

# **Midterm Group Project - Securing an E-Commerce Platform**

## **ASSESSMENT REPORT**

### **GROUP-6**

**Atharva Bhingarkar – 120239366**

**Zhiwen Zhu – 121325471**

**Abhijeet Kumar – 121376701**

# Introduction

In our evaluation of the provided infrastructure YAML configuration, we discovered several critical vulnerabilities that could significantly threaten the confidentiality, integrity, and availability of the deployed system. These vulnerabilities include the absence of encryption mechanisms, inadequate access control settings, lack of network segmentation, and the omission of essential security measures such as logging, firewalls, and backup systems.

The key findings include:

- **Encryption:** Sensitive information, such as database storage and network traffic, is unencrypted, rendering it vulnerable to data breaches.
- **Network Access Control:** The instance has publicly accessible ports, permitting unrestricted traffic connections.
- **Segregation:** The database is located on the same instance as the application, which violates best practices for security isolation.
- **Outdated Software:** There are no update mechanisms in place to ensure that installed packages, such as MariaDB and Node.js, are maintained at their latest secure versions, exposing the instance to potential vulnerabilities.
- **WAF (Web Application Firewall):** The security group configuration lacks a firewall as a whole, as well as any configured rules, leaving critical ports exposed to public access.
- **Load Balancer:** There is an absence of a load balancer to securely manage incoming traffic or to mitigate DDoS attacks.
- **Logging and Monitoring:** The instance lacks any logging and monitoring system, which interferes with the ability to detect and respond to incidents, as well as recovery efforts.
- **Backups:** There is no implemented backup strategy, which heightens the risk of data loss in the event of a system failure or cyberattack.
- **IAM Misconfigurations:** The developer possesses the same privileges as the administrator, which violates the Principle of Least Privilege. Additionally, the guest role has excessive permissions allowed to it.

# Table of Findings

<b>1. Vulnerability Assessment Report</b>	<b>4</b>
1.1 Docker Image Vulnerabilities	
1.2 Vulnerable Dependencies in Application Code	
1.3 Exposed Backup and Certificate Files	
1.4 Insufficient AWS CloudTrail Logging Coverage	
1.5 Insecure EC2 Security Group Configurations	
<b>2. Data security Assessment Report</b>	<b>15</b>
2.1 EBS Volume Not Encrypted	
2.2 MariaDB Database Not Encrypted with Weak password	
2.3 Unencrypted Data Transmission	
2.4 Excessive File Permissions on Database Storage	
2.5 Lack of Automated Backup and Disaster Recovery strategy	
2.6 No Log set ups for sensitive data	
<b>3. Virtual Machine Vulnerability Assessment Report</b>	<b>20</b>
3.1 MariaDB	
3.2 Encryption	
3.3 Express-jwt	
3.4 Cross-Spawn	
3.5 MarsDB	
3.6 VM2	
<b>4. Network security Assessment Report</b>	<b>29</b>
4.1 Unused Private Subnet (Lack of Network Segmentation)	
4.2 Lack of Network Access Control Lists (NACLs)	
4.3 Overly Permissive Security Group Rules	
4.4 Lack of Network Traffic Monitoring and Threat Detection	
4.5 Insecure Web Traffic Due to Missing HTTPS (TLS) Enforcement	
4.6 Lack of Application Load Balancer (ALB) Protection	
4.7 Lack of Auto Scaling for EC2 Instances	
4.8 Excessive IAM Privileges and Insecure Default Account	
<b>5. Disaster Recovery Assessment Report</b>	<b>37</b>
5.1 Lack of Automated Backups & Secure Storage	
5.2 Single-Point-of-Failure Architecture	
5.3 Absent Disaster Recovery Plan	
5.4 Missing Incident Response & Recovery Documentation	
5.5 Absent Auto Scaling & Failover Mechanisms	

# Vulnerability Assessment Report

## 1.1 Docker Image Vulnerabilities

**Tool Used:** Trivy

**Target:** bkimminich/juice-shop (Debian 11.10)

### Finding:

The Docker image hosting the Juice Shop application contains some critical vulnerabilities in both system and application-level dependencies. Using Trivy, 29 vulnerabilities were identified — some affecting Node.js libraries despite the application being actively maintained.

A notable and critical vulnerability was found in the libxmljs package:

- CVE-2024-34391 & CVE-2024-34392: These are confusion-type vulnerabilities while parsing specially designed XML inputs by libxmljs. This vulnerability enables the attackers to manipulate memory and can result in arbitrary code execution within the container.

libxmljs (package.json)	CVE-2024-34391	CRITICAL	affected	1.0.11		libxmljs vulnerable to type confusion when parsing specially crafted XML <a href="https://avd.aquasec.com/nvd/cve-2024-34391">https://avd.aquasec.com/nvd/cve-2024-34391</a>
	CVE-2024-34392					libxmljs vulnerable to type confusion when parsing specially crafted XML <a href="https://avd.aquasec.com/nvd/cve-2024-34392">https://avd.aquasec.com/nvd/cve-2024-34392</a>

Figure 1 – Trivy scan showing critical vulnerabilities (CVE-2024-34391, CVE-2024-34392) in the libxmljs package, due to type confusion when parsing specially crafted XML inputs.

### Recommendation:

- Upgrade or replace the libxmljs dependency to a version that fixes these vulnerabilities.
- Wherever feasible, utilize lighter-weight and safer base images, i.e., Alpine or distroless.
- Periodically scan and rebuild Docker images to incorporate the most recent security patches.
- Consider using software composition analysis (SCA) tools within CI/CD pipelines to catch such bugs sooner.

### Impact:

Successful exploitation may lead to remote code execution within the container, potentially compromising application secrets or being leveraged as a pivot point to attack the system or cloud environment further.

**Risk:** Critical - Exploitation of these types of vulnerabilities would result in full compromise of the container ecosystem, revealing sensitive data and allowing pivoting across the infrastructure.

### Reference:

- [CVE-2024-34391 – libxmljs type confusion](#)
- [CVE-2024-34392 – libxmljs type confusion](#)

## 1.2 Vulnerable Dependencies in Application Code

**Tool Used:** Snyk

**Target:** Juice Shop application source code (via Docker image)

**Scan Path:** bkimminich/juice-shop

### Finding:

The Snyk scan detected 27 high-severity and critical vulnerabilities in the app dependencies. These were detected in Node.js packages on which the Juice Shop application depends, mostly due to outdated third-party libraries. The scan reviewed 1024 dependencies, checking an enormous vulnerability footprint.

### Notable Critical Findings:

- Arbitrary Code Injection in marsdb@0.6.11
  - CVE: [SNYK-JS-MARSDB-480405](#)
  - Impact: Allows injection and execution of malicious code via user-controlled input.
  - Screenshot Evidence:

```
x Arbitrary Code Injection [Critical Severity][https://security.snyk.io/vuln/SNYK-JS-MARSDB-480405] in marsdb@0.6.11
introduced by marsdb@0.6.11
```

- Arbitrary Code Execution in sanitize-html@1.4.2
  - CVE: [SNYK-JS-SANITIZEHTML-585892](#)
  - Impact: Attackers may execute untrusted code inside the application runtime.
  - Screenshot Evidence:

```
x Arbitrary Code Execution [Critical Severity][https://security.snyk.io/vuln/SNYK-JS-SANITIZEHTML-585892] in sanitize-html@1.4.2
introduced by sanitize-html@1.4.2
```

- Remote Code Execution (RCE) in vm2@3.9.17
  - CVE: [SNYK-JS-VM2-5772823](#)
  - Impact: Enables sandbox escapes, leading to full container or host compromise.
  - Screenshot Evidence:

```
x Remote Code Execution (RCE) [Critical Severity][https://security.snyk.io/vuln/SNYK-JS-VM2-5772823] in vm2@3.9.17
introduced by juicy-chat-bot@0.8.0 > vm2@3.9.17
No upgrade or patch available
x Remote Code Execution (RCE) [Critical Severity][https://security.snyk.io/vuln/SNYK-JS-VM2-5772825] in vm2@3.9.17
introduced by juicy-chat-bot@0.8.0 > vm2@3.9.17
No upgrade or patch available
```

### Other High Severity Findings:

- Prototype Pollution in lodash@2.4.2
  - CVE: [SNYK-JS-LODASH-450202](#)
  - Impact: May alter app behavior, escalate privileges, or bypass input validation.

- Screenshot Evidence:

```

x Prototype Pollution [High Severity][https://security.snyk.io/vuln/SNYK-JS-LODASH-450202] in lodash@2.4.2
  introduced by sanitize-html@1.4.2 > lodash@2.4.2
x Prototype Pollution [High Severity][https://security.snyk.io/vuln/SNYK-JS-LODASH-608086] in lodash@2.4.2
  introduced by sanitize-html@1.4.2 > lodash@2.4.2
x Prototype Pollution [High Severity][https://security.snyk.io/vuln/SNYK-JS-LODASH-6139239] in lodash@2.4.2
  introduced by sanitize-html@1.4.2 > lodash@2.4.2
x Prototype Pollution [High Severity][https://security.snyk.io/vuln/SNYK-JS-LODASH-73638] in lodash@2.4.2
  introduced by sanitize-html@1.4.2 > lodash@2.4.2

```

- JWT Authentication Bypass in express-jwt@0.1.3

- CVE: [SNYK-JS-EXPRESSJWT-575022](#)
- Impact: Attackers may bypass authorization mechanisms using forged JWTs.
- Screenshot Evidence:

```

x Authorization Bypass [High Severity][https://security.snyk.io/vuln/SNYK-JS-EXPRESSJWT-575022] in express-jwt@0.1.3
  introduced by express-jwt@0.1.3

```

## Denial of Service (DoS) – Multiple Attack Vectors

Certain DoS vulnerabilities were found in the dependencies that were used by socket.io, which is responsible for handling real-time bidirectional communication in Node.js applications. The vulnerabilities can be used by attackers to flood or crash services with malicious WebSocket messages.

- engine.io@4.1.2

- CVE: [SNYK-JS-ENGINEIO-3136336](#)
- Impact: May allow attackers to send specially crafted payloads that exhaust server resources or cause unhandled exceptions.
- Screenshot Evidence:

```

x Denial of Service (DoS) [High Severity][https://security.snyk.io/vuln/SNYK-JS-ENGINEIO-3136336] in engine.io@4.1.2
  introduced by socket.io@3.1.2 > engine.io@4.1.2

```

- socket.io-parser@4.0.5

- CVE: [SNYK-JS-SOCKETIOPARSER-5596892](#)
- Impact: Vulnerable to crafted payloads that can trigger infinite loops or memory leaks, resulting in downtime.
- Screenshot Evidence:

```

x Denial of Service (DoS) [High Severity][https://security.snyk.io/vuln/SNYK-JS-SOCKETIOPARSER-5596892] in socket.io-parser@4.0.5
  introduced by socket.io@3.1.2 > socket.io-parser@4.0.5
  This issue was fixed in versions: 3.4.3, 4.2.3

```

- ws@7.4.6

- CVE: [SNYK-JS-WS-7266574](#)
- Impact: Improper handling of excessive headers in WebSocket requests could lead to memory exhaustion.
- Screenshot

Evidence:

```

x Denial of Service (DoS) [High Severity][https://security.snyk.io/vuln/SNYK-JS-WS-7266574] in ws@7.4.6
  introduced by socket.io@3.1.2 > engine.io@4.1.2 > ws@7.4.6

```

**Recommendation:**

- Immediately upgrade the vulnerable dependencies to fixed versions if possible.
- For libraries like vm2, for which there are no patches available, look into disabling or sandboxing their usage.
- Automate dependency scanning in the CI/CD pipeline.
- Use Software Composition Analysis (SCA) periodically to minimize exposure due to transitive vulnerabilities.

**Impact:**

These vulnerabilities expose the application to:

- Remote code execution (RCE) and container compromise.
- Authorization bypass and privilege escalation.
- Application logic tampering or total denial-of-service (DoS). Given the sensitive nature of Juice Shop (a intentionally vulnerable app to test security), these bugs are realistic threats applicable to real-world production-ready Node.js applications.

**Risk:** Critical

**Reference:**

- [Snyk Vulnerability Database](#)
- OWASP Top 10 - A06:2021: Vulnerable and Outdated Components

## 1.3 Exposed Backup and Certificate Files

**Tool Used:** Nikto

**Target:** <http://3.239.71.173:3000>

**Scan Output File:** nikto\_juice.html

**Finding:**

The Nikto web vulnerability scan revealed that sensitive backup and certificate files are publicly accessible on the Juice Shop web server. Files like .pem, .jks, .war, and .tar were directly reachable over HTTP. These files often contain private keys, database dumps, or application artifacts.

**Examples of exposed files include:**

- /173.pem – Private key certificate file
- /database.tar – Potentially full database archive
- /site.war – Deployed application package

- /database.pem, /323971.alz, /3.239.71.173.war – Sensitive certs and archives

These indicate a severe misconfiguration in directory access policies and CI/CD hygiene.

### Screenshot Evidences:

- Nikto Target Summary – Confirms the scan target, port, and timestamp

```
- Nikto v2.5.0

+ Target IP:      3.239.71.173
+ Target Hostname: 3.239.71.173
+ Target Port:    3000
+ Start Time:     2025-03-29 23:37:37 (GMT-4)
```

- Detected File Exposures:

```
/173.pem: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /database.tar: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /site.war: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /database.pem: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /323971.alz: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /3.239.71.173.war: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
```

- Nikto Scan Summary Page – Shows 165 findings, validating detection confidence

Host Summary	
Start Time	2025-03-29 23:45:17
End Time	2025-03-29 23:45:51
Elapsed Time	34 seconds
Statistics	661 requests, 19 errors, 165 findings

Scan Summary	
Software Details	<a href="#">Nikto 2.5.0</a>
CLI Options	-h http://3.239.71.173:3000 -o nikto_juice.html
Hosts Tested	1
Start Time	Sat Mar 29 23:45:17 2025
End Time	Sat Mar 29 23:45:51 2025
Elapsed Time	34 seconds

### Recommendation:

- Limit file access: Utilize .htaccess rules, NGINX/Apache directives, or cloud storage bucket policies to limit access to all of the sensitive file types.
- Remove visible artifacts: Remove unnecessary build and configuration files from production servers.
- Auto clean-up: Let CI/CD pipelines clear artifacts after deploy and disable directory listing.



- Keep an eye on access logs for unauthorized downloads of .tar, .pem, .war, .jks, etc.

**Impact:**

Public access to sensitive files such as certificates and backup archives can lead to:

- Remote code execution (through .war, .tar restore)
- TLS spoofing or man-in-the-middle attacks (via leaked .pem, .cer)
- Database intrusion exposing PII or PHI in health care settings
- An attacker can utilize these files for privilege escalation, lateral movement, or full application compromise.

An attacker could exploit these files for privilege escalation, lateral movement, or full application compromise.

**Risk: High**

The exposure of unprotected deployment files and private keys greatly enhances the potential impact of any attack attempted.

**Reference:**

- CWE-530: Exposure of Backup Files to Unauthorized Control Sphere
- OWASP: Insecure File Uploads

## 1.4 Insufficient AWS CloudTrail Logging Coverage

**Tool Used:** ScoutSuite

**Target:** AWS Account hosting the Juice Shop application

**Finding:**

AWS CloudTrail is improperly set up to log all account activity or disabled in some regions, hence resulting in wide logging and auditing visibility gaps. This results in unknown security incidents, especially in unconfigured regions, as well as noncompliance with AWS security standards and best practices such as the CIS AWS Benchmark.

ScoutSuite identified that:

- CloudTrail is not enabled in all regions
- Centralized and tamper-proof logging isn't required
- Potential security events will go undetected in uncovered areas

**Recommendation:**

- Enable AWS CloudTrail in all supported regions to have full coverage.
- Configure trails to publish logs to an encrypted, access-limited S3 bucket.
- Enable log file validation to detect tampering.
- Use retention policies and log integrity monitoring with tools such as AWS Config and GuardDuty.
- Regularly check audit trail configurations for drift or disablement.

**Impact:**

The lack of centralized and regional CloudTrail logging creates a critical blind spot:

- Malicious activity in unlogged regions (e.g., unauthorized access or lateral movement) can go undetected.
- Post-incident analysis is effectively impossible without reliable audit logs.
- In healthcare use cases, this is also a compliance breach (e.g., HIPAA), especially if sensitive PHI access is not tracked.

**Risk: High**

**Reference:**

- [AWS CloudTrail Best Practices](#)
- CIS AWS Foundations Benchmark – Control 2.11

# CloudTrail Service Not Configured

Amazon Web Services > 137068236985

Dashboard

Service	Resources	Rules	Findings	Checks
ACM	0	2	0	0
Lambda	0	0	0	0
CloudFormation	1	1	0	1
CloudFront	0	3	0	0
CloudTrail	0	9	17	17
CloudWatch	0	1	0	0

CloudTrail Service Not Configured

Description

CloudTrail is not configured, which means that API activity is not logged.

Regions checked: 17

Regions flagged: 17

References

<https://docs.aws.amazon.com/awscloudtrail/latest/userguide/best-practices-security.html>

ScoutSuite output showing CloudTrail service not configured across all AWS regions, violating CIS AWS Benchmark controls for audit logging (2.11).

ap-northeast-1

Information

Configured: False

Trails

0

ap-northeast-2

Information

Configured: False

Trails

0

ap-northeast-3

Information

Configured: False

Trails

0

ap-south-1

Information

Configured: False

Trails

0

ap-southeast-1

Information

Configured: False

Trails

0

## 1.5 Insecure EC2 Security Group Configurations

### Description:

ScoutSuite identified excessively permissive security group rules on the EC2 instances supporting the healthcare application. These rules provide broad network access from all IP addresses (0.0.0.0/0) to sensitive ports, including:

- Port 22 (SSH) – critical for administrative access.
- Port 3000 – associated with the application frontend.

These open ports expose the instance to the public internet, making it vulnerable to scanning, brute-force attacks, and potential remote code execution.

### Recommendation:

- Enforce the Principle of Least Privilege on all security groups.
- Restrict access to administrative ports like SSH (22) using whitelisted IP addresses only.
- Isolate application components using segmented security groups.
- Consider using a bastion host or VPN for administrative access instead of exposing SSH directly to the internet.
- Regularly review and audit inbound/outbound rules.

### Impact:

Unrestricted network exposure dramatically increases the attack surface. Attackers can:

- Attempt unauthorized access via brute-force on SSH.
- Exploit vulnerable application ports (e.g., 3000) to gain unauthorized control.
- Use open ports as entry points for pivoting, lateral movement, or launching Denial of Service (DoS) attacks.

**For a healthcare system, such attacks could lead to compromise of Protected Health Information (PHI) or downtime of mission-critical services.**

**Risk: High**

### Reference:

- AWS EC2 Security Best Practices
- [CIS AWS Foundations Benchmark – Section 4.11](#)

## Insecure EC2 Security Group Configurations

● DynamoDB	0	0	0	0
❗ EC2	39	29	93	622
● EFS	0	0	0	0

❗ EBS Volume Not Encrypted	+
❗ Potential Secret in Instance User Data	+
❗ Security Group Opens SSH Port to All	+
⚠ EBS Encryption By Default Is Disabled	+
⚠ Non-empty Rulesets for Default Security Groups	+
⚠ Security Group Opens All Ports	+
⚠ Security Group Opens TCP Port to All	+
⚠ Unrestricted Network Traffic within Security Group	+

## Unencrypted EBS Volume

❗ EBS Volume Not Encrypted	—
<b>Description</b> Enabling encryption of EBS volumes ensures that data is encrypted both at-rest and in-transit (between an instance and its attached EBS storage).	◦ Volumes checked: 1 ◦ Volumes flagged: 1
<b>References</b> <ul style="list-style-type: none"><li>◦ <a href="https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSEncryption.html">https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSEncryption.html</a></li></ul>	

## Potential Secret in User Data

❗ Potential Secret in Instance User Data	—
<b>Description</b> It was detected that the EC2 instance was configured with user data, which could potentially include secrets. Although user data can only be accessed from within the instance itself, the data is not protected by cryptographic methods. Anyone who can access the instance can view its metadata. It should therefore be ensured that sensitive data, such as passwords and SSH keys, are not stored as user data.	◦ Instances checked: 1 ◦ Instances flagged: 1
<b>References</b> <ul style="list-style-type: none"><li>◦ <a href="https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html">https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html</a></li></ul>	

## Finding indicating SSH port (22) is open to all IPs, violating CIS AWS Benchmark recommendations

❗ Security Group Opens SSH Port to All	—
<b>Description</b> The security group was found to be exposing a well-known port to all source addresses. Well-known ports are commonly probed by automated scanning tools, and could be an indicator of sensitive services exposed to Internet. If such services need to be exposed, a restriction on the source address could help to reduce the attack surface of the infrastructure.	◦ Rules checked: 23 ◦ Rules flagged: 1
<b>Remediation</b> Remove the inbound rules that expose open ports	
<b>Compliance</b> <ul style="list-style-type: none"><li>◦ CIS Amazon Web Services Foundations version 1.0.0, reference 4.1</li><li>◦ CIS Amazon Web Services Foundations version 1.0.0, reference 4.2</li><li>◦ CIS Amazon Web Services Foundations version 1.1.0, reference 4.1</li><li>◦ CIS Amazon Web Services Foundations version 1.1.0, reference 4.2</li><li>◦ CIS Amazon Web Services Foundations version 1.2.0, reference 4.1</li><li>◦ CIS Amazon Web Services Foundations version 1.2.0, reference 4.2</li></ul>	

```
# Create a Security Group for the EC2 instance
WebServerSecurityGroup:
  Type: "AWS::EC2::SecurityGroup"
  Properties:
    VpcId: !Ref MidtermVPC
    GroupDescription: "Allow SSH, HTTP, and HTTPS access to the EC2 instance"
    SecurityGroupIngress:
      - IpProtocol: "tcp"
        FromPort: "80"
        ToPort: "80"
        CidrIp: "0.0.0.0/0" # Allow HTTP access
      - IpProtocol: "tcp"
        FromPort: "443"
        ToPort: "443"
        CidrIp: "0.0.0.0/0" # Allow HTTPS access
      - IpProtocol: "tcp"
        FromPort: "22"
        ToPort: "22"
        CidrIp: "0.0.0.0/0" # Allow SSH access
```

Open access to ports 22 (SSH), 3000 (App), etc. via 0.0.0.0/0

```
NetworkInterfaces:
  - AssociatePublicIpAddress: true # Associate public IP here
    SubnetId: !Ref PublicSubnet
```

EC2 instance has a public IP and lives in a public subnet.

```
# Create a route to the Internet in the Public Route Table
PublicRoute:
  Type: "AWS::EC2::Route"
  Properties:
    RouteTableId: !Ref PublicRouteTable
    DestinationCidrBlock: "0.0.0.0/0"
    GatewayId: !Ref InternetGateway
```

The instance launched in a subnet mapped to the internet via a public route.

# Data Security Assessment Report

## 2.1 EBS Volume Not Encrypted

**Impact:** The EBS volumes attached to the EC2 instance are not encrypted, allowing anyone with access to the instance and sufficient permissions to create snapshots and potentially retrieve sensitive data stored on the volume.

**Risk:** Attackers can extract and read stored data, potentially resulting in severe data breaches, compliance issues, and damage to the organization's reputation.

### **Recommendation:**

- Launch a new EC2 instance with EBS encryption enabled.
- Use KMS to store and manage the encryption key.
- Set clear access control of who can access the EBS encrypted data and KMS.

### **Reference:**

Volume ID: vol-0f87b4bb676129b45		
<div><div>Details</div><div>Status checks</div><div>Monitoring</div><div>Tags</div></div>		
<div><div>Volume ID</div><div>vol-0f87b4bb676129b45</div></div> <div><div>AWS Compute Optimizer finding</div><div>Opt-in to AWS Compute Optimizer for recommendations.   <a href="#">Learn more</a></div></div> <div><div>Fast snapshot restored</div><div>No</div></div> <div><div>Attached resources</div><div>i-0fae36c979218135c (VulnerableInstance); /dev/xvda (attached)</div></div> <div><div>Source</div><div>Snapshot ID</div><div>snap-05bb5ec75f10d2a97</div></div> <div><div>Encryption</div><div>Not encrypted</div></div>	<div><div>Size</div><div>8 GiB</div></div> <div><div>Volume state</div><div>In-use</div></div> <div><div>Availability Zone</div><div>us-east-1a</div></div> <div><div>Outposts ARN</div><div>-</div></div> <div><div>KMS key ID</div><div>-</div></div>	<div><div>Type</div><div>gp3</div></div> <div><div>IOPS</div><div>3000</div></div> <div><div>Created</div><div>Tue Apr 01 2025 11:43:16 GMT-04</div></div> <div><div>Managed</div><div>false</div></div> <div><div>KMS key alias</div><div>-</div></div>

I have deployed this template to AWS, as we can see, there is neither encrypted EBS nor KMS used.

## 2.2 MariaDB Database Not Encrypted with Weak password

**Impact:** The MariaDB database does not use Transparent Data Encryption (TDE) or encryption, storing all sensitive data in plaintext on the EC2 instance. It also uses a weak, hardcoded root password (**vulnerable**) during provisioning.

**Risk:** An attacker or any unauthorized user with access to the EC2 instance can directly brute force the database password, thereby also being able to dump or tamper with data and extract customer records.

**Remediation:** Migrate to AWS RDS and enable RDS encryption (KMS) or implement encryption within MariaDB.

- Migrate the database to a newly created AWS RDS.
- Encrypted the AWS RDS at rest using AWS KMS.

### **Reference:**

```
# Start and enable MariaDB
echo "Starting and enabling MariaDB" | tee -a $LOG_FILE
systemctl start mariadb 2>&1 | tee -a $LOG_FILE
systemctl enable mariadb 2>&1 | tee -a $LOG_FILE

# Set up MariaDB (vulnerable root password)
echo "Setting up MariaDB" | tee -a $LOG_FILE
mysql -u root -e "SET PASSWORD FOR root@'localhost' = PASSWORD('vulnerable');" 2>&1 | tee -a $LOG_FILE
mysql -u root -p'vulnerable' -e "CREATE DATABASE juice_shop;" 2>&1 | tee -a $LOG_FILE
```

## 2.3 Unencrypted Data Transmission

**Impact:** The EC2 instances do not enforce HTTPS/TLS, resulting in unencrypted data transmission between the client and the server, which may lead to sensitive information disclosure.

**Risk:** Attackers can perform man-in-the-middle (MITM) attacks to intercept unencrypted credentials, session tokens, and other sensitive data during transmission.

### **Remediation:**

- Deploy a valid SSL/TLS certificate(ACM).
- Restrict all HTTP traffic to HTTPS.
- Disable insecure protocols (use TLS 1.2 or above version)



**Reference:**

sg-05875bfc64eec47c3 - midterm-WebServerSecurityGroup-tjzw9kb9mUDx

Details	Inbound rules	Outbound rules	Sharing - new	VPC associations - new	Tags
---------	---------------	----------------	---------------	------------------------	------

### Inbound rules (3)

<input type="checkbox"/>	Name	Security group rule ID	IP version	Type	Protocol	Port range	Source	Description
<input type="checkbox"/>	-	sgr-00b8af64db066c34f	IPv4	HTTPS	TCP	443	0.0.0.0/0	-
<input type="checkbox"/>	-	sgr-0adf12e510ad8cb0b	IPv4	HTTP	TCP	80	0.0.0.0/0	-
<input type="checkbox"/>	-	sgr-0bc26865f7cbe59e7	IPv4	SSH	TCP	22	0.0.0.0/0	-

## 2.4 Excessive File Permissions on Database Storage

**Impact:** The `/opt` and `/opt/juice-shop/data/` directories have overly permissive permissions (755), allowing unauthorized users to access sensitive database files.

**Risk:** Attackers gaining server-level access can directly read database and log files, leading to unauthorized data disclosure. (potential risk of deleting sensitive data)

**Remediation:** Restrict file permissions (e.g., to 700 or 750) to ensure only database administrators and authorized application users have access.

**Reference:**

```
chown -R ec2-user:ec2-user /opt 2>&1 | tee -a $LOG_FILE
chmod -R 755 /opt 2>&1 | tee -a $LOG_FILE
```

```
# Set permissions for the data directory if it exists
if [ -d "/opt/juice-shop/data" ]; then
    chmod -R 755 /opt/juice-shop/data 2>&1 | tee -a $LOG_FILE
else
    echo "Directory 'data/' does not exist, skipping chmod." | tee -a $LOG_FILE
fi
```

[illegible]

Under this permission, unauthorized users can also retrieve those user information.

## 2.5 Lack of Automated Backup and Disaster Recovery strategy

**Impact:** The current cloud deployment lacks an automated backup and disaster recovery strategy for the MariaDB database hosted on the EC2 instance and the EC2 volume (EBS). No periodic snapshots, backup retention, or multi-region replication are configured. This exposes the infrastructure to the risk of permanent data loss in the event of a system failure, accidental deletion, or ransomware attack.






**Risk:** Without an appropriate backup mechanism, any data loss event could lead to an irreversible loss of customer information, critical business data, and application status.

**Remediation:** Implement automated backups using AWS Backup or RDS automated snapshots, including multi-region backup strategies to ensure disaster recovery capabilities (S3 buckets, and enable block public access and encryption)

- Migrate MariaDB to RDS with automated backups.
- Schedule EBS volume backups.
- Create a S3 bucket to store those backups.
- Make multi-region replication of S3 bucket-based backup storage.
- Block public access to the S3 bucket and use AWS KMS for encryption at rest.

### **Reference:**

**vol-0f87b4bb676129b45**

Details	
<b>Volume ID</b>  <a href="#">vol-0f87b4bb676129b45</a>	<b>Size</b>  8 GiB
<b>AWS Compute Optimizer finding</b>  Opt-in to AWS Compute Optimizer for recommendations.   <a href="#">Learn more</a> 	<b>Volume state</b>  In-use
<b>Fast snapshot restored</b> No	<b>Availability Zone</b> us-east-1a
<b>Attached resources</b> <a href="#">i-Ofae36c979218135c (VulnerableInstance)</a> : /dev/xvda (attached)	<b>Outposts ARN</b> -
<b>▼ Source</b>	

No fast recovery settings, no multi-region backups.

## 2.6 No Log set ups for sensitive data

**Impact:** The lack of logging feature cannot track access to sensitive resources such as databases, EC2 file systems, and configuration files.

**Risk:** Unauthorized access attempts, privilege escalations, or data exfiltration activities are difficult to detect without an appropriate logging mechanism, severely impacting an organization's ability to conduct forensic investigations and comply with security audit requirements.

**Remediation:**

- Enable centralized logging using Amazon CloudWatch Logs.
- Install and configure the CloudWatch Agent on the EC2 instance.
- Define log metric filters and alarms to alert on suspicious activity.
- Use AWS KMS to encrypt and protect the logs.

**Reference:**

**IAM Role**

–

**IMDSv2**

Required

**Operator**

–

We did not find any log groups related to the EC2 instance in CloudWatch. We also found that the EC2 instance does not have any IAM role configurations, so this also proves that the feature is not enabled.

# Virtual Machine Vulnerability Assessment Report

## 3. 1 Finding One: MariaDB

The MariaDB instance does not have network-level restrictions, making it accessible from within the EC2 instance without additional safeguards. Additionally, it has a hardcoded password, which can easily be cracked for access.

### **Impact:**

- A compromised EC2 instance could result in database exposure, unauthorized queries, and potential data loss.
- Hardcoded credentials can easily be extracted from logs

### **Risk: Critical**

Unrestricted database access and a weak password increases the risk of SQL injection attacks and unauthorized exfiltration.

### **Remediation:**

- Deploy the database in a private subnet and restrict access to the EC2 instance only.
- Use security groups and network ACLs to enforce least-privilege access.
- Implement database encryption and regular access monitoring.
- Use AWS Secrets Manager or SSM Parameter Store instead of hardcoding the password.

## **Reference:**

The screenshot shows that MariaDB is installed on the same network, and has a hardcoded password

```
yum install -y nodejs mariadb105-server.x86_64 git 2>&1 | tee -a $LOG_FILE
```

```
# Start and enable MariaDB
```

```
echo "Starting and enabling MariaDB" | tee -a $LOG_FILE
```

```
systemctl start mariadb 2>&1 | tee -a $LOG_FILE
```

```
systemctl enable mariadb 2>&1 | tee -a $LOG_FILE
```

```
# Set up MariaDB (vulnerable root password)
```

```
echo "Setting up MariaDB" | tee -a $LOG_FILE
```

```
mysql -u root -e "SET PASSWORD FOR root@'localhost' = PASSWORD('vulnerable');" 2>&1 | tee -a $LOG_FILE
```

```
mysql -u root -p'vulnerable' -e "CREATE DATABASE juice_shop;" 2>&1 | tee -a $LOG_FILE
```

## **3. 2 Finding Two: Encryption**

The current YAML configuration does not include encryption mechanisms for data at rest or data in transit for the MariaDB database and the application.

### **Impact:**

- **Unencrypted Data at Rest:** If the EC2 instance or database files are compromised, attackers can access sensitive application data, including user credentials and transactional records.
- **Unencrypted Data in Transit:** Without TLS/SSL encryption, database queries, API calls, and HTTP traffic can be intercepted via Man-in-the-Middle (MitM) attacks, leading to data leakage.
- **Regulatory Non-Compliance:** Failing to encrypt sensitive data may violate GDPR, HIPAA, PCI-DSS, and SOC 2 compliance requirements.

### **Risk: High**

Lack of encryption significantly increases the risk of data exposure and unauthorized access, especially if the instance is compromised or traffic is intercepted.

## Remediation:

- Enable Amazon EBS encryption for the volume where MariaDB stores its data.
- Modify the YAML to encrypt the EC2 Instance's EBS Volume.
- Enable InnoDB encryption for database files.
- Enable TLS by generating SSL certificates and configuring MariaDB to use SSL by modifying the my.cnf, as well as modifying the user setup to require SSL.
- Redirect all HTTP traffic (Port 80) to HTTPS (Port 443) via modification of the Nginx configuration.

## Reference:

The "Encrypted: true" is missing inside of the BlockDeviceMappings section

```
# Create an EC2 instance in the Public Subnet
VulnerableEC2Instance:
  Type: "AWS::EC2::Instance"
  Properties:
    InstanceType: "t2.medium"
    KeyName: !Ref KeyPairName
    ImageId: !Ref AMIID
    NetworkInterfaces:
```

Lack of InnoDB and SSL/TLS Encryption

```
# Install required software (Node.js, NPM, and MariaDB)
echo "Installing Node.js and MariaDB" | tee -a $LOG_FILE
curl -sL https://rpm.nodesource.com/setup_18.x | bash - 2>&1 | tee -a $LOG_FILE
yum install -y nodejs mariadb105-server.x86_64 git 2>&1 | tee -a $LOG_FILE

# Start and enable MariaDB
echo "Starting and enabling MariaDB" | tee -a $LOG_FILE
systemctl start mariadb 2>&1 | tee -a $LOG_FILE
systemctl enable mariadb 2>&1 | tee -a $LOG_FILE
```

Application is exposed over HTTP (Port 80)

```
SecurityGroupIngress:
  - IpProtocol: "tcp"
    FromPort: "80"
    ToPort: "80"
    CidrIp: "0.0.0.0/0" # Allow HTTP access
```

### **3.3 Finding Three: Express-jwt**

The virtual machine is running express-jwt v0.1.3, which inherits an authentication bypass and other vulnerabilities from its dependencies. This allows an attacker to gain access to the application via circumventing authentication.

#### **Impact:**

The vulnerable version of express-jwt can potentially allow unauthorized access to your application, compromising sensitive data and operations. This could lead to data breaches, unauthorized data manipulation, and act as a launch pad for further attacks.

#### **Risk: Critical**

Authentication bypass is a serious security risk, and should be patched ASAP, as it can lead the attacker to further attacks which can escalate via privilege escalation.

#### **Remediation:**

Update express-jwt to the latest version

#### **Reference:**

The screenshot shows the dependencies from the developers package.json.bak that was hidden behind /fpt/

```
},  
"dependencies": {  
  "body-parser": "~1.18",  
  "colors": "~1.1",  
  "config": "~1.28",  
  "cookie-parser": "~1.4",  
  "cors": "~2.8",  
  "dottie": "~2.0",  
  "epilogue-js": "~0.7",  
  "errorhandler": "~1.5",  
  "express": "~4.16",  
  "express-jwt": "0.1.3",  
  "fs-extra": "~4.0",  
  "glob": "~5.0",  
  "grunt": "~1.0",  
  "grunt-angular-templates": "~1.1",  
  "grunt-contrib-clean": "~1.1",  
  "grunt-contrib-compress": "~1.4",  
  "grunt-contrib-concat": "~1.0",  
  "grunt-contrib-uglify": "~3.2",  
  "hashids": "~1.1",  
  "helmet": "~3.9",  
}
```

[NVD - CVE-2020-15084](#)



### 3.4 Finding Four: Cross-Spawn

The virtual machine is running an older version (7.0.3) of cross-spawn, which is vulnerable to Regular Expression Denial of Service (ReDoS) due to improper input sanitization. This attack can crash the program via the usage of large crafted strings as input.

**Impact:**

The vulnerable version of cross-spawn enables the instance to be attacked via ReDoS. This enables an attacker to craft a large string in order to slow down the engine, causing service slowdowns or denials as the engine becomes overwhelmed.

**Risk: High**

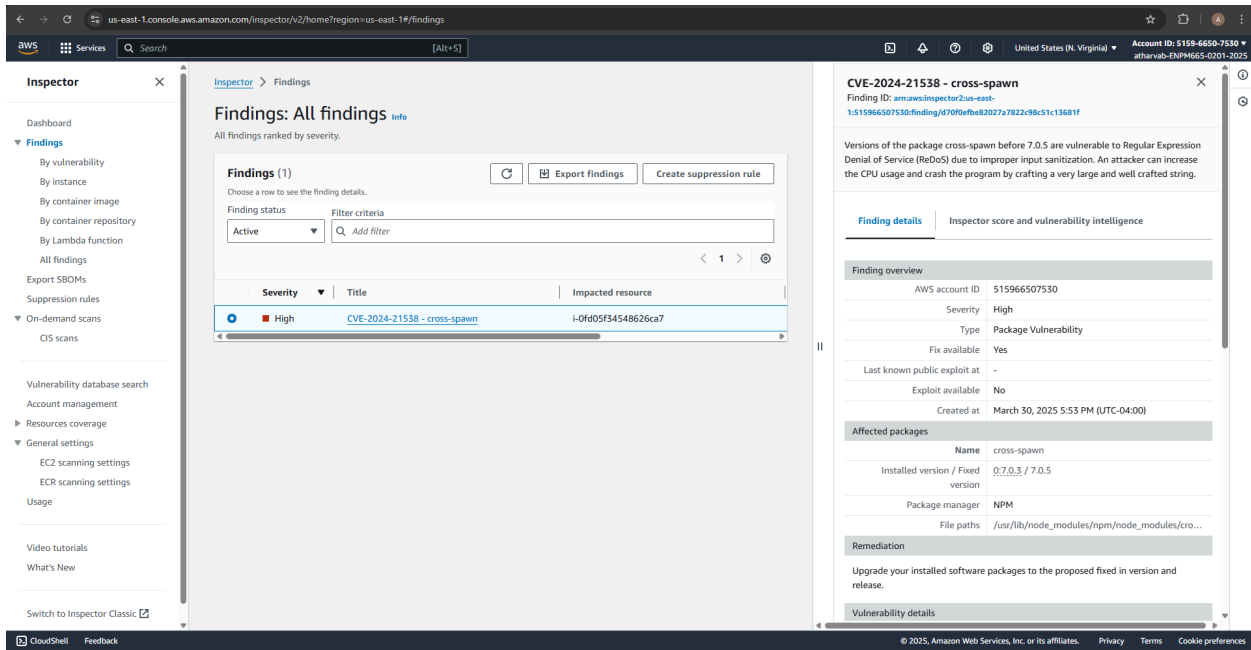
ReDoS can slow down the entire instance to a crawl, or even shut it down if it consumes enough resources.

**Remediation:**

Upgrade cross-spawn to a minimum version of 7.0.5, which fixes the input sanitization issue.

**Reference:**

Amazon inspector shows the package vulnerability via scanning the ec2 instance



CVE Severity and Score

CVE-2024-21538 - cross-spawn



Finding ID: [arn:aws:inspector2:us-east-1:515966507530:finding/d70f0efbe82027a7822c98c51c13681f](#)

Versions of the package cross-spawn before 7.0.5 are vulnerable to Regular Expression Denial of Service (ReDoS) due to improper input sanitization. An attacker can increase the CPU usage and crash the program by crafting a very large and well crafted string.

Finding details

Inspector score and vulnerability intelligence

CVSS v3 (NVD)	Inspector
7.5	7.5

The Inspector score is **the same**. Metrics have not been changed.

CVSS score metrics

Metric	CVSS	Inspector
Attack Vector	Network	Network
Attack Complexity	Low	Low
Privileges Required	None	None
User Interaction	None	None
Scope	Unchanged	Unchanged
Confidentiality	None	None
Integrity	None	None
Availability	High	High

## 3.5 Finding Five: MarsDB

The virtual machine is running MarsDB as a client sided database, however MarsDB is vulnerable to Arbitrary Code Injection due to passing \$where clauses to the function constructor unsanitized.

### **Impact:**

Marsdb is vulnerable to Arbitrary Code Execution, which allows an attacker to execute any command or code of their choosing on the system. This allows the attacker free control over the entire system, and is very dangerous.

### **Risk: Critical**

Arbitrary Code Execution can be used to execute any command of the attackers choosing, as long as it is passed through the \$where clause.

### **Remediation:**

Since this CVE has yet to be resolved, it is advised to stop using Marsdb and switch to another alternative as a whole.

### **Reference:**

Screenshot from the trivy scan

marsdb (package.json)	GHSA-5mrr-rgp6-x4gr	CRITICAL	0.6.11	Command Injection in marsdb	https://github.com/advisories/GHSA-5mrr-rgp6-x4gr
-----------------------	---------------------	----------	--------	-----------------------------	---

<https://security.snyk.io/vuln/SNYK-JS-MARSDB-480405>

<https://github.com/advisories/GHSA-5mrr-rgp6-x4gr>

# 3.6 Finding Six: VM2

The virtual machine is running Vm2, a library that is used to provide a sandbox for executing JavaScript code. The version that is running contains several vulnerabilities, including sandbox escape.

## Impact:

- Sandbox Escape: Could lead to the execution of untrusted code outside the sandbox, compromising the host environment.
- Remote Code Execution (RCE): If an attacker exploits the vulnerability, they might execute arbitrary code in the context of the host application.

## Risk: Critical

The library contains several CVE's, ranging from Sandbox Escape, to Remote Code Execution. This allows the attacker to break out of the restricted environment to attack the host system directly, as well as execute code remotely over there.

## Remediation:

Updating vm2 to 3.9.18 will resolve the most egregious of the vulnerabilities, however there are still some that are not fixed yet. Hence it would be advisable to look for other alternatives till they are fixed.

## Reference:

Screenshot from the trivy scan

vm2 (package.json)	CVE-2023-32314	CRITICAL	3.9.17	3.9.18	vm2: Sandbox Escape https://avd.aquasec.com/nvd/cve-2023-32314
	CVE-2023-37466	affected			vm2: Promise handler sanitization can be bypassed allowing attackers to escape the... https://avd.aquasec.com/nvd/cve-2023-37466
	CVE-2023-37903				vm2: custom inspect function allows attackers to escape the sandbox and run... https://avd.aquasec.com/nvd/cve-2023-37903
	CVE-2023-32313	MEDIUM	fixed	3.9.18	vm2: Inspect Manipulation https://avd.aquasec.com/nvd/cve-2023-32313

[NVD - CVE-2023-32314](#)

[NVD - CVE-2023-37466](#)

[NVD - CVE-2023-37903](#)

# Network Security Assessment Report

## 4.1 Unused Private Subnet (Lack of Network Segmentation)

**Impact:** Network segmentation is missing. The RDS resources are not deployed in a private subnet. Thus, critical resources such as the database and internal components are unnecessarily exposed on the public subnet.

**Risk:** Failing to strictly segment the network increases the attack surface significantly and the risk of unauthorized access to sensitive resources. Attackers can directly access the EC2 instance, and once the EC2 instance is compromised, all information will be retrieved by the attacker.

### Remediation:

- Deploy critical backend resources (e.g., database servers) within the private subnet.
- Configure security groups and routing rules to enforce isolation between public and private subnets.
- The communication between the EC2 instance and the RDS instance that accesses the private network must be encrypted.

### Reference:

```
# Create a Private Subnet
PrivateSubnet:
  Type: "AWS::EC2::Subnet"
  Properties:
    VpcId: !Ref MidtermVPC
    CidrBlock: !Ref PrivateSubnetCidr
    AvailabilityZone: !Select [ 0, !GetAZs "" ]
    Tags:
      - Key: Name
        Value: "Private Subnet"
```

Route tables (2) [Info](#)

Find resources by attribute or tag

<input type="checkbox"/>	Name	Route table ID	Explicit subnet associ...	Edge associations	Main	VPC	Owner ID
<input type="checkbox"/>	-	<a href="#">rtb-0239107909292cba6</a>	-	-	Yes	<a href="#">vpc-001ddb0c8a6c9dd59</a>   <a href="#">ENP...</a>	867344441398
<input type="checkbox"/>	Public Route Table	<a href="#">rtb-07d7e5db9278e3e98</a>	<a href="#">subnet-0e4888ffe694bd8...</a>	-	No	<a href="#">vpc-001ddb0c8a6c9dd59</a>   <a href="#">ENP...</a>	867344441398

As we can see from the figure, we have created a private subnet. In addition, we have not configured anything for the private subnet, not even the route table.

## 4.2 Lack of Network Access Control Lists (NACLs)

**Impact:** Network access control lists (NACLs) are currently deployed using the default, which allows all inbound and outbound traffic. This configuration does not allow for any subnet-level traffic filtering, exposing the environment to potential unauthorized access or lateral movement within the VPC.

**Risk:** Without a properly configured NACL, malicious or unauthorized traffic can traverse subnets freely. If an EC2 instance in a public subnet is compromised, an attacker may be able to scan for or access other internal subnets, especially if the security groups are overly permissive. This increases the likelihood of lateral movement, privilege escalation, and access to sensitive resources such as databases.

### **Remediation:**

- Avoid using the default NACL, which allows all kinds of traffic to flow freely in and out.
- Define well-structured NACLs with explicit allow/deny rules for both inbound and outbound traffic.
- Block unnecessary ports and restrict IP ranges at the subnet level.
- If necessary, configure dedicated NACLs for each subnet.
- Combine NACLs with security groups and VPC Flow Logs to enhance visibility and implement layered network protection.

### **Reference:**


acl-0542a9b7fd20632f7

Actions


Details

Info

Network ACL ID

 acl-0542a9b7fd20632f7

Owner

 867344441398

Associated with

[2 Subnets](#)

Default

Yes

VPC ID

[vpc-001ddb0c8a6c9dd59 / ENPM665-midterm-vpc1](#)

Inbound rules

Outbound rules

Subnet associations

Tags

Inbound rules (2)

Edit inbound rules

Filter inbound rules

< 1 > ⚙

Rule number	Type	Protocol	Port range	Source	Allow/Deny
100	All traffic	All	All	0.0.0.0/0	Allow
*	All traffic	All	All	0.0.0.0/0	Deny

## 4.3 Overly Permissive Security Group Rules

**Impact:** The security group settings allow unrestricted inbound access to ports (e.g. SSH (22), HTTP (80) and HTTPS (443)) without IP restrictions (0.0.0.0/0). This means that anyone on the internet can attempt to connect to these ports, increasing the likelihood of brute force attacks, unauthorized access and exploitation of application vulnerabilities.

**Risk:** Overly permissive security group rules can significantly increase system exposure, increasing the risk of network attacks, which may lead to data breaches and system downtime. Attackers can scan the open ports of the EC2 instance and launch targeted attacks including brute force attacks and vulnerability exploits. If there are no access restrictions at the security group level, we are essentially actively exposing the system to the internet and waiting to be attacked.

### Remediation:

- Restrict SSH access to a trusted IP address range.
- Block HTTP access and ensure that all network traffic is encrypted using HTTPS only.
- Review and modify all security group rules to allow only necessary traffic.

### Reference:

sg-05875bfc64eec47c3 - midterm-WebServerSecurityGroup-tjzw9kb9mUDx Actions

**Details**  
**Security group name**  
midterm-WebServerSecurityGroup-tjzw9kb9mUDx  
**Owner**  
867344441398

**Security group ID**  
sg-05875bfc64eec47c3  
**Inbound rules count**  
4 Permission entries

**Description**  
Allow SSH, HTTP, and HTTPS access to the EC2 instance  
**Outbound rules count**  
1 Permission entry

**VPC ID**  
vpc-001ddb0c8a6c9dd59

[Inbound rules](#) | [Outbound rules](#) | [Sharing - new](#) | [VPC associations - new](#) | [Tags](#)

**Inbound rules (4)** Manage tags Edit inbound rules

	Name	Security group rule ID	IP version	Type	Protocol	Port range	Source	Description
<input type="checkbox"/>	-	sgr-06663a3f5d34f13c7	IPv4	Custom TCP	TCP	3000	0.0.0.0/0	-
<input type="checkbox"/>	-	sgr-0e1c2ebe205b7f129	IPv4	SSH	TCP	22	0.0.0.0/0	-
<input type="checkbox"/>	-	sgr-09b7ccba110c77a4	IPv4	HTTPS	TCP	443	0.0.0.0/0	-
<input type="checkbox"/>	-	sgr-01115fd231facae80	IPv4	HTTP	TCP	80	0.0.0.0/0	-

## 4.4 Lack of Network Traffic Monitoring and Threat Detection

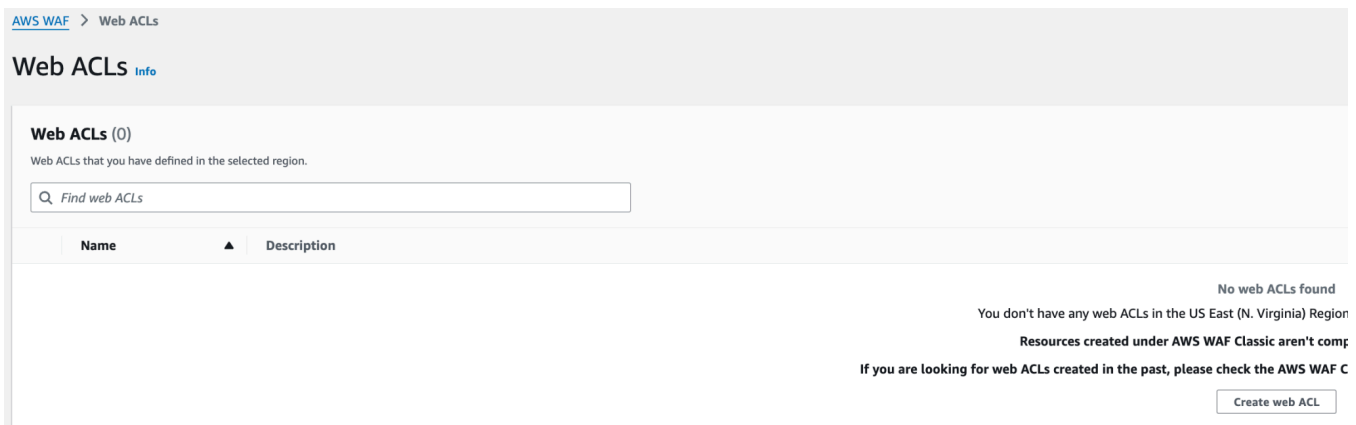
**Impact:** Without network traffic monitoring, we cannot detect network attacks in real time or respond to potential threats. This makes EC2 instances vulnerable to probing, brute-force attacks, and SQL injection attacks, and DevOps staff barely have the ability to respond to unexpected attacks and trace back attack paths.

**Risk:** Unmonitored network activity increases the risk of persistent attacks, delayed detection of violations, and intrusion into sensitive systems. In other words, it is like entering an airport without security checks.

### **Remediation:**

- Deploy AWS Web Application Firewall (WAF) to detect and block common web-based attacks, including SQL injection and cross-site scripting (XSS).
- Enable AWS GuardDuty and CloudWatch to monitor, detect, and alert on suspicious network activities.

### **Reference:**



We have not configured any type of firewall. Based on our service scenarios, we recommend configuring a WAF.



## 4.5 Insecure Web Traffic Due to Missing HTTPS (TLS) Enforcement

**Impact:** The web application on the EC2 instance currently accepts HTTP and HTTPS requests, but does not enforce the use of HTTPS. This allows sensitive data such as login credentials and personal information to be transmitted in plain text, where it can be intercepted by attackers.

**Risk:** Unencrypted communication is vulnerable to man-in-the-middle attacks (MITM), which can lead to the disclosure of user data and session hijacking.

### Remediation:

- Enforce HTTPS for all web traffic by deploying an Application Load Balancer (ALB) configured for SSL/TLS termination.
- Use AWS Certificate Manager to automatically configure, manage, and update SSL/TLS certificates.
- Force HTTPS requests and close the HTTP protocol port.

### Reference:

Instances (1/1) [Info](#)

All states ▾

<input checked="" type="checkbox"/>	Name <a href="#">🔗</a>	Instance ID	Instance state ▾	Instance type ▾	Status check	Alarm status	Availability Zone ▾	Public IPv4 DNS	Public IPv4 ... ▾	Elastic I
<input checked="" type="checkbox"/>	VulnerableInst...	i-Ofae36c979218135c	Running <a href="#">🔍</a>	t2.medium	2/2 checks passsec <a href="#">View alarms +</a>		us-east-1a	ec2-34-201-126-143.co...	34.201.126.143	-

---

**i-Ofae36c979218135c (VulnerableInstance)**

[Details](#) | [Status and alarms](#) | [Monitoring](#) | [Security](#) | [Networking](#) | [Storage](#) | [Tags](#)

► Security details

▼ Inbound rules

Name	Security group rule ID	Port range	Protocol	Source	Security groups	Description
-	sgr-06663a3f5d34f13c7	3000	TCP	0.0.0.0/0	<a href="#">midterm-WebServerSecurityGroup-tj...</a>	-
-	sgr-0e1c2ebe205b7f129	22	TCP	0.0.0.0/0	<a href="#">midterm-WebServerSecurityGroup-tj...</a>	-
-	sgr-09b7ccbf110c77a4	443	TCP	0.0.0.0/0	<a href="#">midterm-WebServerSecurityGroup-tj...</a>	-
-	sgr-01115fd231facae80	80	TCP	0.0.0.0/0	<a href="#">midterm-WebServerSecurityGroup-tj...</a>	-

▼ Outbound rules

Name	Security group rule ID	Port range	Protocol	Destination	Security groups	Description
-	sgr-05c4d372cee5b0f9c	All	All	0.0.0.0/0	<a href="#">midterm-WebServerSecurityGroup-tj...</a>	-

## 4.6 Lack of Application Load Balancer (ALB) Protection

**Impact:** Since the application load balancer is not deployed, all external requests access the EC2 instance directly, which causes it to be completely exposed to the Internet. Because of the lack of ALB as an intermediate protection layer, EC2 instances are more vulnerable to direct access and attacks.

**Risk:** Attackers can bypass the load balancing and attack the EC2 instance directly without going through the ALB as the traffic entry point. It's like leaving the door unlocked, and anyone can come in and try. Even if they don't succeed, it shows that you don't have a first line of defense at all.

### **Remediation:**

- Deploy an Application Load Balancer (ALB) in the public subnet to handle all inbound traffic and forward valid requests to the EC2 instances.
- Ensure that the EC2 instances are deployed in the private subnet and only accessible by the ALB.

### **Reference:**

**Load balancers**  
Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.

🔍

■	Name	▼	DNS name	▼	State	▼	VPC ID	▼	Availability Zones	▼	Type	▼	Date created
---	------	---	----------	---	-------	---	--------	---	--------------------	---	------	---	--------------

No load balancers  
You don't have any load balancers in us-east-1

[Create load balancer](#)

There is no configuration for load balancing in the existing configuration template.

## 4.7 Lack of Auto Scaling for EC2 Instances

**Impact:** A single EC2 instance is provisioned in the template, but no additional auto scaling group (ASG) is configured, which limits the ability to automatically recover from instance failures or handle increased traffic as the application scales. lacks a protective middle layer, making the server more vulnerable to direct access and attacks.

**Risk:** Under high traffic conditions, the performance of EC2 instances may decline or they may go down. If an EC2 instance fails, there is no backup to maintain availability. Without scalability, the system is vulnerable to overload, performance degradation, and denial of service (DoS).

**Remediation:**

- Configure an Auto Scaling group to manage EC2 instances based on traffic load, CPU utilization, or request count.
- Define the minimum, maximum, and desired number of instances to ensure system resilience and efficiency.
- Combine Autoscaling with CloudWatch alarms to trigger scaling actions.

**Reference:**

**AWS Compute Optimizer finding**

[i Opt-in to AWS Compute Optimizer for recommendations. | Learn more](#) 

**Auto Scaling Group name**

—

**Managed**

false

We can verify that the running EC2 instance does not have an autoscaling group configured.

## 4.8 Excessive IAM Privileges and Insecure Default Account

**Impact:** IAM configurations assigned overly broad permissions to the developer role, effectively granting them unrestricted access to all AWS services. In addition, the default `ec2-user` account on the EC2 instance was not removed or restricted.

### **Risk:**

- If a developer account is compromised (ex. phishing), an attacker can gain full administrative control of the entire AWS environment.
- Shared or overly privileged roles make it difficult to trace responsibility and increase the likelihood of accidental deletion, misconfiguration, or data breaches.
- Attackers will use the default `ec2-user` account to gain access to the instance

### **Remediation:**

- Apply the principle of least privilege by reviewing and customizing IAM roles based on job responsibilities.
- Developers should only have access to the specific services and actions they need.
- Delete the `ec2-user` account on the EC2 instance.
- Enable CloudTrail to audit all IAM-related activities to ensure accountability and traceability.

### **Reference:**

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    }
  ]
}
```



```
Midterm — ec2-user@ip-10-0-1-247:/opt/juice-shop/data — ssh -i test1.pem ec2-user@3.94.125.208 — 110x...
/opt/juice-shop/data
[ec2-user@ip-10-0-1-247 data]$ ls
```

# Disaster Recovery Assessment Report

## 5.1 Lack of Automated Backups & Secure Storage

### Impact

Without automated backups—that is, planned EBS snapshots or AWS Backup—lost vital data is a very real threat. Should something go wrong—say, a server crash or an inadvertent deletion—this configuration makes recovery far more difficult—or sometimes impossible. Store backups without appropriate encryption or access rules also opens a door for illegal access.

### Risk

One slip in design or an unanticipated failure can permanently wipe out crucial data, so endangering daily operations. Unencrypted and freely available storage can also draw attackers in.

### Remediation

1. **Regular Snapshots:** Activate AWS Backup or set up scheduled EBS snapshots so you always have an up-to-date copy of your data.
2. **Stricter Security:** Make sure backups are encrypted at rest (using KMS) and review who can access them.
3. **Validation & Drills:** Check your backups every so often to confirm they're usable. Run practice drills to see if your recovery steps actually work in a crisis.

### Reference

- AWS Backup Documentation
- EBS Snapshot Automation (AWS Data Lifecycle Manager)
- AWS Key Management Service (KMS) Guides
- Amazon S3 Security Best Practices

## 5.2 Single-Point-of-Failure Architecture

### Impact

If you're running all services on one EC2 instance in a single Availability Zone, you're flirting with a serious outage risk. A hardware malfunction, zone-wide downtime, or even a local crisis could knock out your entire system.

### Risk

An extended outage isn't just a nuisance—it can translate into frustrated customers, lost revenue, and a damaged reputation. With everything dependent on one machine or zone, there's no fallback.

### Remediation

1. **Diversify Your Deployments:** Use multiple AZs or replicate data to another region.
2. **Scale & Balance:** Implement an Auto Scaling group plus a load balancer so traffic and workloads shift seamlessly if one instance dies.
3. **Multi-AZ Databases:** For crucial databases, consider RDS Multi-AZ configurations to avoid single-instance vulnerabilities.

### Reference

- [AWS Auto Scaling Documentation](#)
- [Elastic Load Balancing Guides](#)
- [Amazon RDS Multi-AZ Instructions](#)

## 5.3 Absent Disaster Recovery Plan

### Impact

Without a proper disaster recovery (DR) strategy in place if something catastrophic occurs—whether that’s a large-scale outage or a major security incident—teams may not know where to turn to in order to respond. That disruption can greatly delay system restoration and increase the chances of irrevocable data loss.

### Risk

With no clear path, downtime could linger indefinitely and, in the worst-case scenario, essential services or records could be lost for good. It puts the business at risk of losing customer trust, and can see revenues nosedive the longer it takes

### Remediation

1. **Build a Real DR Playbook:** Spell out recovery steps, define roles, and rehearse who does what during a disaster scenario.
2. **Frequent Testing:** Conduct mock “fire drills” to confirm whether everyone understands their responsibilities and that recovery works as designed.
3. **Leverage Automation:** Consider Infrastructure-as-Code (like AWS CloudFormation) and well-crafted scripts to relaunch critical resources without guesswork.

### Reference

- AWS Well-Architected Framework
- CIS AWS Foundations Benchmark
- NIST SP 800-34: Contingency Planning Guide

## 5.4 Missing Incident Response & Recovery Documentation

### Impact

When things go haywire—a security breach or system crash—teams must have a straightforward map to respond quickly. Without formal incident response runbooks or monitoring alerts (via CloudWatch or SNS), issues can go unauthedesk (or worse, procreate) before anyone can respond.

### Risk

Small errors can turn into extended downtime if there's disagreement about who does what. Long-term failures inevitably involve time and money, and possibly the loss of customer loyalty.

### Remediation

1. **Set Up Monitoring:** Use CloudWatch to keep tabs on CPU usage, disk space, or custom app metrics. Send alerts through SNS (or a similar service).
2. **Incident Runbooks:** Develop step-by-step instructions that walk responders through how to isolate, contain, and fix issues.
3. **On-Call Training:** Make sure team members know the runbooks inside and out, and confirm that escalations happen swiftly.

### Reference

- AWS CloudWatch Documentation
- AWS Incident Response Playbook
- Operational Excellence Pillar (AWS Well-Architected Framework)



## 5.5 Absent Auto Scaling & Failover Mechanisms

### Impact

A standalone system with no safeguards in place with Auto Scaling or a load balancer can quickly become a bottleneck on an uptick of traffic or simply fail when things go south. When this occurs, user experience plummets and your overall availability suffers.

### Risk

But when under high load—or if a bad actor DDoS attacks—you're left with only one instance that will fail. (If your system isn't able to scale up on demand, or you can't traffic elsewhere, you're out for however long it takes for you to manually fix.)

### Remediation

1. **Auto Scaling Group:** Establish minimum and maximum capacity so new instances can spin up when traffic spikes, and unhealthy ones can be replaced automatically.
2. **Load Balancer:** Distribute traffic evenly across multiple instances, preventing any single point from getting slammed.
3. **Performance Metrics:** Use CloudWatch to observe resource usage and set thresholds that trigger scaling activities.

### Reference

- AWS Auto Scaling Documentation
- Elastic Load Balancing Overviews
- AWS DDoS Resiliency Best Practices