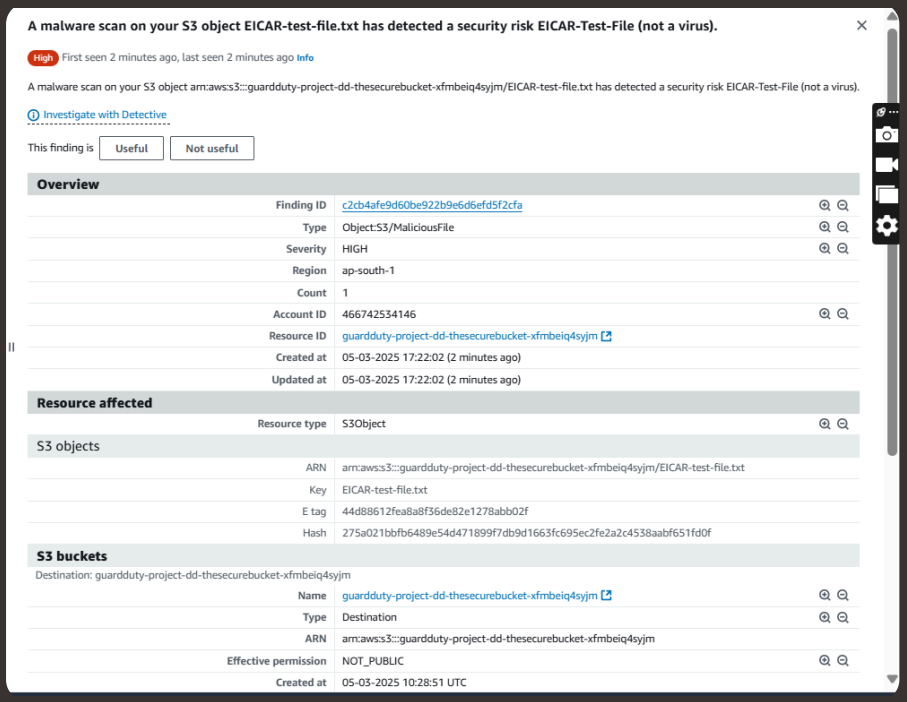
nextwork.org

Threat Detection with GuardDuty

DI Dineshraj Dhanapathy



DI
Dineshraj Dhanapathy
NextWork Student

Introducing Today's Project!

Tools and concepts

The services I used were AWS CloudFormation, S3, EC2, GuardDuty, and CloudShell. Key concepts I learnt include how to simulate real-world attacks like SQL injection and command injection, how credentials can be exfiltrated and misused, and how GuardDuty detects suspicious behavior using anomaly detection and threat intelligence. I also explored malware protection for S3 buckets and verified its effectiveness using a test file, gaining hands-on experience with AWS security monitoring and incident response.

Project reflection

This project took me approximately 2 hours to complete. The most challenging part was simulating realistic attacks while ensuring I followed best practices and didn't disrupt any live environments. Understanding how to properly use the stolen credentials in CloudShell and interpreting GuardDuty findings required careful attention. It was most rewarding to see how AWS GuardDuty accurately detected the simulated threats and to gain hands-on experience with advanced security concepts like malware protection, credential exfiltration, and anomaly detection in a controlled setting.

placeholder

I did this project today to deepen my practical knowledge of AWS security services, especially GuardDuty. My goal was to understand how real-world attacks are detected and mitigated using AWS tools.

This project definitely met my goals—it provided hands-on experience with threats like SQL injection, credential theft, and malware detection, and showed how GuardDuty identifies and reports suspicious activity effectively.

Project Setup

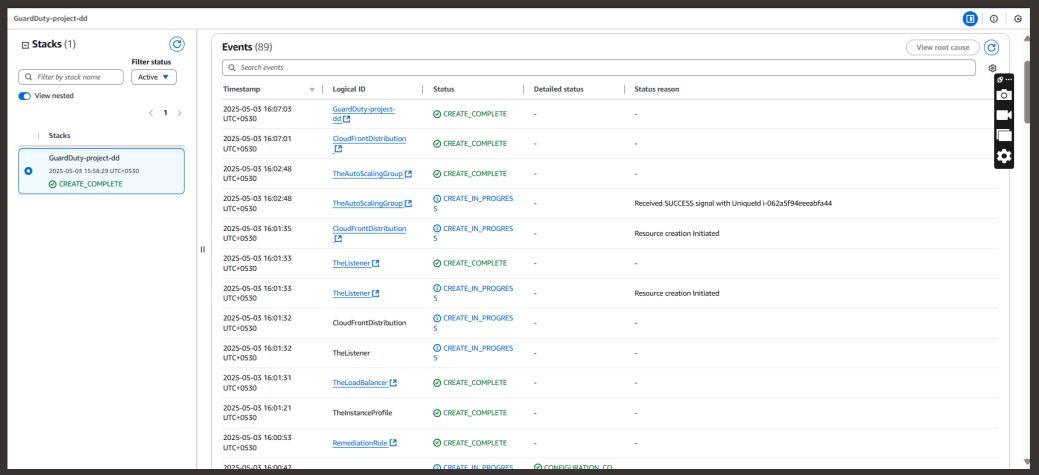
To set up for this project, I deployed a CloudFormation template that launches a fully functional web app environment. The three main components are an EC2 instance that hosts and runs the web application, custom networking resources including a new VPC to isolate and secure the infrastructure, and a CloudFront distribution that delivers the web content globally with low latency. This setup ensures performance, scalability, and security while allowing public access to the app via a unique URL.

The web app deployed is called the OWASP Juice Shop. To practice my GuardDuty skills, I will simulate real-world attacks on this vulnerable application, such as credential theft and unauthorized data access. These actions will generate suspicious behavior that GuardDuty can detect and analyze. By reviewing the alerts and findings, I will better understand how AWS detects threats and how to respond to them. This hands-on experience will help me build my cloud security skills in a safe, controlled environment.

GuardDuty is a threat detection service by AWS that continuously monitors your AWS accounts and workloads for malicious activity and unauthorized behavior. In this project, it will help identify and analyze potential attacks, such as stolen credentials being used or suspicious network traffic. By leveraging machine learning and threat intelligence, GuardDuty provides actionable security findings, allowing me to respond quickly and understand how threats impact my cloud environment.

DI

Dineshraj Dhanapathy
NextWork Student



DI
Dineshraj Dhanapathy
NextWork Student

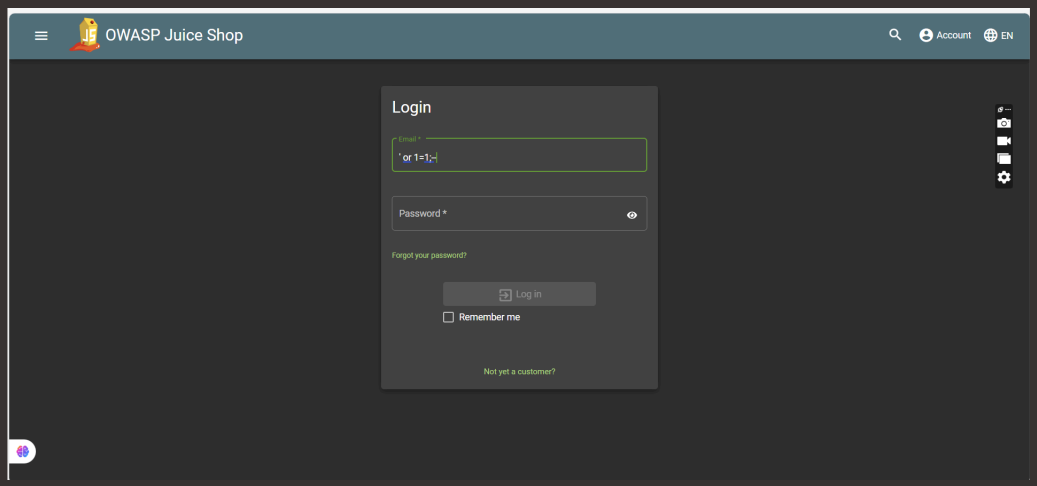
SQL Injection

The first attack I performed on the web app is SQL injection, which means I inserted malicious SQL code into a login field to manipulate the database query. SQL injection is a security risk because it allows attackers to bypass authentication, access unauthorized data, or even alter the database. By entering code like ' or 1=1;--, I tricked the application into thinking the login was valid, highlighting how insecure query handling can be exploited if input is not properly sanitized.

My SQL injection attack involved entering ' or 1=1;-- into the username or email login field. This means I injected SQL code that alters the logic of the application's authentication query. Instead of checking if the entered credentials are valid, the injected or 1=1 makes the condition always true, and -- comments out the rest of the query. As a result, I was able to bypass the login without needing a valid username or password, demonstrating a classic SQL injection vulnerability.

DI

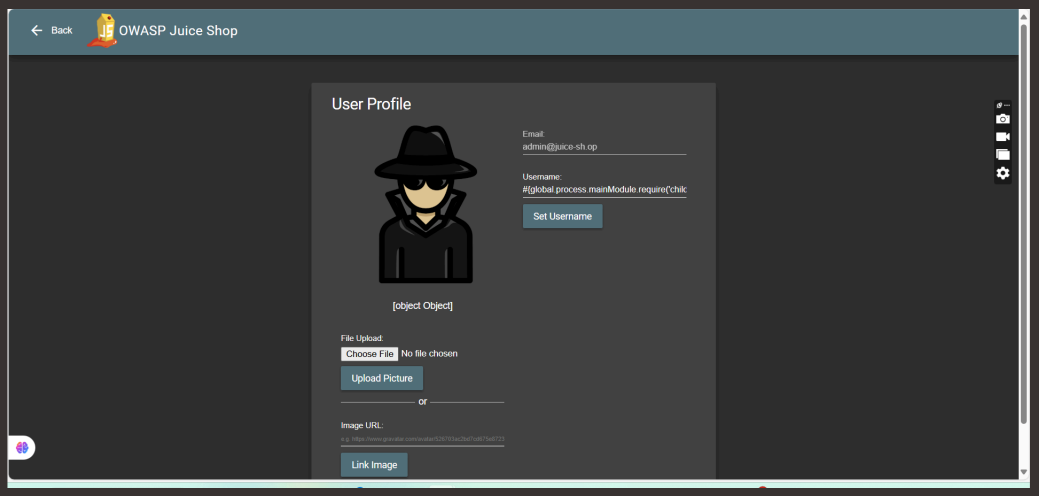
Dineshraj Dhanapathy
NextWork Student



Command Injection

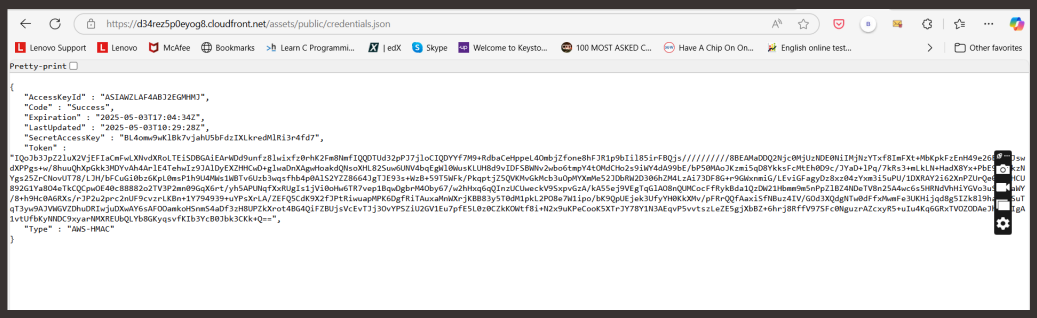
Next, I used command injection, which is an attack where malicious commands are inserted into a program's input to be executed by the system's shell. The Juice Shop web app is vulnerable to this because it doesn't properly validate user input before passing it to system-level commands. This allows attackers to run unauthorized commands on the server, potentially exposing sensitive data, compromising the system, or gaining deeper access into the underlying infrastructure.

To run command injection, I entered a malicious script into the username field that instructed the server to execute a command and create a file named credentials.json containing AWS credentials. The script will run on the server because the app fails to validate user input properly. When the app tries to display the username, it shows [object Object] since it's trying to render a full JavaScript object instead of plain text—proving that the injection worked and the file was created.


Back
OWASP Juice Shop
User Profile
Email:
admin@juice-sh.op
Username:
#{global.process.mainModule.require('chil
Set Username
[object Object]
File Upload:
Choose File
No file chosen
Upload Picture
or
Image URL:
Link Image

Attack Verification

To verify the attack's success, I accessed the credentials file that my malicious script created on the server. The credentials page showed me a JSON object containing the AccessKeyId, SecretAccessKey, and Token, confirming that I had successfully extracted temporary AWS credentials. This proves the command injection worked and that the server executed unauthorized commands. With these credentials, I could now access AWS services as if I were the application, highlighting a serious security risk.

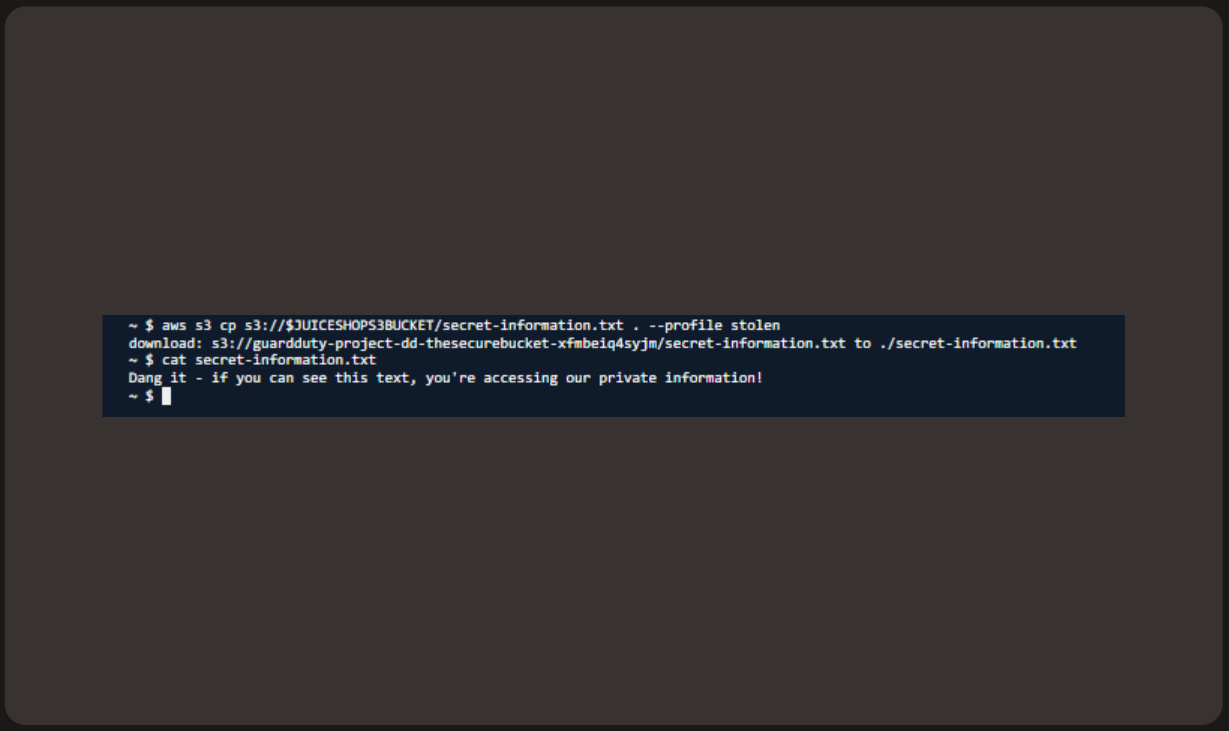


Using CloudShell for Advanced Attacks

The attack continues in CloudShell, because it simulates a scenario where a hacker uses stolen AWS credentials from outside the victim's environment. CloudShell runs in a temporary AWS account with a different account ID, so when I configure the AWS CLI with the compromised credentials and access resources like S3, it appears as if another entity is misusing the web app's credentials. This triggers GuardDuty alerts, helping me test how AWS detects unauthorized and suspicious activity.

In CloudShell, I used wget to download the credentials.json file from the vulnerable web app into my local CloudShell environment, simulating how an attacker would exfiltrate stolen credentials. Next, I ran a command using cat and jq to display and format the contents of the JSON file. This allowed me to clearly view the stolen AWS credentials—such as the AccessKeyId, SecretAccessKey, and Token—which confirmed that the attack was successful and gave me the necessary data to continue accessing AWS resources as the compromised app.

I then set up a profile, called stolen, to use the AWS credentials I extracted from the web application. I had to create a new profile because the default profile in CloudShell is linked to my IAM user's permissions. By using the stolen credentials, I can simulate an attacker's actions in the environment, performing tasks under the guise of the compromised web app. This allows me to test whether GuardDuty detects suspicious activity associated with these credentials.

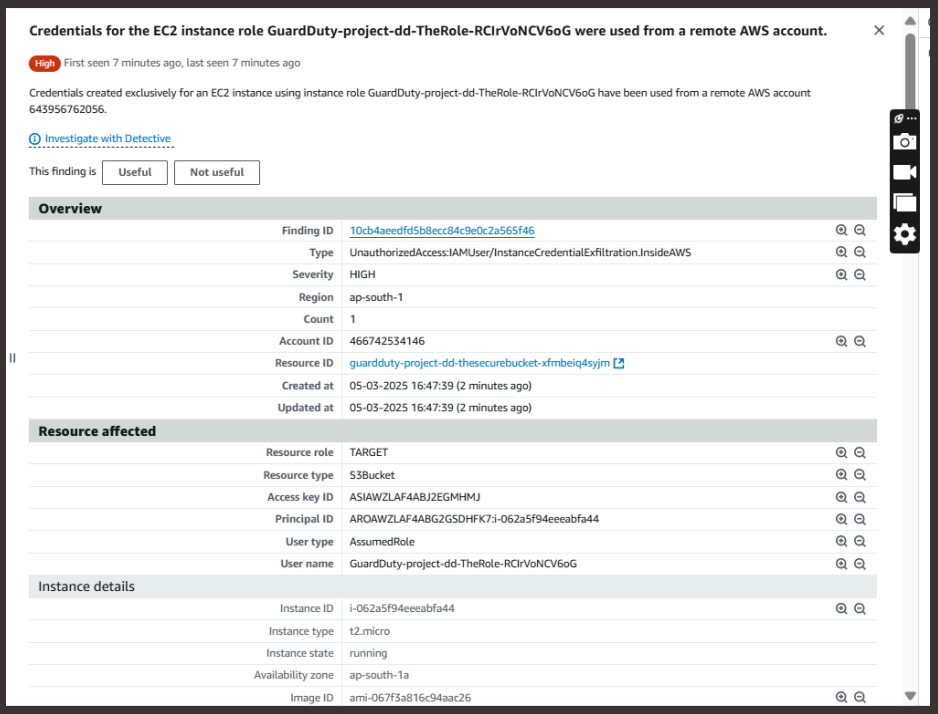

~ $ aws s3 cp s3://$JUICESHOPS3BUCKET/secret-information.txt . --profile stolen
download: s3://guardduty-project-dd-thesecurebucket-xfmbeiq4syjm/secret-information.txt to ./secret-information.txt
~ $ cat secret-information.txt
Dang it - if you can see this text, you're accessing our private information!
~ $

GuardDuty's Findings

After performing the attack, GuardDuty reported a finding within 1 minute. Findings are notifications that alert you to suspicious activity within your AWS environment. They provide crucial details, such as what happened, who carried out the action, and how it was done. This helps you understand the nature of the attack, the compromised resources, and the steps needed to remediate the situation and prevent further incidents.

GuardDuty's finding was called UnauthorizedAccess:IAMUser/InstanceCredentialExfiltration.InsideAWS, which means that GuardDuty detected suspicious use of the credentials assigned to an EC2 instance. These credentials were being used by another AWS account, indicating a potential data breach. Anomaly detection was used because GuardDuty compared the current activity (such as copying data from an S3 bucket) against the usual behavior of the EC2 instance and flagged it as unusual, triggering the alert.

GuardDuty's detailed finding reported that an unauthorized access event occurred involving the use of IAM credentials assigned to an EC2 instance. The attacker, using a different AWS account, made an API call (GetObject) to retrieve a file from an S3 bucket associated with the Juice Shop web app. The finding highlighted the use of the assumed IAM role (GuardDuty-project-dd-TheRole) and provided details like the principal ID and the exact time of the unauthorized action, indicating credential exfiltration.


Credentials for the EC2 instance role GuardDuty-project-dd-TheRole-RCIrVoNCV6oG were used from a remote AWS account.
High First seen 7 minutes ago, last seen 7 minutes ago
Credentials created exclusively for an EC2 instance using instance role GuardDuty-project-dd-TheRole-RCIrVoNCV6oG have been used from a remote AWS account 643956762056.
Investigate with Detective
This finding is Useful Not useful
Overview
Finding ID 10cb4aeedfd5b8ecc84c9e0c2a565f46
Type UnauthorizedAccess:IAMUser/InstanceCredentialExfiltration.InsideAWS
Severity HIGH
Region ap-south-1
Count 1
Account ID 466742534146
Resource ID guardduty-project-dd-thesecurebucket-xfmbeiq4syjm
Created at 05-03-2025 16:47:39 (2 minutes ago)
Updated at 05-03-2025 16:47:39 (2 minutes ago)
Resource affected
Resource role TARGET
Resource type S3Bucket
Access key ID ASIAWZLAF4ABJ2EGMHMJ
Principal ID AROAWZLAF4ABG2GSDHFK7:i-062a5f94eeeabfa44
User type AssumedRole
User name GuardDuty-project-dd-TheRole-RCIrVoNCV6oG
Instance details
Instance ID i-062a5f94eeeabfa44
Instance type t2.micro
Instance state running
Availability zone ap-south-1a
Image ID ami-067f3a816c94aac26

Extra: Malware Protection

For my project extension, I enabled Malware Protection for S3 in GuardDuty. Malware is a significant threat in cloud environments, and enabling this feature allows GuardDuty to scan objects in S3 buckets for potential malware. By doing so, I ensure that malicious files are detected and reported. The Tag scanned objects setting automatically labels scanned files with statuses like NO_THREATS_FOUND or THREATS_FOUND, making it easier to track file security. This helps streamline security management without digging through logs.

To test Malware Protection, I uploaded the EICAR test file to my S3 bucket. The uploaded file won't actually cause damage because it's a safe, standardized file used across the industry to simulate malware detection. It’s recognized by security tools as a test virus, allowing me to verify that GuardDuty can detect and respond to potential threats without introducing any real risk to my environment.

Once I uploaded the file, GuardDuty instantly triggered a malware detection finding on the EICAR test file. This verified that Malware Protection is actively monitoring my S3 buckets and can detect known threats effectively. It proved that GuardDuty can identify suspicious files quickly, even if they are just test signatures like the EICAR file, which simulates malware without causing harm. This confirms that my AWS environment is being continuously scanned for security risks.

DI

Dineshraj Dhanapathy
NextWork Student