Detecting privilege escalation in your AWS environment

Applicability

- Product: Splunk Cloud Platform or Splunk Enterprise
- Feature: Monitoring
- Technology: Splunk App for AWS and Splunk Add-on for AWS
- Data: Amazon Web Services: CloudTrail and CloudWatch

Scenario

You are an Amazon Web Services (AWS) admin who manages AWS resources and services across your organization. As part of your role, you need to be able to query your AWS CloudTrail to detect activities related to privilege escalation.

Amazon Web Services provides a feature called Identity and Access Management (IAM) which enables organizations to manage various AWS services and resources in a secure way. All IAM users have roles, groups and policies associated with them which governs and sets permissions to allow a user to access specific restrictions. However, if these IAM policies are misconfigured and have specific combinations of weak permissions, attackers can gain escalated privileges.

These searches are designed to uncover these potentially malicious events.

- Some commands, parameters, and field names in the searches below may need to be adjusted to match your environment. In addition, to optimize the searches shown below, you should specify an index and a time range when appropriate.
- To run these searches, install the Splunk App for AWS (version 5.1.0 or later) and Splunk Add-on for AWS (version 4.4.0 or later), then configure your CloudTrail inputs.

Detection searches

► AWS create policy version to allow all resources

This search looks for CloudTrail events where a user created a policy version that allows them to access any resource in their account.

While this search has no known false positives, it is possible that an AWS admin has legitimately created a policy to allow a user to access all resources. AWS advises against granting full control to all AWS resources.
| search (errorCode=success eventName=CreatePolicyVersion eventSource=iam.amazonaws.com sourcetype=aws:cloudtrail) | spath input=requestParameters.policyDocument output=key_policy_statements path=Statement[] | mvexpand key_policy_statements | spath input=key_policy_statements output=key_policy_action_1 path=Action | search key_policy_action_1="*" | stats count min(_time) AS firstTime max(_time) AS lastTime values(key_policy_statements) AS policy_added BY eventName eventSource aws_account_id errorCode userAgent eventID awsRegion userIdentity. principalId user_arn | convert timeformat="%Y-%m-%dT%H:%M:%S" ctime(firstTime) | convert timeformat="%Y-%m-%dT%H:%M:%S" ctime(lastTime) |

**AWS create access key**

This search looks for CloudTrail events where user A who has already permission to create access keys makes an API call to create access keys for user B. Attackers have been known to use this technique for privilege escalation in case user B has more permissions than user A.

While this search has no known false positives, it is possible that an AWS admin has legitimately created keys for another user.

| search (userAgent!=console.amazonaws.com userName!=requestParameters.userName errorCode=success eventName=CreateAccessKey sourcetype=aws:cloudtrail) | stats count min(_time) AS firstTime max(_time) AS lastTime BY requestParameters.userName src eventName eventSource aws_account_id errorCode userAgent eventID awsRegion userIdentity.principalId user_arn | convert timeformat="%Y-%m-%dT%H:%M:%S" ctime(firstTime) | convert timeformat="%Y-%m-%dT%H:%M:%S" ctime(lastTime) |

**AWS create login profile**

This search looks for CloudTrail events where user A creates a login profile for user B, followed by an AWS Console login event from user B from the same source IP address as user B. This correlated event can be indicative of privilege escalation because both events happened from the same source IP address.

While this search has no known false positives, it is possible that an AWS admin has legitimately created a login profile for another user.

| search (sourcetype=aws:cloudtrail eventName=CreateLoginProfile) | rename "requestParameters.userName" AS new_login_profile | table src_ip, eventName, new_login_profile, userName | join new_login_profile,src_ip | search (sourcetype=aws:cloudtrail eventName=ConsoleLogin) | rename userName AS new_login_profile | stats count values(eventName) min(_time) AS firstTime max(_time) AS lastTime BY eventSource aws_account_id errorCode userAgent eventID awsRegion userIdentity.principalId user_arn new_login_profile src_ip | convert timeformat="%Y-%m-%dT%H:%M:%S" ctime(firstTime) | convert timeformat="%Y-%m-%dT%H:%M:%S" ctime(lastTime) |
**AWS set default policy version**

This search looks for CloudTrail events where a user has set a default policy version. Attackers have been known to use this technique for privilege escalation in case the previous versions of the policy had permissions to access more resources than the current version of the policy.

While this search has no known false positives, it is possible that an AWS admin has legitimately set a default policy to allow a user to access all resources. AWS advises against granting full control to all AWS resources.

```
| search (eventName=SetDefaultPolicyVersion eventSource=iam.amazonaws.com sourcetype=aws:cloudtrail)
| stats count min(_time) AS firstTime max(_time) AS lastTime values(requestParameters.policyArn) AS policy_arn BY src requestParameters.versionId eventName eventSource aws_account_id errorCode userAgent eventID awsRegion userIdentity.principalId user_arn
| convert timeformat=”%Y-%m-%dT%H:%M:%S” ctime(firstTime)
| convert timeformat=”%Y-%m-%dT%H:%M:%S” ctime(lastTime)
```

**AWS updated login profile**

This search looks for CloudTrail events where user A, who has already permission to update login profile, makes an API call to update the login profile for user B. Attackers have been known to use this technique for privilege escalation when user B has more permissions than user A.

While this search has no known false positives, it is possible that an AWS admin has legitimately created login profiles for another user.

```
| search (userAgent!=console.amazonaws.com userName!=requestParameters.userName errorCode=success eventName=UpdateLoginProfile sourcetype=aws:cloudtrail)
| stats count min(_time) AS firstTime max(_time) AS lastTime BY requestParameters.userName src eventName eventSource aws_account_id errorCode userAgent eventID awsRegion userName user_arn
| convert timeformat=”%Y-%m-%dT%H:%M:%S” ctime(firstTime)
| convert timeformat=”%Y-%m-%dT%H:%M:%S” ctime(lastTime)
```

---

**Additional resources**

This use case is included within Splunk Enterprise Security, a Splunk app that provides prebuilt content and searches to help answer root-cause questions in real-time about malicious and anomalous events in your IT infrastructure. In addition, Splunk Enterprise Security provides a number of other searches to help reinforce your Cloud Security posture, including:

- Detecting AWS cross-account activity
• Detecting AWS network ACL activity
• Detecting AWS suspicious provisioning activities
• Monitoring user activity spikes in AWS
• Monitoring AWS EC2 for suspicious login activities
• Monitoring AWS S3 for suspicious activities
• Monitoring AWS for suspicious traffic
• Monitoring AWS EC2 for unusual modifications