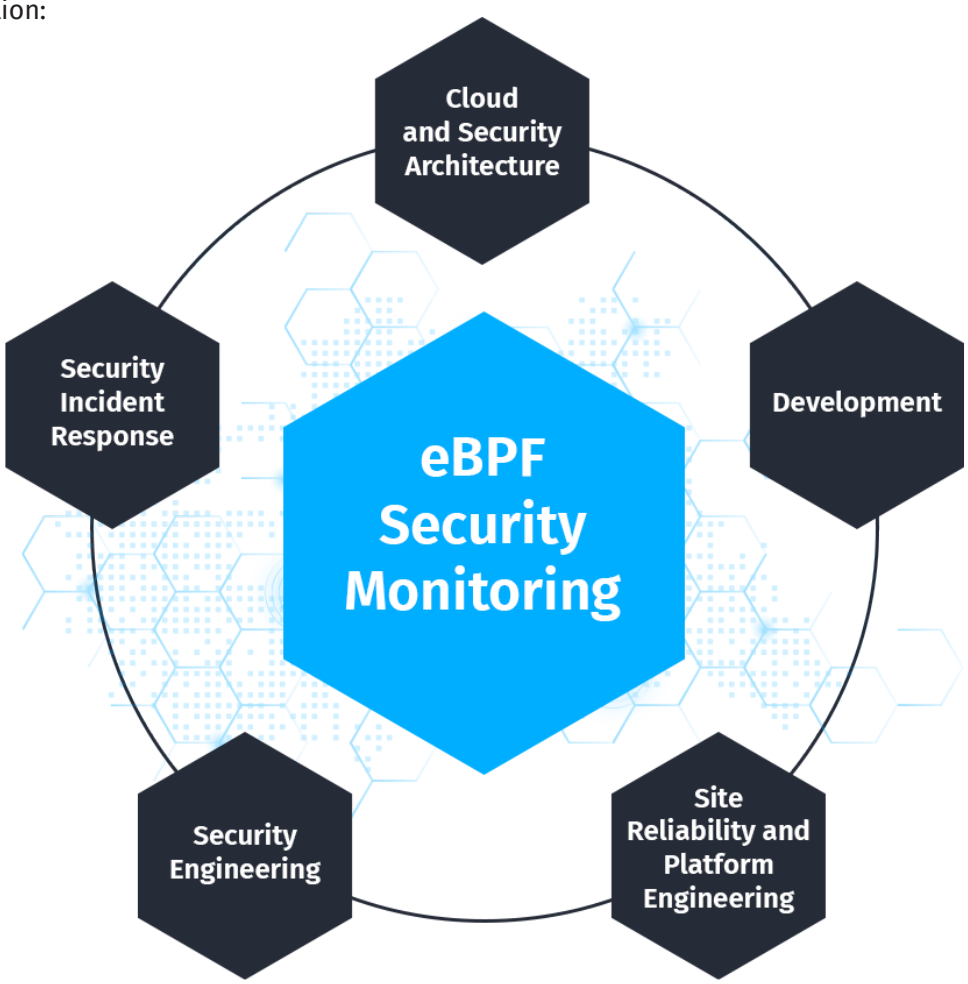
- Creates excessive userspace syscall overhead.
- Often shows invocations such as execveat without revealing what they were called on.
- Is inherently container unaware.

In contrast, eBPF programs run more efficiently than auditd, provide user attribution, and are container aware.

Previous methods to abstract Linux kernel-level data required kernel modification or kernel modules. Altering the Linux kernel raises stability, security, and performance risks. It also requires manual effort to “re-modify” the kernel, with each kernel update creating upgrade challenges and application compatibility conflicts. As a result, eBPF has become the preferred method for Linux instrumentation.

Organizational Impact and Assessing Organizational Readiness

Implementing an eBPF Security Monitoring program impacts different parts of the buyer's organization:



In smaller, 'born-in-the-cloud' organizations, many of these roles and responsibilities are collapsed to a few individuals who operate within engineering. In larger organizations that have these are often distinct teams across engineering, IT, and security groups, possibly with a governing center of excellence. This section will outline the organizational considerations, regardless of company size.

Before implementing an eBPF Security Monitoring program, organizations should assess the impact of monitoring both existing assets and how to account for future feature and application growth.

Cloud and Security Architecture

In most cloud native environments, organizations choose to extend their eBPF Security Monitoring throughout integration, staging, and production environments. There are at least three reasons to focus exclusively on production environments,

- Every environment hosted on a cloud platform includes the same risk of compromise.
- Catch supply chain compromise and insider threats during the development processes.
- Baseline expected application behaviors before moving to production.

To determine the scope of an eBPF Security Monitoring program, teams should:

- Evaluate applications by business risk to the buyer's organization should they become compromised.
- Determine infrastructure coverage for eBPF monitoring (e.g., Linux VMs, managed Kubernetes, etc.). eBPF may not cover serverless infrastructure (e.g., Fargate, Lambdas, etc.)
- Consider which processes (both in terms of workflow and automation) can be implemented to ensure evaluation and monitoring of new functional areas and applications?

Development

Many organizations are shifting security "left," extending some security responsibility to developers and/or DevOps teams who can implement solutions earlier in the development process.

- Will developers/devops validate application and container behaviors as part of their CI/CD process?

Site Reliability and Platform Engineering

Some organizations have elected Platform or Site Reliability Engineering team members to initially triage eBPF Security Monitoring detections since it will not yet be known if the issue is operational or security in nature.

- What responsibility will Site Reliability or Platform Engineering take to triage eBPF security detections, if any?
- How will first responders collaborate with developers, architects, platform engineers, or security engineers?

Security Engineering

The goal of the eBPF Security Monitoring program is to equip security engineers with security context regarding activities in cloud native environments.

- How will eBPF findings be integrated into existing security infrastructure, such as SIEM, SOAR, or ticket tracking systems?

Security Incident Response

The workflow for detections raised by the eBPF Security Monitoring program will need to be defined.

Different workflow examples include:

- Send all eBPF security program detections (regardless of environment) to the security team for initial analysis, then collaborate or escalate to others based on whether the issue appears to be introduced by new runtime code, platform changes, or an external actor.
- Send all eBPF security program detections (regardless of environment) to the Site Reliability Engineering team for initial analysis, then collaborate or escalate to others based on whether the issue appears to have been introduced by new runtime code, platform changes, or an external actor.
- Send eBPF security program detections from integration environments to developers, from staging to site reliability engineers, and from production to security analysts.

To determine an optimal solution:

- Is there a default first responder, or should it vary by environment (e.g., integration, staging, or production)?
- What context is important to first responders (e.g., cloud provider, Kubernetes namespace, internal application name, previous container behavior, etc.)?

Lastly, consider how often workflows should be reviewed and assessed moving forward.

Evaluating eBPF Security Monitoring solutions

eBPF Agent

An eBPF Security Monitoring program requires the use of an agent on Linux nodes running containers (e.g., worker nodes). Solutions should not leverage sidecar or kernel mods/modules given the significant disadvantages to deployment, ongoing management, and visibility. An agent approach provides a means for executing eBPF programs, efficiently collecting the output of these programs, and transferring eBPF program output for centralized monitoring.

eBPF Agent Deployment and Management

Deploying the solution is a critical area of evaluation since in ephemeral cloud and containerized environments, nodes will auto-provision on a regular basis. Evaluation criteria should focus on deployment speed and ease, requiring as few steps as possible prior to seeing value from the solution.

Deployment and Management Worksheet

	Questions to Ask	What to look for
Deployment	• What methods are used to deploy the eBPF Agent? • How quickly will the eBPF Agent begin producing data? • What data entry about the network and Kubernetes environment is required by the operator (if any)?	The solution should support an automated method aligned to the buyer's cloud and/or Kubernetes provisioning method for deployment such as 'helm', Hashicorp Terraform, AWS Cloud Formations, or Ansible. The solution should automatically learn the runtime environments without any data entry.
Management	• Are there different agent types? • Are multiple agents required (network, process, cluster, etc.) • How is the agent upgraded? • How is health information collected about the agent? • Is agent management included with the solution or require third-party services?	Avoid solutions that require different agents for each Cloud provider, Linux distribution and/or kernel version as this creates issues upgrading and expanding. Agents should self-update and be centrally managed within the solution.

Microsoft's project, eBPF for Windows, aims to support eBPF programs on their Operating System.

Endpoint Performance Impact

Evaluation criteria regarding the performance of the eBPF agent sits across five dimensions: CPU, memory, eBPF, data storage, and data transfer.

Endpoint Performance Worksheet

	Questions to Ask	What to look for
CPU	• What is the average CPU load? • Is analysis performed on the endpoint or offloaded to a backend service? • Will more activity equate to higher CPU usage?	CPU load should be <2% during average levels of activity of the node. The solutions' analytics should be offloaded from the buyer's cloud nodes.

	Questions to Ask	What to look for
Memory	• What data is stored in memory? • Is data stored in memory encrypted?	Memory use should be <2.5% during the average activity levels of the node.
eBPF	• What type of data does the eBPF agent capture? • Can the eBPF program lose data during peak times due to buffer overload issues?	With eBPF, it is possible to access exhaustible amounts of detail about Kernel-level transactions. Take precaution, as eBPF will drop system call events if attempting to collect too much data during high levels of activity. Does the eBPF data collected align to meet use cases?
Data Storage	• How much disk space is required on the endpoint? • What happens when disk space is exceeded? • Is stored data encrypted? • Is data compressed? If so, what are average compression rates?	The solution should have customizable disk space controls to prevent overusing data storage. All data stored should be encrypted and compressed.
Data Transfer	• How much bandwidth utilization is required? • Does the agent require communication with a central server? What happens if communication is interrupted? • Is transferred data encrypted? • Is transferred data compressed? If so, what are average compression rates?	Bandwidth should average <100 Kb/ second during average activity levels of the node. In the case of a network interruption, the agent should cache locally to ensure no loss of data. All data should be encrypted and compressed in transit.

Analytics

eBPF and Contextual Data Collection

While eBPF is a powerful data source for observability, observability does not equate to security. To meet security use cases, analytics needs data from eBPF and additional contextual data sources. For this reason, this section is broken into two parts, analytics data and analytics tools. In this first part, we evaluate the eBPF Security Monitoring solution by its level of visibility generated by eBPF and other contextual information.

Analytics Data

It is important to evaluate the eBPF data and contextual data gathered by the solution to meet with use cases.

Analytics Data Worksheet

Captured Activity	Examples
Process information	• Process start time • Process end time • Parent process • Children processes • Complete command line • User and user rights context
Network information	• Directionality • Source and destination IP address/hostname • Fully Qualified Domain Name • Bytes transferred • Connection duration • Related process • Host and container name
User information	• User and effective user rights associated with each process
Host context	• Hostname • IP address(es) • Memory utilization • CPU utilization • OS architecture
Container context	For each container: • Start time • End time • Process details • Network connection details • The image the container launched from
Kubernetes context	• Cluster name • Namespace • Service • Pod • Replica set
Cloud provider context	• Cloud image ID • Instance ID • Region ID • Cloud tags • Identity roles

Analytics Tools

This section evaluates how the buyer's team will interact with the collected data and how the solution identifies areas of concern.

Analytics Tools Worksheet

Available Tools	Questions to Ask	What to look for
Search	• What information is indexed and searchable by the solution? • How are search results presented? • Can the results of multiple queries be combined for analysis?	Search by: • Commands with command line arguments • Commands with user context • Processes with parent process context • IP addresses, hostnames, and fully qualified domain names Combining search results into a common analysis view expedites investigation.
Discover	• Is there a topology view or visualization of the running Kubernetes environment? • Does the visualization support real-time and historic views? What length of history is maintained?	Be cautious of solutions that provide visualizations that work with a few nodes, but become challenging, if not impossible to read with normal levels of activity. Ask the vendor to show real-world usage in their demonstrations. Visualizations should be stateful as of each change rather than periodic snapshots, which may miss key events. The solution should maintain a 90-day history of activity.
Container or Application Profiling	• How granularly does the solution identify application drift at runtime? • How are application drift policies generated?	The solution should be able to capture the exact processes and network details that deviate from validated states at runtime. Solutions that only identify new packages miss ad hoc changes within installed applications (e.g., supply chain compromise). Solutions should suggest policies based on actual application workload behaviors. Be cautious of out-of-box policies for standard applications due to the high levels of tuning required to adhere to policies for actual application use.
Security Detections	• Do security detections follow a known framework? • Does the solution allow customization? If so, what programming level is required?	Security detections should follow a known framework, such as MITRE ATT&CK. Be cautious of black-box machine learning that makes it challenging to understand why detections trigger or that auto-tunes unresolved detections. Complex machine learning models may be more challenging to tune to each application environment.

Additional Considerations

False Positives

While completely avoiding false positives may be an unrealistic goal, the solution should provide means for accounting for each environment in order to reduce false positives without relying on manual tuning.

There are several reasons for false positives:

Type	Details
Criteria is too broad	If the criteria for detecting a security event is defined too broadly, the detection will trigger on innocuous events. For example, the criteria "shell executed in a container" may align to best practices, but in practice, many applications use a shell to execute code.
Anomaly without security context	Machine Learning identifies anomalies that are not necessarily security relevant. For example, the first time an application connects to an S3 bucket may be a true anomaly, but also a legitimate new data source for the application. New events occur frequently in cloud native environments. How does the vendor reduce detecting true anomalies without security relevancy?
Application-Specific Default Policies	Vendors may include application-specific default policies that align with their understanding of how the application should behave. The nature of cloud native development is customization and iteration. Does the vendor include tuning requirements to align application-specific policies with actual usage?

The solution should employ methods for reducing false positives. Some methods are:

- **Risk/Threat scoring:** Risk scores consider other contexts such as the environment, user, etc. to increase or decrease the severity of the detection.
- **Detection chaining:** Chaining individual, atomic detections into a "grouped" detection reduces false positives by corroborating evidence of an attack. Be wary of the methods used to chain detections as it may fall in the 'Criteria is too broad' category of false positives.
- Watch out for systems that "auto-tune" findings with Machine Learning algorithms. Product feedback indicates these systems begin suppressing potentially legitimate concerns simply because human analysts were not able to engage with the findings in a timely manner, effectively 'training' the system to avoid these findings. Be sure to directly control alert tuning to ensure alignment to organizational risk tolerance.

False Negatives

A false negative is a security detection that the solution should have identified but did not. The most common reason for a false negative is the detection criteria are too narrow. For example, an 'Indicator of Compromise' that specifies the file hash for a file containing malware is easily evaded when the threat actor makes a simple change to the file. In practice, false negatives occur from over-tuning rules to become too narrow. When evaluating the methods for managing false negatives (as described previously), consider the impact on false negatives since methods relying on tuning risk higher false negative rates.

Use Cases

While it is important to assess the analytic capabilities of the solution, at the end of the day, the solution needs to solve relevant use cases for the buyer's organization. Below are examples of use cases and how to evaluate the effectiveness of the solution.

Use Case - Supply Chain Compromise

A supply chain compromise is the manipulation of software, or its delivery mechanisms, before receipt by the end user. Modern software development includes dependencies on both third-party and open-source components, creating a greater risk of falling victim to supply chain compromise even when following best security practices.

An eBPF security program can be used to detect evidence of a supply chain compromise early in the development process to avoid the loss of confidential data or other damaging steps. There are minimally two challenges in identifying supply chain compromise:

- Unless previously disclosed, the compromise is a 'zero-day' so vulnerability scanning will not detect it.
- In supply chain attacks, malware is often programmed to wait long periods of time to detonate when used in production environments.

To detect supply chain compromise, the eBPF Security Monitoring solution will need to:

Profile application behavior	Builds a profile of applications at runtime, using the process details, network details, and user context.
Validated application behavior	Enables developers to validate correct application behaviors to avoid accepting compromised behaviors into any baseline.
Identify application behavior deviations at runtime	Recognizes new behaviors (application drift) outside of validated behaviors.
Present validated and deviated runtime behaviors	Presents both the previously accepted workload behavior and deviated behaviors to expedite the investigation.

Use Case - Insider Threat

Insider threats are people with legitimate access to the network who use their access in a way that causes harm to the organization. Examples include a disgruntled employee or a contractor with compromised credentials.

While it is important to implement roles-based access controls in Kubernetes orchestration and cloud identity access management, insider threats may have been provided explicit access or taken advantage of broad access controls. An eBPF security program can identify evidence of insider threat. The challenges of detecting this type of attack are:

- Identifying behavior deviations of legitimate users.
- Identifying correct user attribution when using shared accounts such as ec2-user or root.

To detect insider threats, the eBPF Security Monitoring solution will need to:

Track application behavior with user context	Captures all process details for the authenticated user that triggered the process.
Track changes to effective user rights	Linux commands such as sudo or su allow users to change their effective rights without an authentication event.
Track and connect user sessions across systems or containers	Tracks the original authenticated user even if different accounts are used in subsequent commands or connection methods (e.g., ssh, scp, etc.).
Identifies application behavior deviation at runtime with user context	Presents previously accepted workload behaviors and deviated behaviors with user context (who did what).

Use Case - External Attack

While preventative security methods for patching known vulnerabilities reduce the attack surface available to an external threat actor, it is impossible to completely cleans the environment of all known vulnerabilities. Organizations need to prepare for the inevitable intrusion that evades preventative security measures. Threat actors have varying financial and political motivations, leading to customer and employee data theft, ransomware, cryptojacking, and network disruption.

An eBPF security solution should identify evidence of intrusion to enable quick containment and remediation of attackers that enter the environment. The challenges with identifying external attacks are:

- Reducing false positives while avoiding false negatives.
- Slow and low attacks that span weeks to months.

To detect external attacks, the eBPF Security Monitoring solution will need to:

Detecting behaviors or patterns of known attack tactics	The solution should follow a known framework for identifying attack tactics and techniques.
Connect detected attack tactics across time	The solution should automatically identify connections between attack tactics and techniques.
Suppressing false positives with methods described previously	The solution should reduce false positives to avoid overwhelming the team when responding to real threats.
Present first responders with context	First responders need context of what occurred before, between, and after detections, as well as where, when these detections occur within the environment or across environments.

Automated Response

Due to the ephemeral nature of our cloud environments, and as we enter an age of AI assisted attacks, it becomes increasingly critical that our detection solutions are also capable of automating response.

The effectiveness of the detections is paramount. Without accuracy, it is impossible to enable automation. Confidence in the accuracy of detections enable the transition to automated responses that contain and, in some cases, even fully remediate threats.

Automated Response Worksheet

	Questions to Ask	What to look for
Response Execution	• Where do responses execute? • What permissions are required to enable automated responses?	Automated responses should execute on the worker node itself. Network-based responses risk interference with systems and network-based access controls.
Response Library	• What types of automated responses are available? • Do responses leverage contextual data (e.g., parent processes, namespaces)?	While it is positive for the solution to include a broad set of available responses, ultimately responses need to align with the buyer's specific use cases. Context is important. Killing an individual process may not contain a threat if malware is left running to retry, or the threat actor still has access to backdoors.
Implementation	• Does enabling responses change the agent type and Installation process?	Be cautious if the response requires additional licensing or changes the deployment type.
Customization	• Can automated responses be customized? • If so, what skill or programming level is required?	Solutions should support custom scripts appropriate to the environment and the buyer's use cases.

Third-party integration

From small companies just building their security program to large organizations with established security infrastructure, the eBPF Security Monitoring solution will need to support bi-directional integration into existing and future investments and workflows.

Third-party Integration Worksheet

Questions to Ask	What to look for
Outbound integration • SIEM, SOAR, and ticket management systems • Pagerduty, Slack, Teams	The solution should adhere to each organization's operational workflows, providing full context to their raised alerts with the ability to quickly return to the eBPF Security Monitoring solution for further analysis, when required.
Inbound integration • Kubernetes API • Cloud Provider (e.g., cloud tags, hostnames)	The solution should be able to consume additional context regarding eBPF activities.

Day 2 Operations

Maintenance and Support

Maintenance and Support Worksheet

	Questions to Ask	What to look for
Maintenance	• What levels of support does the vendor offer? • Are professional services included?	The solution should support a maintenance operational model that aligns with the buyer, whether business appropriate time zones or 24x7. Understand the costs to upgrade and expand.
Data storage	• Is the solution SaaS or on-premise? • Are data storage responsibilities the customer's or vendor's? • What are the data retention policies? • Can all data be removed should the service be discontinued?	On-premise storage and management costs can be significant, providing an advantage to SaaS solutions. Ensure SaaS solutions leverage regional data centers and abide by any required data sovereignty regulations.

Open Platform

This document previously covered third-party integrations. Support for custom development is equally important. A bi-directional API enables additional context inputs outside the vendor's supported integrations and the ability to display information appropriately to different consumers within each organization. When evaluating the vendor's API, it is important to understand:

- What is and is not exposed.
- What is and is not documented.
- Includes working examples in the documentation.
- How the API is impacted during upgrades.

Conclusion

An eBPF Security Monitoring program is a critical component for organizations embracing cloud native environments. The adoption of eBPF as a powerful observability tool has the potential to revolutionize security event analysis and incident response by empowering organizations with critical insights at runtime. By leveraging eBPF's capabilities, organizations strengthen their cloud native security posture, mitigate risks, and protect their valuable assets effectively.

Organizations should choose a solution for their eBPF Security Monitoring that excels at identifying security issues, aligns with continuous development workflows and security operations, and integrates with existing security investments.

About Spyderbat

Marc Willebeek-LeMair and Brian Smith stood in the security operations center for a large managed security services firm, observing security analysts respond to alert after alert in a race to meet with the firm's service level agreements. As each alert came in, an analyst immediately delved into logs and audit records in a manual attempt to recreate the scenario and answer fundamental questions - is this a false positive, and if not, what is the scope and entry point of the attack, etc. The manual effort was an exhausting search across esoteric log data manually 'connecting the dots', assuming key events were even logged. Not surprisingly, results often ended inconclusive. The observation led Willebeek-LeMair and Smith to an idea.

Why is every security program facing an uncomfortable tradeoff between accepting high false positive rates, requiring large teams to manually investigate, or tuning signals down and accepting the risk of false negatives? The answer, they concluded, was context.

Alerts trigger from data meeting specific criteria or the parameters of an anomalous outlier, without any understanding of what activities both led to or followed the trigger.

This security formula is ineffective even in traditional networking environments where there are a high number of security controls providing detailed telemetry. What happens in highly ephemeral environments where variables constantly change, and security controls are sparse?

Willebeek-LeMair and Smith realized that cloud-native environments needed a constant state recorder that went beyond audit records - it had to know the causal sequence of activities for every activity.

Spyderbat was founded with this idea.

Using eBPF, Spyderbat constantly builds a Behavioral Context Web that connects every activity to each other along with system, cloud platform, container, and Kubernetes context. Using this Web, Spyderbat simultaneously looks for deviations from 'known-good' workload behaviors while also connecting otherwise disparate security concerns to identify 'known-bad' attack tactics. The result is a cloud-native runtime security solution that removes the uncomfortable trade off by scoring every causal sequence of activities by its risk, exponentially reducing false positives, and surfacing truly concerning sets of activities that programmatic actions can remedy or notify the correct groups across DevOps and SecOps.

Learn more about Spyderbat at www.spyderbat.com