SANS

Analyst Program

Whitepaper

A Comprehensive Approach to Cloud Threat Detection and Response

Written by **Jake Williams**

June 2022

sysdig

Introduction

When it comes to container security and visibility in the cloud, there are two primary models: agent based and agentless. Many organizations spend an inordinate amount of time trying to decide which model is superior. In this paper, we make a strong case that organizations need both agent-based and agentless monitoring solutions to maximize visibility and control. Once practitioners recognize that both are necessary for optimal security outcomes, this logical question emerges: How should these technologies work together? We seek to answer those questions in this paper, using a case study showing the benefits of synergizing data from agent-based and agentless monitoring in finding, focusing, and fixing threats.

Use Cases for Cloud Detection and Response

Before discussing specifics, establishing the use cases for any cloud workload monitoring solution is vital. We can categorize these use cases as finding, focusing, and fixing. Let's examine each of these in detail.

Finding

Visibility is critical across your cloud and container platforms, but traditional security tools (those that aren't built to be cloud-native) don't provide adequate context to perform an investigation and security response. Sure, agentless is the new hotness, but what are you giving up? Do you need agents or agentless? The reality is that in a cloud environment, you need both to adequately secure workloads. (We discuss the trade-offs between agent-based and agentless solutions in the section "Agent or Agentless.")

Do you need agents or agentless? In a cloud environment, you need *both* to adequately secure workloads.

Focusing

Now that you've found the threat, you need to prioritize and rapidly focus on what matters. How do you quickly identify what matters most? It will come as no surprise to anyone with SecOps experience that alert fatigue is a major contributor to analyst burnout. And this isn't limited to just false-positive alerts. When analysts receive alerts, they're expected to take action, but when those alerts lack necessary investigative context, analysts find themselves in a frustrating situation with potentially career-altering implications. Even ignoring analyst happiness and retention (things that organizations absolutely should not ignore), providing the tools to quickly focus an analyst matters for the health of the organization.

In many incident response investigations, we learn that analysts had some piece of relevant data that would have revealed the intrusion sooner—but that wasn't clear at the time. The most common root cause? The alert, as displayed to the analyst, lacked the necessary context to properly understand the intrusion. However, this leads to an obvious question: With so many new platform-as-a-service (PaaS) offerings, even on a familiar cloud service provider's platform, how do analysts contextualize any alert they're staring at? What service even generated it? Without the appropriate context, analysts are left to wonder if the issue is isolated to a particular workload or if it's even really exploitable in the given configuration.

Analysts need to be able to focus on what matters with confidence, eliminating the all-too-familiar guesswork around alert triage so common in the space today. This necessarily includes prioritizing the most critical alerts to developers and engineers to address common root causes—another task rendered nearly impossible without proper context.

Fixing

Prioritization isn't the end of the journey, though. We still need to remediate discovered issues. Just like a good doctor, we need to treat the causes rather than just address the symptoms. Remediation must happen at the source. With widespread adoption of Infrastructure as Code (IaC), fixing the issues at the source becomes more important than ever. Anything less leaves the organization dealing with the same issues repeatedly as teams instantiate new workloads using the same vulnerable configurations. Of course, runtime alert fatigue is not the only factor impacting security teams. Scanning tools used in the continuous integration/continuous delivery (CI/CD) pipeline generate significant additional numbers of security alerts, many of which are not remediated prior to deployment.

Agent or Agentless?

When investigating visibility solutions for cloud platforms, and workload monitoring more specifically, the question of agent-based vs. agentless becomes an inevitable focal point. After decades of everything requiring an agent, the IT department's desire to shift to agentless technologies was predictable. Every system admin has a horror story about *that one time* an agent caused an availability issue. What they always fail to mention are the myriad occasions where the same agent performed some critical function in securing, maintaining, or providing visibility for the same workload. DevOps isn't helping the agent-based vs. agentless debate either. Given the speed with which DevOps teams change deployment parameters, testing to find every edge case with an agent simply isn't realistic.

Although agentless technologies like to claim they can do everything without an agent, let's delve into that a bit. While an agent-based technology stack has direct visibility into the runtime of the system it runs on, comparable agentless solutions are limited to polling or taking trigger-based actions. Polling models suffer from limited visibility between polling intervals, not to mention the difficulty of choosing an appropriate polling interval. Trigger-based actions are limited to the types of triggers (events) the monitored platform can send (and the timeliness with which it does so). Agentless also implies that a target cloud service or workload an organization wants to monitor and control is fully API enabled. This API enablement isn't necessarily a given, depending on the underlying technology stack or the cloud provider's offering.

Given these limitations, should organizations transition from agent-based to agentless solutions, even if it means preventing potential availability issues caused by agents? In most cases, we would argue that they definitely should not. Because agentless solutions rely on existing features of the monitoring target to provide telemetry, they often miss important telemetry that agent-based solutions catch. Perhaps more importantly, the additional data collected by agent-based solutions often proves pivotal in establishing context. An additional point worth mentioning is the extreme rarity of availability issues caused by agent-based solutions in today's environments. Most anecdotes in this area can be traced to immature capabilities, improper deployments, or other extremely self-inflicted circumstances.

Agentless solutions are, however, required to capture data that agent-based solutions cannot. If the organization has workloads configured in AWS, agent-based solutions can (and should) capture telemetry inside containers, but what about understanding the AWS control plane? Because the control plane doesn't offer the capability to run an agent, we should think of this as an ideal use case for agentless technology.

By considering circumstances where agent-based and agentless technology each excel (and don't), it becomes clear that organizations need a unified model of agentless plus agent-based to address the challenges of finding, focusing, and fixing container security issues.

Case Study

In this section, we examine a case study played out across two different scenarios. In the first scenario, the organization has performed a "lift and shift" to the cloud, a common practice in data center consolidations, without considering changes to security tooling. The organization discovers an incident but has difficulty with the key tasks of finding, focusing, and fixing. In the second scenario, the same incident occurs, but due to cloud-native tooling, we see a markedly different outcome.

Scenario 1: Cloud Lift and Shift with No Tooling Changes

Robert (an L1 SOC analyst) receives an alert for a running process **`aws s3api delete-bucket`** running on a host. He recalls a notification from the DevOps team mentioning they were now deploying a mission-critical application in containers. A *bucket* sounds a lot like a *container*, so Robert assesses that this must be important. Robert coordinates with his shift lead, Janet (an L3 SOC analyst), and the two begin to investigate. Figure 1 depicts a simplified view of the architecture and scenario.

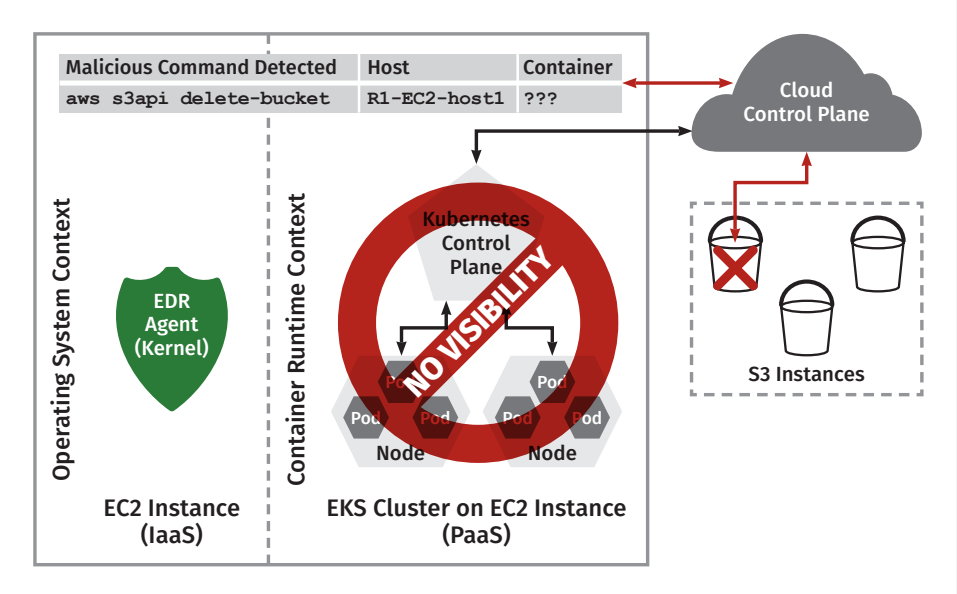

Figure 1. Lift and Shift Architecture Without Tooling Changes

Janet notes that they should definitely care about this command, but she is also able to demystify the confusion around containers vs. buckets. Janet notes that the alert came from the organization's endpoint detection and response (EDR) solution. The EDR solution depends on a kernel-mode agent, and as such, can only function on infrastructure-as-a-service (IaaS) instances. Because containers operate with a shared kernel, the EDR agent must run on the underlying physical or virtual host; it cannot run inside containers themselves. This limitation has also kept the organization from adopting Amazon Elastic Kubernetes Service (EKS) PaaS offerings where host concepts don't apply because it's abstracted by the cloud provider. Although using EKS would doubtless make the DevOps team happier and can substantially reduce OPEX costs, stakeholders have advised against deploying anywhere they can't have visibility (for regulatory reasons). As anyone who has worked in cybersecurity for any length of time will confirm, business units within the organization almost certainly deploy in EKS (and other PaaS) anyway, but the organization is just blind to those instances.

Janet queries the host in the EDR and sees multiple commands that were run before and after the suspicious command that generated the alarm. The sequence of commands appears completely nonsensical, leaving Janet and Robert struggling to understand their meaning. They draw the conclusion that something happened that the DevOps team should know about, but they can't pinpoint exactly what.

Janet finally elevates the alert to the cloud security architect, Satish. The security architect needs to work with the DevOps team to even understand how the application works in its latest deploy and which resources on the cloud platform this application uses. As they inventory resources, they notice several deployed EKS containers as well as Amazon Simple Storage Service (S3) buckets that can't be linked to any authorized project. After further investigation, Satish learns that an over-permissioned API key was compromised and used to deploy EKS instances used to mine cryptocurrency, increasing the organization's cloud service provider bill. The organization had no visibility because it relies on an agent-based monitoring solution that requires kernel mode access, and as such, can't run in containers.

The API key is deprovisioned by Satish, but the mystery of why a threat actor was deleting an S3 bucket remains. The organization investigates usage and billing statements, looking for other rogue use of the account, and discovers that someone was creating and using S3 buckets to distribute stolen digital content. Additional investigation shows that these S3 buckets were created using a *different* API key, suggesting a possible second threat actor. The team deprovisions this API key too, of course. The organization has an S3 Lifecycle policy that expires AWS CloudTrail logs after 14 days, primarily due to the cost of storing logs. Because someone created these rogue EKS containers and S3 buckets 23 and 25 days ago respectively, the organization is left guessing as to the root cause of the issue. Additionally, because the organization processes regulated data on its cloud instances, it is exploring whether it can sufficiently confirm that a breach has not occurred.

Scenario 2: Cloud-Native Tooling

Now let's examine the same scenario but this time with cloud-native tooling using a combination of agent-based and agentless telemetry. Before considering this scenario, it makes sense to enumerate additional telemetry the organization now has and why it matters:

- Sensors can run inside containers, giving the organization focused visibility at the *container* level.
- The agentless technology monitors the AWS control plane for unauthorized actions.
- IaC scanning enables auditing of service, infrastructure, and workload configurations.
- Alerts and aggregated information from the agentless solution are summarized in the SIEM, partially mitigating the short retention of AWS CloudTrail logs.

Figure 2 depicts a simplified view of the architecture and scenario.

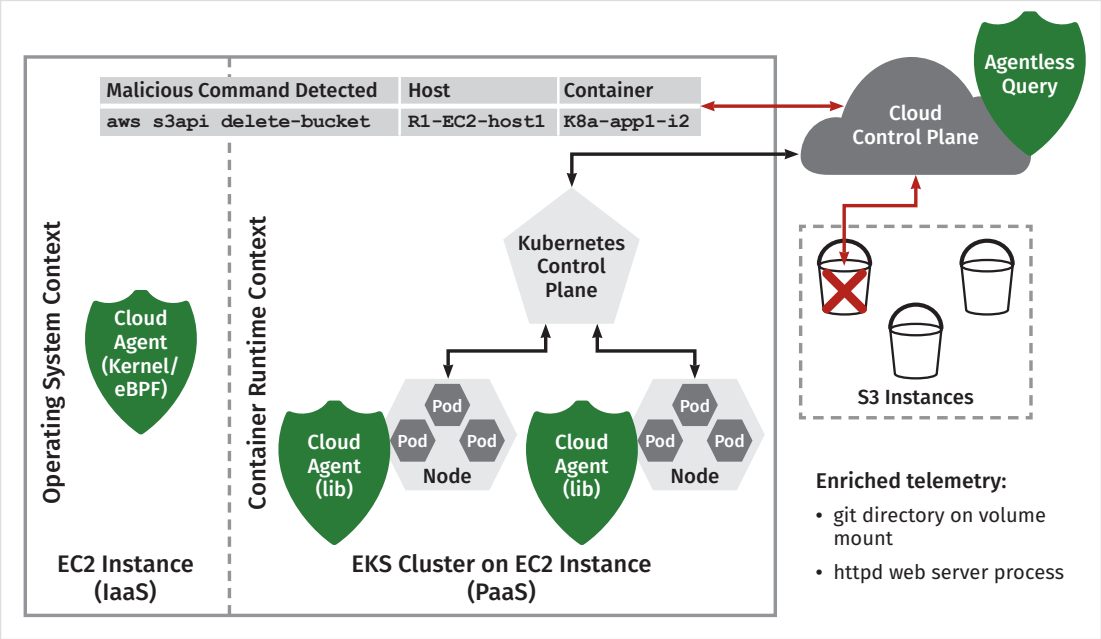

Figure 2. Lift and Shift Architecture with Cloud-Native Tooling

Robert (the L1 SOC analyst) receives an alert from his agentless tooling showing that multiple EKS instances have an abnormally high load. The organization began migrating to primarily EKS after adopting cloud-native agent-based tooling two months ago, although some IaaS hosts running containers still exist. However, Robert notices that he's not receiving any agent-based telemetry from these EKS instances that have significantly higher resources usage than expected. Robert loops in his shift lead, Janet (an L3 analyst), to investigate further.

Janet identifies that the EKS instances aren't running the expected agent (explaining the missing agent-based telemetry), leading her to conclude that they are possibly rogue Kubernetes deployments. But questions remain: What kind of rogue? Was this a threat actor or shadow IT? Janet easily spots the over-permissioned AWS API key used to launch the EKS containers in her tooling and quickly confirms that the API key has not been used historically to launch containers previously. Janet calls in Satish, the cloud security architect, to confirm that this API key should not be launching EKS containers. Satish reviews the data available through the agentless tooling, and after consultation with the DevOps team he confirms these rogue deployments.

The agentless tool does not completely mitigate the use of short retention of AWS CloudTrail data, but it does provide substantial additional information showing where and how threat actors have used the compromised API key. The API key was provisioned more than six months ago (the agentless tooling has six months of retention), but the first anomalous use of the key was two days ago. Armed with this information, Satish works with the DevOps team to review new commits, actions taken, and infrastructure changes backing up from the first anomalous use of the API key.

They discover the key was exposed in a Dockerfile committed to a public repository only four days earlier. Satish resists saying, "I told you so," having just been told to postpone the initiatives to perform more thorough scanning pre-production. He notes that the organization, having just dodged the proverbial bullet, probably won't resist future attempts to shift left. While the threat actor racked up some charges in AWS, they didn't compromise the regulated data they could have using the API key—and Satish can confirm that thanks to his cloud-native agentless tool stack. The organization then deprovisions the compromised API key and makes plans to integrate more scanning into its CI/CD pipeline.

Weeks later, Robert is on shift again and receives multiple alerts involving a deleted S3 bucket. One alert comes from the agentless tooling, and another comes from the agent running on the container, which captures the **`aws s3api delete-bucket`** command. Rather than seeing this only on the host (as in the previous scenario), Robert sees the command from the point of view of the container it is running in. This provides context to focus him on the issue, instead of forcing him to spend time working through commands run in other containers on the host. Robert can't see any reason why an S3 bucket should be deleted, but he checks again with Janet. Together, they review their telemetry and discover that although this AWS key is sometimes used to manipulate S3 buckets, the actions are never performed using the AWS command line tool, especially from a container.

Janet calls in Satish to help with the investigation. Satish's first thought is "oh no, here we go again" and laments how he wishes the organization had already integrated security tooling into the CI/CD process. While the organization is moving toward that approach after the last incident, it sadly hasn't happened yet. Satish begins investigating any other use of the API key in telemetry and discovers multiple other uses in two additional containers running on the same host. Although it is obviously concerning that a threat actor is operating in multiple containers, Satish is relieved that he can easily separate these actions on a per-container basis due to his cloud-native agent-based tooling.

After digging into operations a bit more, Satish notes that the use of the AWS command line tool should generate alerts in the vast majority of cases. In fact, he's only able to find one container where this shouldn't generate an alert and writes a detection rule to deploy through the Falco agent for faster detection in the future. Satish also looks back into logging to try to determine how this new API key was compromised in the first place. Given recent history, he suspects a Dockerfile, but a review of available data doesn't yield any investigative leads.

As Satish continues to investigate, he discovers a web-accessible **`.git`** directory on one of the compromised containers, and unsurprisingly it's the first container where the threat actor ran the AWS command line tool. The **`.git`** directory should not have been included in the production instance and provides the threat actor valuable information, including API keys. As Satish continues to investigate, he looks for any actions performed with the API key in his agentless tooling stack. Satish quickly identifies that the threat actor has created multiple AWS buckets, which are being used to host stolen digital content.

To provide rapid detection of these issues in the future, Satish creates another Falco rule to trigger on any open system calls performed on a **`.git`** directory from an **`httpd`** (web server) process. This won't solve the issue of inadequate predeployment container scanning, but it will ensure that if another **`.git`** directory is deployed in a running instance, the organization will receive an instant alert if a threat actor discovers it. While Satish recognizes that we should always shift left in security to the extent possible, he's stuck waiting on other teams to implement the appropriate processes and tooling. Satish isn't powerless, though, and takes immediate steps to implement runtime detections in his cloud-native tooling stack. If the same detection rules were even possible to create in the organization's chosen EDR, they would be implemented at the host level, still lacking the critical context of which container the observed activity occurred in. As any analyst will tell you: During an investigation, context is the key to quickly focusing on what matters.

Satish is able to focus the DevOps team to the specific container that was compromised (and even the **`.git`** directory as the root cause). Additionally, the DevOps team reviews whether (and under what circumstances) alerts should be generated for new S3 buckets being created. The organization reviews logging in the agentless solution, confident they've discovered all locations where the API key is being used prior to deprovisioning and replacement. In addition to recycling the key, the organization changed how the key is used, removing many permissions and transitioning some operations (as feasible) to identity and access management (IAM) roles.

The rogue buckets hosting stolen digital content were created without running commands inside the container instances, instead directly interacting with the AWS control plane. The organization would have been blind to these bucket creations if using only an agent-based tooling stack and would only be able to trigger a detection when the AWS command line tool was called from within a running container. While this in fact triggered the relevant detection in our scenario, this was purely a tuning issue. The analyst received two alarms for the bucket deletion, each of which provided more context than the single alarm received in the first scenario.

Deprovisioning API Keys

While we should strive to use IAM roles where possible, API keys continue to be required in many use cases (namely machine communication and automation). But as any experienced DevOps engineer will happily explain, one does not "simply" deprovision an API key. For the sake of space, in the first scenario we didn't sufficiently address the myriad issues inherent in deprovisioning an API key identified as in-use by threat actors. While organizations always want to move fast to contain the threat actor, they must balance the risk of self-imposed denial of service if they do not properly enumerate first the legitimate services using the key.

This is a major advantage of agentless tooling: It can help operations teams uncover how these keys are *actually being used* (as opposed to just knowing how they are *supposed to be used based on intended design*). Organizations with the appropriate tooling can move more rapidly through the fixing phase of the "finding, focusing, fixing" model, confident that they understand the impacts of proposed changes. We've witnessed multiple organizations stress over changes for inordinate time periods, extending the length of a threat actor's access to their environments. This is understandable. After all, although nobody wants to deal with a threat actor in the environment, causing a self-inflicted outage in a critical service is often a résumé-updating event (especially during an incident when tensions are already high).

Conclusion

In this paper, we discussed how lifting and shifting workloads to the cloud without tooling changes is a losing proposition. It results in reduced visibility in cloud environments, lack of cloud control plane context, and limited workload context that weaken the security posture of the organization. We discussed how agent-based and agentless solutions work, including some strengths and weaknesses of each approach. After reviewing the case study, the outcomes are clearly superior when using a combination of agent-based and agentless *cloud-native* security tooling. The two technologies aren't mutually exclusive. Organizations that have moved to the cloud without deploying cloud-native monitoring tools should examine these scenarios and ask whether they would have similar outcomes to the observations in the first scenario. Those wanting to avoid those outcomes should strongly consider deploying a cloud-native tooling stack using a combination of agentless and agent-based technologies to maximize their ability to rapidly find, focus, and fix security (and operations) issues in their cloud environments and the spectrum of workload types that exist within modern architecture.

Sponsor

SANS would like to thank this paper's sponsor: