Detecting Credential Compromise in AWS

William Bengtson
Senior Security Engineer, Netflix

Credential compromise is an important concern for anyone operating in the cloud. The concerns become more widespread as more and more organizations adopt cloud resources. Impacts and outcomes from credential compromise vary depending on the motive and skill of the attacker. In some cases, it has led to erroneous AWS service usage for bitcoin mining. In others, the outcome has been much more dire and companies have been forced to close their doors due to the loss of data and infrastructure.

This paper describes an approach for detection of compromised AWS credentials without the need to know all the currently used IP addresses beforehand.

What is a credential?

“Credential” in this paper is the Amazon Web Services (AWS) API credential that is used to describe and make changes within an AWS account. The main focus is the credentials that are used on an AWS Elastic Compute Cloud (EC2) instance, although the outlined approach is valid beyond EC2. AWS provides an ability to assign permissions to an instance through an Identity and Access Management (IAM) Role. This role is attached to an EC2 instance through an instance profile, thus providing credentials to the underlying applications running on the EC2 instance.

What is CloudTrail?

CloudTrail is a service provided by AWS to log API calls that are made by your credentials in order to enable governance, compliance, and auditing. CloudTrail provides the ability to see what calls were made within an account and from where. Actions are logged regardless of whether they are performed through the AWS Management console, AWS SDKs, AWS Services operating on your behalf, or AWS command line tools.

An example record that is logged in CloudTrail is shown below[1]:

```
{
    "Records": [
        {
            "eventVersion": "1.0",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::123456789012:user/Alice",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "accountId": "123456789012",
                "userName": "Alice"
            },
            "eventTime": "2014-03-06T21:22:54Z",
            "eventSource": "ec2.amazonaws.com",
            "eventName": "StartInstances",
            "awsRegion": "us-east-2",
            "sourceIPAddress": "205.251.233.176",
            "userAgent": "ec2-api-tools 1.6.12.2",
            "requestParameters": {
                "instancesSet": {
                    "items": [
                        {
                            "instanceId": "i-ebeaf9e2"
                        }
                    ]
                }
            },
            "responseElements": {}
        }
    ]
}
```

It is important to understand what calls are logged in CloudTrail and where gaps may still exist in this logging service.

Detecting Compromise

The core concept behind detecting credential compromise is detecting use of a credential from an IP address that is not assigned to your AWS account(s) resources. The ability to describe an environment enables understanding of which IPs are currently assigned to an environment. It becomes increasingly difficult to do this as the infrastructure continues to grow in size. As a result, the time to detect compromised credentials becomes significantly longer (30+ minutes). This approach implements a sliding window for CloudTrail analysis while keeping a history of IP addresses allocated in the environment. The cloud, being API driven, allows for large changes to be made very quickly. This factor amplifies the risk of credential compromise.

The proposed alternate method is both a simple and highly effective approach at solving this problem without the inconvenience of an up-to-the-second accurate inventory requirement.

How Does it Work?

This security monitoring approach relies on an underlying understanding of how AWS operates when passing credentials to your EC2 instance and analyzing CloudTrail entries record by record.

The method relies on a data table of all EC2 assumed roles records built from the CloudTrail data. Each table entry shows the instance ID, assumed role, IP address of the API call, and a TTL entry to keep the table lean. Using the IP address of the instance making the call we can quickly and simply determine if the caller is an appropriate instance or a comprised source.

Assume Role

When you launch an EC2 instance in AWS with an IAM Role for permissions, the AWS EC2 service assumes the role specified for the instance and passes those temporary credentials to the EC2 metadata service. This `AssumeRole` action appears in CloudTrail with the following key fields:

```
eventSource: sts.amazonaws.com
eventName: AssumeRole
requestParameters.roleArn: arn:aws:iam::12345678901:role/rolename
requestParameters.roleSessionName: i-12345678901234
sourceIPAddress: ec2.amazonaws.com
```

We can determine the Amazon Resource Name (ARN) for the temporary instance credentials from this Cloud Trail log. Within EC2, AWS refreshes credentials in the EC2 metadata service every 1-6 hours.

When we see an `AssumeRole` action by the EC2 service, let's store it in a table with the following columns:

```
Instance-Id
AssumedRole-Arn
IPs
TTL
```

We can get the `Instance-Id` from the `requestParameters.roleSessionName` field. For each `AssumeRole` action, let's check to see if a row already exists in our table and if not, create one. If the row exists, let's update the TTL to keep it around. We update the TTL because the instance is still up and running so we do not want to age this table entry off. A safe TTL in this case is 6 hours due to the periodic refreshing on instance credentials within AWS, but you may decide to make it longer. You can construct the `AssumedRole-Arn` by taking the `requestParameters.roleArn` and `requestParameters.roleSessionName` from the `AssumeRole` CloudTrail record.
For e.g.: The resulting `AssumedRole-Arn` for the above entry is:

```
arn:aws:iam::12345678901:assumed-role/rolename/i-12345678901234
```

This `AssumeRole-Arn` becomes your `userIdentity.arn` entry in CloudTrail for all calls that use these temporary credentials.

AssumedRole Calls

Now that we have a table of `Instance-ID`s and `AssumeRole-ARN`s, we can start analyzing each CloudTrail record that uses these temporary credentials. Each instance-id/session row starts without an IP address to lock to, which aligns with the initial claim of not needing to know all the infrastructure with this approach.

For each CloudTrail record, we will analyze the type of record to make sure it came from an assumed role. You can do this by checking the value of `userIdentity.type` and making sure it equals `AssumedRole`. If it is an `AssumedRole`, we will grab the `userIdentity.arn` field which is equivalent to the `AssumeRole-Arn` column in the table. Since the `userIdentity.arn` has the `requestParameters.roleSessionName` in the value, we can extract the `instance-id` and do a lookup in the table to see if a row exists. If the row exists, we then check to see if there are any

IPs that this `AssumeRole-Arn` is locked to. If there aren't any then we update the table with the `sourceIPAddress` from the record and this becomes our IP address that all calls should come from. If we see a call with a `sourceIPAddress` that is not this IP, then we have detected a credential not being used on the instance it was assigned to and can assume it has been compromised.

For CloudTrail records that do not have a corresponding row in the table, we discard these as we cannot make a decision based on not having a corresponding entry in the table. This will only be the case for up to six hours due to the way AWS does temporary instance credentials within EC2. After that, we will not have the need to throw away the call because at this point, you will have all assume role entries for your environment .

Edge Cases

There are a few edge cases to this approach that you may want/need to account for in order to prevent false positives. The edge cases are as follows:

- AWS will make calls on your behalf using your credentials if certain API calls are made
- You have an AWS VPC Endpoint(s) for certain AWS Services
- You attach a new ENI or associate a new address to your instance

AWS Makes Calls on Your Behalf

If you look in your CloudTrail records, you may find that you see a `sourceIPAddress` that shows up as `<servicename>.amazonaws.com` outside of the `AssumeRole` action mentioned earlier. You will want to account for these appearing and trust AWS in these calls. You might still want to keep track of these and alert as informational.

AWS VPC Endpoints

When you make an API call in a VPC that has a VPC endpoint for your service, the `sourceIPAddress` will show up as a private IP address instead of the public IP address assigned to your instance or your VPC NAT Gateway. You will most likely need to account for having a `[public IP, private IP]` list in your table for a given `instance-id/AssumeRole-Arn` row.

Attaching a New ENI or Associating a New Address to Your Instance

You might have a use case where you attach additional ENI(s) to your EC2 instance or associate a new address through use of an Elastic IP (EIP). In these cases, you will see additional IP(s) show up in CloudTrail records for your `AssumedRole-Arn`. You will need to account for these actions in order to prevent false positives. One way to address this edge case is to inspect the CloudTrail records which associate new IPs to instances and create a table that has a row for each time a new IP was associated with the instance. This will account for the number of potential IP changes that you come across in CloudTrail. If you see a `sourceIPAddress` that does not match your lock IP, check to see if there was a call that resulted in a new IP for your instance. If so, add this IP to your IP column in your `AssumeRole-Arn` table entry, remove the entry in the additional table where you track associations, and do not alert.

Attacker Gets to it First

You might be asking the question: "Since we set the lock IP to the first API call seen with the credentials, isn't there a case where an *Attacker's* IP is set to the lock IP?" Yes, there is a slight chance that due to this approach you add an *Attacker's* IP to the Lock table because of a compromised credential. In this rare case, you will detect a "compromise" when your EC2 instance makes its first API call after the lock of the *Attacker's* IP. To minimize this rare case, you might add a script that executes the first time your AWS instance boots and makes an AWS API call that is known to be logged in CloudTrail.

Avoiding Detection

Depending on the vector used to compromise the credentials initially, the analysis of IP alone may not be enough to detect the compromise. API calls made from the compromised instance will not be detected directly from this approach without additional monitoring. One such example is with a Server Side Request Forgery (SSRF). An attacker that finds a SSRF vulnerability and gets an application to request the AWS EC2 metadata service credential path will be returned valid temporary AWS credentials that are associated with the EC2 instance. These credentials would match the `AssumeRole-Arn` mentioned earlier. Using the same SSRF attack vector, the attacker could construct API requests to AWS and pass the API call URL.

An example of a URL is below:

```
https://ec2.amazonaws.com/?Action=AssociateAddress&InstanceId=i-1234567890
abcdef0&PublicIp=192.0.2.1&AUTHPARAMS
```

To detect this type of an attack vector, you will need to take into consideration additional fields from CloudTrail such as `userAgent`. Calls resulting from a SSRF vector will not have the correct `userAgent` that the AWS SDK calls have.

Some `userAgent` strings you might find in CloudTrail are:

```
Boto3/1.7.2…
Boto/2.46.1...
aws-sdk-java/1.11.264…
aws-cli/1.11.88…
aws-sdk-go/1.12.79…..
<servicename>.amazonaws.com
```

A whitelist can be built by enumerating `userAgent` strings found in your environment to alert on deviation from.

Conclusion

By understanding how AWS operates and what is recorded in CloudTrail, you can approach detection of credential compromise without the need of having an always up to date list of IPs in your AWS account(s). This approach allows your detection to scale as your environment and infrastructure within AWS scales. From one account to many, attacking the problem through the understanding of the underlying systems involved will result in a straightforward approach to solving a hard problem. This detection mechanism should just be one layer of security in protecting your resources within AWS.

1 https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-log-file-examples.html