# AI-Powered Security Policy Report

Generated: 2025-11-06 12:53:13

**Total Vulnerabilities Scanned:** 26
• SAST: 8
• SCA: 10
• DAST: 8

## LLM Models Used:

• LLaMA 3.3 70B (Groq) - SAST/SCA
• LLaMA 3.1 8B Instant (Groq) - DAST

## Policy #1: SAST

**Title:** Node Sqli
**Severity:** HIGH
**LLM:** LLaMA 3.3 70B

## POLICY IDENTIFIER
SP-2024-001: SQL Injection Protection Policy

## RISK STATEMENT
The organization faces a significant risk of data breach and unauthorized access to sensitive information due to the presence of SQL injection vulnerabilities in our applications. If exploited, this vulnerability could lead to the theft of confidential data, disruption of business operations, and damage to our reputation. The affected systems include our user management application, and the impacted data includes user credentials and personal information.

## COMPLIANCE MAPPING
This policy is aligned with the following compliance requirements:
- NIST CSF: PR.DS-5 (Data Protection), DE.CM-8 (Vulnerability Management)
- ISO 27001: A.8.8 (Management of technical vulnerabilities), A.5.37 (Documented operating procedures), A.5.26 (Response to information security incidents), A.8.16 (Monitoring activities), A.8.26 (Application security requirements)
- Industry standards: OWASP A03:2021 (Injection), CWE-89 (Improper Neutralization of Special Elements used in an SQL Command)

## POLICY REQUIREMENTS
To mitigate the risk of SQL injection attacks, the following security controls must be implemented:
- All user input must be validated and sanitized before being used in SQL queries.
- Parameterized queries or prepared statements must be used to construct SQL queries.

- Regular security testing and code reviews must be performed to identify and remediate potential vulnerabilities.
- Success criteria: All user input is validated and sanitized, and parameterized queries are used throughout the application.
- Validation methods: Code reviews, penetration testing, and vulnerability scanning.

## REMEDIATION PLAN
To remediate the identified vulnerability, the following actions must be taken:
- Technical action: Review and refactor the affected code in src/controllers/UserController.js to use parameterized queries or prepared statements.
- Responsible party: Security Lead
- Timeline: 1 week
- Verification steps: Code review, re-scan, and penetration test to ensure the vulnerability is fully remediated.

## MONITORING AND DETECTION
To detect similar vulnerabilities in the future, the following measures must be implemented:
- Regular security testing and code reviews must be performed to identify potential vulnerabilities.
- Logging and alerting requirements: All security-related events must be logged and alerts must be generated in case of suspected SQL injection attempts.
- Continuous monitoring strategies: Regular vulnerability scanning and penetration testing must be performed to identify and remediate potential vulnerabilities.

# Policy #2: SAST

**Title:** Var In Href
**Severity:** MEDIUM
**LLM:** LLaMA 3.3 70B

**SP-2024-001: Cross-Site Scripting (XSS) Protection Policy**

### 1. POLICY IDENTIFIER
Policy ID: SP-2024-001
Policy Title: Cross-Site Scripting (XSS) Protection Policy

### 2. RISK STATEMENT
The organization is at risk of Cross-Site Scripting (XSS) attacks due to the use of template variables in anchor tags with the 'href' attribute. This vulnerability could allow malicious actors to inject malicious code, potentially leading to unauthorized access, data theft, or system compromise. The affected systems include the profile management module, and the impacted users are all registered users of the application.

### 3. COMPLIANCE MAPPING
This policy aligns with the following standards and guidelines:
- NIST CSF: PR.DS-5 (Data Protection), DE.CM-8 (Vulnerability Management)
- ISO 27001: A.8.23 (Web filtering), A.8.8 (Management of technical vulnerabilities), A.5.7 (Threat intelligence)
- Industry standards: OWASP Top 10 (A07:2021 - Cross-Site Scripting), CWE-79 (Improper Neutralization of Input During Web Page Generation)

### 4. POLICY REQUIREMENTS
To mitigate XSS vulnerabilities, the following security controls must be implemented:
- Validate and sanitize all user input data used in web page generation.
- Implement Content Security Policy (CSP) to define which sources of content are allowed to be executed within a web page.
- Use secure coding practices, such as encoding output and using prepared statements.
Success criteria include regular security audits and penetration testing to ensure the absence of XSS vulnerabilities. Validation methods include code reviews, automated vulnerability scanning, and manual testing.

### 5. REMEDIATION PLAN
To remediate the identified vulnerability:
- Technical actions: Review and fix the template variable usage in the 'href' attribute in the profile management module (src/views/profile.mustache, line 23).
- Responsible party: Development Lead
- Timeline: 2 weeks
- Verification steps: Conduct a code review, re-scan the application using automated vulnerability scanning tools, and perform a penetration test to ensure the vulnerability is resolved.

### 6. MONITORING AND DETECTION
To detect similar vulnerabilities in the future:
- Implement regular automated vulnerability scanning and manual penetration testing.
- Require logging and alerting for suspicious web application activity, such as unusual URL patterns or user input.
- Continuously monitor web application security using Web Application Firewalls (WAFs) and intrusion

detection systems.
Regular security audits and compliance reviews will ensure the effectiveness of this policy and identify areas for improvement.

# Policy #3: SAST

**Title:** Path Join Resolve Traversal
**Severity:** HIGH
**LLM:** LLaMA 3.3 70B

## POLICY IDENTIFIER
SP-2024-001: Path Traversal Protection Policy

## RISK STATEMENT
The Path Traversal vulnerability poses a significant risk to our organization's data security. If exploited, an attacker could potentially access sensitive files and data outside the intended directory, leading to unauthorized data disclosure, modification, or deletion. This could result in reputational damage, financial loss, and legal liabilities. The affected systems include our web application servers, and the impacted users are our customers and employees who rely on these systems for daily operations.

## COMPLIANCE MAPPING
This policy aligns with the following compliance requirements:
- NIST CSF: PR.DS-5 (Data Protection), DE.CM-8 (Vulnerability Management)
- ISO 27001: A.8.28 (Secure Coding), A.8.29 (Security Testing in Development and Acceptance)
- Industry Standards: OWASP A5:2021 (Security Misconfiguration), CWE-22 (Improper Limitation of a Pathname to a Restricted Directory)

## POLICY REQUIREMENTS
To mitigate the Path Traversal vulnerability, the following security controls must be implemented:
- Input validation and sanitization for all user-provided file paths
- Implementation of a whitelist-based approach to restrict file access to authorized directories
- Regular security testing and code reviews to identify and address potential vulnerabilities
- Success criteria: No unauthorized access to sensitive files or data outside the intended directory
- Validation methods: Regular penetration testing, code reviews, and vulnerability scanning

## REMEDIATION PLAN
To remediate the identified vulnerability:
- Technical actions: Review and fix the vulnerable code in src/routes/files.js (line 67) to properly sanitize user input and restrict file access to authorized directories
- Responsible party: Security Lead
- Timeline: 1 week
- Verification steps: Code review, re-scan using vulnerability scanning tools, and penetration testing to ensure the fix is effective

## MONITORING AND DETECTION
To detect similar vulnerabilities in the future:
- Implement logging and alerting mechanisms to monitor file access patterns and detect potential path traversal attempts
- Conduct regular security audits and code reviews to identify and address potential vulnerabilities
- Utilize continuous monitoring strategies, including vulnerability scanning and penetration testing, to ensure the security of our systems and data
- Regularly review and update our secure coding practices and security testing processes to ensure alignment with industry standards and best practices.

# Policy #4: SAST

**Title:** Express Cookie Session No Httponly
**Severity:** MEDIUM
**LLM:** LLaMA 3.3 70B

## SP-2024-001: Secure Session Management Policy
### POLICY IDENTIFIER
The unique policy ID for this section is SP-2024-001, and the policy title is "Secure Session Management Policy," which aligns with the vulnerability category of Session Management.

### RISK STATEMENT
The absence of the 'HttpOnly' flag in cookie sessions poses a significant risk to our organization. In non-technical terms, this vulnerability could allow malicious actors to hijack user sessions, potentially leading to unauthorized access to sensitive information. The potential consequences if exploited include data breaches, financial loss, and reputational damage. The affected systems include our web application, the data includes user session information, and the users impacted are all individuals who interact with our web application.

### COMPLIANCE MAPPING
This policy aligns with the following compliance requirements:
- NIST CSF: PR.DS-5 (Data Protection), which emphasizes the importance of protecting sensitive data.
- ISO 27001 Annex A controls: A.8.1 (User endpoint devices), A.8.5 (Secure authentication), and A.8.23 (Web filtering), which collectively stress the need for secure authentication, protection of user endpoint devices, and management of access to external websites.
- Relevant industry standards include OWASP's guidance on secure cookie practices and CWE-1004, which specifically addresses sensitive cookies without the 'HttpOnly' flag.

### POLICY REQUIREMENTS
To mitigate the identified vulnerability, the following security controls must be implemented:
- All cookie sessions must be configured with the 'HttpOnly' flag to prevent JavaScript access.
- Regular security testing must be performed to identify similar vulnerabilities.
- Success criteria include verification that all cookie sessions have the 'HttpOnly' flag set, which can be validated through code reviews and vulnerability scans.

### REMEDIATION PLAN
To remediate this vulnerability:
- Specific technical action: Set the 'HttpOnly' flag for all cookie sessions in the src/config/session.js file (line 12).
- Responsible party: Dev Lead, given the medium severity of this vulnerability.
- Timeline: Completion within 2 weeks.
- Verification steps: Code review to confirm the 'HttpOnly' flag is set for all cookie sessions, followed by a re-scan to ensure the vulnerability is resolved.

### MONITORING AND DETECTION
To detect similar vulnerabilities in the future:
- Regular security audits and code reviews must be conducted.
- Logging and alerting must be configured to identify potential session hijacking attempts.
- Continuous monitoring strategies include periodic vulnerability scans and penetration testing to ensure the security of our session management practices.
By following this policy, we aim to protect user session information and prevent session hijacking, thereby safeguarding our users' data and maintaining the trust and integrity of our web application.

# Policy #5: SAST

**Title:** Hardcoded Secret
**Severity:** HIGH
**LLM:** LLaMA 3.3 70B

## POLICY IDENTIFIER
Policy ID: SP-2024-001
Policy Title: Secure Coding Practices for Hardcoded Secrets

## RISK STATEMENT
The presence of hardcoded secrets in our codebase poses a significant risk to the security of our systems and data. If exploited, an attacker could gain unauthorized access to sensitive information, compromising the confidentiality, integrity, and availability of our systems. This vulnerability affects our development, testing, and production environments, potentially impacting all users and data.

## COMPLIANCE MAPPING
This policy aligns with the following compliance requirements:
- NIST CSF: PR.DS-5 (Data Protection), PR.IP-1 (Identity and Access Control)
- ISO 27001: A.8.28 (Secure coding), A.8.12 (Data leakage prevention), A.8.27 (Secure system architecture and engineering principles), A.8.31 (Separation of development, test and production environments), A.8.24 (Use of cryptography)
- Industry standards: OWASP, CWE-798 (Use of Hard-coded Credentials)

## POLICY REQUIREMENTS
To mitigate the risk of hardcoded secrets, the following security controls must be implemented:
- All secrets must be stored in environment variables or a secure secret management system.
- Code reviews must be conducted regularly to detect and remove hardcoded secrets.
- Secure coding principles and guidelines must be established and followed.
Success criteria include:
- No hardcoded secrets detected in code reviews.
- All secrets stored in environment variables or a secure secret management system.
Validation methods include:
- Regular code reviews.
- Automated scanning tools.

## REMEDIATION PLAN
To remediate this vulnerability, the following technical actions are required:
- Review and fix the hardcoded secret in src/config/auth.js (line 8).
- Implement a secure secret management system.
Responsible party: Security Lead.
Timeline: 1 week.
Verification steps:
- Code review.
- Automated scanning tools.

## MONITORING AND DETECTION
To detect similar vulnerabilities in the future, the following measures must be implemented:
- Regular code reviews.
- Automated scanning tools.
- Logging and alerting requirements:
- Log all access to sensitive information.

- Alert on suspicious activity.

Continuous monitoring strategies:

- Regularly review and update secure coding principles and guidelines.
- Conduct periodic penetration testing and vulnerability assessments.

# Policy #6: SCA

**Title:** lodash - CWE-1321
**Severity:** HIGH
**LLM:** LLaMA 3.3 70B

**POLICY IDENTIFIER**
SP-2024-001
Lodash Prototype Pollution Policy

**RISK STATEMENT**
The lodash prototype pollution vulnerability poses a significant risk to our organization's intellectual property and sensitive data. If exploited, this vulnerability could lead to unauthorized access, modification, or theft of sensitive information, resulting in reputational damage, financial loss, and legal liabilities. The affected systems include our web applications, development environments, and production servers, which could impact all users and data.

**COMPLIANCE MAPPING**
This policy aligns with the following compliance requirements:
* NIST CSF: PR.DS-5 (Data Protection), DE.CM-1 (Anomaly Detection)
* ISO 27001: A.5.32 (Intellectual Property Rights), A.8.23 (Web Filtering), A.8.31 (Separation of Development, Test, and Production Environments), A.8.30 (Outsourced Development), A.5.10 (Acceptable Use of Information and Other Associated Assets)
* Industry standards: OWASP A9 (Using Components with Known Vulnerabilities), CWE-1321 (Prototype Pollution)

**POLICY REQUIREMENTS**
To mitigate the lodash prototype pollution vulnerability, the following security controls must be implemented:
* Use a secure version of lodash that is not vulnerable to prototype pollution (success criteria: verified through code review and vulnerability scanning)
* Implement web filtering to restrict access to malicious content (success criteria: verified through logging and alerting)
* Separate development, testing, and production environments to prevent unauthorized access (success criteria: verified through environment audits)
* Monitor and review outsourced development activities to ensure compliance with security policies (success criteria: verified through regular audits and reporting)

**REMEDICATION PLAN**
To remediate the lodash prototype pollution vulnerability, the following technical actions are required:
* Update all instances of lodash to a secure version (technical action)
* Implement web filtering and restrict access to malicious content (technical action)
* Responsible party: Security Lead
* Timeline: 1 week
* Verification steps: code review, re-scan, and penetration testing

**MONITORING AND DETECTION**
To detect similar vulnerabilities in the future, the following monitoring and detection strategies will be implemented:
* Regular vulnerability scanning and code reviews
* Logging and alerting for suspicious activity
* Continuous monitoring of web applications and development environments

* Annual penetration testing and security audits to ensure compliance with security policies and procedures.

# Policy #7: SCA

**Title:** express - CWE-601
**Severity:** MEDIUM
**LLM:** LLaMA 3.3 70B

## POLICY IDENTIFIER
Policy ID: SP-2024-001
Policy Title: Express Open Redirect Protection Policy

## RISK STATEMENT
The Express Open Redirect vulnerability poses a significant risk to our organization's web applications, allowing attackers to redirect users to malicious websites, potentially leading to phishing, malware distribution, or other cyber threats. If exploited, this vulnerability could compromise sensitive user data, damage our reputation, and disrupt business operations. The affected systems include all web applications built using the Express framework, and the impacted users are all customers and employees interacting with these applications.

## COMPLIANCE MAPPING
This policy aligns with the following compliance requirements:
- NIST CSF: PR.DS-5 (Data Protection), DE.AE-2 (Anomaly Detection)
- ISO 27001 Annex A controls: A.8.23 (Web filtering), A.5.30 (ICT readiness for business continuity)
- Industry standards: OWASP A10:2021 (Server-Side Request Forgery), CWE-601 (URL Redirection to Untrusted Site)

## POLICY REQUIREMENTS
To mitigate the Express Open Redirect vulnerability, the following security controls must be implemented:
- Validate all user-inputted URLs to ensure they are legitimate and within our domain.
- Implement a web application firewall (WAF) to detect and block suspicious traffic.
- Conduct regular security audits and penetration testing to identify potential vulnerabilities.
Success criteria include:
- No open redirect vulnerabilities detected during security audits.
- All user-inputted URLs are validated and sanitized.
Validation methods include code reviews, automated security scans, and penetration testing.

## REMEDIATION PLAN
To remediate the Express Open Redirect vulnerability, the following technical actions are required:
- Update the Express framework to the latest version.
- Implement URL validation and sanitization using a trusted library.
The responsible party for this remediation is the Development Lead. The timeline for completion is 2 weeks. Verification steps include code review, automated security scans, and penetration testing.

## MONITORING AND DETECTION
To detect similar vulnerabilities in the future, we will implement the following monitoring and detection strategies:
- Regular security audits and penetration testing.
- Logging and alerting for suspicious traffic patterns.
- Continuous monitoring of web application security using automated tools.
We will also maintain up-to-date documentation of our web applications' security configurations and ensure that all developers are trained on secure coding practices, including input validation and sanitization.

# Policy #8: SCA

**Title:** axios - CWE-918
**Severity:** HIGH
**LLM:** LLaMA 3.3 70B

## POLICY IDENTIFIER
SP-2024-001: Server-Side Request Forgery (SSRF) Protection Policy

## RISK STATEMENT
The Server-Side Request Forgery (SSRF) vulnerability in axios poses a significant risk to our organization's security posture. If exploited, this vulnerability could allow attackers to bypass security controls, access sensitive data, and potentially disrupt business operations. The affected systems include all applications utilizing the axios library, and the impacted data includes sensitive user information and internal network resources.

## COMPLIANCE MAPPING
This policy aligns with the following compliance requirements:
- NIST CSF: PR.DS-5 (Data Protection), DE.AE-2 (Anomaly Detection)
- ISO 27001 Annex A controls: A.8.23 (Web filtering), A.8.1 (User endpoint devices), A.5.10 (Acceptable use of information and other associated assets), A.5.33 (Protection of records), A.8.16 (Monitoring activities)
- Industry standards: OWASP A10:2021 (Server-Side Request Forgery), CWE-918 (Server-Side Request Forgery)

## POLICY REQUIREMENTS
To mitigate the SSRF vulnerability, the following security controls must be implemented:
- Validate and sanitize all user-input data used in axios requests
- Implement web filtering to restrict access to external websites and resources
- Configure axios to use a proxy server or a whitelist of allowed domains
- Conduct regular security testing and code reviews to identify potential SSRF vulnerabilities
Success criteria include:
- No SSRF vulnerabilities detected in security scans and code reviews
- All axios requests are validated and sanitized
Validation methods include:
- Regular security scans and code reviews
- Penetration testing and vulnerability assessments

## REMEDIATION PLAN
To remediate the SSRF vulnerability, the following technical actions are required:
- Update axios to the latest version
- Implement input validation and sanitization for all user-input data used in axios requests
- Configure web filtering and proxy servers to restrict access to external resources
Responsible party: Security Lead
Timeline: 1 week
Verification steps:
- Code review to ensure input validation and sanitization are implemented correctly
- Re-scan the application to verify the SSRF vulnerability is resolved

## MONITORING AND DETECTION
To detect similar vulnerabilities in the future, the following monitoring and detection strategies will be implemented:

- Regular security scans and code reviews to identify potential SSRF vulnerabilities
- Logging and alerting requirements:
+ Log all axios requests and responses
+ Alert on suspicious or anomalous traffic patterns
- Continuous monitoring strategies:
+ Regularly review and update web filtering and proxy server configurations
+ Conduct penetration testing and vulnerability assessments to identify potential SSRF vulnerabilities

# Policy #9: SCA

**Title:** jsonwebtoken - CWE-327
**Severity:** MEDIUM
**LLM:** LLaMA 3.3 70B

## POLICY IDENTIFIER
SP-2024-001: Secure JSON Web Token Configuration Policy

## RISK STATEMENT
The use of insecure default key sizes in JSON Web Tokens (JWT) poses a significant risk to our organization's data security. If exploited, an attacker could potentially decipher sensitive information, compromising user authentication and data integrity. This vulnerability affects all systems and users that rely on JWT for authentication, potentially leading to unauthorized access, data breaches, and reputational damage.

## COMPLIANCE MAPPING
This policy aligns with the following compliance requirements:
- NIST CSF: PR.DS-5 (Data Protection), PR.IP-1 (Identity and Access Control)
- ISO 27001: A.8.24 (Use of Cryptography), A.8.26 (Application Security Requirements), A.8.27 (Secure System Architecture and Engineering Principles)
- Industry Standards: OWASP A03:2021 (Injection), CWE-327 (Use of a Broken or Risky Cryptographic Algorithm)

## POLICY REQUIREMENTS
To mitigate this risk, the following security controls must be implemented:
- Use a secure key size (minimum 2048 bits) for all JSON Web Tokens.
- Implement a key rotation policy to ensure regular updates of cryptographic keys.
- Utilize a secure random number generator for key generation.
Success criteria include:
- Verification of secure key sizes through code review and cryptographic analysis.
- Validation of key rotation policies through regular security audits.

## REMEDIATION PLAN
To remediate this vulnerability, the following technical actions are required:
- Update all JSON Web Token configurations to use secure key sizes.
- Implement a key rotation policy and schedule regular key updates.
The responsible party for this remediation is the Development Lead. The timeline for completion is 2 weeks. Verification steps include:
- Code review to ensure secure key sizes are used.
- Cryptographic analysis to validate key strength.
- Penetration testing to ensure the vulnerability is fully remediated.

## MONITORING AND DETECTION
To detect similar vulnerabilities in the future, the following measures will be implemented:
- Regular code reviews to ensure secure coding practices.
- Automated scanning tools to identify potential vulnerabilities.
- Continuous monitoring of system logs to detect suspicious activity.
Logging and alerting requirements include:
- Logging all authentication attempts and errors.
- Alerting the security team of potential security incidents.
By following this policy, we can ensure the secure use of JSON Web Tokens and protect our

organization's data from potential security threats.

# Policy #10: SCA

**Title:** minimist - CWE-1321
**Severity:** CRITICAL
**LLM:** LLaMA 3.3 70B

## POLICY IDENTIFIER
SP-2024-001: Minimist Prototype Pollution Protection Policy

## RISK STATEMENT
The minimist prototype pollution vulnerability poses a significant risk to our organization's intellectual property and sensitive data. If exploited, this vulnerability could allow unauthorized access to our source code, development tools, and software libraries, potentially leading to theft of proprietary information, disruption of business operations, and damage to our reputation. This vulnerability affects our web application development environments, including production, testing, and development systems, and could impact all users and data associated with these environments.

## COMPLIANCE MAPPING
This policy aligns with the following compliance requirements:
- NIST CSF: PR.DS-5 (Data Protection), DE.CM-8 (Vulnerability Management)
- ISO 27001: A.5.32 (Intellectual Property Rights), A.8.23 (Web Filtering), A.8.30 (Outsourced Development), A.8.31 (Separation of Development, Test and Production Environments), A.8.4 (Access to Source Code)
- Industry standards: OWASP A03:2021 (Injection), CWE-1321 (Prototype Pollution)

## POLICY REQUIREMENTS
To mitigate the minimist prototype pollution vulnerability, the following security controls must be implemented:
- Validate and sanitize all user input to prevent malicious data from entering our systems.
- Implement web filtering to restrict access to external websites and reduce exposure to malicious content.
- Enforce separation of development, testing, and production environments to prevent unauthorized access to sensitive data.
- Limit read and write access to source code, development tools, and software libraries to authorized personnel only.
- Conduct regular code reviews and vulnerability assessments to identify and remediate similar vulnerabilities.

## REMEDIATION PLAN
To remediate the minimist prototype pollution vulnerability, the following technical actions are required:
- Update the minimist library to the latest version.
- Implement input validation and sanitization for all user input.
- Conduct a thorough code review to identify and remediate similar vulnerabilities.
The CTO is responsible for overseeing the remediation efforts, which must be completed within 24-48 hours. Verification steps include code review, re-scanning for vulnerabilities, and penetration testing.

## MONITORING AND DETECTION
To detect similar vulnerabilities in the future, we will implement the following monitoring and detection strategies:
- Regular code reviews and vulnerability assessments.
- Logging and alerting for suspicious activity, including unauthorized access attempts and unusual user behavior.

- Continuous monitoring of our web application development environments for signs of vulnerability exploitation.
- Regular updates to our web filtering and access control lists to ensure that our systems remain protected from emerging threats.

# Policy #11: DAST

**Title:** SQL Injection
**Severity:** MEDIUM
**LLM:** LLaMA 3.1 8B Instant

**Security Policy Section: SP-2023-001 - SQL Injection Prevention**

**Policy Identifier:** SP-2023-001

**Policy Title:** Prevention of SQL Injection Attacks

**Risk Statement:**

The potential exploitation of the identified SQL injection vulnerability poses a medium-level risk to the organization. If exploited, this vulnerability could lead to unauthorized access to sensitive user data, compromising the confidentiality, integrity, and availability of our systems. This could result in financial losses, reputational damage, and non-compliance with regulatory requirements. The affected systems, data, and users include the web application's user database, API endpoints, and all users who interact with the application.

**Compliance Mapping:**

* NIST Cybersecurity Framework (CSF):
+ PR.DS-5: Implement secure coding practices to prevent common web application vulnerabilities.
+ PR.DS-6: Implement input validation and sanitization to prevent common web application vulnerabilities.
* ISO 27001 Annex A controls:
+ A.8.8: Management of technical vulnerabilities
+ A.8.16: Monitoring activities
+ A.8.23: Web filtering
+ A.5.33: Protection of records
+ A.6.1: Screening
* Industry standards:
+ OWASP: Preventing SQL Injection
+ CWE: CWE-89: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

**Policy Requirements:**

To prevent SQL injection attacks, the following security controls must be implemented:

1. **Input Validation and Sanitization**: All user input must be validated and sanitized on the server-side to prevent malicious SQL code from being injected.
2. **Use of Prepared Statements**: Prepared statements with parameterized queries must be used instead of string concatenation to build SQL queries.
3. **Code Review**: Regular code reviews must be conducted to ensure that all code changes are secure and free from SQL injection vulnerabilities.
4. **Logging and Alerting**: All SQL injection attempts must be logged and alerted to the security team for further investigation.

**Success Criteria:**

* All user input is validated and sanitized on the server-side.
* Prepared statements with parameterized queries are used for all SQL queries.
* Code reviews are conducted regularly to ensure secure coding practices.
* All SQL injection attempts are logged and alerted to the security team.

**Remediation Plan:**

* **Responsible Party:** Dev Lead
* **Timeline:** 2 weeks
* **Technical Actions:**
1. Implement input validation and sanitization on the server-side.
2. Update all SQL queries to use prepared statements with parameterized queries.
3. Conduct regular code reviews to ensure secure coding practices.
* **Verification Steps:**
1. Code review to ensure secure coding practices.
2. Re-scan the application for SQL injection vulnerabilities.
3. Conduct penetration testing to ensure the application is secure.

**Monitoring and Detection:**

* **Detection:** Regularly scan the application for SQL injection vulnerabilities using automated tools.
* **Logging and Alerting:** Log all SQL injection attempts and alert the security team for further investigation.
* **Continuous Monitoring:** Continuously monitor the application for signs of SQL injection attacks and adjust the security controls as necessary.

# Policy #12: DAST

**Title:** Cross Site Scripting (Reflected)
**Severity:** MEDIUM
**LLM:** LLaMA 3.1 8B Instant

**SP-2023-001: Cross-Site Scripting (XSS) Prevention Policy**

**Risk Statement:**

This policy aims to mitigate the risk of Cross-Site Scripting (XSS) attacks, which can compromise user data and system integrity. XSS attacks occur when malicious code is injected into a web application, allowing attackers to steal sensitive information or take control of user sessions. If exploited, XSS vulnerabilities can lead to unauthorized access, data breaches, and reputational damage. The affected systems and data include web applications, user accounts, and sensitive information stored within the application.

**Compliance Mapping:**

- NIST Cybersecurity Framework (CSF):
- PR.DS-5: Implement secure coding practices to prevent vulnerabilities.
- PR.DS-6: Use secure development lifecycles to identify and address vulnerabilities.
- ISO 27001 Annex A controls:
- A.8.23: Web filtering to manage access to external websites.
- A.8.28: Secure coding principles to prevent vulnerabilities.
- A.8.29: Security testing in development and acceptance to identify vulnerabilities.
- A.8.8: Management of technical vulnerabilities to obtain and evaluate vulnerability information.
- Industry standards:
- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting').
- OWASP: Cross-Site Scripting (XSS) Prevention Cheat Sheet.

**Policy Requirements:**

1. **Input Validation and Sanitization:** All user input must be validated, filtered, and sanitized to prevent malicious code injection.
2. **Use of Vetted Libraries and Frameworks:** Only vetted libraries and frameworks that prevent XSS vulnerabilities will be used for web application development.
3. **Secure Coding Practices:** Secure coding principles must be applied to prevent vulnerabilities, including the use of HTML entities to convert special characters.
4. **Security Testing:** Regular security testing must be performed during the development lifecycle to identify and address vulnerabilities.

**Success Criteria:**

- All web applications will be scanned for XSS vulnerabilities on a quarterly basis.
- No XSS vulnerabilities will be detected in the next 6 months.
- All developers will receive training on secure coding practices and input validation within the next 3 months.

**Remediation Plan:**

- **Responsible Party:** Dev Lead

- **Timeline:** 2 weeks
- **Technical Actions:**
1. Review and update all web applications to use vetted libraries and frameworks.
2. Implement input validation and sanitization mechanisms.
3. Conduct security testing to identify and address vulnerabilities.
- **Verification Steps:**
1. Code review to ensure secure coding practices are applied.
2. Re-scan web applications for XSS vulnerabilities after remediation.

**Monitoring and Detection:**

- Regular security testing and vulnerability scanning will be performed to detect similar vulnerabilities in the future.
- Logging and alerting requirements will be implemented to detect and respond to XSS attacks.
- Continuous monitoring strategies will be implemented to ensure the effectiveness of this policy.

# Policy #13: DAST

**Title:** Absence of Anti-CSRF Tokens
**Severity:** MEDIUM
**LLM:** LLaMA 3.1 8B Instant

**SP-2023-001: Protection Against Cross-Site Request Forgery (CSRF)**

**Risk Statement**

This policy aims to mitigate the risk of Cross-Site Request Forgery (CSRF) attacks, which can compromise the security of our web application and lead to unauthorized actions. If exploited, a CSRF attack can result in data tampering, unauthorized account modifications, or even financial losses. The affected systems, data, and users include our web application, customer accounts, and sensitive business information.

**Compliance Mapping**

* NIST Cybersecurity Framework (CSF):
+ PR.DS-5: Implement secure coding practices
+ PR.DS-6: Implement secure software development practices
* ISO 27001 Annex A controls:
+ A.8.23: Web filtering
+ A.8.8: Management of technical vulnerabilities
+ A.5.14: Information transfer
+ A.6.4: Disciplinary process
* Industry standards:
+ CWE-352: Cross-Site Request Forgery (CSRF)
+ OWASP: Cross-Site Request Forgery (CSRF)

**Policy Requirements**

To protect against CSRF attacks, the following security controls must be implemented:

1. **Technical Requirements**:
* Use a vetted library or framework that provides anti-CSRF tokens, such as OWASP CSRFGuard.
* Ensure that the application is free of cross-site scripting (XSS) issues, as most CSRF defenses can be bypassed using attacker-controlled script.
2. **Procedural Requirements**:
* Conduct regular code reviews to identify and address potential CSRF vulnerabilities.
* Implement secure software development practices, including secure coding guidelines and secure coding training for developers.
3. **Success Criteria and Validation Methods**:
* Conduct regular penetration testing and vulnerability scanning to identify potential CSRF vulnerabilities.
* Review code changes and implement secure coding practices to prevent CSRF attacks.

**Remediation Plan**

To address this vulnerability, the following technical actions must be taken:

1. **Responsible Party**: Development Lead

2. **Timeline**: 2 weeks
3. **Technical Actions**:
* Implement OWASP CSRFGuard in the web application.
* Conduct a code review to identify and address potential CSRF vulnerabilities.
* Update secure coding guidelines and provide secure coding training for developers.
4. **Verification Steps**:
* Conduct a penetration test to verify the effectiveness of the implemented controls.
* Review code changes and implement secure coding practices to prevent CSRF attacks.

**Monitoring and Detection**

To detect similar vulnerabilities in the future, the following monitoring and detection strategies must be implemented:

1. **Logging and Alerting Requirements**:
* Configure logging to capture potential CSRF attacks.
* Set up alerts to notify security teams of potential CSRF attacks.
2. **Continuous Monitoring Strategies**:
* Conduct regular penetration testing and vulnerability scanning to identify potential CSRF vulnerabilities.
* Review code changes and implement secure coding practices to prevent CSRF attacks.

By implementing these security controls and monitoring strategies, we can effectively mitigate the risk of CSRF attacks and protect our web application, customer accounts, and sensitive business information.

# Policy #14: DAST

**Title:** Weak Authentication Method
**Severity:** MEDIUM
**LLM:** LLaMA 3.1 8B Instant

**Security Policy Section: SP-2023-001 - Secure Authentication over Unsecured Connections**

**Policy Identifier:** SP-2023-001

**Policy Title:** Secure Authentication over Unsecured Connections

**Risk Statement:**

The use of HTTP basic or digest authentication over an unsecured connection poses a medium-severity risk to the organization. If exploited, an attacker can intercept and reuse user credentials, compromising the confidentiality and integrity of sensitive information. This vulnerability affects all users accessing the application via the affected endpoint, potentially leading to unauthorized access to sensitive data and systems.

**Compliance Mapping:**

* NIST Cybersecurity Framework (CSF):
+ PR.DS-5: Protect the confidentiality, integrity, and availability of information
+ ID.RA-5: Implement secure authentication and session management
* ISO 27001 Annex A controls:
+ A.8.5: Secure authentication
+ A.7.4: Physical security monitoring
+ A.8.21: Security of network services
+ A.8.1: User endpoint devices
+ A.7.14: Secure disposal or re-use of equipment
* Industry standards:
+ CWE-287: Improper Authentication
+ OWASP Top 10: Authentication and Authorization

**Policy Requirements:**

To mitigate this vulnerability, the following security controls must be implemented:

1. **Technical Requirements:**
* Protect the connection using HTTPS (TLS 1.2 or later) for all authentication requests.
* Use a stronger authentication mechanism, such as OAuth or JWT, for all API requests.
2. **Procedural Requirements:**
* Conduct regular code reviews to ensure secure authentication practices.
* Implement logging and alerting mechanisms to detect potential authentication-related issues.
3. **Success Criteria:**
* All authentication requests are encrypted using HTTPS.
* The application uses a stronger authentication mechanism for all API requests.
* Regular code reviews and logging/alerting mechanisms are in place.

**Remediation Plan:**

* **Responsible Party:** Dev Lead (Medium severity)
* **Timeline:** 2 weeks
* **Technical Actions:**
+ Update the application to use HTTPS for all authentication requests.
+ Implement a stronger authentication mechanism for all API requests.
* **Verification Steps:**
+ Conduct a code review to ensure secure authentication practices.
+ Perform a re-scan of the application using a vulnerability scanner to verify the fix.

**Monitoring and Detection:**

To detect similar vulnerabilities in the future:

1. **Logging and Alerting:** Implement logging and alerting mechanisms to detect potential authentication-related issues.
2. **Continuous Monitoring:** Conduct regular security assessments and penetration testing to identify vulnerabilities.
3. **Code Reviews:** Conduct regular code reviews to ensure secure authentication practices.

By implementing these security controls and procedures, we can mitigate the risk of authentication-related vulnerabilities and ensure the confidentiality, integrity, and availability of sensitive information.

# Policy #15: DAST

**Title:** Cookie Without Secure Flag
**Severity:** MEDIUM
**LLM:** LLaMA 3.1 8B Instant

**SP-2023-001: Secure Cookie Flag Policy**

**Policy Title:** Secure Cookie Flag Implementation

**Risk Statement:**

This policy addresses the risk of sensitive information exposure due to the insecure transmission of cookies. The absence of the secure flag in cookies allows unauthorized parties to access sensitive data, potentially leading to identity theft, unauthorized access, or data breaches. If exploited, this vulnerability can compromise the confidentiality, integrity, and availability of sensitive information stored in cookies. Affected systems, data, and users include all web applications and users who interact with them.

**Compliance Mapping:**

- NIST CSF: PR.DS-5 (Protect Data in Transit)
- ISO 27001: A.8.5 (Secure authentication), A.8.21 (Security of network services), A.8.27 (Secure system architecture and engineering principles), A.8.28 (Secure coding), A.8.1 (User endpoint devices)
- Industry Standards: CWE-614 (Information Exposure Through Browser Side Scripting), OWASP (Secure Cookies)

**Policy Requirements:**

To mitigate the risk of sensitive information exposure, the following security controls must be implemented:

1. **Technical Requirements:**
- Ensure that all cookies containing sensitive information or session tokens are transmitted over an encrypted channel (HTTPS).
- Set the secure flag for cookies containing sensitive information.
- Implement a secure cookie flag validation mechanism to ensure compliance.

2. **Procedural Requirements:**
- Develop and maintain a secure coding standard that includes guidelines for secure cookie flag implementation.
- Conduct regular code reviews to ensure compliance with the secure coding standard.
- Update the secure cookie flag implementation in all web applications.

**Success Criteria and Validation Methods:**

- Success criteria: All cookies containing sensitive information or session tokens are transmitted over an encrypted channel (HTTPS) and have the secure flag set.
- Validation methods: Regular code reviews, penetration testing, and vulnerability scanning.

**Remediation Plan:**

- **Responsible Party:** Dev Lead
- **Timeline:** 2 weeks
- **Technical Actions:**
- Update the secure cookie flag implementation in all web applications.
- Implement a secure cookie flag validation mechanism.
- **Verification Steps:**
- Conduct a code review to ensure compliance with the secure coding standard.
- Perform a penetration test to verify the secure cookie flag implementation.

**Monitoring and Detection:**

- **Detection Method:** Regular vulnerability scanning and penetration testing.
- **Logging and Alerting Requirements:** Configure logging and alerting mechanisms to detect similar vulnerabilities in the future.
- **Continuous Monitoring Strategies:** Implement a continuous monitoring program to detect and respond to security incidents in a timely manner.