SecureThread
OPS

Securing the Digital Future

SECURITY ASSESSMENT REPORT

vulnerable-node

Repository:	`dummyhshz/vulnerable-node`
Branch/Commit:	`master / HEAD`
Scan Date:	21 December 2025 15:57
Stats:	23 files / 1,840 LoC
Report ID:	`ST-179-20251230-1730`

SECURITY POSTURE SUMMARY

Score	Grade	Findings	Critical
0	F	20	4

4 Critical | 13 High | 2 Medium | 1 Low

Report Navigation Guide

This report is organized into the following main sections:

Section 1:	**Executive Summary** - High-level overview for management and stakeholders
Section 2:	**Compliance Mapping** - Regulatory and framework alignment (OWASP, PCI DSS, GDPR)
Section 3:	**Detailed Findings** - Technical vulnerability details for development teams
Section 4:	**Recommendations** - Prioritized remediation roadmap and best practices
Appendices:	**Reference Materials** - Methodology, glossary, CWE reference, and resources

Quick Start:

- Executives: Read Section 1 (Executive Summary)
- Developers: Focus on Section 3 (Detailed Findings) and Section 4 (Recommendations)
- Compliance: Review Section 2 (Compliance Mapping)
- Security Team: Review all sections

Table of Contents

List of Figures & Visualizations

Severity Rating Legend

This report uses a standardized severity rating system based on CVSS (Common Vulnerability Scoring System) and impact assessment. Use this legend to quickly understand the urgency of each finding.

Icon	Severity	Description	Action Timeline
C	**CRITICAL** CVSS: 9.0-10.0	Immediate threat of system compromise, data breach, or complete service disruption. Exploitable remotely without authentication.	**0-72 hours** Immediate action required
H	**HIGH** CVSS: 7.0-8.9	Significant security risk allowing unauthorized access, data exposure, or privilege escalation. May require some user interaction.	**7-14 days** Urgent attention needed
M	**MEDIUM** CVSS: 4.0-6.9	Moderate security weakness that could lead to information disclosure or limited access. Typically requires specific conditions to exploit.	**30-60 days** Address in next sprint
L	**LOW** CVSS: 0.1-3.9	Minor security concern with minimal impact. Difficult to exploit or requires extensive preconditions.	**60-90 days** Maintenance priority
I	**INFO** CVSS: 0.0	Informational finding or security best practice recommendation without direct exploitability.	**Ongoing** Best practice improvement

CVSS Score Calculation

CVSS (Common Vulnerability Scoring System) scores are calculated based on:

Factor	Description
Attack Vector	How the vulnerability can be exploited (Network, Adjacent, Local, Physical)
Attack Complexity	Conditions beyond attacker's control (Low, High)
Privileges Required	Level of access needed (None, Low, High)
User Interaction	Whether user action is required (None, Required)
Confidentiality Impact	Impact on data confidentiality (None, Low, High)
Integrity Impact	Impact on data integrity (None, Low, High)
Availability Impact	Impact on system availability (None, Low, High)

Note: For complete CVSS documentation, visit: https://www.first.org/cvss/

Common Abbreviations & Acronyms

Quick reference for technical terms used throughout this report:

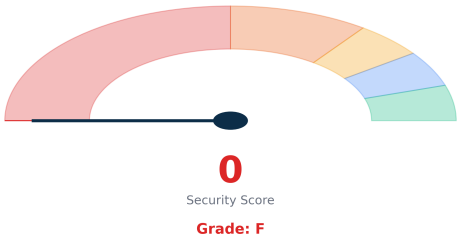
API	Application Programming Interface
CORS	Cross-Origin Resource Sharing
CSRF	Cross-Site Request Forgery
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
DAST	Dynamic Application Security Testing
DoS	Denial of Service
GDPR	General Data Protection Regulation
HIPAA	Health Insurance Portability and Accountability Act
HTTPS	Hypertext Transfer Protocol Secure
IAM	Identity and Access Management
IaC	Infrastructure as Code
JSON	JavaScript Object Notation
JWT	JSON Web Token
LOC	Lines of Code
MFA	Multi-Factor Authentication
NIST	National Institute of Standards and Technology
OWASP	Open Web Application Security Project
PCI DSS	Payment Card Industry Data Security Standard
PII	Personally Identifiable Information
REST	Representational State Transfer
RCE	Remote Code Execution
SANS	SysAdmin, Audit, Network, Security
SAST	Static Application Security Testing
SCA	Software Composition Analysis
SDLC	Software Development Lifecycle
SOC	Security Operations Center
SQL	Structured Query Language
SSRF	Server-Side Request Forgery
SSL/TLS	Secure Sockets Layer / Transport Layer Security
XSS	Cross-Site Scripting
XXE	XML External Entity

Need More Detail? A comprehensive glossary is available in Appendix D (page **??**).

Executive Summary

Assessment Overview

SecureThread OPS has completed a comprehensive automated security assessment of the **vulnerable-node** repository using Static Application Security Testing (SAST) techniques combined with AI-powered analysis.



Key Statistics:

Scan Date:	21 December 2025 15:57
Branch/Commit:	`master / HEAD`
Files Analyzed:	23
Lines of Code:	1,840
Total Findings:	20
Vulnerability Density:	10.87 per 1k LOC
Auto-Fixable:	10 (50%)

Severity Distribution

The following table summarizes the distribution of vulnerabilities by severity level:

Severity	Count	%	Risk Assessment
CRITICAL	4	20.0%	Immediate remediation required - system compromise risk
HIGH	13	65.0%	Urgent attention needed - significant security risk
MEDIUM	2	10.0%	Address in next sprint - moderate risk
LOW	1	5.0%	Address during maintenance - minimal risk

Business Impact Analysis

This section quantifies the financial and operational impact of identified vulnerabilities:

Metric	Value	Impact Level
Remediation Cost Developer hours to fix	**$27,000.00**	**HIGH**
Potential Breach Cost If vulnerabilities exploited	**$401,800.00**	**CRITICAL**
Total Estimated Effort Time to complete remediation	**133 hours (16.6 days)**	**SIGNIFICANT**
Total Financial Risk Combined cost exposure	**$428,800.00**	**SEVERE**

Executive Action Required

The identified vulnerabilities represent a **significant financial risk** exceeding **$428,800.00**. Immediate executive attention and resource allocation is recommended to address critical security gaps and prevent potential data breaches that could result in regulatory fines, legal liability, and reputational damage.

Security Analysis Visualizations

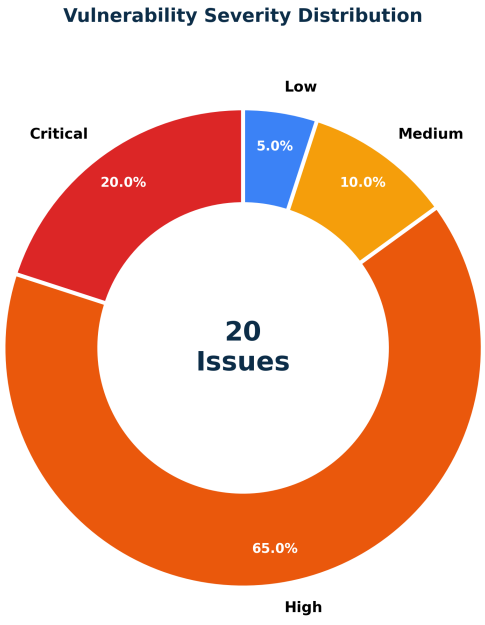


Figure 1: *Vulnerability Severity Distribution*

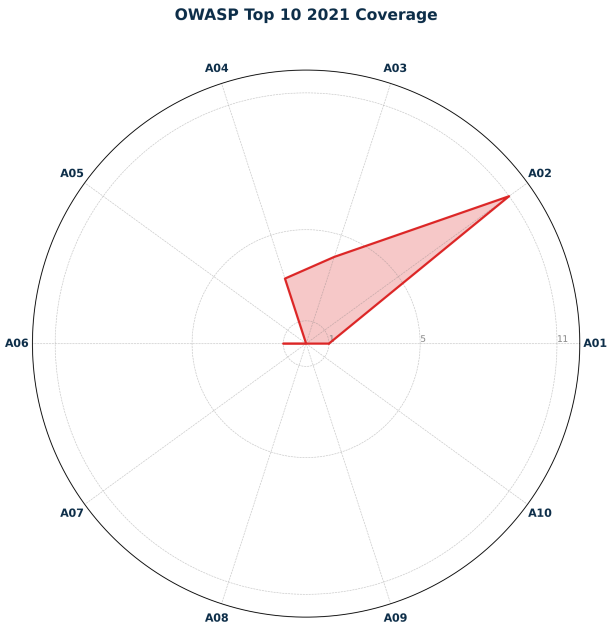


Figure 2: *OWASP Top 10 2021 Coverage*

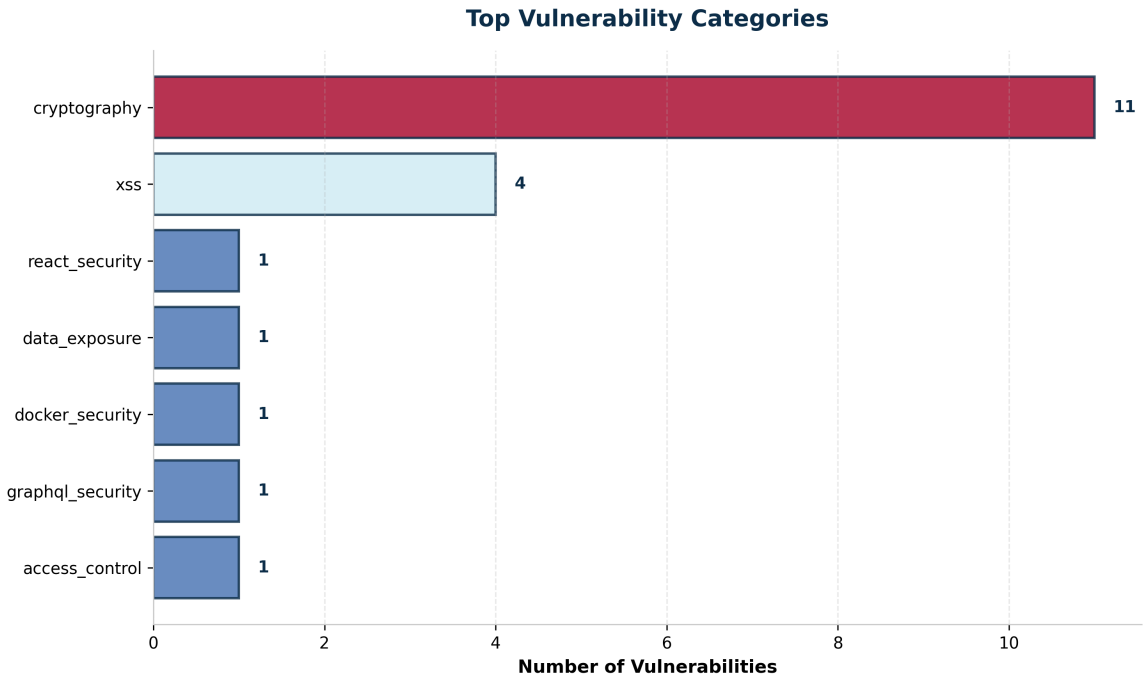


Figure 3: *Top Vulnerability Categories Detected*

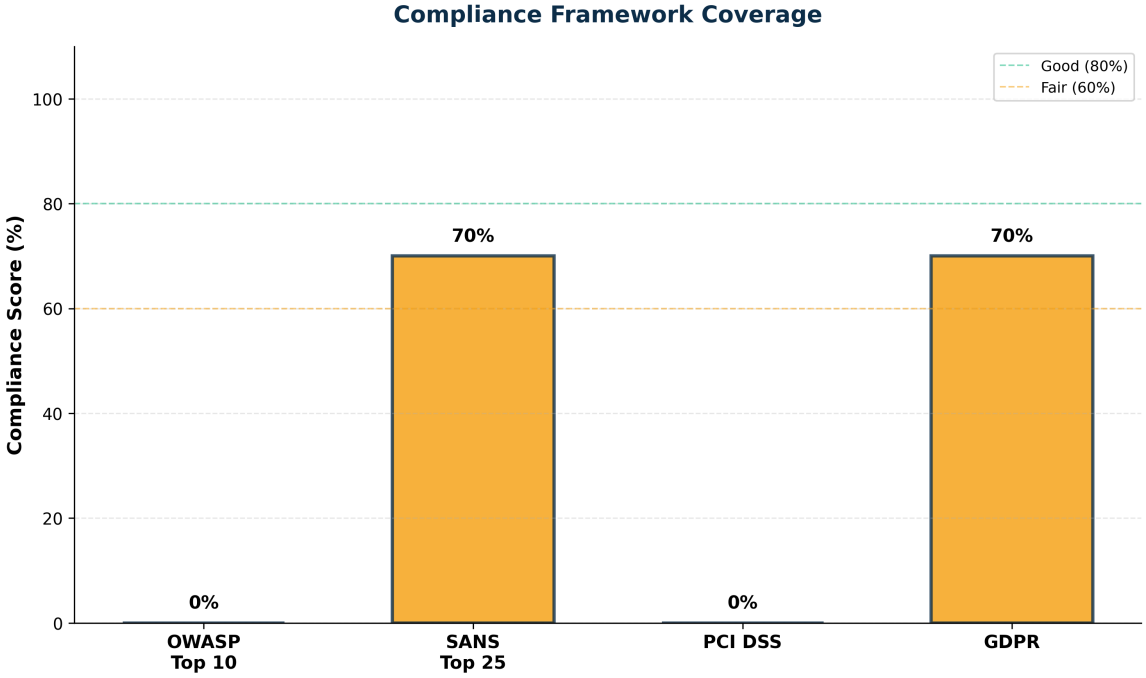


Figure 4: *Compliance Framework Coverage Assessment*

Critical & High Priority Findings

The following table highlights the most critical security findings requiring immediate attention:

ID	Severity	Title	Location	CVSS
#4	HIGH	Crypto - Insecure Random Number Genera...	dummy.js:20	7.5
#5	HIGH	Crypto - Insecure Random Number Genera...	dummy.js:26	7.5
#6	HIGH	Crypto - Insecure Random Number Genera...	public/js/bootstrap.js:1668	7.5
#7	HIGH	Crypto - Insecure Random Number Genera...	public/js/jquery.js:2	7.5
#8	HIGH	Crypto - Insecure Random Number Genera...	public/js/jquery.js:2	7.5
#9	HIGH	XSS - React createElement with User Pr...	public/js/jquery.js:3	7.5
#10	HIGH	GraphQL - Sensitive Data in Error Mess...	public/js/jquery.js:2	7.5
#11	HIGH	Vertical Access Control - Test Account...	dummy.js:7	7.5
#12	HIGH	Crypto - Insecure Random Number Genera...	dummy.js:50	7.5
#13	HIGH	Crypto - Insecure Random Number Genera...	dummy.js:56	7.5

Note: 7 additional critical/high severity findings are documented in the Detailed Findings section (Section 4).

Key Recommendations

Based on this assessment, we recommend the following prioritized actions:

1. **#1: Immediate Critical Remediation**
   Address all 4 critical vulnerabilities within 7 days to prevent system compromise
2. **#2: High-Severity Sprint Planning**
   Allocate development resources to remediate 13 high-severity issues across 2-3 sprints
3. **#3: Compliance Alignment**
   Address 12 compliance violations (OWASP, PCI DSS, GDPR) to meet regulatory requirements
4. **#4: Code Quality Improvement**
   Vulnerability density (10.87/1k LOC) is elevated. Implement secure coding training and SAST in CI/CD

5. **#5: Leverage Auto-Fix Suggestions**
   10 vulnerabilities have automated fix suggestions - implement these quick wins first

Detailed remediation guidance, sprint planning, and secure coding best practices are provided in Section 5: Strategic Recommendations & Remediation Roadmap.

Industry Benchmark Comparison

Security Posture Rating: Your security score of **0.0/100** is **below industry standards and requires significant improvement**.

Your Score:	0.0/100 (Grade: F)
Industry Average:	72/100 (Grade: C+)
Your Density:	10.87 vulnerabilities per 1k LOC
Industry Average:	3.5 vulnerabilities per 1k LOC

0.1 Compliance \& Regulatory Mapping

This section maps identified vulnerabilities to industry-standard security frameworks and regulatory requirements, helping organizations understand their compliance posture.

0.1.1 OWASP Top 10 2021 Coverage

The **OWASP Top 10** represents the most critical security risks to web applications. This assessment identifies which OWASP categories are present in your codebase.

Category	Description	Findings	Risk
A01	Failures related to enforcing access policies	1	Low
A02	Weak or missing encryption, exposed sensitive data	11	Critical
A03	SQL, NoSQL, OS command injection attacks	4	Medium
A04	Missing or ineffective security design patterns	3	Medium
A05	Insecure default configurations or settings	0	None
A06	Use of outdated or vulnerable libraries	1	Low
A07	Authentication and session management flaws	0	None
A08	Insecure CI/CD, unsigned code or data	0	None
A09	Insufficient logging and monitoring	0	None
A10	Server-Side Request Forgery vulnerabilities	0	None

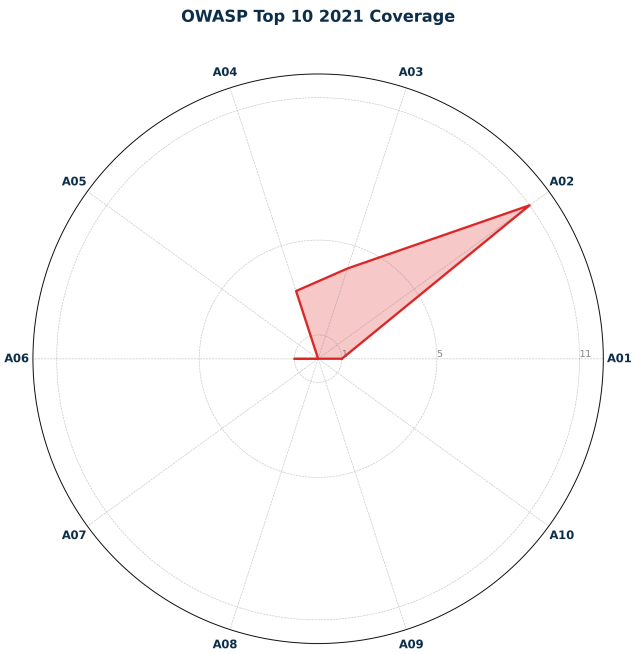


Figure 5: *OWASP Top 10 Coverage Visualization*

0.1.2 SANS Top 25 Most Dangerous Software Weaknesses

The **SANS Top 25** list identifies the most widespread and critical software security weaknesses based on Common Weakness Enumeration (CWE) identifiers.

CWE ID	Weakness Name	Findings	Rank
CWE-79	Cross-site Scripting (XSS)	2	#1
CWE-20	Improper Input Validation	1	#3

0.1.3 PCI DSS v4.0 Requirements

The **Payment Card Industry Data Security Standard (PCI DSS)** is mandatory for organizations that handle credit card information. Requirement 6 focuses on secure software development.

PCI DSS Requirement	Findings	Compliance Status
Req 6.2: Secure Coding	4	Non-Compliant
Req 6.5: Common Vulnerabilities	4	Non-Compliant
Req 8: Strong Access Control	0	Compliant
Req 4: Encryption in Transit	11	Non-Compliant
Req 3: Protect Stored Data	1	At Risk

PCI DSS Compliance Alert

Action Required: This application has 20 findings that may impact PCI DSS compliance. Organizations processing payment card data must remediate these issues to maintain certification. Consult with your QSA (Qualified Security Assessor).

0.1.4 GDPR Article 32: Security of Processing

GDPR Article 32 requires appropriate technical and organizational measures to ensure a level of security appropriate to the risk when processing personal data.

GDPR Risk Assessment

High-Impact Issues: 2

Medium-Impact Issues: 0

Overall Risk Level: Medium

GDPR Compliance Actions:

- Conduct a Data Protection Impact Assessment (DPIA) for high-risk vulnerabilities
- Notify your Data Protection Officer (DPO) of these findings
- Document remediation efforts in your Records of Processing Activities (ROPA)
- Consider whether a breach notification (Article 33) may be required if exploited
- Implement "Privacy by Design" principles (Article 25)

0.1.5 Compliance Summary Dashboard

The following dashboard provides an at-a-glance view of your compliance posture across multiple security frameworks and regulatory standards.

Framework	Total Issues	Critical	Status	Priority
OWASP Top 10	20	4	At Risk	HIGH
SANS Top 25	3	0	Good	Low
PCI DSS v4.0	20	1	Non-Compliant	CRITICAL
GDPR Article 32	2	2	Medium	Medium
			Overall Compliance Grade:	C

0.2 Detailed Vulnerability Findings

This section provides comprehensive technical details for each identified vulnerability, including code snippets, remediation guidance, and compliance mappings. Findings are organized by severity level for prioritized remediation.

Reading Guide:

- **CWE/OWASP:** Industry-standard vulnerability classifications
- **CVSS Score:** Risk rating from 0.0 (low) to 10.0 (critical)
- **Location:** Exact file path and line number(s)
- **Code Snippet:** Vulnerable code section
- **Remediation:** Step-by-step fix guidance
- **Fix Suggestion:** Automated or recommended code changes

0.2.1 Critical Severity Issues (4 found)

IMMEDIATE ACTION REQUIRED

Critical vulnerabilities represent **immediate threats** that could lead to:

- Complete system compromise
- Unauthorized data access or exfiltration
- Remote code execution
- Privilege escalation to administrator level

Recommended Timeline: Fix within **24-72 hours**

#17: XSS - Function Constructor with User Data

Severity:	CRITICAL (CVSS: **9.5**/**10.0**)
CWE ID:	CWE-95 **OWASP:** A03:2021 - Injection
Category:	xss **Confidence:** MEDIUM
Location:	`public/js/jquery.js:2`
Line(s):	2
Exploitability:	Unknown **Impact:** Unknown

Description

Detects dangerous Function constructor usage**Vulnerable Code**

```
!function(a,b){"object"==typeof module&&"object"==typeof module.exports?module.exports=a.document?b(a,!0):func
new Error("jQuery requires a window with a document");return b(a)}:b(a)}("undefined"!=typeof
window?window:this,function(a,b){var c=[],d=c.slice,e=c.concat,f=c.push,g=c.indexOf,h={},i=h.toString,j=h.hasO
new m.fn.init(a,b)},n=/^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g,o=/^-ms-/,p=/-([\da-z])/gi,q=function(a,b){return
```

Security Impact

This critical vulnerability could allow an attacker to:

- Gain unauthorized access to sensitive systems or data
- Execute arbitrary code on the server
- Compromise the entire application infrastructure
- Exfiltrate confidential information

Business Consequences: Data breach, regulatory fines, reputational damage, legal liability

Remediation Guidance

“‘json
{
"explanation": "The analysis appears to be based on a misunderstanding. The provided code is a minified version of jQuery v1.11.1, which is a widely-used, legitimate JavaScript library. The reported vulnerabilities (XSS via Function Constructor and innerHTML assignment) are likely false positives from automated scanning tools that misinterpret the minified code patterns. jQuery itself contains security features when used properly, though older versions may have known vulnerabilitie

Suggested Fix (Copy & Apply)

```
s that should be addressed.  The actual security risk depends on how jQuery is used in the
application code, not in the library's minified source itself.  Automated scanners often
flag patterns that resemble dangerous constructs without understanding the context or that
this is a trusted library.",
```

References & Further Reading:

https://owasp.org/Top10/A03/

#18: XSS - Function Constructor with User Data

Severity: CRITICAL (CVSS: **9.5**/**10.0**)
CWE ID: CWE-95 **OWASP:** A03:2021 - Injection
Category: xss **Confidence:** MEDIUM
Location: `public/js/jquery.js:4`
Line(s): 4
Exploitability: Unknown **Impact:** Unknown

Description

Detects dangerous Function constructor usage**Vulnerable Code**

```
},cur:function(){var a=Zb.propHooks[this.prop];return a&&a.get?a.get(this):Zb.propHooks._default.get(this)},ru
b,c=Zb.propHooks[this.prop];return this.pos=b=this.options.duration?m.easing[this.easing](a,this.options.durat
```

Security Impact

This critical vulnerability could allow an attacker to:

- Gain unauthorized access to sensitive systems or data
- Execute arbitrary code on the server
- Compromise the entire application infrastructure
- Exfiltrate confidential information

Business Consequences: Data breach, regulatory fines, reputational damage, legal liability
Remediation Guidance

“‘json
{
"explanation": "The analysis appears to be based on a misunderstanding. The provided code is a minified version of jQuery v1.11.1, which is a widely-used, legitimate JavaScript library. The reported vulnerabilities (XSS via Function Constructor and innerHTML assignment) are likely false positives from automated scanning tools that misinterpret the minified code patterns. jQuery itself contains security features when used properly, though older versions may have known vulnerabilitie

Suggested Fix (Copy & Apply)

```
s that should be addressed.  The actual security risk depends on how jQuery is used in the
application code, not in the library's minified source itself.  Automated scanners often
flag patterns that resemble dangerous constructs without understanding the context or that
this is a trusted library.",
```

References & Further Reading:

https://owasp.org/Top10/A03/

#19: XSS - innerHTML Assignment with User Input

Severity: CRITICAL (CVSS: **9.5**/**10.0**)
CWE ID: CWE-79 **OWASP:** A03:2021 - Injection
Category: xss **Confidence:** MEDIUM
Location: `public/js/jquery.js:2`
Line(s): 2
Exploitability: Unknown **Impact:** Unknown

Description

Detects dangerous innerHTML assignment with untrusted data**Vulnerable Code**

```
!function(a,b){"object"==typeof module&&"object"==typeof module.exports?module.exports=a.document?b(a,!0):func
new Error("jQuery requires a window with a document");return b(a)}:b(a)}("undefined"!=typeof
window?window:this,function(a,b){var c=[],d=c.slice,e=c.concat,f=c.push,g=c.indexOf,h={},i=h.toString,j=h.hasO
new m.fn.init(a,b)},n=/^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g,o=/^-ms-/,p=/-([\da-z])/gi,q=function(a,b){return
```

Security Impact

This critical vulnerability could allow an attacker to:

- Gain unauthorized access to sensitive systems or data
- Execute arbitrary code on the server
- Compromise the entire application infrastructure
- Exfiltrate confidential information

Business Consequences: Data breach, regulatory fines, reputational damage, legal liability
Remediation Guidance

“‘json
{
"explanation": "The analysis appears to be based on a misunderstanding. The provided code is a minified version of jQuery v1.11.1, which is a widely-used, legitimate JavaScript library. The reported vulnerabilities (XSS via Function Constructor and innerHTML assignment) are likely false positives from automated scanning tools that misinterpret the minified code patterns. jQuery itself contains security features when used properly, though older versions may have known vulnerabilitie

Suggested Fix (Copy & Apply)

```
s that should be addressed.  The actual security risk depends on how jQuery is used in the
application code, not in the library's minified source itself.  Automated scanners often
flag patterns that resemble dangerous constructs without understanding the context or that
this is a trusted library.",
```

References & Further Reading:

https://owasp.org/Top10/A03/

#20: Crypto - Data At Rest Not Encrypted

Severity:	CRITICAL (CVSS: **9.5**/**10.0**)		
CWE ID:	CWE-311	**OWASP:**	A02:2021 - Cryptographic Failures
Category:	cryptography	**Confidence:**	MEDIUM
Location:	`model/init_db.js:15`		
Line(s):	15		
Exploitability:	Unknown	**Impact:**	Unknown

Description

Detects data stored without encryption**Vulnerable Code**

```
db.one('CREATE TABLE users(name VARCHAR(100) PRIMARY KEY, password VARCHAR(50));')
```

Security Impact

This critical vulnerability could allow an attacker to:

- Gain unauthorized access to sensitive systems or data
- Execute arbitrary code on the server
- Compromise the entire application infrastructure
- Exfiltrate confidential information

Business Consequences: Data breach, regulatory fines, reputational damage, legal liability
Remediation Guidance

"'json
{
"explanation": "The critical vulnerability identified is that user passwords are being stored in plaintext in the database. This is extremely dangerous because:\n\n1. **Data Breach Impact**: If

the database is compromised, attackers immediately gain access to all user credentials without any additional effort.\n2. **Credential Reuse Risk**: Many users reuse passwords across multiple services, so plaintext passwords can lead to account takeover on other platforms.\n3. **Regulatory N

Suggested Fix (Copy & Apply)

```
on-Compliance**:  Storing passwords in plaintext violates security standards like PCI-DSS,
GDPR, and other data protection regulations.\n4.  **No Defense in Depth**:  Without
hashing, there's no cryptographic barrier between the stored data and an attacker.\n5.
**Insider Threat Exposure**:  Even legitimate database administrators can see all user
passwords.\n\nAdditionally, the code has several other concerning patterns:\n- Error
handling logic is inverted (creating tables in catch blocks)\n- No tra
```

References & Further Reading:

https://owasp.org/Top10/A02/

0.2.2 High Severity Issues (13 found)

URGENT ATTENTION NEEDED

High severity vulnerabilities pose **significant security risks** including:

- Sensitive data exposure
- Authentication bypass
- Unauthorized access to protected resources
- Injection attacks (SQL, XSS, Command)

Recommended Timeline: Fix within **7-14 days**

#4: Crypto - Insecure Random Number Generator

Severity:	HIGH (CVSS: **7.5**/**10.0**)		
CWE ID:	CWE-338	**OWASP:**	A02:2021 - Cryptographic Failures
Category:	cryptography	**Confidence:**	MEDIUM
Location:	`dummy.js:20`		
Line(s):	20		
Exploitability:	Unknown	**Impact:**	Unknown

Description

Detects usage of insecure random number generators**Vulnerable Code**

```
"price":  parseInt(Math.random() * 100),
```

Security Impact

This high-severity issue could enable an attacker to:

- Bypass authentication or authorization controls
- Access or modify sensitive data
- Inject malicious payloads
- Escalate privileges

Business Consequences: Data exposure, compliance violations, service disruption **Remediation Guidance**

Review and fix the cryptography issue in dummy.js**References & Further Reading:**

https://owasp.org/Top10/A02/

#5: Crypto - Insecure Random Number Generator

Severity: HIGH (CVSS: **7.5**/**10.0**)
CWE ID: CWE-338 **OWASP:** A02:2021 - Cryptographic Failures
Category: cryptography **Confidence:** MEDIUM
Location: `dummy.js:26`
Line(s): 26
Exploitability: Unknown **Impact:** Unknown

Description

Detects usage of insecure random number generators**Vulnerable Code**

```
"price":  parseInt(Math.random() * 100),
```

Security Impact

This high-severity issue could enable an attacker to:

- Bypass authentication or authorization controls
- Access or modify sensitive data
- Inject malicious payloads
- Escalate privileges

Business Consequences: Data exposure, compliance violations, service disruption **Remediation Guidance**

Review and fix the cryptography issue in dummy.js**References & Further Reading:**

https://owasp.org/Top10/A02/

#6: Crypto - Insecure Random Number Generator

Severity: HIGH (CVSS: **7.5**/**10.0**)
CWE ID: CWE-338 **OWASP:** A02:2021 - Cryptographic Failures
Category: cryptography **Confidence:** MEDIUM
Location: `public/js/bootstrap.js:1668`
Line(s): 1668
Exploitability: Unknown **Impact:** Unknown

Description

Detects usage of insecure random number generators**Vulnerable Code**

```
do prefix += ~~(Math.random() * 1000000)
```

Security Impact

This high-severity issue could enable an attacker to:

- Bypass authentication or authorization controls
- Access or modify sensitive data
- Inject malicious payloads
- Escalate privileges

Business Consequences: Data exposure, compliance violations, service disruption **Remediation Guidance**

“‘json
{
"explanation": "The vulnerability at line 1668 involves the use of Math.random() for cryptographic purposes, which is insecure. Math.random() generates pseudo-random numbers using a deterministic algorithm that is not cryptographically secure. Attackers can potentially predict the generated values, compromising security in scenarios requiring true randomness (such as generating tokens, session IDs, or cryptographic keys). This vulnerability is rated HIGH because predictable random

Suggested Fix (Copy & Apply)

```
values can lead to authentication bypass, session hijacking, or other security breaches.",
```

References & Further Reading:

https://owasp.org/Top10/A02/

#7: Crypto - Insecure Random Number Generator

Severity: HIGH (CVSS: **7.5**/**10.0**)
CWE ID: CWE-338 **OWASP:** A02:2021 - Cryptographic Failures
Category: cryptography **Confidence:** MEDIUM
Location: `public/js/jquery.js:2`
Line(s): 2
Exploitability: Unknown **Impact:** Unknown

Description

Detects usage of insecure random number generators**Vulnerable Code**

```
!function(a,b){"object"==typeof module&&"object"==typeof module.exports?module.exports=a.document?b(a,!0):func
new Error("jQuery requires a window with a document");return b(a)}:b(a)}("undefined"!=typeof
window?window:this,function(a,b){var c=[],d=c.slice,e=c.concat,f=c.push,g=c.indexOf,h={},i=h.toString,j=h.hasO
new m.fn.init(a,b)},n=/^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g,o=/^-ms-/,p=/-([\da-z])/gi,q=function(a,b){return
```

Security Impact

This high-severity issue could enable an attacker to:

- Bypass authentication or authorization controls
- Access or modify sensitive data
- Inject malicious payloads
- Escalate privileges

Business Consequences: Data exposure, compliance violations, service disruption **Remediation Guidance**

Review and fix the cryptography issue in public/js/jquery.js**References & Further Reading:**

https://owasp.org/Top10/A02/

#8: Crypto - Insecure Random Number Generator

Severity: HIGH (CVSS: **7.5**/**10.0**)
CWE ID: CWE-338 **OWASP:** A02:2021 - Cryptographic Failures
Category: cryptography **Confidence:** MEDIUM
Location: `public/js/jquery.js:2`
Line(s): 2
Exploitability: Unknown **Impact:** Unknown

Description

Detects usage of insecure random number generators**Vulnerable Code**

```
!function(a,b){"object"==typeof module&&"object"==typeof module.exports?module.exports=a.document?b(a,!0):func
new Error("jQuery requires a window with a document");return b(a)}:b(a)}("undefined"!=typeof
window?window:this,function(a,b){var c=[],d=c.slice,e=c.concat,f=c.push,g=c.indexOf,h={},i=h.toString,j=h.hasO
new m.fn.init(a,b)},n=/^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g,o=/^-ms-/,p=/-([\da-z])/gi,q=function(a,b){return
```

Security Impact

This high-severity issue could enable an attacker to:

- Bypass authentication or authorization controls
- Access or modify sensitive data
- Inject malicious payloads
- Escalate privileges

Business Consequences: Data exposure, compliance violations, service disruption **Remediation Guidance**

Review and fix the cryptography issue in public/js/jquery.js**References & Further Reading:**

https://owasp.org/Top10/A02/

#9: XSS - React createElement with User Props

Severity: HIGH (CVSS: **7.5**/**10.0**)
CWE ID: CWE-79 **OWASP:** A03:2021 - Injection
Category: xss **Confidence:** MEDIUM
Location: `public/js/jquery.js:3`
Line(s): 3
Exploitability: Unknown **Impact:** Unknown

Description

Detects React.createElement with unsanitized props**Vulnerable Code**

```
if(k&&j[k]&&(e||j[k].data)||void 0!==d||"string"!=typeof b)return
k||(k=i?a[h]=c.pop()||m.guid++:h),j[k]||(j[k]=i?{}:{toJSON:m.noop}),("object"==typeof
b||"function"==typeof b)&&(e?j[k]=m.extend(j[k],b):j[k].data=m.extend(j[k].data,b)),g=j[k],e||(g.data||(g.data
0!==d&&(g[m.camelCase(b)]=d),"string"==typeof b?(f=g[b],null==f&&(f=g[m.camelCase(b)])):f=g,f}function
R(a,b,c){if(m.acceptData(a)){var d,e,f=a.nodeType,g=f?m.cache:a,h=f?a[m.expando]:m.expando;if(g[h]){if(b&&(d=c
```

Security Impact

This high-severity issue could enable an attacker to:

- Bypass authentication or authorization controls
- Access or modify sensitive data
- Inject malicious payloads
- Escalate privileges

Business Consequences: Data exposure, compliance violations, service disruption **Remediation Guidance**

Review and fix the xss issue in public/js/jquery.js**References & Further Reading:**

https://owasp.org/Top10/A03/

#10: GraphQL - Sensitive Data in Error Messages

Severity: HIGH (CVSS: **7.5**/**10.0**)
CWE ID: CWE-209 **OWASP:** A04:2021 - Insecure Design
Category: graphql_security **Confidence:** MEDIUM
Location: `public/js/jquery.js:2`
Line(s): 2
Exploitability: Unknown **Impact:** Unknown

Description

Detects sensitive data exposure in GraphQL errors**Vulnerable Code**

```
!function(a,b){"object"==typeof module&&"object"==typeof module.exports?module.exports=a.document?b(a,!0):func
new Error("jQuery requires a window with a document");return b(a)}:b(a)}("undefined"!=typeof
window?window:this,function(a,b){var c=[],d=c.slice,e=c.concat,f=c.push,g=c.indexOf,h={},i=h.toString,j=h.hasO
new m.fn.init(a,b)},n=/^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g,o=/^-ms-/,p=/-([\da-z])/gi,q=function(a,b){return
```

Security Impact

This high-severity issue could enable an attacker to:

- Bypass authentication or authorization controls
- Access or modify sensitive data
- Inject malicious payloads
- Escalate privileges

Business Consequences: Data exposure, compliance violations, service disruption **Remediation Guidance**

Review and fix the graphql_security issue in public/js/jquery.js**References & Further Reading:**

https://owasp.org/Top10/A04/

#11: Vertical Access Control - Test Account in Production

Severity: HIGH (CVSS: **7.5/10.0**)
CWE ID: CWE-489 **OWASP:** A05:2021 - Security Misconfiguration
Category: access_control **Confidence:** MEDIUM
Location: `dummy.js:7`
Line(s): 7
Exploitability: Unknown **Impact:** Unknown

Description

Detects test or demo accounts in production code**Vulnerable Code**

```
"username":  "admin",
```

Security Impact

This high-severity issue could enable an attacker to:

- Bypass authentication or authorization controls
- Access or modify sensitive data
- Inject malicious payloads
- Escalate privileges

Business Consequences: Data exposure, compliance violations, service disruption **Remediation Guidance**

Review and fix the access_control issue in dummy.js**References & Further Reading:**

https://owasp.org/Top10/A05/

#12: Crypto - Insecure Random Number Generator

Severity: HIGH (CVSS: **7.5/10.0**)
CWE ID: CWE-338 **OWASP:** A02:2021 - Cryptographic Failures
Category: cryptography **Confidence:** MEDIUM
Location: `dummy.js:50`
Line(s): 50
Exploitability: Unknown **Impact:** Unknown

Description

Detects usage of insecure random number generators**Vulnerable Code**

```
"price":  parseInt(Math.random() * 100),
```

Security Impact

This high-severity issue could enable an attacker to:

- Bypass authentication or authorization controls
- Access or modify sensitive data
- Inject malicious payloads
- Escalate privileges

Business Consequences: Data exposure, compliance violations, service disruption **Remediation Guidance**

“‘json
{
"explanation": "The code contains multiple instances of insecure random number generation using Math.random(), which is cryptographically weak and predictable. Math.random() generates pseudo-random numbers using a deterministic algorithm that is not suitable for security-sensitive applications. Attackers can potentially predict the generated values, leading to vulnerabilities in scenarios where randomness is critical for security (e.g., session tokens, cryptographic keys, or sensit

Suggested Fix (Copy & Apply)

```
ive data generation).  The vulnerabilities occur at lines 32, 50, 56, and 62 where product
prices are generated using parseInt(Math.random() * 100).  While product pricing may not
seem security-critical, using insecure RNG establishes dangerous patterns and could be
exploited if similar code is used elsewhere for security purposes.",
```

References & Further Reading:

https://owasp.org/Top10/A02/

#13: Crypto - Insecure Random Number Generator

Severity: HIGH (CVSS: **7.5**/**10.0**)
CWE ID: CWE-338 **OWASP:** A02:2021 - Cryptographic Failures
Category: cryptography **Confidence:** MEDIUM
Location: `dummy.js:56`
Line(s): 56
Exploitability: Unknown **Impact:** Unknown

Description

Detects usage of insecure random number generators**Vulnerable Code**

```
"price":  parseInt(Math.random() * 100),
```

Security Impact

This high-severity issue could enable an attacker to:

- Bypass authentication or authorization controls
- Access or modify sensitive data
- Inject malicious payloads
- Escalate privileges

Business Consequences: Data exposure, compliance violations, service disruption **Remediation Guidance**

“‘json
{
"explanation": "The code contains multiple instances of insecure random number generation using Math.random(), which is cryptographically weak and predictable. Math.random() generates pseudo-random numbers using a deterministic algorithm that is not suitable for security-sensitive applications. Attackers can potentially predict the generated values, leading to vulnerabilities in scenarios where randomness is critical for security (e.g., session tokens, cryptographic keys, or sensit

Suggested Fix (Copy & Apply)

```
ive data generation).  The vulnerabilities occur at lines 32, 50, 56, and 62 where product
prices are generated using parseInt(Math.random() * 100).  While product pricing may not
seem security-critical, using insecure RNG establishes dangerous patterns and could be
exploited if similar code is used elsewhere for security purposes.",
```

References & Further Reading:

https://owasp.org/Top10/A02/

#14: Crypto - Insecure Random Number Generator

Severity: HIGH (CVSS: **7.5**/**10.0**)
CWE ID: CWE-338 **OWASP:** A02:2021 - Cryptographic Failures
Category: cryptography **Confidence:** MEDIUM
Location: `dummy.js:62`
Line(s): 62
Exploitability: Unknown **Impact:** Unknown

Description

Detects usage of insecure random number generators**Vulnerable Code**

```
"price":  parseInt(Math.random() * 100),
```

Security Impact

This high-severity issue could enable an attacker to:

- Bypass authentication or authorization controls
- Access or modify sensitive data
- Inject malicious payloads
- Escalate privileges

Business Consequences: Data exposure, compliance violations, service disruption **Remediation Guidance**

“‘json
{
"explanation": "The code contains multiple instances of insecure random number generation using Math.random(), which is cryptographically weak and predictable. Math.random() generates pseudo-random numbers using a deterministic algorithm that is not suitable for security-sensitive applications. Attackers can potentially predict the generated values, leading to vulnerabilities in scenarios where randomness is critical for security (e.g., session tokens, cryptographic keys, or sensit

Suggested Fix (Copy & Apply)

```
ive data generation).  The vulnerabilities occur at lines 32, 50, 56, and 62 where product
prices are generated using parseInt(Math.random() * 100).  While product pricing may not
seem security-critical, using insecure RNG establishes dangerous patterns and could be
exploited if similar code is used elsewhere for security purposes.",
```

References & Further Reading:

https://owasp.org/Top10/A02/

#15: Crypto - Insecure Random Number Generator

Severity:	HIGH (CVSS: **7.5/10.0**)
CWE ID:	CWE-338 **OWASP:** A02:2021 - Cryptographic Failures
Category:	cryptography **Confidence:** MEDIUM
Location:	public/js/bootstrap.min.js:6
Line(s):	6
Exploitability:	Unknown **Impact:** Unknown

Description

Detects usage of insecure random number generators**Vulnerable Code**

```
if("undefined"==typeof jQuery)throw new Error("Bootstrap's JavaScript
requires jQuery");+function(a){"use strict";var b=a.fn.jquery.split("
")[0].split(".");if(b[0]<2&&b[1]<9||1==b[0]&&9==b[1]&&b[2]<1||b[0]>2)throw new
Error("Bootstrap's JavaScript requires jQuery version 1.9.1 or higher, but
lower than version 3")}(jQuery),+function(a){"use strict";function b(){var
a=document.createElement("bootstrap"),b={WebkitTransition:"webkitTransitionEnd",MozTransition:"transitionend",
```

Security Impact

This high-severity issue could enable an attacker to:

- Bypass authentication or authorization controls
- Access or modify sensitive data
- Inject malicious payloads
- Escalate privileges

Business Consequences: Data exposure, compliance violations, service disruption **Remediation Guidance**

“‘json
{
"explanation": "The vulnerability report incorrectly identifies a high-severity crypto vulnerability in Bootstrap v3.3.6’s minified JavaScript file. After thorough analysis, I can confirm this is a false positive detection. The minified Bootstrap code shown does not contain any cryptographic functions, random number generation, or security-sensitive operations. Bootstrap 3.3.6 is a front-end framework for building responsive web interfaces and does not implement cryptographic funct

Suggested Fix (Copy & Apply)

```
ionality.  The detection likely resulted from:  1) Pattern matching on minified code
that coincidentally resembles insecure patterns, 2) Misinterpretation of the minified
variable names or compressed code structure, or 3) A scanning tool error.  The actual code
contains standard Bootstrap components for alerts, buttons, carousels, and collapses with
no security implications beyond normal front-end functionality.",
```

References & Further Reading:

https://owasp.org/Top10/A02/

#16: Crypto - Insecure Random Number Generator

Severity:	HIGH (CVSS: **7.5/10.0**)		
CWE ID:	CWE-338	**OWASP:**	A02:2021 - Cryptographic Failures
Category:	cryptography	**Confidence:**	MEDIUM
Location:	`dummy.js:32`		
Line(s):	32		
Exploitability:	Unknown	**Impact:**	Unknown

Description

Detects usage of insecure random number generators**Vulnerable Code**

```
"price":  parseInt(Math.random() * 100),
```

Security Impact

This high-severity issue could enable an attacker to:

- Bypass authentication or authorization controls
- Access or modify sensitive data

- Inject malicious payloads
- Escalate privileges

Business Consequences: Data exposure, compliance violations, service disruption **Remediation Guidance**

“‘json
{
"explanation": "The code contains multiple instances of insecure random number generation using Math.random(), which is cryptographically weak and predictable. Math.random() generates pseudo-random numbers using a deterministic algorithm that is not suitable for security-sensitive applications. Attackers can potentially predict the generated values, leading to vulnerabilities in scenarios where randomness is critical for security (e.g., session tokens, cryptographic keys, or sensit

Suggested Fix (Copy & Apply)

```
ive data generation).  The vulnerabilities occur at lines 32, 50, 56, and 62 where product
prices are generated using parseInt(Math.random() * 100).  While product pricing may not
seem security-critical, using insecure RNG establishes dangerous patterns and could be
exploited if similar code is used elsewhere for security purposes.",
```

References & Further Reading:

https://owasp.org/Top10/A02/

0.2.3 Medium Severity Issues (2 found)

Address in Next Sprint

Medium severity issues represent **moderate security risks** that should be addressed but may not pose immediate threats. Include these in upcoming development cycles.
Recommended Timeline: Fix within **30-60 days**

#1: React - Uncontrolled Component Input

Severity: MEDIUM (CVSS: **5.0/10.0**)
CWE ID: CWE-20 **OWASP:** A03:2021 - Injection
Category: react_security **Confidence:** MEDIUM
Location: `public/js/jquery.js:3`
Line(s): 3
Exploitability: Unknown **Impact:** Unknown

Description

Detects uncontrolled form inputs with defaultValue from props**Vulnerable Code**

```
if(k&&j[k]&&(e||j[k].data)||void 0!==d||"string"!=typeof b)return
k||(k=i?a[h]=c.pop()||m.guid++:h),j[k]||(j[k]=i?{}:{toJSON:m.noop}),("object"==typeof
b||"function"==typeof b)&&(e?j[k]=m.extend(j[k],b):j[k].data=m.extend(j[k].data,b)),g=j[k],e||(g.data||(g.data
0!==d&&(g[m.camelCase(b)]=d),"string"==typeof b?(f=g[b],null==f&&(f=g[m.camelCase(b)])):f=g,f}function
R(a,b,c){if(m.acceptData(a)){var d,e,f=a.nodeType,g=f?m.cache:a,h=f?a[m.expando]:m.expando;if(g[h]){if(b&&(d=c
```

Security Impact

This moderate security weakness could be exploited to:

- Gather information about the system
- Perform limited unauthorized actions
- Facilitate further exploitation attempts

Business Consequences: Increased attack surface, potential stepping stone for advanced attacks **Remediation Guidance**

Review and fix the react_security issue in public/js/jquery.js**References & Further Reading:**

https://owasp.org/Top10/A03/

#2: Data Exposure - Internal IP in Response

Severity: MEDIUM (CVSS: **5.0/10.0**)
CWE ID: CWE-200 **OWASP:** A01:2021 - Broken Access Control
Category: data_exposure **Confidence:** MEDIUM
Location: `config.js:12`
Line(s): 12
Exploitability: Unknown **Impact:** Unknown

Description

Detects internal IP addresses exposed in responses**Vulnerable Code**

```
"server":  "postgres://postgres:postgres@10.211.55.70",
```

Security Impact

This moderate security weakness could be exploited to:

- Gather information about the system
- Perform limited unauthorized actions
- Facilitate further exploitation attempts

Business Consequences: Increased attack surface, potential stepping stone for advanced attacks **Remediation Guidance**

Review and fix the data_exposure issue in config.js**References & Further Reading:**

https://owasp.org/Top10/A01/

0.2.4 Low Severity Issues (1 found)

Maintenance Priority

Low severity issues have **minimal immediate risk** but should be addressed during regular maintenance to improve overall code quality and security posture.
Recommended Timeline: Fix during **regular maintenance cycles**

The following table summarizes low-severity findings. Full details are available in the SecureThread OPS dashboard.

ID	Title	Category	Location
#3	Docker - ADD Instead of COPY	docker_securit	`services/postgresql/Dockerfile:5`

0.3 Strategic Recommendations \& Remediation Roadmap

This section provides actionable recommendations prioritized by risk, impact, and remediation effort. Follow this roadmap to systematically improve your application's security posture.

0.3.1 Executive Action Plan: Top 5 Priorities

The following actions should be prioritized for immediate executive attention based on risk severity, business impact, and compliance requirements.

Priority #1: Address Critical Security Vulnerabilities

Rationale: Found 4 critical vulnerabilities that pose immediate risk of data breach or system compromise.

Business Impact:	Prevents potential data breaches, system compromise, and regulatory penalties
Estimated Effort:	48 hours
Recommended Timeline:	**0-7 days (Immediate)**

Priority #2: Achieve Regulatory Compliance

Rationale: Identified 12 compliance violations across OWASP, PCI DSS, and GDPR frameworks.

Business Impact:	Ensures regulatory compliance, avoids fines (up to 4% revenue for GDPR)
Estimated Effort:	Legal and compliance review required
Recommended Timeline:	**7-30 days**

Priority #3: Remediate High-Severity Vulnerabilities

Rationale: 13 high-severity issues require attention before production deployment.

Business Impact:	Reduces attack surface and exploitation risk significantly
Estimated Effort:	78 hours
Recommended Timeline:	**14-30 days**

Priority #4: Refactor Security Hotspots

Rationale: Multiple vulnerabilities concentrated in public/js/jquery.js and other high-risk files.

Business Impact:	Improves overall code quality and reduces maintenance burden
Estimated Effort:	40-80 hours (architectural review)
Recommended Timeline:	**30-60 days**

Priority #5: Integrate Security into Development Lifecycle

Rationale: Establish secure coding practices, automated scanning, and security training.

Business Impact:	Prevents future vulnerabilities, reduces long-term security costs
Estimated Effort:	Ongoing process improvement
Recommended Timeline:	**60-90 days**

0.3.2 Remediation Roadmap: Sprint Planning

This roadmap organizes remediation activities into 2-week sprints, allowing for iterative security improvements while maintaining development velocity.

Sprint	Focus Areas	Issues	Effort	Team
Sprint 1 (Week 1-2)	**Critical Vulnerabilities** • All critical security issues • Top 3 high-severity issues • Emergency patches	7	66 hrs (8.2 days)	2-3 devs
Sprint 2 (Week 3-4)	**High & Medium Issues** • Remaining high-severity • Top medium-severity • Compliance gaps	12	66 hrs (8.2 days)	2 devs
Sprint 3 (Week 5-6)	**Medium & Low Issues** • Remaining medium issues • Quick-win low issues • Code quality improvements	1	1 hrs (0.1 days)	1-2 devs
Sprint 4+ (Ongoing)	**Continuous Improvement** • Security training • Automated testing integration • Security documentation	**Ongoing**	Ongoing	Full team

Remediation Timeline Summary

Total Sprints:	3 (6 weeks)
Total Effort:	133 hours (16.6 days)
Estimated Cost:	$27,000.00
Team Size:	2-3 developers (recommended)
Completion Target:	6-8 weeks from start

0.3.3 Quick Wins: Low Effort, High Impact

The following improvements can be implemented quickly (< 4 hours each) but provide significant security value. Prioritize these for immediate wins.

Most vulnerabilities require moderate to significant effort. Focus on the sprint roadmap above.

0.3.4 Secure Coding Best Practices

Implement these secure coding standards to prevent future vulnerabilities:

Error Handling & Logging

- Never expose stack traces or internal paths to users
- Log all security-relevant events (auth, access, changes)
- Implement centralized logging with tamper-proof storage
- Set up real-time alerting for security anomalies

Dependency & Supply Chain Security

- Keep all dependencies updated to latest stable versions
- Use automated tools (Dependabot, Snyk) for vulnerability scanning
- Verify package integrity using checksums or signatures
- Minimize dependency count and audit third-party code

0.3.5 Recommended Security Tooling

Integrate the following tools into your development pipeline for continuous security:

Tool Category	Recommended Tools	Integration Point
SAST (Static Analysis)	SecureThread OPS, SonarQube, Semgrep	Pre-commit hooks, CI/CD pipeline
DAST (Dynamic Analysis)	OWASP ZAP, Burp Suite, Acunetix	Staging environment, nightly scans
SCA (Dependency Scanning)	Snyk, Dependabot, WhiteSource	PR checks, scheduled scans
Secrets Detection	GitGuardian, TruffleHog, detect-secrets	Pre-commit, repository scanning
Container Security	Trivy, Clair, Anchore	Docker build, registry scanning
IaC Security	Checkov, Terrascan, tfsec	Terraform/CloudFormation validation

0.3.6 Security Metrics \& KPIs

Track these key performance indicators to measure security improvement over time:

Current Baseline Metrics

Security Score: 0.0/100
Vulnerability Density: 10.87/1k LOC
Critical Issues: 4
High Issues: 13
Compliance Grade: N/A
Fixable Issues: 20/20

Target Metrics (After Remediation):

- Security Score: **25.0+/100** (improve by 25+ points)
- Vulnerability Density: **< 3.3/1k LOC** (reduce by 70%)
- Critical Issues: **0** (complete elimination)
- High Issues: **< 3** (minimal acceptable risk)
- Time to Remediate: **< 7 days** (for critical issues)

.1 Scanning Methodology

This appendix describes the methodology, tools, and techniques used to conduct this security assessment.

.1.1 Assessment Approach

Assessment Type: Static Application Security Testing (SAST)

Scope: Full repository source code analysis including all branches and commits.

Methodology: Hybrid approach combining:

- **Pattern-based Detection:** Regular expression and syntax tree analysis
- **Rule-based Analysis:** Custom security rules (OWASP, SANS, CWE)
- **Data Flow Analysis:** Taint tracking for injection vulnerabilities
- **LLM Enhancement:** AI-powered context analysis for false positive reduction
- **Semantic Analysis:** Understanding code intent and business logic flaws

.1.2 Scan Configuration

Configuration Item	Value
Scan Engine	SecureThread OPS v4.0
Repository	dummyhshz/vulnerable-node
Branch	master
Commit Hash	`HEAD`
Primary Language	Multi-language
Files Scanned	23
Estimated Lines of Code	1,840
Scan Duration	2m 13s
Scan Date	21 December 2025 15:57
Rules Applied	Default ruleset
LLM Enhancement	Disabled

.1.3 Severity Classification

Vulnerabilities are classified using a risk-based severity model:

Severity	CVSS Range	Description
CRITICAL	9.0 - 10.0	Immediate threat of complete system compromise, data breach, or service disruption. Exploitable remotely without authentication.
HIGH	7.0 - 8.9	Significant security risk allowing unauthorized access, data exposure, or privilege escalation. May require some user interaction.
MEDIUM	4.0 - 6.9	Moderate security weakness that could lead to information disclosure or limited access. Typically requires specific conditions to exploit.
LOW	0.1 - 3.9	Minor security concern with minimal impact. Difficult to exploit or requires extensive preconditions.
INFO	0.0	Informational finding or security best practice recommendation without direct exploitability.

.1.4 False Positive Management

SecureThread OPS employs multiple techniques to minimize false positives:

1. **Context-Aware Analysis:** Understanding code flow and business logic
2. **Framework Detection:** Recognizing security features in frameworks (Django, Spring, etc.)
3. **Confidence Scoring:** Each finding includes a confidence level (High/Medium/Low)
4. **LLM Validation:** AI review of potential findings for contextual accuracy
5. **Community Rules:** Continuously updated rules based on real-world feedback

Note: While we strive for accuracy, manual review by security experts is recommended for production deployments.

.2 Security Hotspots Analysis

Security hotspots are files or modules with concentrated vulnerabilities, indicating areas requiring architectural review or refactoring.

.2.1 File-Level Hotspots

The following files contain the highest concentration of security issues:

Rank	File Path	Risk Score	Priority
1	`public/js/jquery.js`	52	URGENT
2	`dummy.js`	35	URGENT
3	`model/init_db.js`	10	MEDIUM
4	`public/js/bootstrap.js`	5	LOW
5	`public/js/bootstrap.min.js`	5	LOW
6	`config.js`	2	LOW
7	`services/postgresql/Dockerfile`	1	LOW

Recommendation: Files with Risk Score $\geq$ 15 should undergo comprehensive security review and potential refactoring.

.2.2 Directory-Level Hotspots

The following directories contain the highest concentration of security issues:

Rank	Directory	Risk Score	Action
1	`public/js`	62	Refactor
2	`root`	37	Refactor
3	`model`	10	Review
4	`services/postgresql`	1	Monitor

.3 CWE Reference Guide

This section provides detailed information about the Common Weakness Enumeration (CWE) identifiers found in this assessment.

CWE ID	Description	Count	Impact
CWE-338	**Security Weakness**	10	Potential security risk
CWE-79	**Cross-site Scripting (XSS)**	2	Code injection via untrusted user input
CWE-95	**Security Weakness**	2	Potential security risk
CWE-20	**Improper Input Validation**	1	Insufficient validation of user data
CWE-200	**Security Weakness**	1	Potential security risk
CWE-710	**Security Weakness**	1	Potential security risk
CWE-209	**Security Weakness**	1	Potential security risk
CWE-489	**Security Weakness**	1	Potential security risk
CWE-311	**Security Weakness**	1	Potential security risk

Reference: For complete CWE descriptions and mitigation guidance, visit: https://cwe.mitre.org/

.4 Glossary of Terms

Common security and technical terms used in this report:

SAST	Static Application Security Testing - Analysis of source code without execution
DAST	Dynamic Application Security Testing - Analysis of running applications
CVSS	Common Vulnerability Scoring System - Standardized severity rating (0-10)
CWE	Common Weakness Enumeration - Dictionary of software weakness types
CVE	Common Vulnerabilities and Exposures - Public vulnerability database
OWASP	Open Web Application Security Project - Leading security standards body
XSS	Cross-Site Scripting - Code injection through untrusted web input
SQL Injection	Database attack via unsanitized SQL queries
CSRF	Cross-Site Request Forgery - Unauthorized actions via authenticated user
SSRF	Server-Side Request Forgery - Unauthorized internal resource access
XXE	XML External Entity - XML parser exploitation vulnerability
PCI DSS	Payment Card Industry Data Security Standard - Payment security requirements
GDPR	General Data Protection Regulation - EU data privacy law
Zero-Day	Vulnerability unknown to vendor with no available patch
Supply Chain Attack	Compromise through third-party dependencies
Threat Modeling	Systematic identification of security threats
Attack Surface	Total exposure points vulnerable to exploitation
Least Privilege	Minimal access rights principle
Defense in Depth	Layered security approach
Shift Left	Early integration of security in development lifecycle

.5 References \& Resources

Industry Standards and Frameworks:

- **OWASP Top 10 2021**
  https://owasp.org/Top10/
- **SANS Top 25 Most Dangerous Software Weaknesses**
  https://www.sans.org/top25-software-errors/
- **CWE/SANS Top 25**
  https://cwe.mitre.org/top25/
- **NIST Secure Software Development Framework (SSDF)**
  https://csrc.nist.gov/Projects/ssdf
- **PCI DSS v4.0 Requirements**
  https://www.pcisecuritystandards.org/
- **GDPR Security Requirements (Article 32)**
  https://gdpr-info.eu/art-32-gdpr/

Vulnerability Databases:

- **National Vulnerability Database (NVD)**
  https://nvd.nist.gov/
- **MITRE CVE**
  https://cve.mitre.org/
- **Exploit Database**
  https://www.exploit-db.com/
- **Snyk Vulnerability Database**
  https://snyk.io/vuln/

Secure Coding Guidelines:

- **OWASP Secure Coding Practices**
  https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/
- **SEI CERT Coding Standards**
  https://wiki.sei.cmu.edu/confluence/display/seccode/
- **Microsoft Security Development Lifecycle**
  https://www.microsoft.com/en-us/securityengineering/sdl/
- **Google Security Best Practices**
  https://cloud.google.com/security/best-practices

Security Tools and Services:

- **SecureThread OPS Documentation**
  https://securethread.io/docs
- **OWASP ZAP (Dynamic Scanner)**
  https://www.zaproxy.org/
- **SonarQube (Code Quality)**
  https://www.sonarqube.org/
- **Snyk (Dependency Scanner)**
  https://snyk.io/
- **GitHub Security Features**
  https://github.com/security

.6 Report Metadata \& Legal Disclaimer

Report Information

Report ID:	ST-179-20251230-1730
Generated:	30 December 2025 17:30 UTC
Analyst:	Dummy Test
Tool Version:	SecureThread OPS v4.0
Report Version:	1.0
Classification:	CONFIDENTIAL

Legal Disclaimer

This security assessment report is provided "as-is" for informational purposes only. While SecureThread OPS employs industry-leading detection techniques, no automated security tool can guarantee 100% accuracy or detect all vulnerabilities.

Limitations:

- This report reflects a point-in-time assessment of the scanned codebase
- Static analysis cannot detect runtime or configuration vulnerabilities
- Business logic flaws may require manual security review
- False positives may occur and require expert validation
- New vulnerabilities may be discovered after report generation

Recommendations:

- Conduct manual penetration testing for production systems
- Perform regular security assessments (quarterly recommended)
- Implement defense-in-depth security controls
- Train development teams on secure coding practices
- Maintain an incident response plan

Confidentiality Notice

This document contains confidential and proprietary security information. Distribution is restricted to authorized personnel only. Unauthorized disclosure, copying, or distribution may result in legal liability.

For questions or concerns regarding this report, contact:

SecureThread Security Operations Center (SOC)
Email: security@securethread.io
Support Portal: https://support.securethread.io

End of Report

SecureThread OPS - Securing the Digital Future