Generate Strong Security Signals with Sumo Logic & AWS Cloudtrail

By Dan Whalen

Published: 2019-09-10 · Archived: 2026-04-05 21:57:26 UTC

As orgs increasingly shift some of their workloads to cloud providers like Amazon Web Services (AWS), it's often challenging to get the right level of visibility into these new environments for security monitoring purposes. Sure, security professionals have had decades of experience monitoring traditional enterprise networks, but services like AWS, Microsoft Azure and Google Cloud Platform come with additional sources of valuable data — which is frustratingly unfamiliar if you're used to racking and stacking your own servers. Combine that uncertainty with an already long laundry list and the result is this: Most organizations using cloud platforms are not taking full advantage of the signals available to them.

But there's some good news: There are lots of great technology solutions we can use to help us get a better handle on those signals. In this post, I'll show you how Expel uses a SIEM (in this example, we'll take a look at Sumo Logic) to generate security leads from AWS signals. Regardless of what SIEM you use, I'll share some detection use cases (with examples!) that you can try out in your own environment.

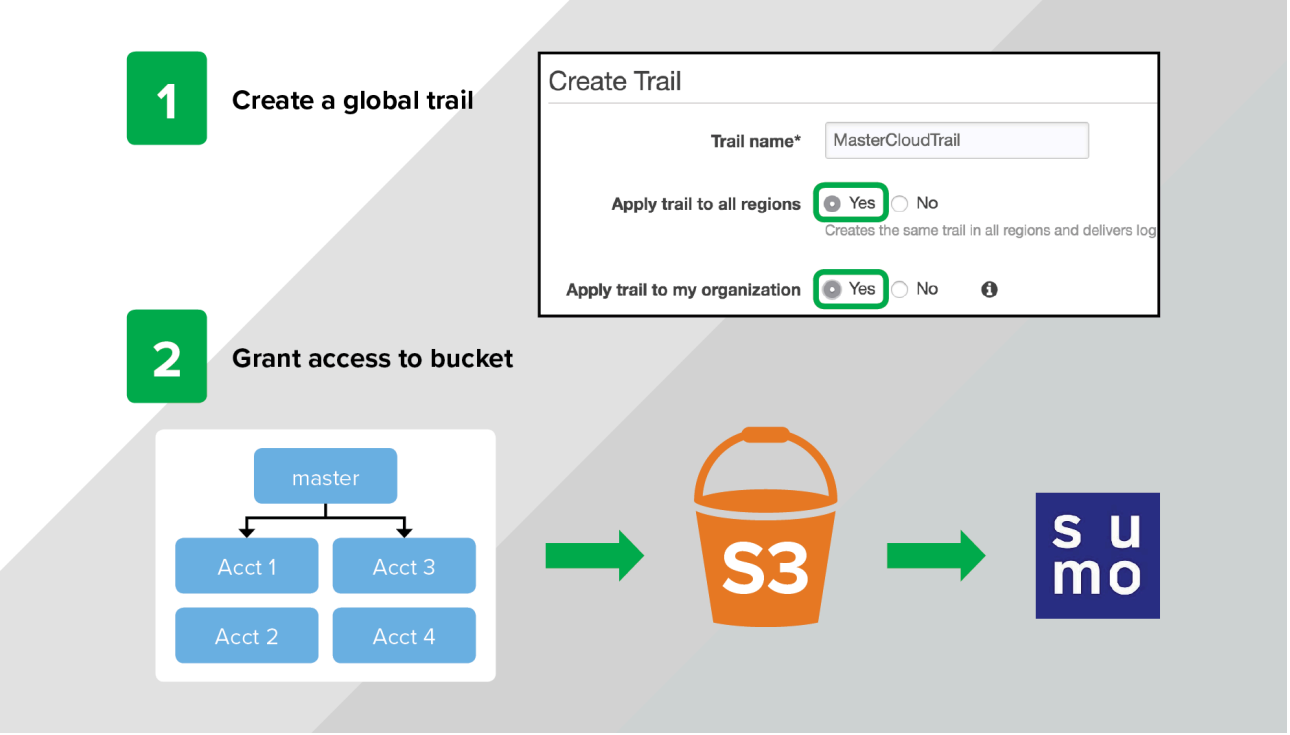
How does a SIEM help?

Anyone who works in security knows that there are two high-level problems that need to be solved to effectively monitor an environment:

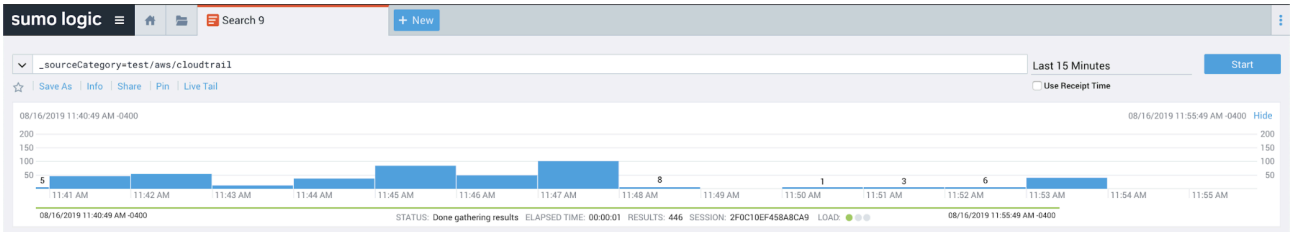
1. Collecting the data you need; and
2. Drawing actionable insights from the data

A log management (aka SIEM) solution like Sumo Logic does all of the heavy lifting, connecting up to your sources of data and providing an intuitive search interface that lets you generate alerts and perform investigations.

For example, you can easily onboard Amazon CloudTrail data from AWS with the built-in connector (more on CloudTrail in a moment). In a few easy steps, you can create a trail and get data flowing by granting Sumo Logic access to the S3 bucket containing the logs. This can be done right in the AWS console with a few button clicks or via the CloudTrail API and takes about five minutes.



Once you've hooked up Sumo Logic, you can validate data flow by issuing queries against the CloudTrail data like so:



Now that you've got data flowing, the next step is making sense of it.

Building a detection strategy for Amazon CloudTrail

Why Amazon CloudTrail is useful

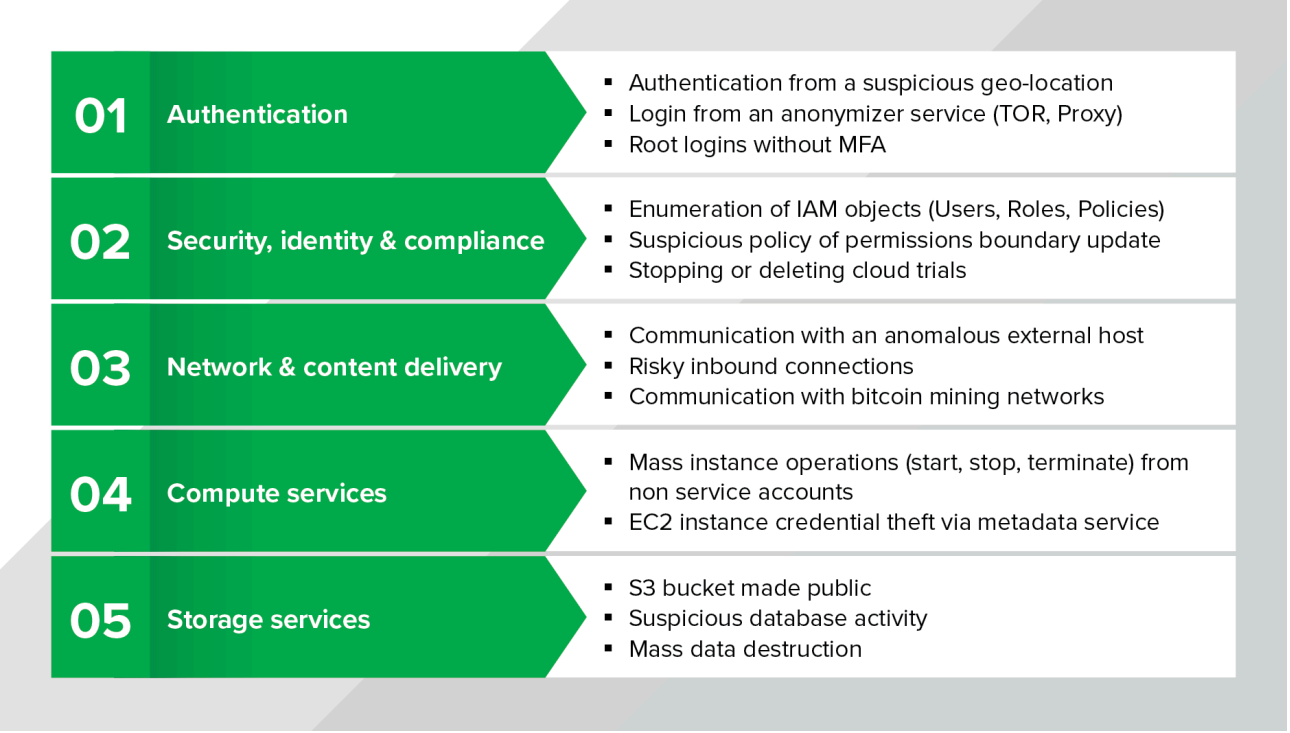
If you aren't familiar with Amazon CloudTrail, think of it as an audit log of all AWS activities that happen in your account. By default, AWS enables a default CloudTrail for every account — it records the most essential events and retains them for 90 days.

This is helpful as a default, but as a best practice it's important to create your own CloudTrail that sends events to a S3 bucket of your choosing. This allows you to control the granularity of the events that get logged and the length of time those logs are retained. You can also use AWS Organizations to enforce a global CloudTrail that sends audit data from all of your AWS accounts (including all regions) to one master S3 bucket. This ensures that you never end up in a situation where you're missing audit data needed for a compliance requirement (or — oh noes! — an active security investigation).

Making sense of CloudTrail data

Getting data flowing is relatively easy, but making sense of it all can be overwhelming (especially if it's the first time you're working with it).

At Expel, we maintain a detection and response strategy for AWS that defines what signals we look for and how our team responds. To build a detection strategy, we consider the attack lifecycle and make sure we have layered signals in place to detect attacker or risky behaviors. We think about signal in terms of these three categories, which helps us identify areas of risk and brainstorm detection use cases. Access (how does someone enter the environment), Movement (how does data move around), and Storage (where does data live). Below are a few examples that show what some of these use cases look like:



Brainstorming at a high level where your areas of risk are and what compensating controls (including detection use cases) you have in place is a healthy exercise that orgs should do routinely (We do this!).

Generating valuable AWS signals

Now I'll move on to the fun part. Let's dig into a few examples of CloudTrail-based signals that we've found valuable that might be helpful to you and your team (if you're lucky enough to have one) as well.

Suspicious logins

Credential theft isn't a new attack vector by any means, but it's still an issue for orgs of all shapes and sizes. Particularly as orgs start to use AWS, managing developer credentials gets challenging. As a result, a compromise of a developer workstation can quickly lead to a greater compromise of an AWS environment if you don't have the right controls and signals in place.

So how can you use CloudTrail logs to help zero in on this problem? Check out the query below as a starting point:

```
_sourceCategory = {{sourceCategory}}
| parse regex "(?<raw>{.*)"
| json field=raw "eventName", "sourceIPAddress", "userAgent", "userIdentity.type", "userIdentity.arn", "userIdentity.userName", "additionalEventData.MFAUsed" as event_name, src_ip, user_agent, user_type, user_arn, user_name, mfa nodrop
| lookup country_code from geo://location on ip = src_ip
| where event_name = "ConsoleLogin" and mfa != "Yes" and country_code not in ("US")
| count by country_code, mfa, user_type, user_arn, src_ip, user_agent
```

What we're looking for:

- AWS console logins where MFA wasn't used
- Unusual geo-location for the source IP address (customizable for your org)

Tips and tricks:

- Consider time of day/day of week as a condition. Should anyone be logging in on the weekend?

Account discovery

Let's assume that an attacker gets authenticated access to an environment. What happens next? In our experience, an attacker usually starts to enumerate the AWS account, listing IAM Users, Groups and Roles, and generally poking around the account to understand it. One common side effect is a burst in API failures as the attacker may not have the necessary permissions to green light all of their attempted actions.

Using this as a hypothesis, let's build a signal to alert when this happens:

```
_sourceCategory = {{sourceCategory}}
| parse regex "(?<raw>{.*)"
| json field=raw "eventName", "errorCode", "errorMessage", "sourceIPAddress", "userIdentity.userName", "userIdentity.type", "eventSource" as event_name, error_code, error_msg, src_ip, user_name, user_type, event_source nodrop
| where error_code = "AccessDenied"
| timeslice 1h
| count as failures, count_distinct(event_name) as methods, count_distinct(event_source) as sources by
```

```
_timeslice, user_type, user_name, event_source, error_msg, src_ip
| where failures > 5 and methods > 1 and sources > 1
```

What we're looking for:

- A burst in AccessDenied errors over a period of one hour
- Multiple unique failed API calls
- Multiple unique AWS services generating failures

Tips and tricks:

- You may find some noisy service accounts when first implementing this signal. This is a good opportunity to fix any broken policy documents or exclude specific accounts that you expect to generate failures.

Maintaining access

If an attacker gains an initial foothold in an environment, the attack's next objective is to figure out a way to maintain that access. In a traditional enterprise, that might look like installing a service, setting up a scheduled task or creating a backdoor user account.

What does this look like in AWS? It turns out there are some parallels – Rhino Security Labs has done a great job with Pacu, an open-source AWS exploitation toolkit (think Metasploit for AWS) that illustrates some of these behaviors.

As an example, AWS Lambda can be used as a persistence mechanism:

```
 _sourceCategory = {{sourceCategory}}
| parse regex "(?<raw>{.*)"
| json field=raw "eventName", "userIdentity.invokedBy" as event_name, invoked_by nodrop
| where invoked_by = "lambda.amazonaws.com" and event_name in ("CreateAccessKey",
"AuthorizeSecurityGroupIngress", "UpdateAssumeRolePolicy")
```

What we're looking for:

- A lambda function executing a suspicious action by:
    - Creating an access key to backdoor an IAM user
    - Updating a security group to allow ingress on a port
    - Updating an assume role policy to allow external access

Tips and tricks:

- Implement least privilege to prevent these persistence mechanisms. Don't grant users access to services they don't need.

Evading defenses

As a final example, determined attackers attempt to evade detection. Since CloudTrail logs nearly everything an attacker might want to do in an environment, CloudTrail is also an attack target. If an attacker has the right permissions, he or she can stop and/or delete a CloudTrail, making it more difficult for an organization to identify threats and respond.

You can identify this activity by looking for the following API calls:

```
_sourceCategory = {{sourceCategory}}
| parse regex "(?<raw>{.*)"
| json field=raw "eventName", "sourceIPAddress", "userIdentity.arn", "userIdentity.type",
"eventSource" as event_name, src_ip, user_arn, user_type, event_source nodrop
| where event_name in ("DeleteTrail", "StopLogging", "DeleteLogGroup", "DeleteLogStream",
"DeleteDestination")
```

What we're looking for:

- An attacker deleting audit data by:
  - Stopping or deleting a CloudTrail
  - Deleting a log group or stream from CloudWatch
  - Deleting a CloudWatch destination

Tips and tricks:

- Using AWS Organization trails are a huge help here. An attacker can't stop or delete a trail that is enforced by the master account. He or she also can't disable the default EventHistory trail that exists for each AWS account.

Bringing it all together

Getting all of this set up is a large step forward if you're just getting started with AWS monitoring, but I'd be remiss if I didn't mention one last piece of this puzzle: operationalizing these alerts. All of this work would go to waste without a well thought-out process for alert triage and investigation (we'll dive deeper into that topic in a future blog post).

Generating alerts is great, but you've got to make sure they're getting in front of the right people. Sumo Logic (and most SIEM technologies) have multiple options including dashboards, in-console workflows and other ways to send notifications to external services via email or webhooks. You may decide, for example, that you'd like to send a Slack message to a team of developers for activity that occurs in your development account.

Alternatively, if you and/or your team is bogged down already with a million other to-dos and you simply don't have the bandwidth to develop and respond to alerts, consider working with a third party. Yes, we're biased, but we'd love to chat if you think we might be able to help.

Getting a handle on your cloud infrastructure isn't always easy. Collecting and making use of audit data that these platforms generate — like AWS CloudTrail — is an important part of that mission and a good place to start.

Source: https://expel.io/blog/following-cloudtrail-generating-aws-security-signals-sumo-logic/