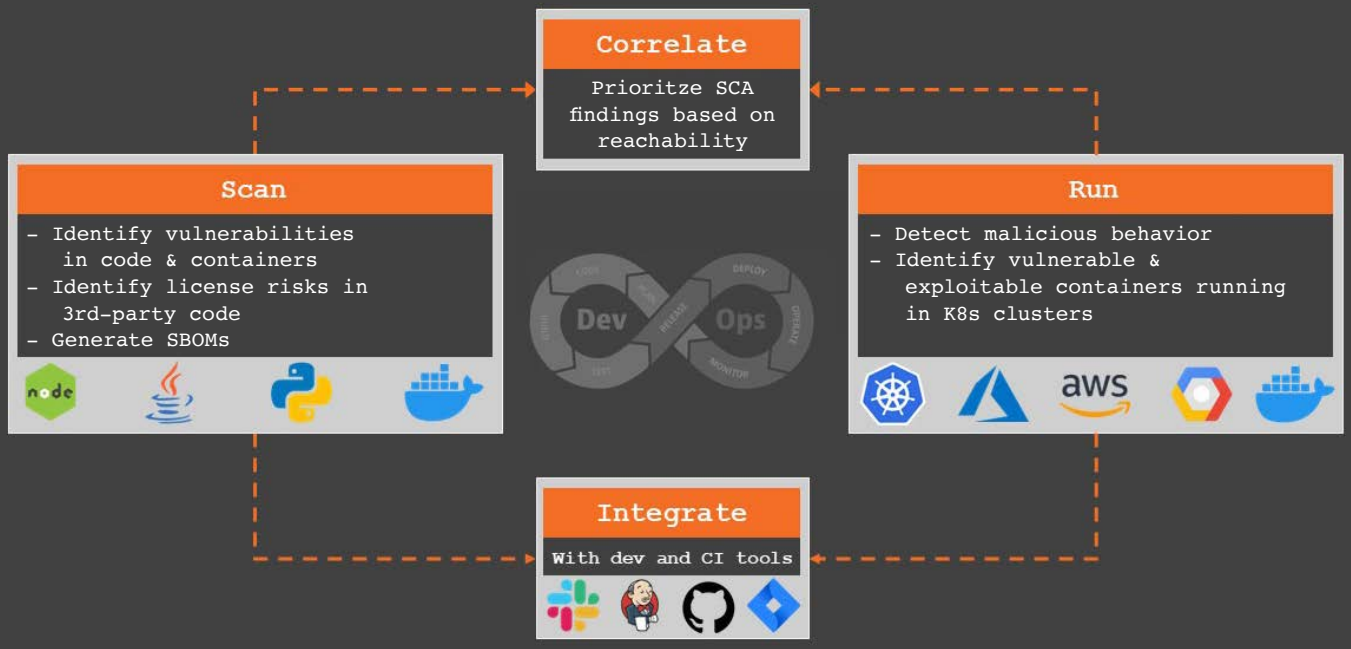
Next-Generation Container Runtime Security

Implementing effective next-generation container runtime security in Kubernetes and cloud native applications.



// Contents

// Executive Summary

This whitepaper provides a comprehensive review of the challenges, advantages, and characteristics of implementing effective next-generation container runtime security in Kubernetes and cloud native applications. In addition to perimeter protection such as Web Application Firewalls (WAF), security teams need to observe and detect high-risk runtime behaviors in containers to effectively mitigate new threats and ensure compliance with regulatory standards such as SOC 2 Type 2 and PCI.

Container runtime security is vital to detecting insecure behaviors across file operations, network communications, process execution, and memory usage. Container runtime security is not only crucial during the development and testing phases for vulnerability identification and remediation before deployment to production but also for continuous monitoring of applications running in production. It provides real-time attack detection and mitigation, helps meet compliance and regulatory requirements, and provides runtime context for development, QA, and security.

Traditional runtime security tools are often challenging to deploy in modern cloud native environments because they have limited deployment options and no correlation with AppSec tools used by developers and in production. Modern solutions like the Deepfactor Application Security platform overcome these issues with increased visibility and protection, flexible deployment models and most importantly, a way to improve actionability in security remediation by correlating behavior and usage with vulnerabilities found in the CI pipeline (by Deepfactor or by other tools).

Deepfactor's approach to container runtime security is a next-generation solution that provides analysis during development and testing, as well as monitoring during production. **Using a patented API Interception technique**, Deepfactor provides rich insights into application behavior without requiring intrusive agents, sidecars, or kernel modifications.

// The Challenges of Implementing Effective Container Runtime Security

What is Container Runtime Security?

Before we delve into the details of container runtime security, let's define the exact scope of what we are talking about.

Container runtime security means monitoring and analyzing activities within containers, enabling the detection of insecure behaviors across various facets like file operations, network communications, process execution, and memory usage. Ideally, runtime security controls should be integrated throughout the application development process, identifying and addressing security risks during development and testing stages prior to deployment in production environments.

Nonetheless, organizations should also adopt a defense-in-depth approach, continuously monitoring applications in production to guard against both new zero-day vulnerabilities and known vulnerabilities that might have been overlooked during development. By actively observing high-risk container behavior in real time, DevOps and security teams can promptly identify and respond to potential security threats, while mitigating application vulnerabilities. This approach facilitates comprehensive visibility into containerized workloads, enabling the detection of malicious activities, unauthorized access attempts, anomalous behavior, and other indicators of compromise.

5 Reasons to Implement Container Runtime Security

1. Detect and Mitigate Real-Time Attacks

Container runtime security facilitates the real-time observation of container processes, allowing for the detection of malicious activities. By analyzing file operations, network communications, process execution, and memory usage, organizations can promptly identify and respond to security threats, preventing potential data breaches or unauthorized actions.

For example, consider a malicious component that was accidentally imported into an application by a developer due to typosquatting or repo hijacking. This component might make unwanted network connections to command-and-control servers, exposing the organization to data exfiltration. Leaking sensitive customer data may lead to reputation loss, loss of sales, or possible civil liabilities for the organization.

2. Identify and Remediate Vulnerabilities Before Deployment

Container runtime security enables organizations to effectively manage vulnerabilities within their cloud native applications. By continuously monitoring container activities in dev, test, and staging environments, it becomes easier to identify and remediate security flaws before applications are deployed to production. In some cases, runtime analysis can help identify security risks in application or third-party code that SAST or SCA tools may not detect. This greatly reduces the number of vulnerabilities reaching production, as well as attack incidents.

3. Compliance and Regulatory Requirements

Many industries have specific compliance and regulatory requirements regarding data security and privacy. Container runtime security helps organizations meet these requirements by implementing the necessary controls and monitoring mechanisms, ensuring that applications adhere to the required standards, and protecting sensitive data. With container runtime security, an application's behavior can be comprehensively cataloged to ensure that sensitive files are not accessed, and if they are, when that happened. A good container runtime security tool will also provide the developer and application security teams with contextual data (like source code modules, file names, and line numbers that triggered the data access).

4. **Defense-in-Depth Approach**

Incorporating container runtime security into cloud native applications provides a defense-in-depth approach to application security. It adds an additional layer of protection alongside other security measures, such as secure coding practices, network security, and access controls. This multi-layered approach reduces the risk of successful attacks and strengthens the overall security posture of cloud native applications.

5. **Reduce Software Composition Analysis Noise and False Positives With Runtime Usage and Reachability**

Container runtime security provides valuable runtime application behavior context to developers, QA, and security teams, enabling them to improve the performance, functionality, and security of the application. For example, a good container runtime security tool can track which modules and dependencies are actually called by the application (e.g., which modules are loaded into memory and which modules may have been imported previously, now unused and forgotten). This information can be used to remove unused OS packages, application components, and dependencies that are not used, thereby reducing the attack surface for security, eliminating the need to test those components, reducing the size of the application, and potentially improving performance.

Key Challenges to Implementing Traditional Runtime Security

Deployment

Traditional runtime security tools originated as host-based agent implementations, deployed on the physical server running the application workload. However, with the emergence of Kubernetes, microservices-based architectures, immutable infrastructure, and ephemeral application environments, full agent-based architectures on each host have become impractical for cloud native applications. This approach hinders development speed to the extent that DevOps teams resist implementing runtime security measures. Modern container runtime security tools address these challenges by utilizing sidecar containers or lightweight plug-ins integrated directly into container images. This simplifies and expedites the implementation of runtime security measures. It's important to note that some runtime security tools may not be compatible with managed Kubernetes platforms or deployments without access to the underlying host, such as AWS Fargate, AWS ECS or Docker Swarm.

Integration and Correlation With AppSec Tooling

Runtime security tools are often inadequately integrated into the continuous integration (CI) pipeline and lack correlation capabilities with AppSec controls such as static application security testing (SAST) and software composition analysis (SCA) scanning tools. This gap inhibits the seamless integration of runtime security measures with the overall application security framework, limiting the ability to proactively identify and address vulnerabilities before they reach the production environment.

> According to a Gartner® report, "With modern cloud-native applications, it can be difficult if not impossible to use a traditional host-OS-based agent approach. In some cases, the DevOps product teams won't accept them, and in other cases, the value of runtime visibility into ephemeral workloads is not offset by the operational overhead of deploying and managing agents."
>
> - ***Gartner***
>
> *Market Guide for Cloud-Native Application Protection Platforms*
>
> *Published 14 March 2023 - ID G00785751.*
>
> *GARTNER is a registered trademark and service mark of Gartner, Inc. and/or its affiliates in the U.S. and internationally and is used herein with permission. All rights reserved.*

Maintaining Agility and Application Delivery Speed

Traditional runtime security tools were initially intended for Ops teams to identify and safeguard applications from attacks in production environments. However, with the advent of DevSecOps principles, the responsibility for discovering and addressing security vulnerabilities has shifted left in the development process, giving development and QA teams the primary responsibility for improving the security of their applications rather than security teams. Traditional runtime security tools are often not accessible to development and QA teams and require leaving their native development and testing tools, resulting in constant requests to the Ops team and slowing down of the development process.

> According to Gartner, "Adversarial relationship between developers and security: Security teams are perceived as slowing down modern DevOps style development. Security controls weren't designed for the speed and scale of cloud-native applications and weren't designed with the developer as the central customer (not security). The result historically has been poorly integrated testing that required the developer to leave their development environment, slowed development and often wasted developer time with false positives or asking them to remediate low-risk vulnerabilities."
>
> - ***Gartner***
>
> *Market Guide for Cloud-Native Application Protection Platforms*
>
> *Published 14 March 2023 - ID G00785751.*

// Key Characteristics of Next-Generation Container Runtime Security

Increased Visibility and Protection

Modern or next-generation container runtime security solutions have expanded their scope to go beyond basic system call monitoring. They now include the capability to observe higher-level operating system API behavior, which is where many new vulnerabilities, such as those found in OpenSSL and Log4J, often reside. This contrasts with other tools that monitor "too low" (at the system call level) to be actionable by the developer.

Flexible Deployment Models

Cloud native applications are deployed using various models, such as self-managed Kubernetes, managed Kubernetes (e.g., Amazon EKS Fargate), non-Kubernetes cloud platforms (like AWS ECS, Docker Swarm, Lambda), and traditional monolithic non-containerized applications. DevOps teams often employ a mix of these models, necessitating seamless integration of runtime security tools across the entire environment. Modern container security solutions offer flexibility by supporting multiple deployment models, including mutating webhooks, embedding the security tool into container images, or launching it through scripts or command lines. This adaptability ensures that container runtime security measures can be effectively implemented regardless of the deployment model employed.

Maintaining Speed Through Integration Into the Dev Pipeline

Next-generation container runtime security tools must integrate into the existing dev and test pipeline and native tools to avoid slowing down the process. Manual review and intervention by the security team or gating of builds that significantly delays releases undermines the objectives of DevOps teams. To maintain the accelerating pace of development, runtime analysis and security testing should be integrated into developer and QA workflows to surface vulnerabilities automatically and file tickets in the dev and QA systems with all the relevant information necessary (stack traces, usage information, links to CVEs and remediation suggestions, for example) to remediate the vulnerabilities so there are no delays to the build process.

> According to Gartner, "There is a desire to integrate security and compliance testing seamlessly and transparently into modern DevOps (referred to as DevSecOps) in a manner that balances security and speed and doesn't unnecessarily slow down digital innovation. Information security's role shifts to one of providing the guardrails across the entire development pipeline, not gates. An analogy would be a racetrack where the guardrails are encountered by the driver only if there is a serious issue. Likewise, developers are allowed to innovate at their desired speed with little or no friction from security, unless a critical risk issue is identified."
>
> - ***Gartner***
>
> *Market Guide for Cloud-Native Application Protection Platforms*
>
> *Published 14 March 2023 - ID G00785751.*

Reducing SCA Alert Fatigue Using Runtime Correlation

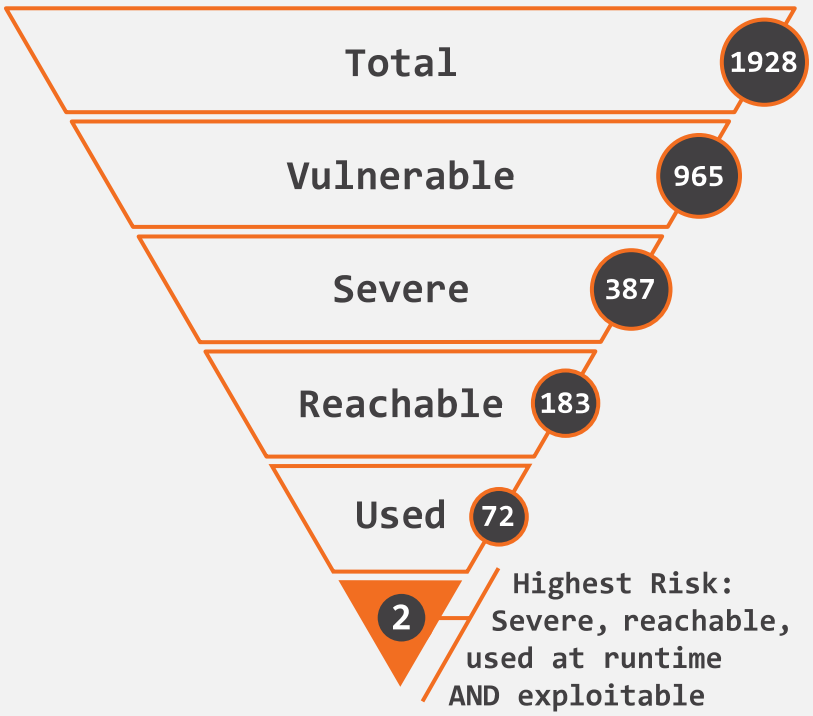
Modern container runtime security tools leverage their unique visibility into the behavior of applications to enhance application security. For instance, they can assist SCA tools in prioritizing which vulnerabilities should be addressed first based on their runtime usage or reachability, down to the level of which class was loaded by a certain vulnerable dependency.

Achieving Compliance Objectives With Runtime Security Controls

Additionally, container runtime security tools play a vital role in ensuring and validating compliance with regulatory requirements and standards like SOC 2, NIST, ISO, PCI-DSS, and others. These tools enable the implementation of necessary security controls to monitor and generate audit logs for activities such as file transfers, use of encryption, privilege escalation, and the presence of critical vulnerabilities in the application.

Using Deepfactor to Prioritize Alerts in the Bank of Anthos Example

The Bank of Anthos example application consists of 11 microservices (split among multiple individual containers), collectively containing 1928 OS (container image) packages and components. Among these, Deepfactor has identified 965 as having known vulnerabilities based on the initial SCA scan of the container images comprising the application. Out of these 965, 387 had a CVSS score of > 8.0. During the development, testing, and production stages, after the app was run with Deepfactor enabled, just 2 of the original 1928 were determined to be severe, reachable, used at runtime (loaded into memory), and exploitable. This filtering process allows the user to prioritize the two highest-risk vulnerabilities.



// Deepfactor Container Runtime Security

Overview

The Deepfactor Application Security platform offers a modern approach to container runtime security that provides runtime security analysis during development and testing, runtime correlation with SCA and container scanning, and runtime monitoring during production.

Runtime Analysis During Dev and Test

Deepfactor identifies unknown vulnerabilities, those that may not yet be CVEs and therefore might go undetected by SAST or SCA tools, by analyzing running applications in the development and test environments before releasing code to production. Based on deep visibility of every thread and process, Deepfactor can detect application risks based on customizable policies that monitor for vulnerabilities such as insecure execution, file system behavior, remote code execution, and buffer overflows that can lead to data breaches or be used by zero-day attacks. A configurable policy engine, as shown in Figure 1, provides the ability to ensure rules defined by the AppSec team are being met.

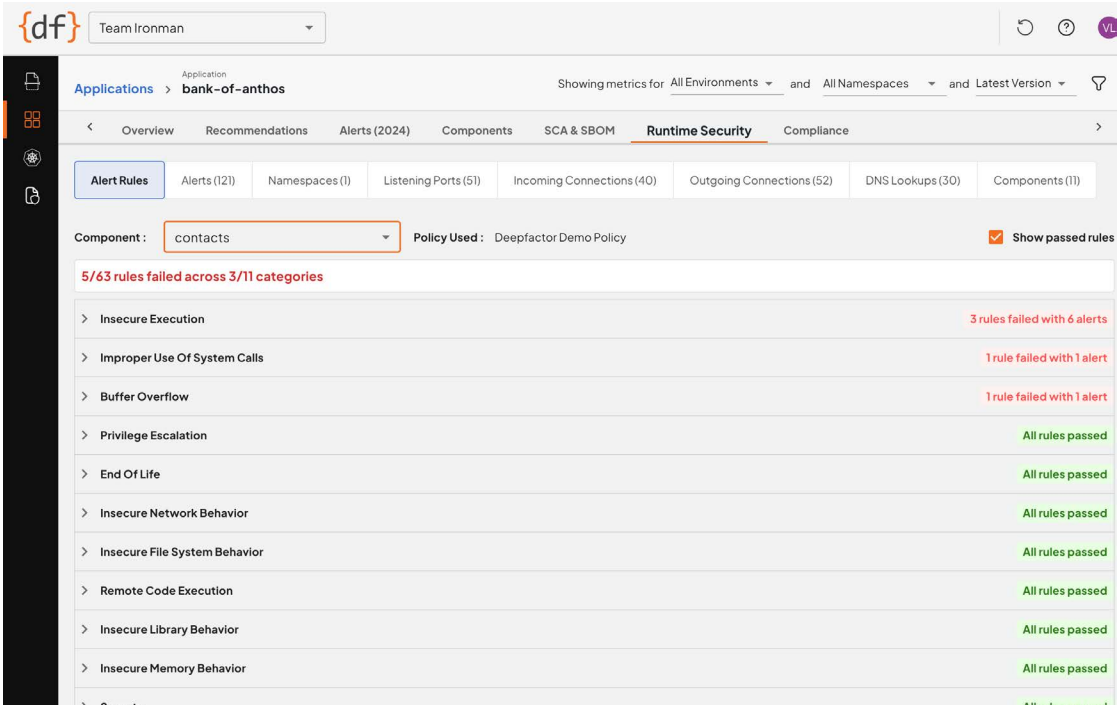


Figure 1 *Deepfactor detecting violations of runtime policy rules*

Deepfactor observes application behavior to determine reachability and runtime usage of dependencies as well as packages installed in container images. It then correlates this information with SCA and container scan output, as shown in Figure 2. This helps AppSec teams prioritize which vulnerable components need to be fixed and which ones could be considered for removal based on lack of usage, resulting in a highly effective way to rapidly resolve CVEs. To learn more, read this whitepaper: **SCA 2.0: A Framework to Prioritize Risk, Reduce False Positives, and Eliminate SCA Alert Fatigue.**

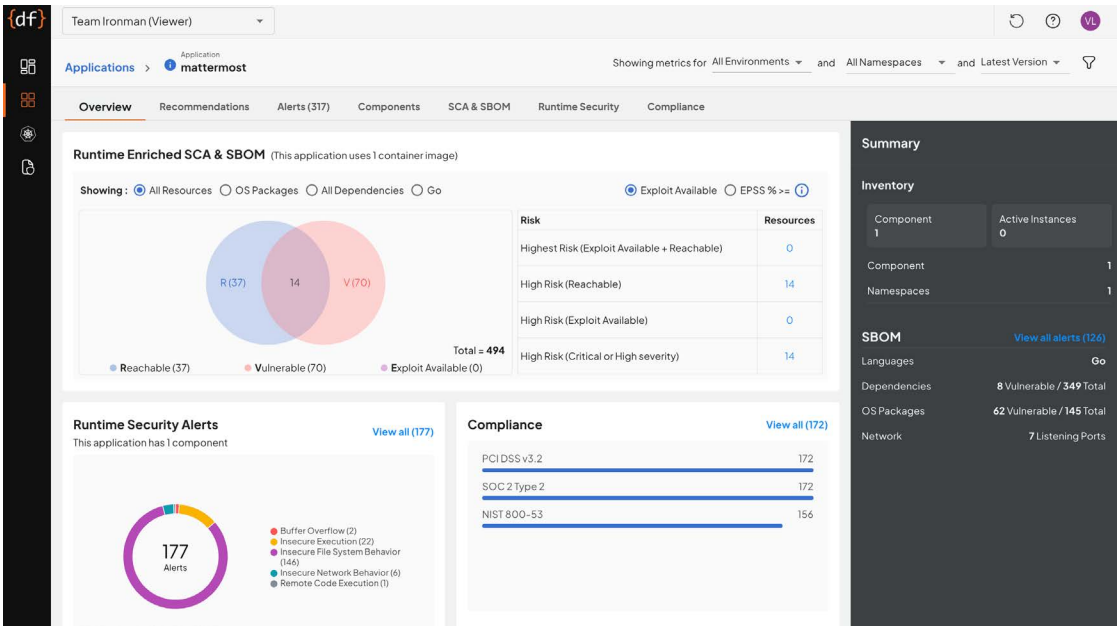


Figure 2 *Prioritization of SCA results with runtime usage analysis*

Runtime Monitoring During Production

Deepfactor's approach to container runtime security monitors applications in production environments to detect runtime security risks in filesystem, network, process, and memory behavior including exposing sensitive information, privilege escalation, and prohibited network communications. Runtime security in production environments helps demonstrate compliance (SOC 2, etc.) and uncovers indicators of compromise by pinpointing suspicious file, network, and memory behaviors. These risks can be configured to raise alerts if seen in runtime, based on policies set by the AppSec team, as shown in Figure 3.

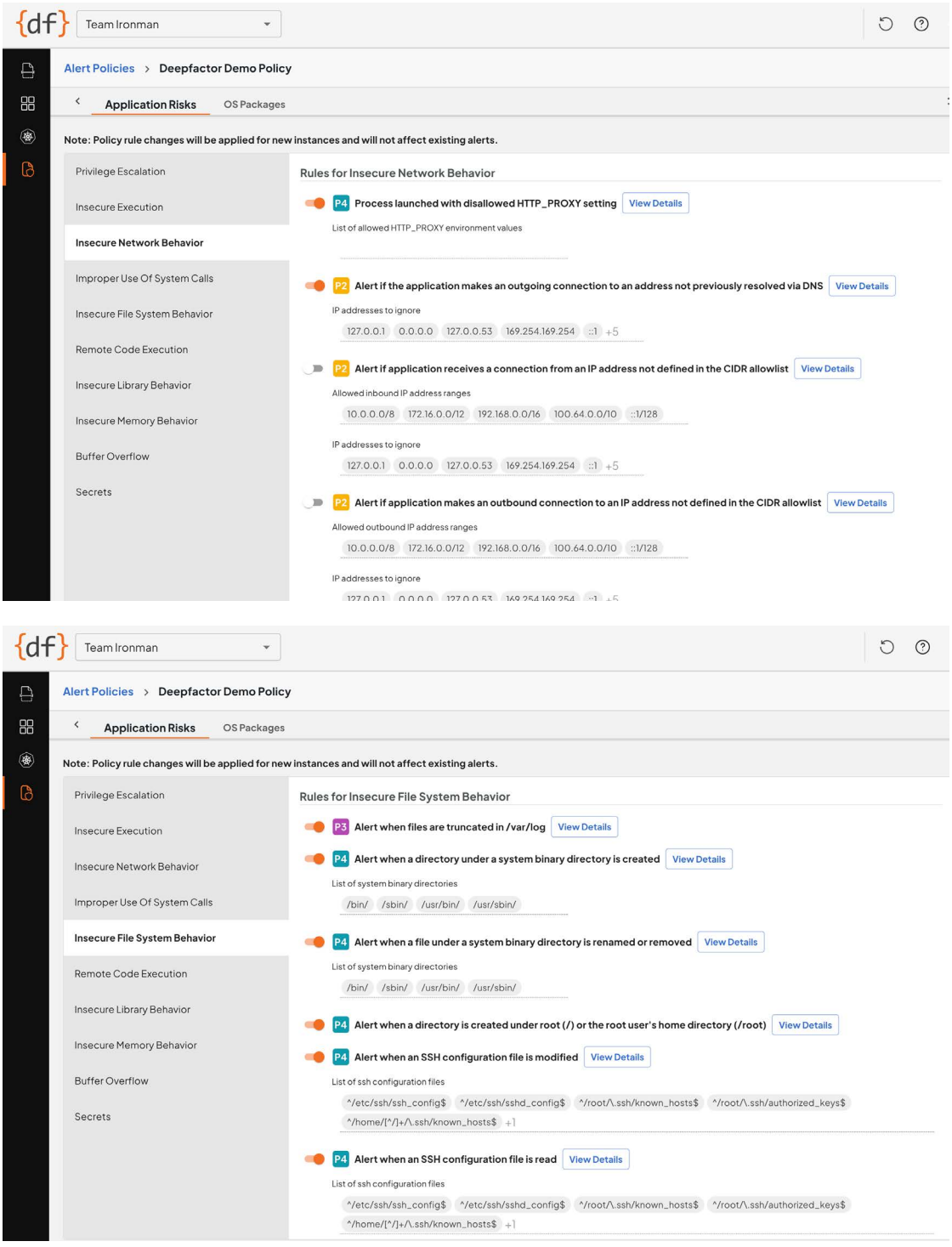


Figure 3 *Configurable runtime alert policy rules*

Increased Visibility Through API Interception

Deepfactor has developed a unique, patented approach to API interception that provides rich insight into application behavior to help developers, QA engineers, and security teams find and fix vulnerabilities faster. Deepfactor dynamically injects a small library into the user space of each container's operating system that monitors hundreds of system calls, library calls, and Web APIs. In addition, Deepfactor auto-detects languages and inserts language-specific interception that provide a deeper level of analysis (to provide language-aware stack traces for Java and Python, for example). Deepfactor is able to correlate the low-level telemetry information with higher level language-specific information to not only detect vulnerabilities but also provide pinpoint locations in the code where the developer can focus. This makes it easier for developers to identify the root of the issue and remediate it quickly, without having to manually triage and debug the issue. This approach not only saves developers time but also motivates them to adopt the tool since it provides results with minimal effort.

Read the whitepaper **Gain Security Insights by Observing Application Behavior via API Interception** for more detailed technical information.

Deployment Flexibility

With a single command, Deepfactor seamlessly loads a robust language-agnostic library into cloud native workloads and environments to provide comprehensive container runtime security. Deepfactor can be deployed in the cloud or on-premises with self-managed Kubernetes, managed Kubernetes (e.g., EKS Fargate), non-Kubernetes container orchestration tools (e.g., Amazon ECS, Docker Swarm, Lambda), and traditional monolithic non-containerized applications. With Deepfactor, there are no host-based agents to install, no sidecar containers, and no kernel modules to install, simplifying and accelerating deployment.

For example, to scan a container image, populate an SBOM for that container, and show vulnerabilities in the components in that container, a single command (shown in Figure 4) can be run. This will scan the container and make the results available in the Deepfactor management portal. The SBOM can then be exported into a variety of standardized formats.

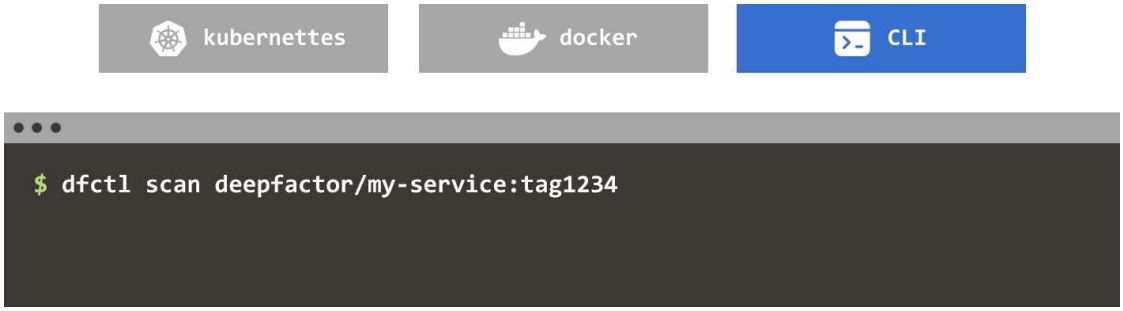


Figure 4 *Scanning a container image to produce an SBOM and SCA report*

Kubernetes Deployments

For Kubernetes deployments, Deepfactor provides a mutating admission controller (shown in Figure 5) which seamlessly adds Deepfactor's interception library into the application workloads. With this more seamless approach, the user does not need to modify their applications or their deployment YAML files (or run the manual scan command shown in Figure 4, since Deepfactor deploys a scan pod which automatically scans the container images used in application pods). Deepfactor provides flexible configuration options for the webhook that allow the user to select which of the workloads should be observed with Deepfactor and how pods should be grouped into applications. For more information on how to observe K8s workloads with Deepfactor, refer to **this article**.

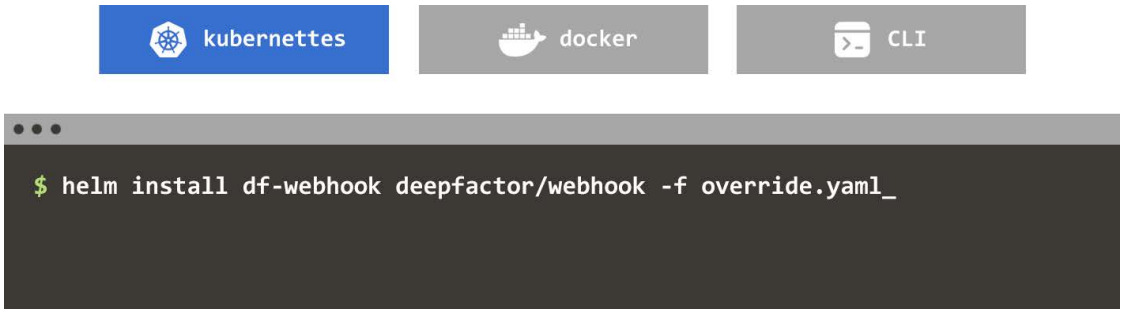


Figure 5 *Installing Deepfactor's Kubernetes admission controller*

Non-Kubernetes Container Orchestration Tools

For those who use other container orchestration tools like AWS Elastic Container Service (ECS), Docker Swarm or even running simple container images, Deepfactor provides a Dockerfile which does a multi-stage build and generates a Deepfactor-enabled container image. This instrumented image can then be run anywhere, such as AWS ECS or Docker Swarm and the instrumented container will send telemetry to the Deepfactor portal. This strategy provides portability across different container orchestration platforms. For more information on how to add Deepfactor to your container images, please refer to **this article**.

If the container image has already been built (but lacks Deepfactor), it can be run using the Deepfactor CLI as shown in Figure 6. For more information on how to run your pre-built container images with Deepfactor, please refer to **this article**.

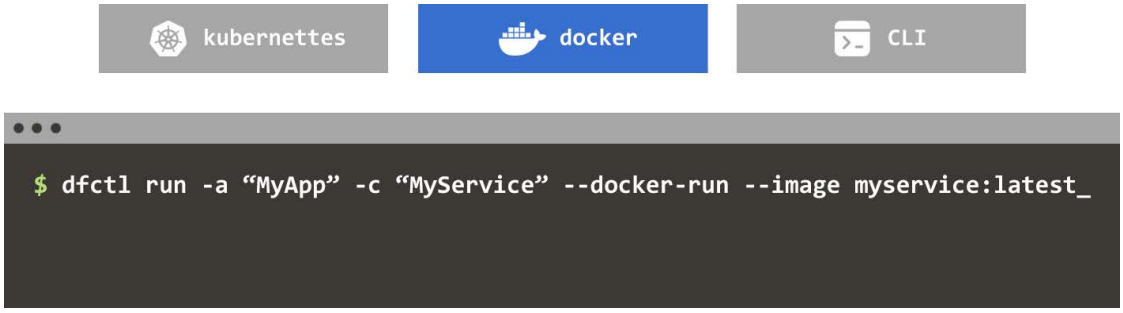


Figure 6 *Running a prebuilt container image using dfctl*

Monolithic Application Deployments

Deepfactor also supports monolithic/non-containerized applications by using the 'dfctl' command line tool, as shown in Figure 7. This command launches the specified command line with Deepfactor; runtime analysis in this case is grouped under the "my application" hierarchy in the Deepfactor management portal.

```
dfctl run -a "my application" -c "my component" --version "1.1.1" -p "Built-in
Policy- Max Alert" -v --cmd COMMAND_WITH_ARGUMENTS
```

Figure 7 *Running a monolithic/non-containerized application using dfctl*

Correlation With AppSec Tooling

Unlike traditional container runtime security tools, the Deepfactor Application Security platform correlates runtime usage behavior to filter and prioritize vulnerability alerts, as shown in Figure 8. Deepfactor's correlation engine observes the application while it's running in test, dev, staging, and production environments to determine whether vulnerable dependencies and packages are actually loaded in memory and used by the application at runtime. This provides valuable context that helps prioritize vulnerabilities, accelerate remediation, and offer guidance on slimming down unused components.

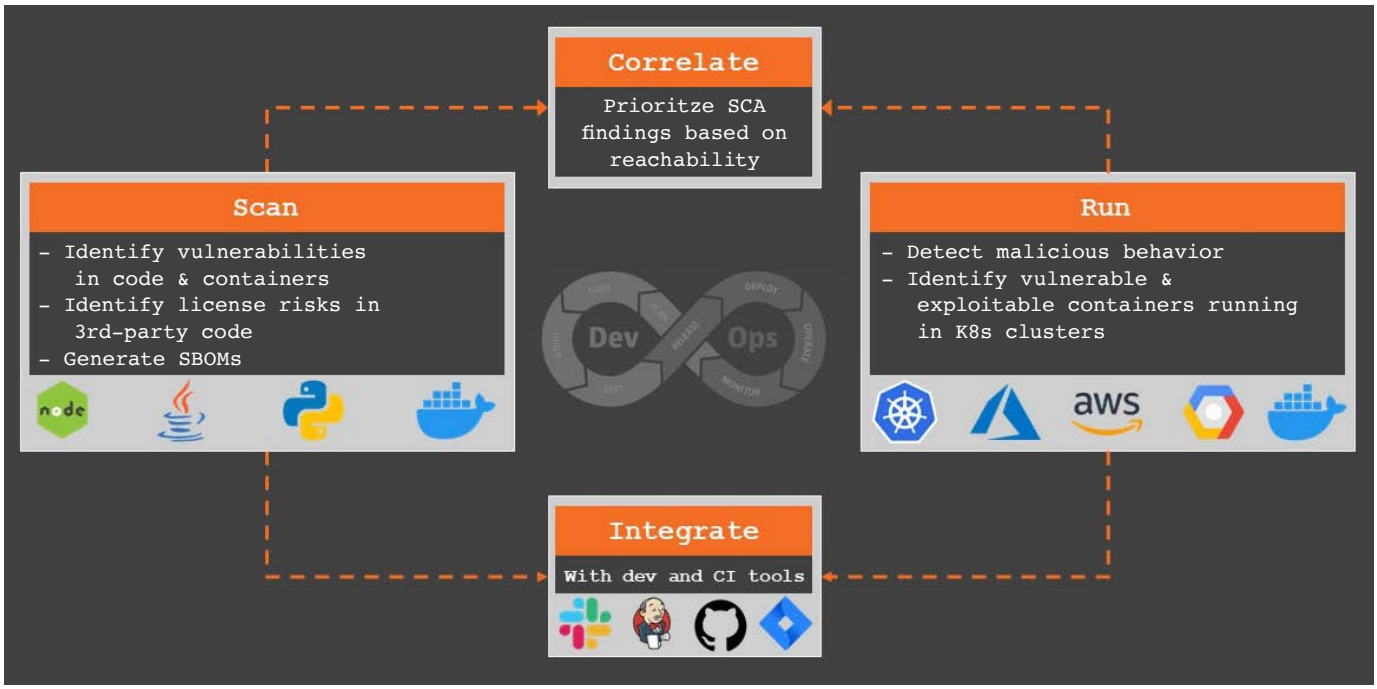
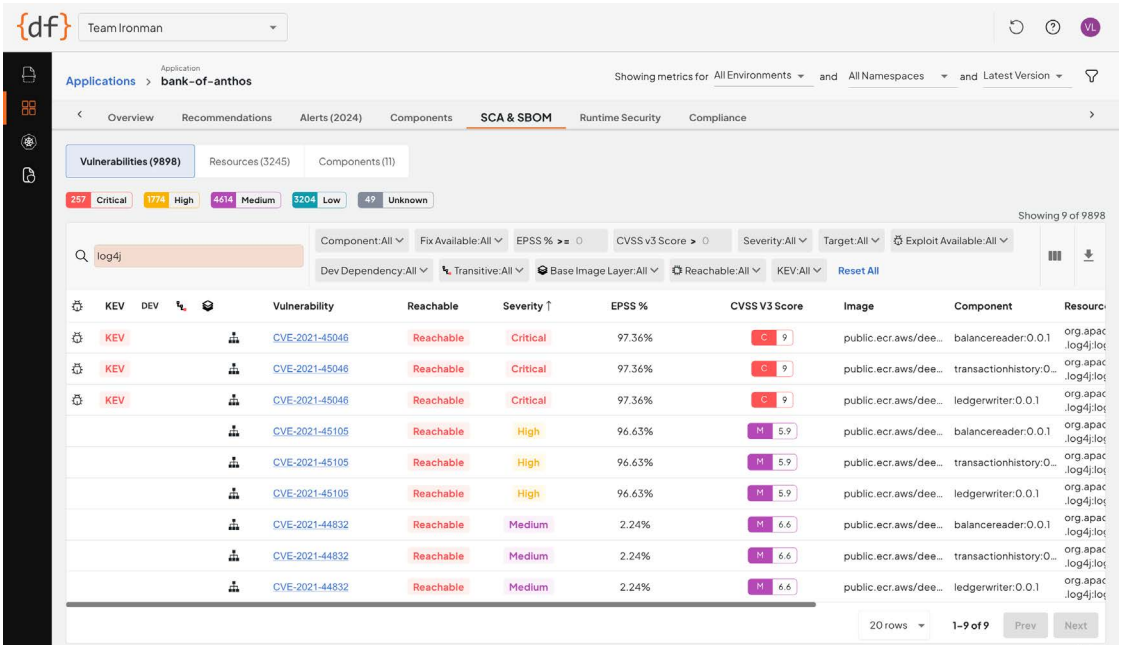


Figure 8 *Continuous runtime and correlation-based reports*

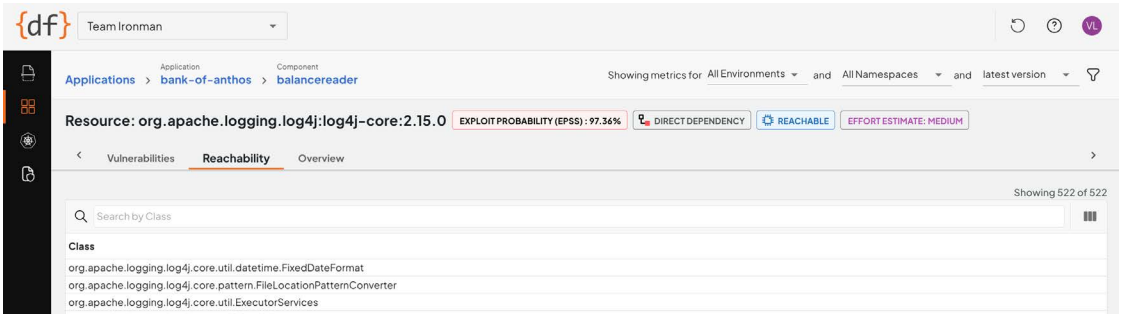
Log4j Example: Dependency Runtime Context

Bank of Anthos is a sample HTTP-based web app created by Google that simulates a bank's payment processing network, allowing users to create artificial bank accounts and complete transactions. The application contains nine common microservices and is written in a combination of Python and Java with PostgreSQL databases. The application contains many known vulnerabilities that make it ideal to use as an example application to illustrate the importance of prioritizing and filtering vulnerabilities.

In December, 2021, the Log4j vulnerability surfaced. The Deepfactor platform determines which containers (or subcomponents) have the vulnerable version of Log4j installed.



In the Dependencies tab, you can see that the Bank of Anthos application has the Log4j vulnerability in nine of its services. Deepfactor detected that all of these services indeed loaded the Log4j dependency into memory at runtime.



Deepfactor provides even deeper insight into the usage of this dependency. In the Log4j dependency details page, Deepfactor provides the list of classes within Log4j which were actually used at runtime. Developers can search for the classes where the specific CVE exists and determine if those classes are being used by their application. In the case of Log4j, the CVE didn't have that class information, which means the fact that Log4j loaded at least one class should be enough reason to remediate the vulnerability. For CVEs where this information is available, a developer could use it to further prioritize remediation.

Read the Deepfactor Blog on Log4j.

// Compliance Customer Case Study: Open Lending

Open Lending (NASDAQ: LPRO) provides loan analytics, risk-based pricing, risk modeling, and default insurance to auto lenders throughout the United States. For 20 years, Open Lending has been empowering financial institutions to create profitable auto loan portfolios with less risk and more reward. Before going public in 2020, Open Lending ranked among Austin's fastest-growing, privately held companies.

Using Runtime Security in Dev and Test

Open Lending uses Deepfactor across dev, test, and production environments. In dev and test, the focus is on eliminating known vulnerabilities before the application is shipped to production. There is also an opportunity to remove any unused packages or dependencies to slim down and clean up the containers as much as possible to reduce the attack surface of the application.

In the on-demand webinar **Container Runtime Security: Detect Malicious Application Behavior & Comply with SOC 2**, Jeff shares best practices and lessons learned based on Open Lending's implementation of container runtime security and their compliance program.

> "If you have a set of tools or dependencies within an application and unused components have vulnerabilities, you should ask yourself if you truly need that component within that image or artifact. There's an opportunity to pull that out and reduce risk."
>
> - ***Jeff Deverna***
> *Vice President of Cyber Security*
> *Open Lending*

Using Container Runtime Security to Save Security and Engineering Time

Open Lending uses the container runtime security functionality provided by Deepfactor to monitor multiple applications and clusters for security incidents that require further investigation. Deepfactor correlates events across all their application components and clusters to determine whether there is a potential security incident and provides information that the security team uses to prioritize investigations and remediation activity. With all the runtime context correlated within the alert, engineers don't have to waste any time combing through logs and history to determine the cause of the issue and recommended remediation steps.

> "Having the proper runtime security tool in place will make it easier for your security and engineering teams to collaborate and assess your application posture."
>
> - ***Jeff Deverna***
> *Vice President of Cyber Security*
> *Open Lending*

Regulatory Compliance in Kubernetes and Cloud Native Applications

Open Lending is required to comply with the SOC 2 Type 2 regulation. As a result, they needed to implement a solution that could monitor the runtime security of applications in the production environment and detect threats such as privilege escalation, unauthorized file transfers, and file changes. In addition, all access and modifications around file, network, memory, and API calls need to be auditable to prove compliance with the relevant regulatory requirements. Deepfactor provides a mapping of all security alerts that includes which, if any, specific sections of specific requirements a particular vulnerability may impact. Using this information, the Open Lending security team can raise the priority to fix a vulnerability that may negatively impact their upcoming audit, avoiding regulatory penalties or negative audit findings.

> "A lot of auditors are starting to get more integrated within Kubernetes and cloud native environments and how what you have may or may not translate from legacy or traditional environments such as virtual machines and physical servers."
>
> - ***Jeff Deverna***
> *Vice President of Cyber Security*
> *Open Lending*

The Engineering Benefits of Increased Visibility With Deepfactor

As part of its container runtime security capabilities, Deepfactor observes hundreds of application event types spanning file, network, memory, and API calls that not only help uncover high-risk behavior and malicious activity but also uniquely valuable information to the engineering team about what the application is doing in production. Developers can use that same information to improve their application performance and functionality or reduce the attack surface by slimming down the application based on the runtime information they have access to through Deepfactor.

> "There are things that shouldn't happen if you are in a containerized or Kubernetes environment. Did someone drop a shell, make a permissions change on a file or executable? There shouldn't be changes made to production containers without approvals and following change management procedures. Container runtime security can help you reinforce those best practices."
>
> - ***Jeff Deverna***
> *Vice President of Cyber Security*
> *Open Lending*

// Conclusion

In a rapidly evolving digital landscape, container runtime security remains pivotal for organizations to ensure comprehensive security, mitigate potential vulnerabilities, and achieve regulatory compliance. As traditional runtime security tools grapple with adapting to the complexity of cloud native applications, Deepfactor is purpose-built for cloud native container runtime security, delivering extensive visibility, flexible deployment models, and comprehensive correlation with application security tools and compliance frameworks.

Deepfactor's unique API interception technology, devoid of intrusive agents or kernel modifications, dramatically simplifies the deployment of container runtime security. Its unique runtime analysis and security testing approach aligns with DevOps principles and workflows, delivering security that doesn't impede the agility and speed of application delivery. By providing critical insights into application's runtime usage and reachability, vulnerabilities can be efficiently prioritized and remediated, reducing the attack surface, bolstering security by burning down CVE debt, and accelerating the delivery of resilient applications.

Real-world case studies, like Open Lending, highlight the effectiveness of the Deepfactor approach, with the solution helping to identify vulnerabilities, save precious engineering time, and maintain regulatory compliance. By effectively integrating security into the development process, modern container runtime security solutions like Deepfactor are paving the way for safer, more secure cloud native applications and Kubernetes environments. As the digital landscape continues to evolve, so too will the role and importance of next-generation container runtime security.

To see all of the Deepfactor Application Security capabilities today, you can:

Request a Demo

Start a Free Trial

Check out this **related webinar**, with guest speaker Jeff Deverna, Vice President, Cyber Security, Open Lending

{deepfactor}

Deepfactor is an application security platform that combines SBOM, software composition analysis, container scans, and container runtime security into a powerful integrated platform. With Deepfactor's unique runtime software composition analysis, you can now correlate static scans with runtime analysis, and prioritize risks based on reachability with actual usage.