
# DevSecOps and Detection Engineering: New Approaches To Security

Jamie Lewis, Chenxi Wang

December 2018

RAIN CAPITAL

# DevSecOps and Detection Engineering: New Approaches To Security

**A Rain Capital Research Note**

Jamie Lewis, Dr. Chenxi Wang

## Introduction: DevSecOps in a Cloud-Native World

The rapid move to the cloud is driving significant changes to application development models and operational processes. DevOps and Continuous Integration/Continuous Deployment (CI/CD) lead to higher degrees of automation, while containerization, microservices, serverless computing, and the more recent advent of 'the service mesh' enable faster deployments, more dynamic execution environments, and rapid scale.

These changes don't just challenge the relevance of the traditional data center. They challenge enterprise IT culture at its core. As organizations adopt cloud technology stacks and DevOps models, the role and priorities of the IT professional must evolve as well. Security is no exception, and the move to cloud-native is having a profound impact on both security posture and operations, introducing the following issues:

- **Distributed architectures create new challenges:** Diverse components interacting within a distributed architecture introduce unpredictable dynamics and unanticipated failure modes.
- **Ephemeral workloads challenge static security approaches:** Dynamic and short-lived work loads require security controls that can change and adapt as quickly as the environments in which they run.
- **The DevOps mindset is upending the status quo:** The DevOps model is challenging organizing principles that have long driven security operations in many enterprises.

While some security professionals have been slow or resistant to change, others are embracing these challenges, seeing an opportunity to apply DevOps technologies to security, and to blend development and security operations. **Hence, the term "DevSecOps."**

While the term doesn't exactly roll off the tongue, "DevSecOps" encapsulates two important trends. From an organizational standpoint, DevSecOps involves decentralizing operational security roles and functions, leading to what some have called the "SOC-less enterprise." On the technology side, it utilizes DevOps techniques to bring security controls into alignment with the cloud-native stack.

## The Rain Forecast: Cloud-Native Demands Realignment of Security Architecture and Priorities

***Bringing security practices and technologies into architectural and organizational alignment with the systems they are meant to protect is one of the most important tasks of enterprise security today.***

Simply put, organizational silos and traditional security architectures cannot support the business effectively as it moves to cloud-native platforms. To protect business assets in cloud-native environments, organizations must establish new methods, capabilities, and instrumentation. Any potential solution must include the following functions:

- **Extensive, real-time visibility:** Partial or after-the-fact visibility will not suffice. Both the infrastructure layer and applications, wherever they are, must be visible.
- **Rapid, iterative feedback loops:** Feedback loops allow security measures to adapt continually to rapidly changing environments.
- **An engineering approach to solving security problems:** Automation, continuous measurement, and controlled experimentation will be the predominant method of solving security problems across the enterprise, replacing manual analysis and control updates.

**DevSecOps embodies this new approach.** It also represents a significant opportunity to build security into much earlier stages of the development and deployment process. That's something security professionals have long aspired to, moving the often adversarial relationship between security practitioners and business application (and risk) owners to a more collaborative footing.

**Detection engineering is a pivotal aspect of this alignment.** Detection engineering uses automation and leverages the cloud-native stack to discover threats and orchestrate responses before they can do significant damage. As part of a move to DevSecOps, detection engineering can improve security posture. It also has implications for how enterprises organize security programs and people. Consequently, organizations making the transition to cloud-native architectures should consider how and when to incorporate detection into their security programs.

# A Primer on Cloud-Native Systems

As distributed systems, cloud-native architectures are both complex and dynamic. Some of the fundamental components of the cloud-native architecture are microservices, containers, and orchestration; serverless computing; and the service mesh.

**Here's a quick primer on these cloud-native concepts:**

1. **Microservices, Containers, and Orchestration:** The microservice architecture is a methodology for developing and deploying distributed applications. Each service represents a discrete component of the application. Decomposing a monolithic application into microservices allows organizations to develop, update, and deploy components independently of each other. Development and subsequent updates can move more rapidly, making the organization more agile.

2. **Containers, such as Docker, have emerged as the infrastructure to deploy microservices.** **Kubernetes**, the leading container orchestration system, can automatically deploy, scale, and maintain containerized microservices. Kubernetes has built-in management and security plugins that allow administrators to specify pod-level or network-level policies.

3. **Serverless computing:** In the serverless execution model, developers deploy applications in a pay-per-execution, just-in-time-resourcing model to a new kind of cloud platform. Amazon's **Lambda**, for example, is an "event-driven" platform that "runs code in response to events and automatically manages the computing resources required by that code". Developers can focus on the core application, while scalability, availability, fault tolerance, and server management are all built into the infrastructure. Serverless applications include scripts that allow service providers to dynamically provision resources at run time. This more efficient utilization of server resources leads to lower costs and more scalable computing.

4. **Service mesh:** A recent innovation, the service mesh provides an abstraction layer for networking and security between microservices. Service mesh functions include service discovery, load balancing, and rules-based routing that allow it to dynamically manage the communications between services. With additional capabilities such as isolation of failing instances, strong authentication, authorization, and real-time metrics, the service mesh is poised to become the primary fabric for running and managing cloud native systems. **Istio** is an open source service mesh implementation that grew out of internal projects at Lyft, Google, and IBM. Version 1.0 of Istio was released in July of 2018.

## Cloud-Native Security Implications: DevSecOps

The combination of containers, microservices, and orchestration frameworks has ushered in a new era of application development and deployment, creating significant implications for enterprise security. In the cloud-native environment, the notions of an "application" running on a "machine" in a persistent "state" are obsolete. The application, or service, is now a distributed system, consisting of multiple components running on a highly variable number of nodes, in a nearly constant state of change. Traditional security controls that rely on machine isolation and a predictable system state are ineffective. Security policies that are blind to service-to-service communications and controls that lack horizontal scalability simply cannot keep pace with today's microservice applications.

In cloud-native and DevOps-driven environments, security controls must be both agile and scalable, providing new capabilities that match the new environment. **DevSecOps, then, is a logical step in this evolution, applying the lessons of DevOps and SRE to security.** Just as DevOps enables continuous development and deployment pipelines, **DevSecOps must enable continuous security pipelines.**

This means making security engineers accountable for the operational implications of security controls. It also means engaging DevOps people in security pipelines, blurring the distinctions between the security operations center (SOC) and DevOps. It means building new controls, real-time metrics, and rapid feedback loops. Accomplishing these goals requires realignment in both the architectural and organizational domains.

Detection engineering is an example of these new capabilities, one that organizations can apply to their increasingly cloud-native environments.

# Detection Engineering

**Detection engineering is the continuous process of deploying, tuning, and operating automated infrastructure for finding active threats in systems and orchestrating responses.** Indeed, both the terms "detection" and "engineering" carry important connotations when it comes to the new approaches to security we're discussing:

1. **Detection:** The debate over preventative versus detective controls isn't new. The key is finding the right balance between the two given an organization's risk profile. But most enterprises have invested significantly more in prevention than they have in detection. In fact, much of the cybersecurity industry has focused on the creation, marketing, and sale of prevention technologies and products.

   **The Challenge:** But products focused on prevention are failing. Repeatedly. Insider threats, social engineering, zero-day attacks, determined and state-sponsored attackers, and many other factors have made an over-reliance on prevention a losing bet. It's simply smarter, and more effective, for security managers to focus on detection rather than attempting to build impenetrable systems.

2. **Engineering:** The goal of any successful alerting system is to separate signal from noise, distilling meaningful and actionable alerts from the collection of event information, moving them up the chain for remediation and response. In a typical security operations center (SOC), analysts process those alerts, determining their severity and whether to escalate them to a higher-level analyst or incident response team. Processing alerts involves compiling contextual data (who, what system, how it's used, what roles, and so on), filtering according to some or all of that contextual data, comparing events with threat feeds, and assembling a coherent picture of what happened--all before deciding what to do about the alert.

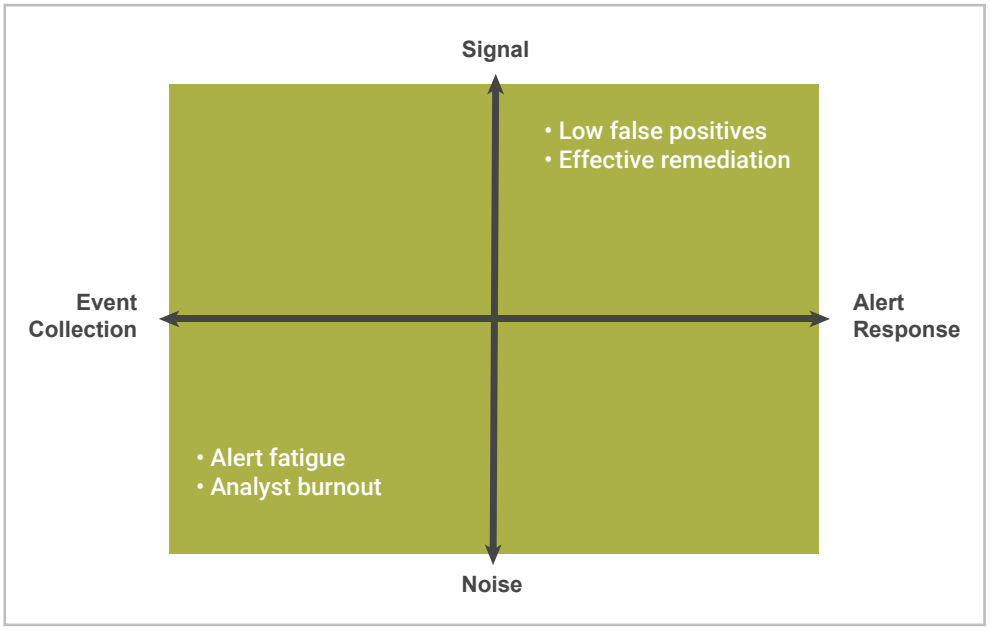


**Figure 1: Security Alert Matrix**

**The Challenge:** However, given the sheer volume of alerts and event logs complex systems can create, this is a staggering task at best. Overwhelmed by mountains of busy work, security programs suffer from analyst burnout and **alert fatigue**. As analysts become desensitized to the staggering data load, **real problems slip through their fingers**. As Ryan McGeehan said in **a recent post,** "when a human being is needed to manually receive an alert, contextualize it, investigate it, and mitigate it...it is a declaration of failure."

Consequently, organizations such as Netflix, Lyft, and Square have started treating threat detection as an engineering problem, using automation to avoid these pitfalls and make security teams more effective. They are also avoiding the silos that separate detection, response, and development teams, following the DevOps mindset when building detection mechanisms and integrating them with response orchestrations.

While organizations could use SIEM or a centralized data lake and analytics to perform security detection, true detection engineering requires a different structure, focusing on a smaller group of skilled engineers rather than a large group of centralized analysts. It also requires a different infrastructure.

## Detection Engineering Infrastructure

In practice, implementing detection engineering requires an integrated infrastructure that consists of the following components:

- Data sources
- Event pipelines
- The correlation engine
- Response orchestrations
- Testing and feedback loops

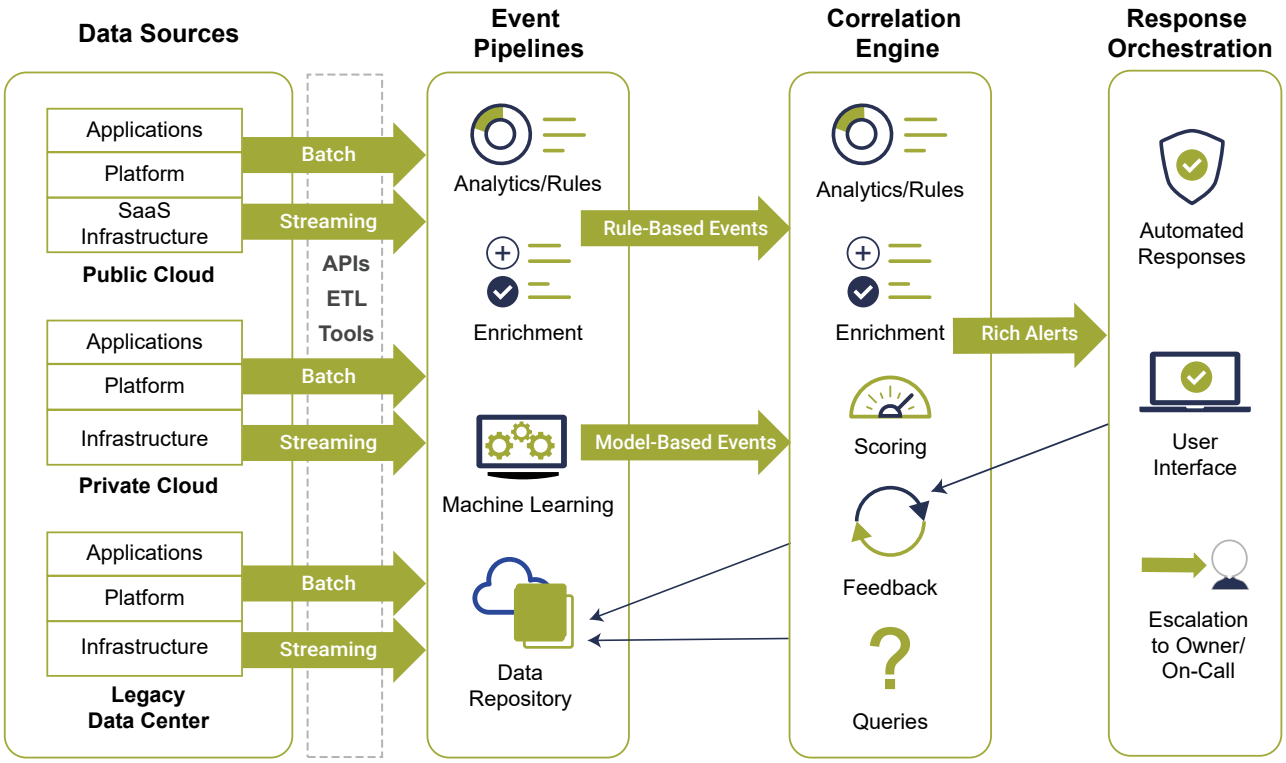


**Figure 2: Detection Engineering Infrastructure**

Our definitions and examples of detection engineering infrastructure components rely heavily on the work of **Alex Maestretti**, engineering manager of Netflix's detection and response team, and **Ryan McGeehan**. We thank them for sharing their insights with us and allowing us to reference them here.

## 1. Data Sources

An effective detection system must start with data about the applications and infrastructure that it is protecting. But getting the right (and good) data is not always easy. Some of the key aspects to consider are:

- **Cloud-native visibility:** The cloud-native architecture has introduced new layers and components such as containers, orchestrators, service mesh, and others. Collecting data from these systems will likely require new tools and instrumentation.
- **Align instrumentation and log consumption:** Instead of creating a central team that deals with logs written by others, security engineers should work directly with DevOps to instrument data collection. The security organization should also hold those engineers accountable for the results. Neflix for instance, holds the philosophy that the person who logs the data should be the person who consumes that data. This brings a new level of rigor and discipline to the process of creating useful logs.
- **Find the best sources:** Different systems support different levels of instrumentation and use different data schemas and models. Security managers need to find and understand the different logs and formats and decide which are relevant to threat detection.
- **Decentralizing detection:** The traditional detection architecture focuses on gathering a massive amount of data into a centralized data lake and run detection algorithm on it. This means detection can hardly be in real time as the amount of analysis needed is enormous. Utilizing a distributed architecture that performs detection closer to major sources of data allows for much faster detection and higher quality of data to be gathered.

## 2. Event Pipelines

Event pipelines gather event data, streamline them via automated mechanisms, and prepare them for further examination before a human ever has to see them. To ensure effective detective controls, security teams must consider these issues when building event pipelines:

- **Stream vs. Batch:** Today, many SaaS products and most legacy systems can't push log and instrumentation data to security system, making batch processes a necessity. But organizations should instrument services to stream data in real time to event pipelines when practical and possible.
- **Normalization vs. Workflows:** Security teams should avoid the difficult work of normalizing event data by creating pipelines for each data source, basing its workflow on the data in that system. Templates and reusable modules to streamline work on common data types can streamline these efforts.
- **Alert Frameworks:** A rigorous framework for creating events and alerts, and the rules that drive them, is essential. The process should follow the same engineering standards that govern software projects, making alerts subject to peer review and using a version-controlled repository. And the

person who writes an alert should be accountable for its results. Palantir's incident response team **posted an excellent write up on such a framework**.

- **Enrichment:** The low quality of security alerts is a major contributor to alert fatigue. Automation should enrich event data, adding critical information, eliminating manual labor, and improving alert quality. In general, less-expensive enrichments--such as a health and schema check or lookups for specific values, such as the geo-location of an IP address -- occur in the event pipeline.
- **Machine Learning:** Proven machine learning techniques increase the ability of the system to detect active threats and conditions that warrant further investigation, reducing false positives. Algorithms for feature extraction run on the data, building models of actions, looking for anomalies and conditions, generating model-based events that drive event triggers.
- **Forensic Data Storage:** Detection engineering infrastructure should store and archive event data, which can be crucial when it comes to forensic activity.

## 3. The Correlation Engine

The event pipeline passes only the events warranting further inspection onto the correlation engine, which will ultimately determine automated responses to alerts, including notifying humans. Key factors include:

- **Choice of Platform:** The correlation engine is typically a data analytics platform. Commercial detection engineering products such as **Capsule8** include their own data analytics engine. Some organizations rely on **Splunk** while others use **Elasticsearch**. (**Disclosure:** Rain Capital has an investment in Capsule8.)
- **Further Enrichment:** The correlation engine uses rules and more expensive enrichment to determine whether the system triggers an alert sent to a human or invokes automated response mechanisms. Security teams will need additional tools and services to gather and include data such as the user accounts involved in an event, their security classification, who they work for, and their contact information. Context-specific information can include screenshots of what the user saw (in the case of a phishing attack, for example), how a given operation was launched (manually vs. automatically), what privilege levels were used, and any privilege escalation that occurred. Automated communication mechanisms, such as Slackbots, can reach out to get confirmations of activity or more information from users.

## 4. Response Orchestration

Automation should do as much as possible to mitigate events as part of response orchestration, before an alert reaches a human. When it does reach a human, a properly enriched alert should contain a reasonable set of response actions. The right set of response actions will depend, of course, on the application and the type of event. In general, however, these are some of the factors to consider:

- **Automate common fixes:** If a common temporary fix involves re-deploying a cluster in Kubernetes, for example, a workflow could re-deploy it automatically, triggered by a rule. Alternatively, an alert could include re-deployment as an option, allowing the human receiving the alert to do so with just the click of a mouse, saving a great deal of time and effort.
- **Build in communication:** Effective response and escalation orchestration should include mechanisms that can quickly bring users, developers, and other relevant players into the communication loop. Integration with Slackbots is often a key component of detection engineering. Dropbox recently released **Securitybot**, a communication mechanism designed specifically for integration with detection engineering systems, under an open source license.

## 5. Testing and Feedback Loops

Given a set of response options, the security team should capture feedback on how alerts are working for alert tuning and improvement. These feedback loops should include end-to-end integration testing. This includes working with red team and other testing efforts. McGeehan says security teams should “treat detection the same way you’d treat a build pipeline supported by CI/CD platforms like Jenkins.” He recommends simple scenario tests and writing direct attacks on the detection mechanisms. Canary testing is another useful technique for testing changes to controls. More often than not, the team will gain valuable insight into how the system works, driving improvements in both existing and future detection controls.

# Case Study: The Netflix Implementation

As we noted earlier, Netflix's detection and response team has been publicly discussing its detection engineering efforts. Due to its large scale, Netflix finds it more economical to build its own infrastructure than to buy off-the-shelf products. Detection engineering is no exception. It's also important to note that Netflix's detection engineering infrastructure is still a work-in-progress.

That said, Netflix is pioneering capabilities that point the way for commercial products and other enterprises as they make the transition to cloud-native systems. In this case study, we cover the components of the Netflix infrastructure as an example of the functionality strong detection engineering systems should provide. Figure 3 (below) illustrates the detection infrastructure architecture defined by Netflix.

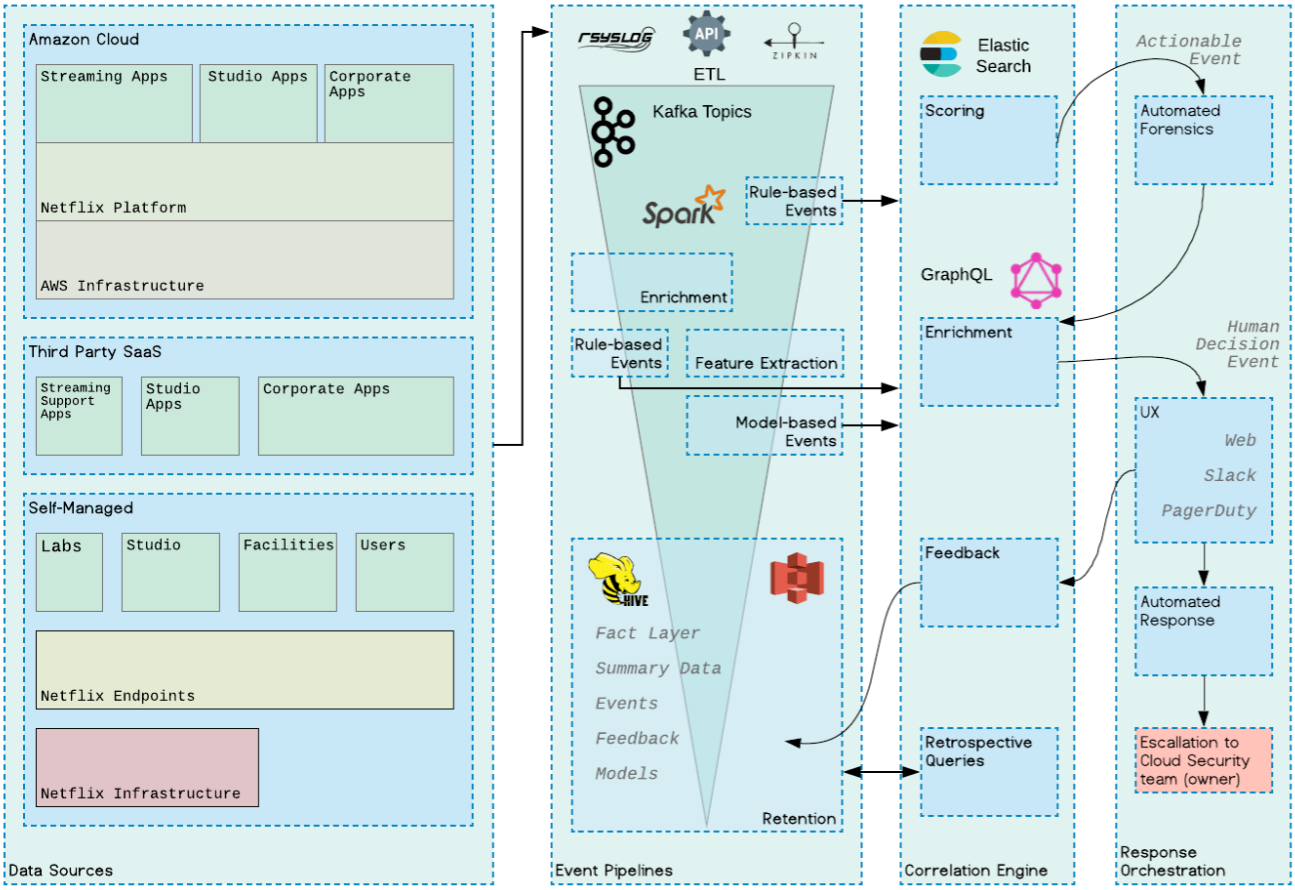


**Figure 3: The Netflix Detection Engineering Infrastructure (image courtesy of Netflix)**

## 1. Data Sources

At Netflix, data sources include applications running on the Netflix platform in AWS, applications in third-party clouds, and the internal infrastructure and systems that Netflix manages for itself.

For monitoring host systems, Netflix relies on custom development based on open source capabilities such as the **extended Berkeley Packet Filter (eBPF)** and **osquery**. eBPF allows developers to gather performance metrics, behavior data, and errors at all levels of the operating system. Osquery is an instrumentation framework for Windows, macOS, Linux, and FreeBSD that exacts low-level system data and allows security managers to explore them via SQL queries.

## 2. Event Pipelines

Instead of normalizing event data upfront, Netflix creates a workflow for each data type. The team then takes what has been a customized process and streamlines it by creating templates and reusing as much code as possible from previous onboarding efforts. As the team repeats the process with common data sources, Alex Maestretti, engineering manager of Netflix's detection and response team, thinks the percentage of code and template reuse will jump significantly, a metric he watches closely.

While he acknowledges each data source may require a bit more effort to set up, Maestretti says the upside is far more flexibility. In addition, Maestretti says Netflix is working to make data sources as self-serve as possible, allowing software engineers who aren't data engineers to access and onboard the data for their event pipelines.

**Other important elements of Netflix's event pipeline infrastructure include:**

- **Batch and Stream:** Today, Netflix still runs batch processes on some data, using ETL tools and APIs as necessary. But since Netflix prefers streaming data whenever possible, the team works with **Zipkin**, a distributed trace system for microservice architectures. Using Zipkin, developers can instrument applications to report data such as timing and dependency, and detection controls can dynamically tell Zipkin which data they need on a particular trace.
- **Kafka:** Each workflow loads event data into **Kafka,** the backbone for Netflix's event pipelines. As a distributed pub/sub messaging system, Kafka allows developers to define topics, or queues, for particular message flows. The workflows load data sources onto Kafka topics, and other components of the detection engineering infrastructure subscribe to the topics important to them.
- **Alert Framework:** Netflix has extended the **Palantir framework** to ensure that any given alert is created in a consistent and disciplined fashion.
- **Accountability:** In the ideal state, system owners will use the framework to write rules. While the security team has responsibility for overall security, system owners have primary responsibility for the reliability and security of their systems. System owners create "alert packs", which define the alerts they deem important. Developers who need to use a given service can then leverage these alert packs in a self-serve model.
- **Rule Refinement:** Netflix is making rule development and refinement part of its standard security processes, such as post-incident reviews. All of this work must blend back into the framework, following the process for updating an alert and documenting those changes.

- **Enrichment:** As Figure 3 illustrates, Netflix has used **Apache Spark** as its rules and analytics engine, driving the enrichment of data in the event pipeline. Netflix has used Spark primarily for batch-mode data sources, but is moving to **Apache Flink** as more data sources are streamed to the event pipe line. **Apache Hive** is the data warehouse, storing event data, machine models, logging data, performance and feedback data on the models, and other relevant data. It also supports forensic investigation on historical data.

## 3. The Correlation Engine

Netflix has targeted **Elasticsearch** for its correlation engine. Using pre-defined templates and APIs, Elasticsearch can load and transform data from the event pipelines, support searches of that data, score an event based on queries and rules. The system further enriches event data via **GraphQL**, which enables queries to APIs, on relevant systems.

Still, Maestretti says Netflix has work to do in determining implementation specifics. Netflix has a substantial investment in Elasticsearch, which is the data analytics platform for its video streaming service, and Netflix's detection and response team is familiar with the Elasticsearch platform. The security team thinks it can repurpose the existing platform, generalizing its structures to support security domains, but has yet to prove that theory. The detection and response team also plans to leverage the work of Netflix's data science team, working from a menu of algorithms, and applying them to data sources to improve the models and reduce false positives.

## 4. Response Orchestration

As Figure 3 shows, "automated forensics" is one of many possible response orchestrations. Python scripts running AWS Lambda functions automate this process, using specialized Amazon Machine Instances (AMIs), S3 buckets, and roles to create a trusted forensics environment. An alert can come to a human with the option for running this automation. In some cases, it can run automatically, without human intervention.

In either case, the automation uses AWS APIs to snapshot the drive(s) in an instance (or many instances), and share it with a special forensics account. The forensics account creates a copy of the volume, which is mounted in a predefined AMI that includes trusted forensics tools. Inside the AMI, a tool known as Diffy takes a hash of the suspect volume and compares it to known good file hashes. Diffy then captures any anomalous files, outputting them to YAML as forensic artifacts.

Diffy can also reveal if instances are listening on an unexpected port, running an unusual process, or have inserted an unknown kernel module. It can collect a functional baseline from a known clean running instance and compare it against an instance group, or survey all instances, revealing any outliers.

*More detail on this topic is available in the **video of Maestretti's 2017 AWS re:Invent talk**. Netflix also recently released Diffy under an open software license, and you can get **more information here**.*

### 5. Testing and Feedback Loops

Netflix has been evaluating several products in atomic red team testing with the intent to automate end-to-end testing of the detection and response system. Maestretti says Netflix has created a proof of concept for a couple of attacks with these products, deploying those attacks via its automated CI/CD pipeline, **Spinnaker**. Currently, those test attacks flow all the way through to a human. While it's important to test human responders, he says, it's not something he wants to do continuously. The team wants to modify the mechanisms so it can isolate the testing to machine and software components, allowing them to do it more often.

## Culture and Decentralization

Simply investing in technology and products will not enable an organization to suddenly become more proficient at detection and response. For many enterprises, success will require fundamental shifts in company culture and mindset. Key stakeholders--such as legal and compliance teams--must be aligned, and development teams must buy into the DevOps mentality. This means leadership must be committed to, and actively involved in any such undertaking. The first step in that effort is taking a hard look at the role of the traditional SOC.

As an organizational construct, the SOC is predicated on the assumption that the analyst team understands most (or all) of an organization's systems, has experience with most (or all) of the organization's security systems, and can define and analyze threats. Given the inherent complexity of cloud-native systems, however, these assumptions simply don't hold up in the DevOps world. And when one considers the rapid change that characterizes cloud-native operations, it's fair to conclude that a centralized SOC may be incapable of keeping up with a business based on cloud-native systems.

### The Decentralized, SOC-less Enterprise

When Netflix's Maestretti and others talk about the "SOC-less enterprise," then, they're really talking about decentralizing the SOC, integrating security functions and people more directly with the DevOps process. That means moving alert triage from a centralized SOC to the system owner/on-call, whether it's a security or an application team. Instead of the traditional SOC, then, organizations have more of an on-call rotation, operating with the DevOps mindset.

Maestretti thinks that with solid detection engineering capabilities, Netflix can bring developers and system owners up to speed on a security alert more easily than it can teach a security person the details of a production system. By providing high levels of context and enrichment, alerts can tell developers and system owners what the alert means and give them a set of response options. (He also emphasizes that overall responsibility for both the quality of and the response to the alert rests with the security team.)

**While for some enterprises, wholesale or overnight adoption of this approach may not be realistic, building a roadmap that leads to a more decentralized or hybrid security organization may be critical to remaining competitive in the future.** As systems grow more decentralized and agile, it becomes difficult to imagine a future that does not involve a shift towards cloud-native systems, with all the unique security implications arising from them.

## Staffing Will Always Be The Biggest Pain Point In Security

The shortage of capable security personnel is well-documented. While automation may reduce the need for some lower skilled analysts, detection engineering increases the need for skilled engineers and analysts. Today, many organizations lack staff with the skill sets necessary for detection engineering.

Over time, outside resources may emerge that can augment internal capabilities. But it's equally clear that organizations that move to cloud-native architectures should ensure that they build and maintain a solid internal capabilities in these areas. Existing staff may need training to expand their capabilities. People who are good at operations, for example, may need additional skill sets such as cross-functional program management.

And while a company like Netflix may find it more economical to build its own infrastructure, many enterprises will need build a core team of "detection engineers" around a solid commercial product. Some of the technical solutions that are helping organizations achieve detection engineering competency include **Capsule8**, **Sysdig**, **Elastic Stack (ELK)**, **MistNet**, and **Splunk**.

## Conclusion

As organizations move to cloud-native systems, security must evolve, gaining higher degrees of alignment in terms of the technology stack and DevOps mindset. That means creating continuous security pipelines that accompany the continuous development pipelines inherent to the cloud-native ecosystem. Detection engineering is one way organizations can accomplish that goal. Enterprises moving to cloud-native technologies should consider how and when to incorporate these practices into their security programs, being careful to understand the importance that culture change plays in successfully making that transition.

# About the Authors

### Jamie Lewis

**Jamie Lewis** has studied core infrastructure technologies for more than 25 years. He was founder and CEO of Burton Group, an enterprise IT research and advisory firm, which Gartner acquired in 2009. He began working with startups through angel investing. He joined Rain Capital as a Venture Partner in 2018.

### Dr. Chenxi Wang

**Dr. Chenxi Wang** is Founder and General Partner of Rain Capital. She has held numerous executive positions in the Cybersecurity industry prior to founding Rain, including VP of strategy at Intel Security and VP of research at Forrester.  Dr. Wang is a trusted advisor to many security leaders and serves on the global BoD for OWASP foundation.

# About Rain Capital


Our mission at Rain Capital is to identify and invest in disruptive companies that push the boundaries of Cyber-security capabilities. We look for:

- **Defensive technologies** that empower the modern defenders; delivering actionable insight, automated response, and defense at scale.
- **Trust disruptions** that leverage blockchain or identity innovations that redefine digital trust.
- **The Scaling Factor** that enables security operations of massive scales with automation, Artificial Intelligence, and a service-first mindset.

For more information, you can check out our website at **www.raincapital.vc** or contact us at **info@raincapital.vc**.