Pacific Northwest
NATIONAL LABORATORY

PNNL-XXXXX

# FIC Vulnerability Profile

Provided by Shamrock Cyber

May 2022

Ryan Bays
Josh Bigler
Angie Chastain
Paul Francik
Catie Himes
Emma Lancaster
Danielle Nodine
Patrick O'Connell
Aaron Phillips
Shawn Ricketts
Garret Seppala
Torri Simmons
Chance Younkin

PNNL-32881

U.S. DEPARTMENT OF ENERGY

Prepared for the U.S. Department of Energy
under Contract DE-AC05-76RL01830

## DISCLAIMER



PACIFIC NORTHWEST NATIONAL LABORATORY
*operated by*
BATTELLE
*for the*
UNITED STATES DEPARTMENT OF ENERGY
*under Contract DE-AC05-76RL01830*

**Printed in the United States of America**

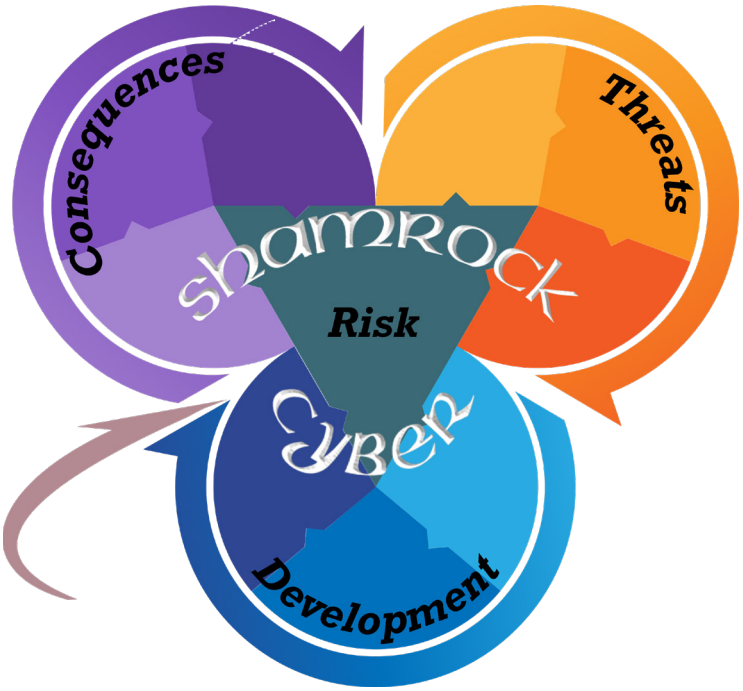
**Available to DOE and DOE contractors from the**
**Office of Scientific and Technical Information,**
**P.O. Box 62, Oak Ridge, TN 37831-0062;**
**ph: (865) 576-8401**
**fax: (865) 576-5728**
**email: reports@adonis.osti.gov**

**Available to the public from the National Technical Information Service**
**5301 Shawnee Rd., Alexandria, VA 22312**
**ph: (800) 553-NTIS (6847)**
**email: orders@ntis.gov <https://www.ntis.gov/about>**
**Online ordering: http://www.ntis.gov**

# FIC Vulnerability Profile

May 2022

Ryan Bays
Josh Bigler
Angie Chastain
Paul Francik
Catie Himes
Emma Lancaster
Danielle Nodine
Patrick O'Connell
Aaron Phillips
Shawn Ricketts
Garret Seppala
Torri Simmons
Chance Younkin




Prepared for
the U.S. Department of Energy
under Contract
DE-AC05-76RL01830

Pacific Northwest National Laboratory
Richland, Washington 99354

# Contents

Contents ..... ii
Acronyms and Abbreviations, and Terms of Reference ..... iii
1.0 Introduction ..... 1
1.1 Purpose of a Vulnerability Profile ..... 1
1.2 Shamrock Cyber Analysis ..... 1
2.0 Static Analysis Security Testing (SAST) Profile ..... 2
3.0 Conclusion ..... 7
Appendix A Brief on Consequence-Based Analysis ..... A.1
Appendix B Brief on Threat-Based Analysis ..... B.1
Appendix C Brief on Security-Based Development ..... C.1
Appendix D Full Checkmarx Scan Results ..... D.1

# Figures

Figure 1. Shamrock Cyber services. ..... 1
Figure 2. The CBA leaf of Shamrock Cyber. ..... A.1
Figure 3. The TBA leaf of Shamrock Cyber. ..... B.1
Figure 4. Lockheed Martin's methodology. ..... B.1
Figure 5. The CIA triad. ..... B.1
Figure 6. The SBD leaf of Shamrock Cyber. ..... C.1

# Tables

Table 1. Static Analysis Security Testing Results ..... 2

# Acronyms and Abbreviations, and Terms of Reference

| | |
|---|---|
| CIA | Confidentiality, Integrity, Availability |
| CSRF | Cross-Site Request Forgery |
| IDDIL-ATC | Identify Assets, Define the Attack Surface, Decompose the System, Identify Attack Vectors, List the Threat Actors, Analysis & Assessment, Triage, Controls |
| OSA | Open-Source Analysis |
| PNNL | Pacific Northwest National Laboratory |
| SAST | Static Analysis Security Testing |
| STRIDE | Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege |
| TBA | Threat-Based Analysis |
| TMT | Threat Modeling Tool |

# 1.0 Introduction

The FIC team is engaged with Pacific Northwest National Laboratory's (PNNL's) ***Shamrock Cyber*** Team to provide cybersecurity analyses of the FIC software. Shamrock offers both Threat-Based Analysis services and Secure Software Development services, as defined in Figure 1. These services are ultimately used to understand and mitigate threats against software and to reduce vulnerabilities in software, thus improving overall cybersecurity and informing decision makers. Shamrock's Secure Software Development services, specifically Static Analysis Security Testing (SAST) and Open-Source Analysis (OSA), produced this Vulnerability Profile.

***Consequence Based Analysis*** – analyzes the abuse, misuse, and hazards that determine risks of developing and deploying a system. The result is a dossier outlining the consequence-based analysis.
***Threat Based Software Analysis*** – determines and prioritizes threats against the system and recommends mitigations. The result is a Threat Profile that contains a threat model, threat findings, and mitigations.
***Security Based Development*** – applies security best practices to the system development life cycle. This includes secure design, secure implementation, and security testing.

Figure 1. Shamrock Cyber services.

## 1.1 Purpose of a Vulnerability Profile

The purpose of this Vulnerability Profile is to provide concise, clear actions that the FIC team can take to reduce vulnerabilities in the application code itself. Vulnerability Profiles are based on automated vulnerability scans, which can be performed at the programming stage as well as the testing stage of the software development life cycle. They are designed to be completed regularly during software development, with the intent of eliminating vulnerabilities before deploying in a production environment.

## 1.2 Shamrock Cyber Analysis

The Vulnerability Profile is based on a commercial off-the-shelf vulnerability scanner called Checkmarx,[1] which scans application source code for security vulnerabilities. It was adopted by PNNL, and Shamrock Cyber uses it to perform SAST scans. Checkmarx also provides OSA, which scans dependencies and third-party libraries used by the source code against a knowledge base. Any libraries found in the knowledge base that are outdated or vulnerable are listed in the results. Shamrock analyzes both SAST and OSA scan results to:

- Determine if a vulnerability is valid, possibly valid, or invalid
- Provide justification for this determination
- Recommend possible fixes to the vulnerability
- Make additional suggestions not provided by Checkmarx, such as refactoring duplicate code, implementing best practices, checking authorization access, formatting, etc.

The Shamrock Cyber analysis is an informal code review across sections of the code near vulnerabilities highlighted by Checkmarx. These feedback points and suggestions are performed by a software engineer experienced in the source code language and are not part of the Checkmarx scan. This provides additional code quality reviews that provide suggestions on best practices to save time and effort for the FIC team.

[1] https://www.checkmarx.com

## 2.0 Static Analysis Security Testing (SAST) Profile

The details for all vulnerabilities found in the SAST scan, as well as Shamrock recommendations, are outlined in the SAST Profile Table. This table is a concise list of vulnerabilities found in the Checkmarx scan with the details necessary to find and fix vulnerable code.

Table 1. Static Analysis Security Testing Results

| # | Vulnerability Type (Checkmarx) | Vulnerability | Location | Mitigation | Explanation |
|---|---|---|---|---|---|
| **HIGH** | | | | | |
| **1** | SQL_Injection | The application's **insertInstallationPermit** method executes an SQL query with **insertStatement**, at line **26** of **app\server\postgresQueries.js**. The application constructs this SQL query by embedding an untrusted string into the query without proper sanitization. The concatenated string is submitted to the database, where it is parsed and executed accordingly. | Source Line: 48 of \app \server.js<br>Destination Line: 26 \app\server\postgresQueries.js | Sanitize input. Consider using query-builder library. | |

| MEDIUM | | | | |
|---|---|---|---|---|
| **2** | Missing_HSTS_ Header | Web servers without the Strict-Transport-Security header expose clients to man-in-the-middle attacks by forcing an initial connection over HTTP before being redirected to use HTTPS. Strict-Transport-Security ensures the connection will always happen over HTTPS. | Source Line: 35 of \app\server.js<br>Destination Line: 35 of \app\server.js | Consider adding Strict-Transport-Security header in server configuratio n. Validate that browser receives Strict-Transport-Security header in response from server. |
| **3** | Use_Of_Hardco ded_Password | The application uses a hard-coded password, allowing database access to anyone with access to the source code. | Source Line: 7 \app\server\postgresQueries.js<br>Destination Line: 7 \app\server\postgresQueries.js | Move the password out of source code. Consider using secret manageme nt library or similar method for managing passwords. |

| | | | | | |
|---|---|---|---|---|---|
| **4** | Missing_CSP_Header | Web servers without the Strict-Transport-Security header are vulnerable to attacks such as Cross-Site Scripting. | Source Line: 35 \app \server.js<br>Destination Line: 35 \app \server.js | Consider adding Strict-Transport-Security header in server configuration. Validate that browser receives Strict-Transport-Security header in response from server. | |
| **LOW** | | | | | |
| **5** | Client_Insecure_Randomness | Javascript's Math.random() method is not cryptographically secure | Source Line: 68 of \app\src\assets\DummyData.js<br>Destination Line: 116-119 \app\src\assets\DummyData.js | Consider replacing Math.random() with a cryptographically strong random number generator | This is likely low-impact, as it is only being used to generate test data in the place of actual user data. |
| **6** | Potential_Clickjacking_on_Legacy_Browsers | HTML does not protect against clickjacking attacks, which could result in a user clicking a malicious link unintentionally. | Source Line: 1 \app\dist\index.html<br>Destination Line: 1 \app\dist\index.html | Add a framebusting script (see https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html#best-for-now-legacy- | |

| | | | | |
|---|---|---|---|---|
| | | | | browser-frame-breaking-script) |
| **7** | Client_Hardcoded_Domain | A resource is being loaded from a remote domain, allowing an attacker to replace its contents. | Source Line: 4 \app\dist/index.html<br>Destination Line: 4 \app\dist\index.html | Consider serving resources from the web server, rather than directing clients to external URLs. |
| **8** | Potentially_Vulnerable_To_Csrf | This parameter value flows through the code and is eventually used to access application state altering functionality. This may enable Cross-Site Request Forgery (CSRF) | Source Line: 11 \app \server.js<br>Destination Line: 11 \app \server.j | Consider adding a CSRF token https://levelup.gitconnected.com/how-to-implement-csrf-tokens-in-express-f867c9e95af0 |
| **9** | React_Deprecated | The ReactDOM.render() is deprecated | Source Line: 59 \app\src\index.jsx<br>Destination Line: 59 \app\src\index.jsx | Consider replacing with updated React rendering methods. |
| **10** | React_Deprecated | The ReactDOM.render() is deprecated | Source Line: 28 \app\src\components\Header.jsx<br>Destination Line: 28 \app\src\components\Header.jsx | Consider replacing with updated React rendering methods. |

| INFO | | | | | |
|---|---|---|---|---|---|
| **11** | Log_Forging | Method app.post at line 48 of app\server.js gets user input from element body. This element's value flows through the code without being properly sanitized or validated and is eventually used in writing an audit log in, .then at line 49 of app\server.js. This may enable Log Forging. | Source Line: 48 of \app\server.js<br>Destination Line: 49,50 \app\server.js | Either sanitize user input before writing to log or avoid writing user data to log. | Likely false positive, as data is being written to browser console log, not a log file (and therefore will not be able to corrupt log file that will be processed later) |

# 3.0 Conclusion

This Vulnerability Profile is delivered with the intention of highlighting areas of concern based off code scans of the FIC software that will be incorporated into the final system to ensure the confidentiality, integrity, and availability of flagpole installation data for both employees and the public. The order of importance in addressing vulnerabilities and their impact on the provided code was based on the CIA prioritization of (1) Confidentiality, (2) Integrity, and (3) Availability. After analyzing the codebase through an automated process (Checkmarx) and manual review, a total of 1 high, 3 medium, 6 low, and 1 informative vulnerabilities were found to be of potential consequence.

The vulnerability of most concern would be Vulnerability 1: SQL Injection, which involves embedding an untrusted string into a query without proper sanitization. If the user input is not sanitized and validated, this vulnerability could allow an attacker or user to exploit the database and read files for which they are not authorized or modify the database. This would affect confidentiality. This vulnerability can be remediated by validating/sanitizing all user input and using whitelisting for valid input.

This Vulnerability Profile provides a set of vulnerability remediations that can be applied to the application code of the FIC team immediately. By doing so, the cost of finding and fixing vulnerabilities in production is drastically reduced. The contents of this Vulnerability Profile will inform software developers and software testers before deployment, thus reducing the cost of finding and fixing vulnerabilities in production.

This effort leads to more secure software and better-understood security, and the FIC team is to be commended for their rigorous approach to employing cybersecurity in their products.

# Appendix A Brief on Consequence-Based Analysis

The Shamrock Cyber Team uses Consequence-Based Analysis (CBA) to assess risk to mission or business operations. Figure 1, shows the three categories of CBA, each of which is composed of three elements: a system function, a negative outcome, and a technical capability that, through the system function, enables the negative outcome. The elements, when present and combined in a system have the potential to impart harm to some part of the system, its operation, its mission, or its stakeholders. The negative outcome is a plausible consequence of something going wrong with the system or its operation. The technical element is the link that could transform normal operations into the identified negative outcome. Each of these cases is constructed as follows:

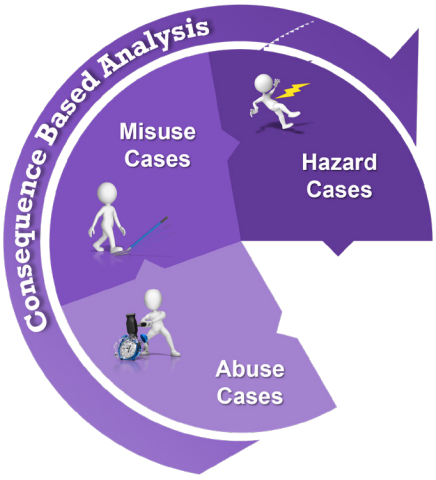



Figure 2. The CBA leaf of Shamrock Cyber.

***Abuse Case*** – damage caused by intentional acts of an adversary

- Adversaries and their Motives (A&M) – who wants to do damage and why
- Functional Use Element (FUE) -- what the system does
- Functional Abuse Element (FAE) – the harmful outcome
- Technical Abuse Element (TAE) – how the system can be "hacked" intentionally

***Misuse Case*** – damage caused by unintentional acts and human error

- Mistakes and Misbehavior (M&M) – foreseeable user mistakes or misuse
- Functional Use Element (FUE) – what the system does
- Functional Misuse Element (FME) – the harmful outcome
- Technical Misuse Element (TME) – how the system errors when misused

***Hazard Case*** – damage caused by non-human events in the system's operating environment

- Environmental Events (EE) – something that occurs naturally in the environment
- Functional Use Element (FUE) – what the system does
- Functional Hazard Element (FHE) – harmful outcome)
- Technical Hazard Element (THE) – how the system could malfunction due to a hazard

The Shamrock Cyber team engages with customers (owners, operators, and other stakeholders) to understand system operations, use cases, and missions. The team also gathers stakeholders' unacceptable mission outcomes and conditions. From this, plausible scenarios are derived that could lead to unacceptable consequences. Assessments for threats and vulnerabilities are then either gathered or performed and used to build the various "cases." The Shamrock Cyber team engages customers as needed throughout the process.

When analysis is complete, narratives such as ***Adversary Dossiers*** are developed to explain in simple, non-technical terms, the risks and consequences those risks can have on stakeholder equities. This allows greater stakeholder access to risk assessment and management processes and discussions. At the same time, each case is directly linked to one or more technical elements which directs system security and defense personnel in the identification, design, and implementation of security controls, vulnerability remediations, or risk mitigations

# Appendix B Brief on Threat-Based Analysis

The Shamrock Cyber team combines three stages of Threat-Based Analysis (TBA), as shown in Figure 2. TBA utilizes portions of Lockheed Martin's IDDIL-ATC methodology (Figure 3) to perform threat analysis. Shamrock optimizes IDDIL-ATC for more cost-effective, time-efficient results that lead to immediately actionable controls. Using the Lockheed Martin nomenclature, Shamrock actually begins with ***Decompose the System***. To accomplish this, Shamrock often requests that ***Usage Narratives*** be written by members of the project team. The narratives provide the Shamrock team with valuable context in simple, non-jargon terms. With this context, the next step is to develop a set of use cases and data flow diagrams that represent the system. Generally, the assets and the attack surface can be identified using these diagrams, thus addressing the ***Identify Assets*** and ***Define the Attack Surface*** steps. From there, Shamrock attempts to ***List Threat Actors***, but this is not yet a rigorous exercise. The use cases, abuse cases, and data flow diagrams represent the ***Shamrock Cyber Threat Model***, which is the foundation for developing the Threat Profile.

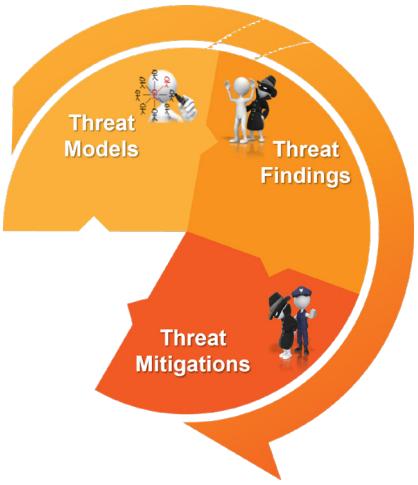



Figure 3. The TBA leaf of Shamrock Cyber.

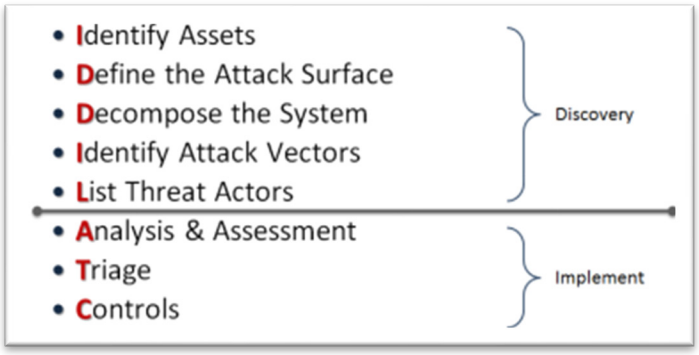



Figure 4. Lockheed Martin's methodology.

Shamrock asks the project team to set an initial expectation of threat priority based on Confidentiality, Integrity, and Availability (CIA). The CIA Triad (see Figure 4) is a commonly used cybersecurity model.

The Shamrock Cyber team uses the data flow diagrams as input to Microsoft's Threat Modeling Tool (TMT). The TMT is a free download that comes with standard threat templates used by Shamrock. The TMT reads the diagrams and uses the templates to provide initial ***Analysis and Assessment*** as well as ***Triage*** results. The TMT also uses Microsoft's STRIDE model to categorize threats. The initial results from the TMT are then analyzed by Shamrock subject matter experts to complete the ***Shamrock Cyber Threat Findings*** for review by the project team.


Figure 5. The CIA triad.

With the Threat Findings in hand, Shamrock goes back to the project team to collaboratively analyze and determine mitigations (***Controls***). When this exercise is complete, the Shamrock Cyber team organizes the information into the final product, the ***Shamrock Cyber Threat Profile***.

# Appendix C Brief on Security-Based Development

The Shamrock Cyber Team is establishing Security-Based Development (SBD) best practices in the areas depicted in Figure 5. While Shamrock will at some point offer ***Secure Design*** and ***Security Test*** services, the current focus is on ***Secure Implementation***. For Shamrock, secure implementation of software combines Static Application Security Testing (SAST) and Open-Source Analysis (OSA). SAST involves scanning source code to identify known vulnerabilities, while OSA entails scanning 3rd party software developed outside the project team. The objective of Shamrock cyber secure implementation is to use a SAST scan and possibly an OSA scan to perform an analysis that eliminates false positives, summarizes the vulnerabilities, and makes recommendations. The result of this analysis enables the software development team to prioritize vulnerabilities and address them in order of priority.

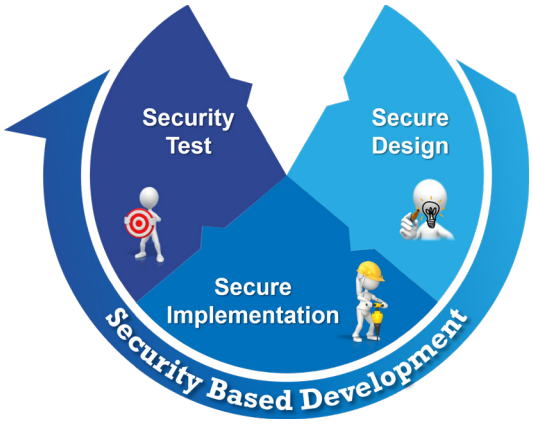



Figure 6. The SBD leaf of Shamrock Cyber.

Shamrock Cyber makes use of Checkmarx, a commercial software scanning tool adopted by PNNL, that performs both SAST and OSA scanning. The Shamrock process is a straightforward set of steps:

1. ***Receive source code***

   The source code comes from the customer development team in the form of a zip file or a URL to a code repository The source code will be used as input to the Checkmarx scanner.

2. ***Execute a Checkmarx SAST scan***

   Every file contained in software (from the repo or the zip file) will be scanned and the results form the foundation for Shamrock Cyber analysis.

3. ***Execute a Checkmarx OSA scan***

   Dependency libraries will be scanned by Checkmarx, and vulnerable libraries along with out-of-date libraries will be documented, forming the foundation for Shamrock analysis.

4. ***Analyze SAST scan results***

   The results of Shamrock analysis of SAST go into the final report.

5. ***Analyze OSA scan results***

   The results of Shamrock analysis of OSA go into the final report

When this process is complete, the Shamrock Cyber team organizes the information into the final product, the ***Shamrock Cyber Vulnerability Profile***.

# Appendix D Full Checkmarx Scan Results

The Vulnerability Profile is derived from a source code scan by the Checkmarx SAST tool. The full, unaltered scan produced by the Checkmarx scanner is provided in this appendix. The scan results are comprehensive and include details from several standards such as OWASP, NIST, and FISMA. Details in the scan results can be useful to further understand the vulnerabilities and to gain insight into the details of the scan itself. However, the Vulnerability Profile contains all “action items” for fixing vulnerabilities. And the scan results are provided as extra information.

# FIC Scan Report

| | |
|---|---|
| Project Name | FIC |
| Scan Start | Friday, March 11, 2022 11:38:48 AM |
| Preset | Checkmarx Default |
| Scan Time | 00h:00m:36s |
| Lines Of Code Scanned | 9257 |
| Files Scanned | 22 |
| Report Creation Time | Tuesday, March 15, 2022 1:28:51 PM |
| Online Results | https://cxmanager.pnl.gov/CxWebClient/ViewerMain.aspx?scanid=1017574&projectid=1025 |
| Team | PNNL |
| Checkmarx Version | 9.4.3 |
| Scan Type | Full |
| Source Origin | LocalPath |
| Density | 2/1000 (Vulnerabilities/LOC) |
| Visibility | Public |

# Filter Settings

**Severity**

Included: High, Medium, Low, Information
Excluded: None

**Result State**

Included: To Verify, Not Exploitable, Confirmed, Urgent, Proposed Not Exploitable
Excluded: None

**Assigned to**

Included: All

**Categories**

Included:

| | |
|---|---|
| Uncategorized | All |
| Custom | All |
| PCI DSS v3.2.1 | All |
| OWASP Top 10 2013 | All |
| FISMA 2014 | All |
| NIST SP 800-53 | All |
| OWASP Top 10 2017 | All |
| OWASP Mobile Top 10 2016 | All |
| ASD STIG 4.10 | All |
| OWASP Top 10 API | All |
| OWASP Top 10 2010 | All |
| OWASP Top 10 2021 | All |

Excluded:

| | |
|---|---|
| Uncategorized | None |
| Custom | None |
| PCI DSS v3.2.1 | None |
| OWASP Top 10 2013 | None |
| FISMA 2014 | None |

| | |
|---|---|
| NIST SP 800-53 | None |
| OWASP Top 10 2017 | None |
| OWASP Mobile Top 10 2016 | None |
| ASD STIG 4.10 | None |
| OWASP Top 10 API | None |
| OWASP Top 10 2010 | None |
| OWASP Top 10 2021 | None |

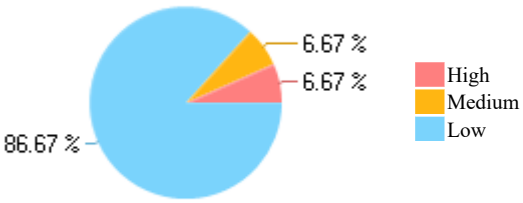
**Results Limit**

Results limit per query was set to 50

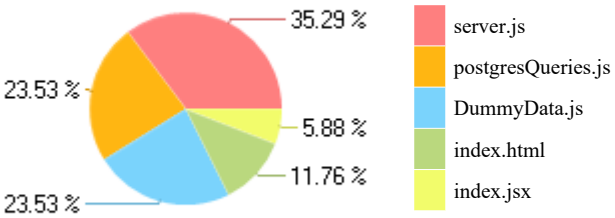
**Selected Queries**

Selected queries are listed in Result Summary

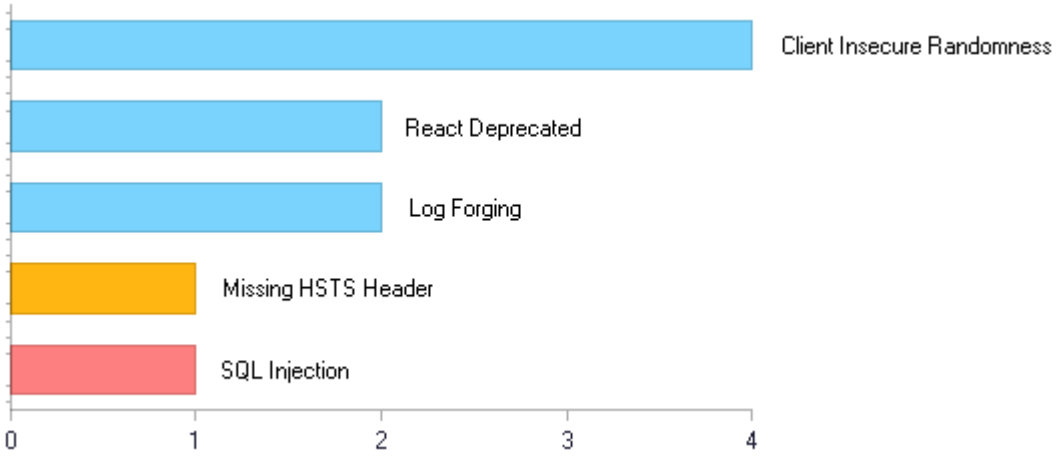
## Result Summary




## Most Vulnerable Files




## Top 5 Vulnerabilities



# Scan Summary - OWASP Top 10 2017

Further details and elaboration about vulnerabilities and risks can be found at: OWASP Top 10 2017

| Category | Threat Agent | Exploitability | Weakness Prevalence | Weakness Detectability | Technical Impact | Business Impact | Issues Found | Best Fix Locations |
|---|---|---|---|---|---|---|---|---|
| A1-Injection | App. Specific | EASY | COMMON | EASY | SEVERE | App. Specific | 3 | 2 |
| A2-Broken Authentication | App. Specific | EASY | COMMON | AVERAGE | SEVERE | App. Specific | 0 | 0 |
| A3-Sensitive Data Exposure | App. Specific | AVERAGE | WIDESPREAD | AVERAGE | SEVERE | App. Specific | 1 | 1 |
| A4-XML External Entities (XXE) | App. Specific | AVERAGE | COMMON | EASY | SEVERE | App. Specific | 0 | 0 |
| A5-Broken Access Control* | App. Specific | AVERAGE | COMMON | AVERAGE | SEVERE | App. Specific | 0 | 0 |
| A6-Security Misconfiguration | App. Specific | EASY | WIDESPREAD | EASY | MODERATE | App. Specific | 1 | 1 |
| A7-Cross-Site Scripting (XSS)* | App. Specific | EASY | WIDESPREAD | EASY | MODERATE | App. Specific | 0 | 0 |
| A8-Insecure Deserialization | App. Specific | DIFFICULT | COMMON | AVERAGE | SEVERE | App. Specific | 0 | 0 |
| A9-Using Components with Known Vulnerabilities | App. Specific | AVERAGE | WIDESPREAD | AVERAGE | MODERATE | App. Specific | 6 | 3 |
| A10-Insufficient Logging & Monitoring | App. Specific | AVERAGE | WIDESPREAD | DIFFICULT | MODERATE | App. Specific | 0 | 0 |

* Project scan results do not include all relevant queries. Presets and\or Filters should be changed to include all relevant standard queries.

# Scan Summary - OWASP Top 10 2021

| Category | Issues Found | Best Fix Locations |
|---|---|---|
| A1-Broken Access Control* | 1 | 1 |
| A2-Cryptographic Failures | 4 | 1 |
| A3-Injection* | 1 | 1 |
| A4-Insecure Design | 0 | 0 |
| A5-Security Misconfiguration | 0 | 0 |
| A6-Vulnerable and Outdated Components | 2 | 2 |
| A7-Identification and Authentication Failures | 3 | 3 |
| A8-Software and Data Integrity Failures* | 1 | 1 |
| A9-Security Logging and Monitoring Failures | 2 | 1 |
| A10-Server-Side Request Forgery | 0 | 0 |

* Project scan results do not include all relevant queries. Presets and\or Filters should be changed to include all relevant standard queries.

# Scan Summary - OWASP Top 10 2013

Further details and elaboration about vulnerabilities and risks can be found at: OWASP Top 10 2013

| Category | Threat Agent | Attack Vectors | Weakness Prevalence | Weakness Detectability | Technical Impact | Business Impact | Issues Found | Best Fix Locations |
|---|---|---|---|---|---|---|---|---|
| A1-Injection | EXTERNAL, INTERNAL, ADMIN USERS | EASY | COMMON | AVERAGE | SEVERE | ALL DATA | 1 | 1 |
| A2-Broken Authentication and Session Management | EXTERNAL, INTERNAL USERS | AVERAGE | WIDESPREAD | AVERAGE | SEVERE | AFFECTED DATA AND FUNCTIONS | 0 | 0 |
| A3-Cross-Site Scripting (XSS)* | EXTERNAL, INTERNAL, ADMIN USERS | AVERAGE | VERY WIDESPREAD | EASY | MODERATE | AFFECTED DATA AND SYSTEM | 0 | 0 |
| A4-Insecure Direct Object References* | SYSTEM USERS | EASY | COMMON | EASY | MODERATE | EXPOSED DATA | 0 | 0 |
| A5-Security Misconfiguration | EXTERNAL, INTERNAL, ADMIN USERS | EASY | COMMON | EASY | MODERATE | ALL DATA AND SYSTEM | 0 | 0 |
| A6-Sensitive Data Exposure | EXTERNAL, INTERNAL, ADMIN USERS, USERS BROWSERS | DIFFICULT | UNCOMMON | AVERAGE | SEVERE | EXPOSED DATA | 0 | 0 |
| A7-Missing Function Level Access Control | EXTERNAL, INTERNAL USERS | EASY | COMMON | AVERAGE | MODERATE | EXPOSED DATA AND FUNCTIONS | 0 | 0 |
| A8-Cross-Site Request Forgery (CSRF)* | USERS BROWSERS | AVERAGE | COMMON | EASY | MODERATE | AFFECTED DATA AND FUNCTIONS | 0 | 0 |
| A9-Using Components with Known Vulnerabilities | EXTERNAL USERS, AUTOMATED TOOLS | AVERAGE | WIDESPREAD | DIFFICULT | MODERATE | AFFECTED DATA AND FUNCTIONS | 0 | 0 |
| A10-Unvalidated Redirects and Forwards | USERS BROWSERS | AVERAGE | WIDESPREAD | DIFFICULT | MODERATE | AFFECTED DATA AND FUNCTIONS | 0 | 0 |

* Project scan results do not include all relevant queries. Presets and\or Filters should be changed to include all relevant standard queries.

# Scan Summary - PCI DSS v3.2.1

| Category | Issues Found | Best Fix Locations |
|---|---|---|
| PCI DSS (3.2.1) - 6.5.1 - Injection flaws - particularly SQL injection | 1 | 1 |
| PCI DSS (3.2.1) - 6.5.2 - Buffer overflows | 0 | 0 |
| PCI DSS (3.2.1) - 6.5.3 - Insecure cryptographic storage | 0 | 0 |
| PCI DSS (3.2.1) - 6.5.4 - Insecure communications | 0 | 0 |
| PCI DSS (3.2.1) - 6.5.5 - Improper error handling | 0 | 0 |
| PCI DSS (3.2.1) - 6.5.7 - Cross-site scripting (XSS) | 0 | 0 |
| PCI DSS (3.2.1) - 6.5.8 - Improper access control | 0 | 0 |
| PCI DSS (3.2.1) - 6.5.9 - Cross-site request forgery* | 0 | 0 |
| PCI DSS (3.2.1) - 6.5.10 - Broken authentication and session management | 0 | 0 |

* Project scan results do not include all relevant queries. Presets and\or Filters should be changed to include all relevant standard queries.

# Scan Summary - FISMA 2014

| Category | Description | Issues Found | Best Fix Locations |
|---|---|---|---|
| Access Control | Organizations must limit information system access to authorized users, processes acting on behalf of authorized users, or devices (including other information systems) and to the types of transactions and functions that authorized users are permitted to exercise. | 0 | 0 |
| Audit And Accountability | Organizations must: (i) create, protect, and retain information system audit records to the extent needed to enable the monitoring, analysis, investigation, and reporting of unlawful, unauthorized, or inappropriate information system activity; and (ii) ensure that the actions of individual information system users can be uniquely traced to those users so they can be held accountable for their actions. | 0 | 0 |
| Configuration Management | Organizations must: (i) establish and maintain baseline configurations and inventories of organizational information systems (including hardware, software, firmware, and documentation) throughout the respective system development life cycles; and (ii) establish and enforce security configuration settings for information technology products employed in organizational information systems. | 1 | 1 |
| Identification And Authentication | Organizations must identify information system users, processes acting on behalf of users, or devices and authenticate (or verify) the identities of those users, processes, or devices, as a prerequisite to allowing access to organizational information systems. | 1 | 1 |
| Media Protection | Organizations must: (i) protect information system media, both paper and digital; (ii) limit access to information on information system media to authorized users; and (iii) sanitize or destroy information system media before disposal or release for reuse. | 4 | 1 |
| System And Communications Protection | Organizations must: (i) monitor, control, and protect organizational communications (i.e., information transmitted or received by organizational information systems) at the external boundaries and key internal boundaries of the information systems; and (ii) employ architectural designs, software development techniques, and systems engineering principles that promote effective information security within organizational information systems. | 0 | 0 |
| System And Information Integrity* | Organizations must: (i) identify, report, and correct information and information system flaws in a timely manner; (ii) provide protection from malicious code at appropriate locations within organizational information systems; and (iii) monitor information system security alerts and advisories and take appropriate actions in response. | 3 | 2 |

* Project scan results do not include all relevant queries. Presets and\or Filters should be changed to include all relevant standard queries.

# Scan Summary - NIST SP 800-53

| Category | Issues Found | Best Fix Locations |
|---|---|---|
| AC-12 Session Termination (P2) | 0 | 0 |
| AC-3 Access Enforcement (P1) | 0 | 0 |
| AC-4 Information Flow Enforcement (P1) | 0 | 0 |
| AC-6 Least Privilege (P1) | 0 | 0 |
| AU-9 Protection of Audit Information (P1) | 2 | 1 |
| CM-6 Configuration Settings (P2) | 0 | 0 |
| IA-5 Authenticator Management (P1) | 0 | 0 |
| IA-6 Authenticator Feedback (P2) | 0 | 0 |
| IA-8 Identification and Authentication (Non-Organizational Users) (P1) | 0 | 0 |
| SC-12 Cryptographic Key Establishment and Management (P1) | 0 | 0 |
| SC-13 Cryptographic Protection (P1) | 0 | 0 |
| SC-17 Public Key Infrastructure Certificates (P1) | 0 | 0 |
| SC-18 Mobile Code (P2) | 1 | 1 |
| SC-23 Session Authenticity (P1)* | 0 | 0 |
| SC-28 Protection of Information at Rest (P1) | 5 | 2 |
| SC-4 Information in Shared Resources (P1) | 0 | 0 |
| SC-5 Denial of Service Protection (P1) | 0 | 0 |
| SC-8 Transmission Confidentiality and Integrity (P1) | 1 | 1 |
| SI-10 Information Input Validation (P1)* | 1 | 1 |
| SI-11 Error Handling (P2) | 0 | 0 |
| SI-15 Information Output Filtering (P0)* | 0 | 0 |
| SI-16 Memory Protection (P1) | 0 | 0 |

* Project scan results do not include all relevant queries. Presets and\or Filters should be changed to include all relevant standard queries.

# Scan Summary - OWASP Mobile Top 10 2016

| Category | Description | Issues Found | Best Fix Locations |
|---|---|---|---|
| M1-Improper Platform Usage | This category covers misuse of a platform feature or failure to use platform security controls. It might include Android intents, platform permissions, misuse of TouchID, the Keychain, or some other security control that is part of the mobile operating system. There are several ways that mobile apps can experience this risk. | 0 | 0 |
| M2-Insecure Data Storage | This category covers insecure data storage and unintended data leakage. | 0 | 0 |
| M3-Insecure Communication | This category covers poor handshaking, incorrect SSL versions, weak negotiation, cleartext communication of sensitive assets, etc. | 0 | 0 |
| M4-Insecure Authentication | This category captures notions of authenticating the end user or bad session management. This can include:<br>-Failing to identify the user at all when that should be required<br>-Failure to maintain the user's identity when it is required<br>-Weaknesses in session management | 0 | 0 |
| M5-Insufficient Cryptography | The code applies cryptography to a sensitive information asset. However, the cryptography is insufficient in some way. Note that anything and everything related to TLS or SSL goes in M3. Also, if the app fails to use cryptography at all when it should, that probably belongs in M2. This category is for issues where cryptography was attempted, but it wasnt done correctly. | 0 | 0 |
| M6-Insecure Authorization | This is a category to capture any failures in authorization (e.g., authorization decisions in the client side, forced browsing, etc.). It is distinct from authentication issues (e.g., device enrolment, user identification, etc.).<br>If the app does not authenticate users at all in a situation where it should (e.g., granting anonymous access to some resource or service when authenticated and authorized access is required), then that is an authentication failure not an authorization failure. | 0 | 0 |
| M7-Client Code Quality | This category is the catch-all for code-level implementation problems in the mobile client. That's distinct from server-side coding mistakes. This would capture things like buffer overflows, format string vulnerabilities, and various other code-level mistakes where the solution is to rewrite some code that's running on the mobile device. | 0 | 0 |
| M8-Code Tampering | This category covers binary patching, local resource modification, method hooking, method swizzling, and dynamic memory modification. Once the application is delivered to the mobile device, the code and data resources are resident there. An attacker can either directly modify the code, change the contents of memory dynamically, change or replace the system APIs that the application uses, or modify the application's data and resources. This can provide the attacker a direct method of subverting the intended use of the software for personal or monetary gain. | 0 | 0 |
| M9-Reverse Engineering | This category includes analysis of the final core binary to determine its source code, libraries, algorithms, and other assets. Software such as IDA Pro, Hopper, otool, and other binary inspection tools give the attacker insight into the inner workings of the application. This may be used to exploit other nascent vulnerabilities in the application, as well as revealing information about back end servers, cryptographic constants and ciphers, and intellectual property. | 0 | 0 |
| M10-Extraneous Functionality | Often, developers include hidden backdoor functionality or other internal development security controls that are | 0 | 0 |

| | | | |
|---|---|---|---|
| | not intended to be released into a production environment. For example, a developer may accidentally include a password as a comment in a hybrid app. Another example includes disabling of 2-factor authentication during testing. | | |

# Scan Summary - Custom

| Category | Issues Found | Best Fix Locations |
|---|---|---|
| Must audit | 0 | 0 |
| Check | 0 | 0 |
| Optional | 0 | 0 |

# Scan Summary - ASD STIG 4.10

| Category | Issues Found | Best Fix Locations |
|---|---|---|
| APSC-DV-000640 - CAT II The application must provide audit record generation capability for the renewal of session IDs. | 0 | 0 |
| APSC-DV-000650 - CAT II The application must not write sensitive data into the application logs. | 0 | 0 |
| APSC-DV-000660 - CAT II The application must provide audit record generation capability for session timeouts. | 0 | 0 |
| APSC-DV-000670 - CAT II The application must record a time stamp indicating when the event occurred. | 0 | 0 |
| APSC-DV-000680 - CAT II The application must provide audit record generation capability for HTTP headers including User-Agent, Referer, GET, and POST. | 0 | 0 |
| APSC-DV-000690 - CAT II The application must provide audit record generation capability for connecting system IP addresses. | 0 | 0 |
| APSC-DV-000700 - CAT II The application must record the username or user ID of the user associated with the event. | 0 | 0 |
| APSC-DV-000710 - CAT II The application must generate audit records when successful/unsuccessful attempts to grant privileges occur. | 0 | 0 |
| APSC-DV-000720 - CAT II The application must generate audit records when successful/unsuccessful attempts to access security objects occur. | 0 | 0 |
| APSC-DV-000730 - CAT II The application must generate audit records when successful/unsuccessful attempts to access security levels occur. | 0 | 0 |
| APSC-DV-000740 - CAT II The application must generate audit records when successful/unsuccessful attempts to access categories of information (e.g., classification levels) occur. | 0 | 0 |
| APSC-DV-000750 - CAT II The application must generate audit records when successful/unsuccessful attempts to modify privileges occur. | 0 | 0 |
| APSC-DV-000760 - CAT II The application must generate audit records when successful/unsuccessful attempts to modify security objects occur. | 0 | 0 |
| APSC-DV-000770 - CAT II The application must generate audit records when successful/unsuccessful attempts to modify security levels occur. | 0 | 0 |
| APSC-DV-000780 - CAT II The application must generate audit records when successful/unsuccessful attempts to modify categories of information (e.g., classification levels) occur. | 0 | 0 |
| APSC-DV-000790 - CAT II The application must generate audit records when successful/unsuccessful attempts to delete privileges occur. | 0 | 0 |
| APSC-DV-000800 - CAT II The application must generate audit records when successful/unsuccessful attempts to delete security levels occur. | 0 | 0 |
| APSC-DV-000810 - CAT II The application must generate audit records when successful/unsuccessful attempts to delete application database security objects occur. | 0 | 0 |
| APSC-DV-000820 - CAT II The application must generate audit records when successful/unsuccessful attempts to delete categories of information (e.g., classification levels) occur. | 0 | 0 |
| APSC-DV-000830 - CAT II The application must generate audit records when successful/unsuccessful logon attempts occur. | 0 | 0 |
| APSC-DV-000840 - CAT II The application must generate audit records for privileged activities or other system-level access. | 0 | 0 |
| APSC-DV-000850 - CAT II The application must generate audit records showing starting and ending time for user access to the system. | 0 | 0 |
| APSC-DV-000860 - CAT II The application must generate audit records when successful/unsuccessful accesses to objects occur. | 0 | 0 |
| APSC-DV-000870 - CAT II The application must generate audit records for all direct access to the information system. | 0 | 0 |
| APSC-DV-000880 - CAT II The application must generate audit records for all account creations, modifications, disabling, and termination events. | 0 | 0 |
| APSC-DV-000910 - CAT II The application must initiate session auditing upon startup. | 0 | 0 |
| APSC-DV-000940 - CAT II The application must log application shutdown events. | 0 | 0 |

| | | |
|---|---|---|
| APSC-DV-000950 - CAT II The application must log destination IP addresses. | 0 | 0 |
| APSC-DV-000960 - CAT II The application must log user actions involving access to data. | 0 | 0 |
| APSC-DV-000970 - CAT II The application must log user actions involving changes to data. | 0 | 0 |
| APSC-DV-000980 - CAT II The application must produce audit records containing information to establish when (date and time) the events occurred. | 0 | 0 |
| APSC-DV-000990 - CAT II The application must produce audit records containing enough information to establish which component, feature or function of the application triggered the audit event. | 0 | 0 |
| APSC-DV-001000 - CAT II When using centralized logging; the application must include a unique identifier in order to distinguish itself from other application logs. | 0 | 0 |
| APSC-DV-001010 - CAT II The application must produce audit records that contain information to establish the outcome of the events. | 0 | 0 |
| APSC-DV-001020 - CAT II The application must generate audit records containing information that establishes the identity of any individual or process associated with the event. | 0 | 0 |
| APSC-DV-001030 - CAT II The application must generate audit records containing the full-text recording of privileged commands or the individual identities of group account users. | 0 | 0 |
| APSC-DV-001040 - CAT II The application must implement transaction recovery logs when transaction based. | 0 | 0 |
| APSC-DV-001050 - CAT II The application must provide centralized management and configuration of the content to be captured in audit records generated by all application components. | 0 | 0 |
| APSC-DV-001070 - CAT II The application must off-load audit records onto a different system or media than the system being audited. | 0 | 0 |
| APSC-DV-001080 - CAT II The application must be configured to write application logs to a centralized log repository. | 0 | 0 |
| APSC-DV-001090 - CAT II The application must provide an immediate warning to the SA and ISSO (at a minimum) when allocated audit record storage volume reaches 75% of repository maximum audit record storage capacity. | 0 | 0 |
| APSC-DV-001100 - CAT II Applications categorized as having a moderate or high impact must provide an immediate real-time alert to the SA and ISSO (at a minimum) for all audit failure events. | 0 | 0 |
| APSC-DV-001110 - CAT II The application must alert the ISSO and SA (at a minimum) in the event of an audit processing failure. | 0 | 0 |
| APSC-DV-001120 - CAT II The application must shut down by default upon audit failure (unless availability is an overriding concern). | 0 | 0 |
| APSC-DV-001130 - CAT II The application must provide the capability to centrally review and analyze audit records from multiple components within the system. | 0 | 0 |
| APSC-DV-001140 - CAT II The application must provide the capability to filter audit records for events of interest based upon organization-defined criteria. | 0 | 0 |
| APSC-DV-001150 - CAT II The application must provide an audit reduction capability that supports on-demand reporting requirements. | 0 | 0 |
| APSC-DV-001160 - CAT II The application must provide an audit reduction capability that supports on-demand audit review and analysis. | 0 | 0 |
| APSC-DV-001170 - CAT II The application must provide an audit reduction capability that supports after-the-fact investigations of security incidents. | 0 | 0 |
| APSC-DV-001180 - CAT II The application must provide a report generation capability that supports on-demand audit review and analysis. | 0 | 0 |
| APSC-DV-001190 - CAT II The application must provide a report generation capability that supports on-demand reporting requirements. | 0 | 0 |
| APSC-DV-001200 - CAT II The application must provide a report generation capability that supports after-the-fact investigations of security incidents. | 0 | 0 |
| APSC-DV-001210 - CAT II The application must provide an audit reduction capability that does not alter original content or time ordering of audit records. | 0 | 0 |
| APSC-DV-001220 - CAT II The application must provide a report generation capability that does not alter original content or time ordering of audit records. | 0 | 0 |
| APSC-DV-001250 - CAT II The applications must use internal system clocks to generate time stamps for audit records. | 0 | 0 |
| APSC-DV-001260 - CAT II The application must record time stamps for audit records that can be mapped to Coordinated Universal Time (UTC) or Greenwich Mean Time (GMT). | 0 | 0 |
| APSC-DV-001270 - CAT II The application must record time stamps for audit records that meet a granularity of one | 0 | 0 |

| | | |
|---|---|---|
| second for a minimum degree of precision. | | |
| APSC-DV-001280 - CAT II The application must protect audit information from any type of unauthorized read access. | 0 | 0 |
| APSC-DV-001290 - CAT II The application must protect audit information from unauthorized modification. | 0 | 0 |
| APSC-DV-001300 - CAT II The application must protect audit information from unauthorized deletion. | 0 | 0 |
| APSC-DV-001310 - CAT II The application must protect audit tools from unauthorized access. | 0 | 0 |
| APSC-DV-001320 - CAT II The application must protect audit tools from unauthorized modification. | 0 | 0 |
| APSC-DV-001330 - CAT II The application must protect audit tools from unauthorized deletion. | 0 | 0 |
| APSC-DV-001340 - CAT II The application must back up audit records at least every seven days onto a different system or system component than the system or component being audited. | 0 | 0 |
| APSC-DV-001570 - CAT II The application must electronically verify Personal Identity Verification (PIV) credentials. | 0 | 0 |
| APSC-DV-001350 - CAT II The application must use cryptographic mechanisms to protect the integrity of audit information. | 0 | 0 |
| APSC-DV-001360 - CAT II Application audit tools must be cryptographically hashed. | 0 | 0 |
| APSC-DV-001370 - CAT II The integrity of the audit tools must be validated by checking the files for changes in the cryptographic hash value. | 0 | 0 |
| APSC-DV-001390 - CAT II The application must prohibit user installation of software without explicit privileged status. | 0 | 0 |
| APSC-DV-001410 - CAT II The application must enforce access restrictions associated with changes to application configuration. | 0 | 0 |
| APSC-DV-001420 - CAT II The application must audit who makes configuration changes to the application. | 0 | 0 |
| APSC-DV-001430 - CAT II The application must have the capability to prevent the installation of patches, service packs, or application components without verification the software component has been digitally signed using a certificate that is recognized and approved by the orga | 0 | 0 |
| APSC-DV-001440 - CAT II The applications must limit privileges to change the software resident within software libraries. | 0 | 0 |
| APSC-DV-001460 - CAT II An application vulnerability assessment must be conducted. | 0 | 0 |
| APSC-DV-001480 - CAT II The application must prevent program execution in accordance with organization-defined policies regarding software program usage and restrictions, and/or rules authorizing the terms and conditions of software program usage. | 0 | 0 |
| APSC-DV-001490 - CAT II The application must employ a deny-all, permit-by-exception (whitelist) policy to allow the execution of authorized software programs. | 0 | 0 |
| APSC-DV-001500 - CAT II The application must be configured to disable non-essential capabilities. | 0 | 0 |
| APSC-DV-001510 - CAT II The application must be configured to use only functions, ports, and protocols permitted to it in the PPSM CAL. | 0 | 0 |
| APSC-DV-001520 - CAT II The application must require users to reauthenticate when organization-defined circumstances or situations require reauthentication. | 0 | 0 |
| APSC-DV-001530 - CAT II The application must require devices to reauthenticate when organization-defined circumstances or situations requiring reauthentication. | 0 | 0 |
| APSC-DV-001540 - CAT I The application must uniquely identify and authenticate organizational users (or processes acting on behalf of organizational users). | 0 | 0 |
| APSC-DV-001550 - CAT II The application must use multifactor (Alt. Token) authentication for network access to privileged accounts. | 0 | 0 |
| APSC-DV-001560 - CAT II The application must accept Personal Identity Verification (PIV) credentials. | 0 | 0 |
| APSC-DV-001580 - CAT II The application must use multifactor (e.g., CAC, Alt. Token) authentication for network access to non-privileged accounts. | 0 | 0 |
| APSC-DV-001590 - CAT II The application must use multifactor (Alt. Token) authentication for local access to privileged accounts. | 0 | 0 |
| APSC-DV-001600 - CAT II The application must use multifactor (e.g., CAC, Alt. Token) authentication for local access to non-privileged accounts. | 0 | 0 |
| APSC-DV-001610 - CAT II The application must ensure users are authenticated with an individual authenticator prior to using a group authenticator. | 0 | 0 |
| APSC-DV-001620 - CAT II The application must implement replay-resistant authentication mechanisms for network access to privileged accounts. | 0 | 0 |

| | | |
|---|---|---|
| APSC-DV-001630 - CAT II The application must implement replay-resistant authentication mechanisms for network access to non-privileged accounts. | 0 | 0 |
| APSC-DV-001640 - CAT II The application must utilize mutual authentication when endpoint device non-repudiation protections are required by DoD policy or by the data owner. | 0 | 0 |
| APSC-DV-001650 - CAT II The application must authenticate all network connected endpoint devices before establishing any connection. | 0 | 0 |
| APSC-DV-001660 - CAT II Service-Oriented Applications handling non-releasable data must authenticate endpoint devices via mutual SSL/TLS. | 0 | 0 |
| APSC-DV-001670 - CAT II The application must disable device identifiers after 35 days of inactivity unless a cryptographic certificate is used for authentication. | 0 | 0 |
| APSC-DV-001680 - CAT I The application must enforce a minimum 15-character password length. | 0 | 0 |
| APSC-DV-001690 - CAT II The application must enforce password complexity by requiring that at least one upper-case character be used. | 0 | 0 |
| APSC-DV-001700 - CAT II The application must enforce password complexity by requiring that at least one lower-case character be used. | 0 | 0 |
| APSC-DV-001710 - CAT II The application must enforce password complexity by requiring that at least one numeric character be used. | 0 | 0 |
| APSC-DV-001720 - CAT II The application must enforce password complexity by requiring that at least one special character be used. | 0 | 0 |
| APSC-DV-001730 - CAT II The application must require the change of at least 8 of the total number of characters when passwords are changed. | 0 | 0 |
| APSC-DV-001740 - CAT I The application must only store cryptographic representations of passwords. | 1 | 1 |
| APSC-DV-001850 - CAT I The application must not display passwords/PINs as clear text. | 0 | 0 |
| APSC-DV-001750 - CAT I The application must transmit only cryptographically-protected passwords. | 0 | 0 |
| APSC-DV-001760 - CAT II The application must enforce 24 hours/1 day as the minimum password lifetime. | 0 | 0 |
| APSC-DV-001770 - CAT II The application must enforce a 60-day maximum password lifetime restriction. | 0 | 0 |
| APSC-DV-001780 - CAT II The application must prohibit password reuse for a minimum of five generations. | 0 | 0 |
| APSC-DV-001790 - CAT II The application must allow the use of a temporary password for system logons with an immediate change to a permanent password. | 0 | 0 |
| APSC-DV-001795 - CAT II The application password must not be changeable by users other than the administrator or the user with which the password is associated. | 0 | 0 |
| APSC-DV-001800 - CAT II The application must terminate existing user sessions upon account deletion. | 0 | 0 |
| APSC-DV-001820 - CAT I The application, when using PKI-based authentication, must enforce authorized access to the corresponding private key. | 0 | 0 |
| APSC-DV-001830 - CAT II The application must map the authenticated identity to the individual user or group account for PKI-based authentication. | 0 | 0 |
| APSC-DV-001870 - CAT II The application must uniquely identify and authenticate non-organizational users (or processes acting on behalf of non-organizational users). | 0 | 0 |
| APSC-DV-001810 - CAT I The application, when utilizing PKI-based authentication, must validate certificates by constructing a certification path (which includes status information) to an accepted trust anchor. | 0 | 0 |
| APSC-DV-001840 - CAT II The application, for PKI-based authentication, must implement a local cache of revocation data to support path discovery and validation in case of the inability to access revocation information via the network. | 0 | 0 |
| APSC-DV-001860 - CAT II The application must use mechanisms meeting the requirements of applicable federal laws, Executive Orders, directives, policies, regulations, standards, and guidance for authentication to a cryptographic module. | 0 | 0 |
| APSC-DV-001880 - CAT II The application must accept Personal Identity Verification (PIV) credentials from other federal agencies. | 0 | 0 |
| APSC-DV-001890 - CAT II The application must electronically verify Personal Identity Verification (PIV) credentials from other federal agencies. | 0 | 0 |
| APSC-DV-002050 - CAT II Applications making SAML assertions must use FIPS-approved random numbers in the generation of SessionIndex in the SAML element AuthnStatement. | 0 | 0 |
| APSC-DV-001900 - CAT II The application must accept FICAM-approved third-party credentials. | 0 | 0 |
| APSC-DV-001910 - CAT II The application must conform to FICAM-issued profiles. | 0 | 0 |
| APSC-DV-001930 - CAT II Applications used for non-local maintenance sessions must audit non-local maintenance | 0 | 0 |

| | | |
|---|---|---|
| and diagnostic sessions for organization-defined auditable events. | | |
| APSC-DV-000310 - CAT III The application must have a process, feature or function that prevents removal or disabling of emergency accounts. | 0 | 0 |
| APSC-DV-001940 - CAT II Applications used for non-local maintenance sessions must implement cryptographic mechanisms to protect the integrity of non-local maintenance and diagnostic communications. | 0 | 0 |
| APSC-DV-001950 - CAT II Applications used for non-local maintenance sessions must implement cryptographic mechanisms to protect the confidentiality of non-local maintenance and diagnostic communications. | 0 | 0 |
| APSC-DV-001960 - CAT II Applications used for non-local maintenance sessions must verify remote disconnection at the termination of non-local maintenance and diagnostic sessions. | 0 | 0 |
| APSC-DV-001970 - CAT II The application must employ strong authenticators in the establishment of non-local maintenance and diagnostic sessions. | 0 | 0 |
| APSC-DV-001980 - CAT II The application must terminate all sessions and network connections when non-local maintenance is completed. | 0 | 0 |
| APSC-DV-001995 - CAT II The application must not be vulnerable to race conditions. | 0 | 0 |
| APSC-DV-002000 - CAT II The application must terminate all network connections associated with a communications session at the end of the session. | 0 | 0 |
| APSC-DV-002010 - CAT II The application must implement NSA-approved cryptography to protect classified information in accordance with applicable federal laws, Executive Orders, directives, policies, regulations, and standards. | 0 | 0 |
| APSC-DV-002020 - CAT II The application must utilize FIPS-validated cryptographic modules when signing application components. | 0 | 0 |
| APSC-DV-002030 - CAT II The application must utilize FIPS-validated cryptographic modules when generating cryptographic hashes. | 0 | 0 |
| APSC-DV-002040 - CAT II The application must utilize FIPS-validated cryptographic modules when protecting unclassified information that requires cryptographic protection. | 0 | 0 |
| APSC-DV-002150 - CAT II The application user interface must be either physically or logically separated from data storage and management interfaces. | 0 | 0 |
| APSC-DV-002210 - CAT II The application must set the HTTPOnly flag on session cookies. | 0 | 0 |
| APSC-DV-002220 - CAT II The application must set the secure flag on session cookies. | 0 | 0 |
| APSC-DV-002230 - CAT I The application must not expose session IDs. | 0 | 0 |
| APSC-DV-002240 - CAT I The application must destroy the session ID value and/or cookie on logoff or browser close. | 0 | 0 |
| APSC-DV-002250 - CAT II Applications must use system-generated session identifiers that protect against session fixation. | 0 | 0 |
| APSC-DV-002260 - CAT II Applications must validate session identifiers. | 0 | 0 |
| APSC-DV-002270 - CAT II Applications must not use URL embedded session IDs. | 0 | 0 |
| APSC-DV-002280 - CAT II The application must not re-use or recycle session IDs. | 0 | 0 |
| APSC-DV-002290 - CAT II The application must use the Federal Information Processing Standard (FIPS) 140-2-validated cryptographic modules and random number generator if the application implements encryption, key exchange, digital signature, and hash functionality. | 4 | 1 |
| APSC-DV-002300 - CAT II The application must only allow the use of DoD-approved certificate authorities for verification of the establishment of protected sessions. | 0 | 0 |
| APSC-DV-002310 - CAT I The application must fail to a secure state if system initialization fails, shutdown fails, or aborts fail. | 0 | 0 |
| APSC-DV-002320 - CAT II In the event of a system failure, applications must preserve any information necessary to determine cause of failure and any information necessary to return to operations with least disruption to mission processes. | 0 | 0 |
| APSC-DV-002330 - CAT II The application must protect the confidentiality and integrity of stored information when required by DoD policy or the information owner. | 1 | 1 |
| APSC-DV-002340 - CAT II The application must implement approved cryptographic mechanisms to prevent unauthorized modification of organization-defined information at rest on organization-defined information system components. | 0 | 0 |
| APSC-DV-002350 - CAT II The application must use appropriate cryptography in order to protect stored DoD information when required by the information owner or DoD policy. | 0 | 0 |
| APSC-DV-002360 - CAT II The application must isolate security functions from non-security functions. | 0 | 0 |
| APSC-DV-002370 - CAT II The application must maintain a separate execution domain for each executing process. | 0 | 0 |

| | | |
|---|---|---|
| APSC-DV-002380 - CAT II Applications must prevent unauthorized and unintended information transfer via shared system resources. | 0 | 0 |
| APSC-DV-002390 - CAT II XML-based applications must mitigate DoS attacks by using XML filters, parser options, or gateways. | 0 | 0 |
| APSC-DV-002400 - CAT II The application must restrict the ability to launch Denial of Service (DoS) attacks against itself or other information systems. | 0 | 0 |
| APSC-DV-002410 - CAT II The web service design must include redundancy mechanisms when used with high-availability systems. | 0 | 0 |
| APSC-DV-002420 - CAT II An XML firewall function must be deployed to protect web services when exposed to untrusted networks. | 0 | 0 |
| APSC-DV-002610 - CAT II The application must remove organization-defined software components after updated versions have been installed. | 0 | 0 |
| APSC-DV-002440 - CAT I The application must protect the confidentiality and integrity of transmitted information. | 1 | 1 |
| APSC-DV-002450 - CAT II The application must implement cryptographic mechanisms to prevent unauthorized disclosure of information and/or detect changes to information during transmission unless otherwise protected by alternative physical safeguards, such as, at a minimum, a Prot | 0 | 0 |
| APSC-DV-002460 - CAT II The application must maintain the confidentiality and integrity of information during preparation for transmission. | 0 | 0 |
| APSC-DV-002470 - CAT II The application must maintain the confidentiality and integrity of information during reception. | 0 | 0 |
| APSC-DV-002480 - CAT II The application must not disclose unnecessary information to users. | 0 | 0 |
| APSC-DV-002485 - CAT I The application must not store sensitive information in hidden fields. | 0 | 0 |
| APSC-DV-002490 - CAT I The application must protect from Cross-Site Scripting (XSS) vulnerabilities. | 1 | 1 |
| APSC-DV-002500 - CAT II The application must protect from Cross-Site Request Forgery (CSRF) vulnerabilities.* | 0 | 0 |
| APSC-DV-002510 - CAT I The application must protect from command injection. | 0 | 0 |
| APSC-DV-002520 - CAT II The application must protect from canonical representation vulnerabilities. | 0 | 0 |
| APSC-DV-002530 - CAT II The application must validate all input. | 0 | 0 |
| APSC-DV-002540 - CAT I The application must not be vulnerable to SQL Injection. | 1 | 1 |
| APSC-DV-002550 - CAT I The application must not be vulnerable to XML-oriented attacks. | 0 | 0 |
| APSC-DV-002560 - CAT I The application must not be subject to input handling vulnerabilities.* | 2 | 1 |
| APSC-DV-002570 - CAT II The application must generate error messages that provide information necessary for corrective actions without revealing information that could be exploited by adversaries. | 0 | 0 |
| APSC-DV-002580 - CAT II The application must reveal error messages only to the ISSO, ISSM, or SA. | 0 | 0 |
| APSC-DV-002590 - CAT I The application must not be vulnerable to overflow attacks. | 0 | 0 |
| APSC-DV-002630 - CAT II Security-relevant software updates and patches must be kept up to date. | 0 | 0 |
| APSC-DV-002760 - CAT II The application performing organization-defined security functions must verify correct operation of security functions. | 0 | 0 |
| APSC-DV-002900 - CAT II The ISSO must ensure application audit trails are retained for at least 1 year for applications without SAMI data, and 5 years for applications including SAMI data. | 0 | 0 |
| APSC-DV-002770 - CAT II The application must perform verification of the correct operation of security functions: upon system startup and/or restart; upon command by a user with privileged access; and/or every 30 days. | 0 | 0 |
| APSC-DV-002780 - CAT III The application must notify the ISSO and ISSM of failed security verification tests. | 0 | 0 |
| APSC-DV-002870 - CAT II Unsigned Category 1A mobile code must not be used in the application in accordance with DoD policy. | 0 | 0 |
| APSC-DV-002880 - CAT II The ISSO must ensure an account management process is implemented, verifying only authorized users can gain access to the application, and individual accounts designated as inactive, suspended, or terminated are promptly removed. | 0 | 0 |
| APSC-DV-002890 - CAT I Application web servers must be on a separate network segment from the application and database servers if it is a tiered application operating in the DoD DMZ. | 0 | 0 |
| APSC-DV-002910 - CAT II The ISSO must review audit trails periodically based on system documentation recommendations or immediately upon system security events. | 0 | 0 |
| APSC-DV-002920 - CAT II The ISSO must report all suspected violations of IA policies in accordance with DoD information system IA procedures. | 0 | 0 |
| APSC-DV-002930 - CAT II The ISSO must ensure active vulnerability testing is performed. | 0 | 0 |

| | | |
|---|---|---|
| APSC-DV-002980 - CAT II New IP addresses, data services, and associated ports used by the application must be submitted to the appropriate approving authority for the organization, which in turn will be submitted through the DoD Ports, Protocols, and Services Management (DoD PPS | 0 | 0 |
| APSC-DV-002950 - CAT II Execution flow diagrams and design documents must be created to show how deadlock and recursion issues in web services are being mitigated. | 0 | 0 |
| APSC-DV-002960 - CAT II The designer must ensure the application does not store configuration and control files in the same directory as user data. | 0 | 0 |
| APSC-DV-002970 - CAT II The ISSO must ensure if a DoD STIG or NSA guide is not available, a third-party product will be configured by following available guidance. | 0 | 0 |
| APSC-DV-002990 - CAT II The application must be registered with the DoD Ports and Protocols Database. | 0 | 0 |
| APSC-DV-002990 - CAT II The application must be registered with the DoD Ports and Protocols Database. | 0 | 0 |
| APSC-DV-002995 - CAT II The Configuration Management (CM) repository must be properly patched and STIG compliant. | 0 | 0 |
| APSC-DV-003000 - CAT II Access privileges to the Configuration Management (CM) repository must be reviewed every three months. | 0 | 0 |
| APSC-DV-003010 - CAT II A Software Configuration Management (SCM) plan describing the configuration control and change management process of application objects developed by the organization and the roles and responsibilities of the organization must be created and maintained. | 0 | 0 |
| APSC-DV-003020 - CAT II A Configuration Control Board (CCB) that meets at least every release cycle, for managing the Configuration Management (CM) process must be established. | 0 | 0 |
| APSC-DV-003030 - CAT II The application services and interfaces must be compatible with and ready for IPv6 networks. | 0 | 0 |
| APSC-DV-003040 - CAT II The application must not be hosted on a general purpose machine if the application is designated as critical or high availability by the ISSO. | 0 | 0 |
| APSC-DV-003050 - CAT II A disaster recovery/continuity plan must exist in accordance with DoD policy based on the applications availability requirements. | 0 | 0 |
| APSC-DV-003060 - CAT II Recovery procedures and technical system features must exist so recovery is performed in a secure and verifiable manner. The ISSO will document circumstances inhibiting a trusted recovery. | 0 | 0 |
| APSC-DV-003070 - CAT II Data backup must be performed at required intervals in accordance with DoD policy. | 0 | 0 |
| APSC-DV-003080 - CAT II Back-up copies of the application software or source code must be stored in a fire-rated container or stored separately (offsite). | 0 | 0 |
| APSC-DV-003090 - CAT II Procedures must be in place to assure the appropriate physical and technical protection of the backup and restoration of the application. | 0 | 0 |
| APSC-DV-003100 - CAT II The application must use encryption to implement key exchange and authenticate endpoints prior to establishing a communication channel for key exchange. | 0 | 0 |
| APSC-DV-003110 - CAT I The application must not contain embedded authentication data. | 0 | 0 |
| APSC-DV-003120 - CAT I The application must have the capability to mark sensitive/classified output when required. | 0 | 0 |
| APSC-DV-003130 - CAT III Prior to each release of the application, updates to system, or applying patches; tests plans and procedures must be created and executed. | 0 | 0 |
| APSC-DV-003150 - CAT II At least one tester must be designated to test for security flaws in addition to functional testing. | 0 | 0 |
| APSC-DV-003140 - CAT II Application files must be cryptographically hashed prior to deploying to DoD operational networks. | 0 | 0 |
| APSC-DV-003160 - CAT III Test procedures must be created and at least annually executed to ensure system initialization, shutdown, and aborts are configured to verify the system remains in a secure state. | 0 | 0 |
| APSC-DV-003170 - CAT II An application code review must be performed on the application. | 0 | 0 |
| APSC-DV-003180 - CAT III Code coverage statistics must be maintained for each release of the application. | 0 | 0 |
| APSC-DV-003190 - CAT II Flaws found during a code review must be tracked in a defect tracking system. | 0 | 0 |
| APSC-DV-003200 - CAT II The changes to the application must be assessed for IA and accreditation impact prior to implementation. | 0 | 0 |
| APSC-DV-003210 - CAT II Security flaws must be fixed or addressed in the project plan. | 0 | 0 |
| APSC-DV-003215 - CAT III The application development team must follow a set of coding standards. | 0 | 0 |
| APSC-DV-003220 - CAT III The designer must create and update the Design Document for each release of the application. | 0 | 0 |

| | | |
|---|---|---|
| APSC-DV-003230 - CAT II Threat models must be documented and reviewed for each application release and updated as required by design and functionality changes or when new threats are discovered. | 0 | 0 |
| APSC-DV-003235 - CAT II The application must not be subject to error handling vulnerabilities. | 0 | 0 |
| APSC-DV-003250 - CAT I The application must be decommissioned when maintenance or support is no longer available. | 0 | 0 |
| APSC-DV-003236 - CAT II The application development team must provide an application incident response plan. | 0 | 0 |
| APSC-DV-003240 - CAT I All products must be supported by the vendor or the development team. | 0 | 0 |
| APSC-DV-003260 - CAT III Procedures must be in place to notify users when an application is decommissioned. | 0 | 0 |
| APSC-DV-003270 - CAT II Unnecessary built-in application accounts must be disabled. | 0 | 0 |
| APSC-DV-003280 - CAT I Default passwords must be changed. | 0 | 0 |
| APSC-DV-003330 - CAT II The system must alert an administrator when low resource conditions are encountered. | 0 | 0 |
| APSC-DV-003285 - CAT II An Application Configuration Guide must be created and included with the application. | 0 | 0 |
| APSC-DV-003290 - CAT II If the application contains classified data, a Security Classification Guide must exist containing data elements and their classification. | 0 | 0 |
| APSC-DV-003300 - CAT II The designer must ensure uncategorized or emerging mobile code is not used in applications. | 0 | 0 |
| APSC-DV-003310 - CAT II Production database exports must have database administration credentials and sensitive data removed before releasing the export. | 0 | 0 |
| APSC-DV-003320 - CAT II Protections against DoS attacks must be implemented. | 0 | 0 |
| APSC-DV-003340 - CAT III At least one application administrator must be registered to receive update notifications, or security alerts, when automated alerts are available. | 0 | 0 |
| APSC-DV-003360 - CAT III The application must generate audit records when concurrent logons from different workstations occur. | 0 | 0 |
| APSC-DV-003345 - CAT III The application must provide notifications or alerts when product update and security related patches are available. | 0 | 0 |
| APSC-DV-003350 - CAT II Connections between the DoD enclave and the Internet or other public or commercial wide area networks must require a DMZ. | 0 | 0 |
| APSC-DV-003400 - CAT II The Program Manager must verify all levels of program management, designers, developers, and testers receive annual security training pertaining to their job function. | 0 | 0 |
| APSC-DV-000010 - CAT II The application must provide a capability to limit the number of logon sessions per user. | 0 | 0 |
| APSC-DV-000060 - CAT II The application must clear temporary storage and cookies when the session is terminated. | 0 | 0 |
| APSC-DV-000070 - CAT II The application must automatically terminate the non-privileged user session and log off non-privileged users after a 15 minute idle time period has elapsed. | 0 | 0 |
| APSC-DV-000080 - CAT II The application must automatically terminate the admin user session and log off admin users after a 10 minute idle time period is exceeded. | 0 | 0 |
| APSC-DV-000090 - CAT II Applications requiring user access authentication must provide a logoff capability for user initiated communication session. | 0 | 0 |
| APSC-DV-000100 - CAT III The application must display an explicit logoff message to users indicating the reliable termination of authenticated communications sessions. | 0 | 0 |
| APSC-DV-000110 - CAT II The application must associate organization-defined types of security attributes having organization-defined security attribute values with information in storage. | 0 | 0 |
| APSC-DV-000120 - CAT II The application must associate organization-defined types of security attributes having organization-defined security attribute values with information in process. | 0 | 0 |
| APSC-DV-000130 - CAT II The application must associate organization-defined types of security attributes having organization-defined security attribute values with information in transmission. | 0 | 0 |
| APSC-DV-000160 - CAT II The application must implement DoD-approved encryption to protect the confidentiality of remote access sessions. | 0 | 0 |
| APSC-DV-000170 - CAT II The application must implement cryptographic mechanisms to protect the integrity of remote access sessions. | 0 | 0 |
| APSC-DV-000190 - CAT I Messages protected with WS_Security must use time stamps with creation and expiration times. | 0 | 0 |
| APSC-DV-000180 - CAT II Applications with SOAP messages requiring integrity must include the following message elements:-Message ID-Service Request-Timestamp-SAML Assertion (optionally included in messages) and | 0 | 0 |

| | | |
|---|---|---|
| all elements of the message must be digitally signed. | | |
| APSC-DV-000200 - CAT I Validity periods must be verified on all application messages using WS-Security or SAML assertions. | 0 | 0 |
| APSC-DV-000210 - CAT II The application must ensure each unique asserting party provides unique assertion ID references for each SAML assertion. | 0 | 0 |
| APSC-DV-000220 - CAT II The application must ensure encrypted assertions, or equivalent confidentiality protections are used when assertion data is passed through an intermediary, and confidentiality of the assertion data is required when passing through the intermediary. | 0 | 0 |
| APSC-DV-000230 - CAT I The application must use the NotOnOrAfter condition when using the SubjectConfirmation element in a SAML assertion. | 0 | 0 |
| APSC-DV-000240 - CAT I The application must use both the NotBefore and NotOnOrAfter elements or OneTimeUse element when using the Conditions element in a SAML assertion. | 0 | 0 |
| APSC-DV-000250 - CAT II The application must ensure if a OneTimeUse element is used in an assertion, there is only one of the same used in the Conditions element portion of an assertion. | 0 | 0 |
| APSC-DV-000260 - CAT II The application must ensure messages are encrypted when the SessionIndex is tied to privacy data. | 0 | 0 |
| APSC-DV-000290 - CAT II Shared/group account credentials must be terminated when members leave the group. | 0 | 0 |
| APSC-DV-000280 - CAT II The application must provide automated mechanisms for supporting account management functions. | 0 | 0 |
| APSC-DV-000300 - CAT II The application must automatically remove or disable temporary user accounts 72 hours after account creation. | 0 | 0 |
| APSC-DV-000320 - CAT III The application must automatically disable accounts after a 35 day period of account inactivity. | 0 | 0 |
| APSC-DV-000330 - CAT II Unnecessary application accounts must be disabled, or deleted. | 0 | 0 |
| APSC-DV-000420 - CAT II The application must automatically audit account enabling actions. | 0 | 0 |
| APSC-DV-000340 - CAT II The application must automatically audit account creation. | 0 | 0 |
| APSC-DV-000350 - CAT II The application must automatically audit account modification. | 0 | 0 |
| APSC-DV-000360 - CAT II The application must automatically audit account disabling actions. | 0 | 0 |
| APSC-DV-000370 - CAT II The application must automatically audit account removal actions. | 0 | 0 |
| APSC-DV-000380 - CAT III The application must notify System Administrators and Information System Security Officers when accounts are created. | 0 | 0 |
| APSC-DV-000390 - CAT III The application must notify System Administrators and Information System Security Officers when accounts are modified. | 0 | 0 |
| APSC-DV-000400 - CAT III The application must notify System Administrators and Information System Security Officers of account disabling actions. | 0 | 0 |
| APSC-DV-000410 - CAT III The application must notify System Administrators and Information System Security Officers of account removal actions. | 0 | 0 |
| APSC-DV-000430 - CAT III The application must notify System Administrators and Information System Security Officers of account enabling actions. | 0 | 0 |
| APSC-DV-000440 - CAT II Application data protection requirements must be identified and documented. | 0 | 0 |
| APSC-DV-000520 - CAT II The application must audit the execution of privileged functions. | 0 | 0 |
| APSC-DV-000450 - CAT II The application must utilize organization-defined data mining detection techniques for organization-defined data storage objects to adequately detect data mining attempts. | 0 | 0 |
| APSC-DV-000460 - CAT I The application must enforce approved authorizations for logical access to information and system resources in accordance with applicable access control policies. | 0 | 0 |
| APSC-DV-000470 - CAT II The application must enforce organization-defined discretionary access control policies over defined subjects and objects. | 0 | 0 |
| APSC-DV-000480 - CAT II The application must enforce approved authorizations for controlling the flow of information within the system based on organization-defined information flow control policies. | 0 | 0 |
| APSC-DV-000490 - CAT II The application must enforce approved authorizations for controlling the flow of information between interconnected systems based on organization-defined information flow control policies. | 0 | 0 |
| APSC-DV-000500 - CAT II The application must prevent non-privileged users from executing privileged functions to include disabling, circumventing, or altering implemented security safeguards/countermeasures. | 0 | 0 |
| APSC-DV-000510 - CAT I The application must execute without excessive account permissions. | 0 | 0 |

| | | |
|---|---|---|
| APSC-DV-000530 - CAT I The application must enforce the limit of three consecutive invalid logon attempts by a user during a 15 minute time period. | 0 | 0 |
| APSC-DV-000560 - CAT III The application must retain the Standard Mandatory DoD Notice and Consent Banner on the screen until users acknowledge the usage conditions and take explicit actions to log on for further access. | 0 | 0 |
| APSC-DV-000540 - CAT II The application administrator must follow an approved process to unlock locked user accounts. | 0 | 0 |
| APSC-DV-000550 - CAT III The application must display the Standard Mandatory DoD Notice and Consent Banner before granting access to the application. | 0 | 0 |
| APSC-DV-000570 - CAT III The publicly accessible application must display the Standard Mandatory DoD Notice and Consent Banner before granting access to the application. | 0 | 0 |
| APSC-DV-000580 - CAT III The application must display the time and date of the users last successful logon. | 0 | 0 |
| APSC-DV-000630 - CAT II The application must provide audit record generation capability for the destruction of session IDs. | 0 | 0 |
| APSC-DV-000590 - CAT II The application must protect against an individual (or process acting on behalf of an individual) falsely denying having performed organization-defined actions to be covered by non-repudiation. | 0 | 0 |
| APSC-DV-000600 - CAT II For applications providing audit record aggregation, the application must compile audit records from organization-defined information system components into a system-wide audit trail that is time-correlated with an organization-defined level of tolerance | 0 | 0 |
| APSC-DV-000610 - CAT II The application must provide the capability for organization-identified individuals or roles to change the auditing to be performed on all application components, based on all selectable event criteria within organization-defined time thresholds. | 0 | 0 |
| APSC-DV-000620 - CAT II The application must provide audit record generation capability for the creation of session IDs. | 0 | 0 |

***** Project scan results do not include all relevant queries. Presets and\or Filters should be changed to include all relevant standard queries.

# Scan Summary - OWASP Top 10 API

| Category | Issues Found | Best Fix Locations |
|---|---|---|
| API1-Broken Object Level Authorization | 0 | 0 |
| API2-Broken Authentication | 0 | 0 |
| API3-Excessive Data Exposure | 0 | 0 |
| API4-Lack of Resources and Rate Limiting | 0 | 0 |
| API5-Broken Function Level Authorization | 0 | 0 |
| API6-Mass Assignment | 0 | 0 |
| API7-Security Misconfiguration | 0 | 0 |
| API8-Injection | 0 | 0 |
| API9-Improper Assets Management | 0 | 0 |
| API10-Insufficient Logging and Monitoring | 0 | 0 |

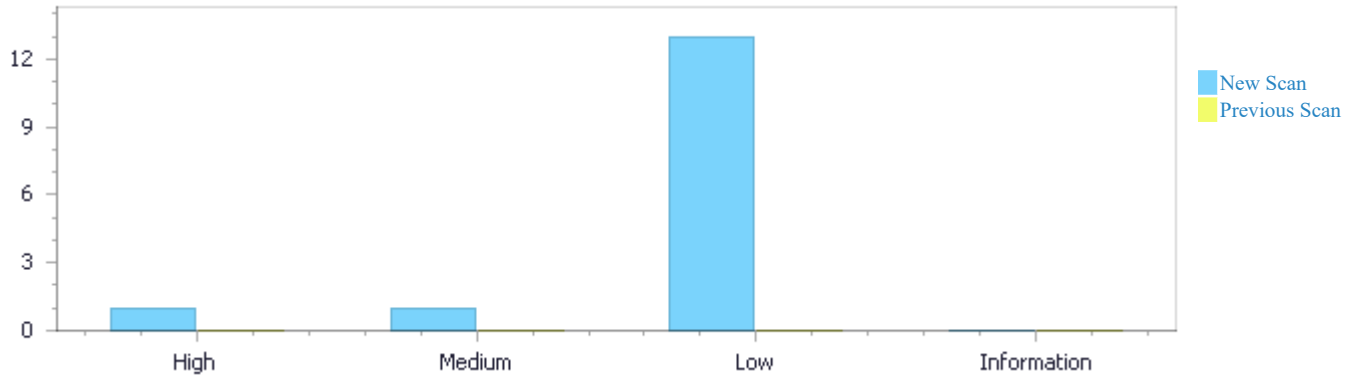
# Scan Summary - OWASP Top 10 2010

| Category | Issues Found | Best Fix Locations |
|---|---|---|
| A1-Injection* | 0 | 0 |
| A2-Cross-Site Scripting (XSS) | 0 | 0 |
| A3-Broken Authentication and Session Management | 0 | 0 |
| A4-Insecure Direct Object References | 0 | 0 |
| A5-Cross-Site Request Forgery (CSRF) | 0 | 0 |
| A6-Security Misconfiguration* | 1 | 1 |
| A7-Insecure Cryptographic Storage | 4 | 1 |
| A8-Failure to Restrict URL Access | 0 | 0 |
| A9-Insufficient Transport Layer Protection | 0 | 0 |
| A10-Unvalidated Redirects and Forwards | 0 | 0 |

* Project scan results do not include all relevant queries. Presets and\or Filters should be changed to include all relevant standard queries.

## Results Distribution By Status First scan of the project

| | High | Medium | Low | Information | Total |
|---|---|---|---|---|---|
| New Issues | 1 | 1 | 13 | 0 | 15 |
| Recurrent Issues | 0 | 0 | 0 | 0 | 0 |
| Total | 1 | 1 | 13 | 0 | 15 |
| | | | | | |
| Fixed Issues | 0 | 0 | 0 | 0 | 0 |




## Results Distribution By State

| | High | Medium | Low | Information | Total |
|---|---|---|---|---|---|
| To Verify | 1 | 1 | 13 | 0 | 15 |
| Not Exploitable | 0 | 0 | 0 | 0 | 0 |
| Confirmed | 0 | 0 | 0 | 0 | 0 |
| Urgent | 0 | 0 | 0 | 0 | 0 |
| Proposed Not Exploitable | 0 | 0 | 0 | 0 | 0 |
| Total | 1 | 1 | 13 | 0 | 15 |

## Result Summary

| Vulnerability Type | Occurrences | Severity |
|---|---|---|
| SQL Injection | 1 | High |
| Missing HSTS Header | 1 | Medium |
| Client Insecure Randomness | 4 | Low |
| Log Forging | 2 | Low |
| React Deprecated | 2 | Low |
| Client Hardcoded Domain | 1 | Low |
| Missing CSP Header | 1 | Low |
| Potential Clickjacking on Legacy Browsers | 1 | Low |
| Potentially Vulnerable To Csrf | 1 | Low |

| | | |
|---|---|---|
| Use Of Hardcoded Password | 1 | Low |

# 10 Most Vulnerable Files

## High and Medium Vulnerabilities

| File Name | Issues Found |
|---|---|
| app/server.js | 2 |
| app/server/postgresQueries.js | 1 |

# Scan Results Details

## SQL Injection

Query Path:
JavaScript\Cx\JavaScript Server Side Vulnerabilities\SQL Injection Version:3

### Categories

PCI DSS v3.2.1: PCI DSS (3.2.1) - 6.5.1 - Injection flaws - particularly SQL injection
OWASP Top 10 2013: A1-Injection
FISMA 2014: System And Information Integrity
NIST SP 800-53: SI-10 Information Input Validation (P1)
OWASP Top 10 2017: A1-Injection
ASD STIG 4.10: APSC-DV-002540 - CAT I The application must not be vulnerable to SQL Injection.
OWASP Top 10 2021: A3-Injection

*Description*

**SQL Injection\Path 1:**

| | |
|---|---|
| Severity | High |
| Result State | To Verify |
| Online Results | https://cxmanager.pnl.gov/CxWebClient/ViewerMain.aspx?scanid=1017574&projectid=1025&pathid=1 |
| Status | New |
| Detection Date | 3/11/2022 11:39:24 AM |

The application's insertInstallationPermit method executes an SQL query with insertStatement, at line 19 of app/server/postgresQueries.js. The application constructs this SQL query by embedding an untrusted string into the query without proper sanitization. The concatenated string is submitted to the database, where it is parsed and executed accordingly.
An attacker would be able to inject arbitrary syntax and data into the SQL query, by crafting a malicious payload and providing it via the input body; this input is then read by the app.post method at line 47 of app/server.js. This input then flows through the code, into a query and to the database server - without sanitization.
This may enable an SQL Injection attack.

| | Source | Destination |
|---|---|---|
| File | app/server.js | app/server/postgresQueries.js |
| Line | 48 | 26 |
| Object | body | insertStatement |

Code Snippet

File Name app/server.js

Method app.post('/api/addPermit', function (req, res) {

```
....
48.     insertInstallationPermit(req.body)
```

File Name app/server/postgresQueries.js

Method function insertInstallationPermit(permit) {

```
....
26.       return postgresPool.query(insertStatement);
```

## Missing HSTS Header

Query Path:
JavaScript\Cx\JavaScript Medium Threat\Missing HSTS Header Version:1

### Categories

ASD STIG 4.10: APSC-DV-002440 - CAT I The application must protect the confidentiality and integrity of transmitted information.
OWASP Top 10 2010: A6-Security Misconfiguration
OWASP Top 10 2021: A7-Identification and Authentication Failures

*Description*

**Missing HSTS Header\Path 1:**

| | |
|---|---|
| Severity | Medium |
| Result State | To Verify |
| Online Results | https://cxmanager.pnl.gov/CxWebClient/ViewerMain.aspx?scanid=1017574&projectid=1025&pathid=2 |
| Status | New |
| Detection Date | 3/11/2022 11:39:24 AM |

The web-application does not define an HSTS header, leaving it vulnerable to attack.

| | Source | Destination |
|---|---|---|
| File | app/server.js | app/server.js |
| Line | 35 | 35 |
| Object | send | send |

Code Snippet
File Name app/server.js
Method app.get('/api/getPermit', function (req, res) {

```
....
35.     return res.send('addPermit');
```

## Client Insecure Randomness

Query Path:
JavaScript\Cx\JavaScript Low Visibility\Client Insecure Randomness Version:3

### Categories

FISMA 2014: Media Protection
NIST SP 800-53: SC-28 Protection of Information at Rest (P1)
OWASP Top 10 2017: A9-Using Components with Known Vulnerabilities
ASD STIG 4.10: APSC-DV-002290 - CAT II The application must use the Federal Information Processing Standard (FIPS) 140-2-validated cryptographic modules and random number generator if the application implements encryption, key exchange, digital signature, and hash functionality.
OWASP Top 10 2010: A7-Insecure Cryptographic Storage
OWASP Top 10 2021: A2-Cryptographic Failures

*Description*

**Client Insecure Randomness\Path 1:**

| | |
|---|---|
| Severity | Low |
| Result State | To Verify |
| Online Results | https://cxmanager.pnl.gov/CxWebClient/ViewerMain.aspx?scanid=1017574&projectid=1025&pathid=3 |
| Status | New |
| Detection Date | 3/11/2022 11:39:24 AM |

Method getRandomNumberBetween0And at line 67 of app/src/assets/DummyData.js uses a weak method random to produce random values. These values might be used as personal identifiers, session tokens or cryptographic input; however, due to their insufficient randomness, an attacker may be able to derive their value.

| | Source | Destination |
|---|---|---|
| File | app/src/assets/DummyData.js | app/src/assets/DummyData.js |
| Line | 68 | 116 |
| Object | random | getRandomAddress |

Code Snippet
File Name app/src/assets/DummyData.js
Method function getRandomNumberBetween0And(max) {

```
....
68.       return Math.ceil((Math.random() * max)) - 1;
```

File Name app/src/assets/DummyData.js
Method function makeRandomInstallationPermit() {

```
....
116.       const contact = getRandomAddress();
```

**Client Insecure Randomness\Path 2:**

| | |
|---|---|
| Severity | Low |
| Result State | To Verify |
| Online Results | https://cxmanager.pnl.gov/CxWebClient/ViewerMain.aspx?scanid=1017574&projectid=1025&pathid=4 |
| Status | New |
| Detection Date | 3/11/2022 11:39:24 AM |

Method getRandomNumberBetween0And at line 67 of app/src/assets/DummyData.js uses a weak method random to produce random values. These values might be used as personal identifiers, session tokens or cryptographic input; however, due to their insufficient randomness, an attacker may be able to derive their value.

| | Source | Destination |
|---|---|---|
| File | app/src/assets/DummyData.js | app/src/assets/DummyData.js |
| Line | 68 | 117 |
| Object | random | getRandomAddress |

Code Snippet
File Name app/src/assets/DummyData.js
Method function getRandomNumberBetween0And(max) {

```
....
68.       return Math.ceil((Math.random() * max)) - 1;
```

File Name app/src/assets/DummyData.js
Method function makeRandomInstallationPermit() {

```
....
117.      const installation = getRandomAddress([contact.index]);
```

**Client Insecure Randomness\Path 3:**

| | |
|---|---|
| Severity | Low |
| Result State | To Verify |
| Online Results | https://cxmanager.pnl.gov/CxWebClient/ViewerMain.aspx?scanid=1017574&projectid=1025&pathid=5 |
| Status | New |
| Detection Date | 3/11/2022 11:39:24 AM |

Method getRandomNumberBetween0And at line 67 of app/src/assets/DummyData.js uses a weak method random to produce random values. These values might be used as personal identifiers, session tokens or cryptographic input; however, due to their insufficient randomness, an attacker may be able to derive their value.

| | Source | Destination |
|---|---|---|
| File | app/src/assets/DummyData.js | app/src/assets/DummyData.js |
| Line | 68 | 118 |
| Object | random | getRandomAddress |

Code Snippet

File Name app/src/assets/DummyData.js

Method function getRandomNumberBetween0And(max) {

```
....
68.      return Math.ceil((Math.random() * max)) - 1;
```

File Name app/src/assets/DummyData.js

Method function makeRandomInstallationPermit() {

```
....
118.      const contractor = getRandomAddress([contact.index,
installation.index]);
```

**Client Insecure Randomness\Path 4:**

| | |
|---|---|
| Severity | Low |
| Result State | To Verify |
| Online Results | https://cxmanager.pnl.gov/CxWebClient/ViewerMain.aspx?scanid=1017574&projectid=1025&pathid=6 |
| Status | New |
| Detection Date | 3/11/2022 11:39:24 AM |

Method getRandomNumberBetween0And at line 67 of app/src/assets/DummyData.js uses a weak method random to produce random values. These values might be used as personal identifiers, session tokens or cryptographic input; however, due to their insufficient randomness, an attacker may be able to derive their value.

| | Source | Destination |
|---|---|---|
| File | app/src/assets/DummyData.js | app/src/assets/DummyData.js |
| Line | 68 | 119 |
| Object | random | getRandomAddress |

| | |
|---|---|
| Code Snippet | |
| File Name | app/src/assets/DummyData.js |
| Method | function getRandomNumberBetween0And(max) { |

```
....
68.       return Math.ceil((Math.random() * max)) - 1;
```

| | |
|---|---|
| File Name | app/src/assets/DummyData.js |
| Method | function makeRandomInstallationPermit() { |

```
....
119.      const electrician = getRandomAddress([contact.index,
installation.index, contractor.index]);
```

## React Deprecated

Query Path:
JavaScript\Cx\JavaScript Low Visibility\React Deprecated Version:3

### Categories

OWASP Top 10 2017: A9-Using Components with Known Vulnerabilities
OWASP Top 10 2021: A6-Vulnerable and Outdated Components

*Description*

**React Deprecated\Path 1:**

| | |
|---|---|
| Severity | Low |
| Result State | To Verify |
| Online Results | https://cxmanager.pnl.gov/CxWebClient/ViewerMain.aspx?scanid=1017574&projectid=1025&pathid=8 |
| Status | New |
| Detection Date | 3/11/2022 11:39:24 AM |

Method ReactDOM.render in app/src/index.jsx, at line 59, calls an obsolete API, render. This has been deprecated, and should not be used in a modern codebase.

| | Source | Destination |
|---|---|---|
| File | app/src/index.jsx | app/src/index.jsx |
| Line | 59 | 59 |
| Object | render | render |

| | |
|---|---|
| Code Snippet | |
| File Name | app/src/index.jsx |
| Method | ReactDOM.render( |

```
....
59.  ReactDOM.render(
```

**React Deprecated\Path 2:**

| | |
|---|---|
| Severity | Low |
| Result State | To Verify |
| Online Results | https://cxmanager.pnl.gov/CxWebClient/ViewerMain.aspx?scanid=1017574&projectid=1025&pathid=9 |

| | |
|---|---|
| Status | New |
| Detection Date | 3/11/2022 11:39:24 AM |

Method Header in app/src/components/Header.jsx, at line 6, calls an obsolete API, CxAssociativeArray_dc8fb2be. This has been deprecated, and should not be used in a modern codebase.

| | Source | Destination |
|---|---|---|
| File | app/src/components/Header.jsx | app/src/components/Header.jsx |
| Line | 28 | 28 |
| Object | CxAssociativeArray_dc8fb2be | CxAssociativeArray_dc8fb2be |

| | |
|---|---|
| Code Snippet | |
| File Name | app/src/components/Header.jsx |
| Method | export default function Header() { |

```
....
28.                     <img src={Logo} alt="LOGO" className="logo"></img>
```

## Log Forging

Query Path:
JavaScript\Cx\JavaScript Server Side Vulnerabilities\Log Forging Version:3

### Categories

FISMA 2014: System And Information Integrity
NIST SP 800-53: AU-9 Protection of Audit Information (P1)
OWASP Top 10 2017: A1-Injection
ASD STIG 4.10: APSC-DV-002560 - CAT I The application must not be subject to input handling vulnerabilities.
OWASP Top 10 2021: A9-Security Logging and Monitoring Failures

*Description*

**Log Forging\Path 1:**

| | |
|---|---|
| Severity | Low |
| Result State | To Verify |
| Online Results | https://cxmanager.pnl.gov/CxWebClient/ViewerMain.aspx?scanid=1017574&projectid=1025&pathid=11 |
| Status | New |
| Detection Date | 3/11/2022 11:39:24 AM |

Method app.post at line 47 of app/server.js gets user input from element body. This element's value flows through the code without being properly sanitized or validated, and is eventually used in writing an audit log in .then at line 49 of app/server.js.
This may enable Log Forging.

| | Source | Destination |
|---|---|---|
| File | app/server.js | app/server.js |
| Line | 48 | 49 |
| Object | body | log |

| | |
|---|---|
| Code Snippet | |
| File Name | app/server.js |
| Method | app.post('/api/addPermit', function (req, res) { |

```
....
48.     insertInstallationPermit(req.body)
```

File Name app/server.js

Method .then(res => console.log(res.rows[0]))

```
....
49.       .then(res => console.log(res.rows[0]))
```

**Log Forging\Path 2:**

| | |
|---|---|
| Severity | Low |
| Result State | To Verify |
| Online Results | https://cxmanager.pnl.gov/CxWebClient/ViewerMain.aspx?scanid=1017574&projectid=1025&pathid=12 |
| Status | New |
| Detection Date | 3/11/2022 11:39:24 AM |

Method app.post at line 47 of app/server.js gets user input from element body. This element's value flows through the code without being properly sanitized or validated, and is eventually used in writing an audit log in .catch at line 50 of app/server.js.
This may enable Log Forging.

| | Source | Destination |
|---|---|---|
| File | app/server.js | app/server.js |
| Line | 48 | 50 |
| Object | body | error |

Code Snippet

File Name app/server.js

Method app.post('/api/addPermit', function (req, res) {

```
....
48.     insertInstallationPermit(req.body)
```

File Name app/server.js

Method .catch(e => console.error(e.stack));

```
....
50.       .catch(e => console.error(e.stack));
```

## Client Hardcoded Domain

Query Path:
JavaScript\Cx\JavaScript Low Visibility\Client Hardcoded Domain Version:4

### Categories

NIST SP 800-53: SC-18 Mobile Code (P2)
ASD STIG 4.10: APSC-DV-002490 - CAT I The application must protect from Cross-Site Scripting (XSS) vulnerabilities.
OWASP Top 10 2021: A8-Software and Data Integrity Failures

*Description*

**Client Hardcoded Domain\Path 1:**

| | |
|---|---|
| Severity | Low |
| Result State | To Verify |
| Online Results | https://cxmanager.pnl.gov/CxWebClient/ViewerMain.aspx?scanid=1017574&projectid=1025&pathid=7 |
| Status | New |
| Detection Date | 3/11/2022 11:39:24 AM |

The JavaScript file imported in https://static2.sharepointonline.com/files/fabric/office-ui-fabric-core/11.0.0/css/fabric.min.css in app/dist/index.html at line 4 is from a remote domain, which may allow attackers to replace its contents with malicious code.

| | Source | Destination |
|---|---|---|
| File | app/dist/index.html | app/dist/index.html |
| Line | 4 | 4 |
| Object | https://static2.sharepointonline.com/files/fabric/office-ui-fabric-core/11.0.0/css/fabric.min.css | https://static2.sharepointonline.com/files/fabric/office-ui-fabric-core/11.0.0/css/fabric.min.css |

Code Snippet

File Name app/dist/index.html

Method <link rel="stylesheet" href="https://static2.sharepointonline.com/files/fabric/office-ui-fabric-core/11.0.0/css/fabric.min.css" />

```
....
4.          <link rel="stylesheet"
href="https://static2.sharepointonline.com/files/fabric/office-ui-
fabric-core/11.0.0/css/fabric.min.css" />
```

## Potential Clickjacking on Legacy Browsers

Query Path:

JavaScript\Cx\JavaScript Low Visibility\Potential Clickjacking on Legacy Browsers Version:3

### Categories

FISMA 2014: Configuration Management
NIST SP 800-53: SC-8 Transmission Confidentiality and Integrity (P1)
ASD STIG 4.10: APSC-DV-002330 - CAT II The application must protect the confidentiality and integrity of stored information when required by DoD policy or the information owner.

*Description*

**Potential Clickjacking on Legacy Browsers\Path 1:**

| | |
|---|---|
| Severity | Low |
| Result State | To Verify |
| Online Results | https://cxmanager.pnl.gov/CxWebClient/ViewerMain.aspx?scanid=1017574&projectid=1025&pathid=10 |
| Status | New |
| Detection Date | 3/11/2022 11:39:24 AM |

The application does not protect the web page app/dist/index.html from clickjacking attacks in legacy browsers, by using framebusting scripts.

| | Source | Destination |
|---|---|---|
| File | app/dist/index.html | app/dist/index.html |
| Line | 1 | 1 |

| | | |
|---|---|---|
| Object | < | < |

Code Snippet
File Name app/dist/index.html
Method <!DOCTYPE html>

```
....
1.  <!DOCTYPE html>
```

## Missing CSP Header

Query Path:
JavaScript\Cx\JavaScript Server Side Vulnerabilities\Missing CSP Header Version:3

### Categories

OWASP Top 10 2017: A6-Security Misconfiguration
OWASP Top 10 2021: A7-Identification and Authentication Failures

*Description*
**Missing CSP Header\Path 1:**

| | |
|---|---|
| Severity | Low |
| Result State | To Verify |
| Online Results | https://cxmanager.pnl.gov/CxWebClient/ViewerMain.aspx?scanid=1017574&projectid=1025&pathid=13 |
| Status | New |
| Detection Date | 3/11/2022 11:39:24 AM |

A Content Security Policy is not explicitly defined within the web-application.

| | Source | Destination |
|---|---|---|
| File | app/server.js | app/server.js |
| Line | 35 | 35 |
| Object | send | send |

Code Snippet
File Name app/server.js
Method app.get('/api/getPermit', function (req, res) {

```
....
35.     return res.send('addPermit');
```

## Potentially Vulnerable To Csrf

Query Path:
JavaScript\Cx\JavaScript Server Side Vulnerabilities\Potentially Vulnerable To Csrf Version:3

### Categories

OWASP Top 10 2021: A1-Broken Access Control

*Description*
**Potentially Vulnerable To Csrf\Path 1:**

| | |
|---|---|
| Severity | Low |
| Result State | To Verify |
| Online Results | https://cxmanager.pnl.gov/CxWebClient/ViewerMain.aspx?scanid=1017574&projectid=1025&pathid=14 |
| Status | New |

| | |
|---|---|
| Detection Date | 3/11/2022 11:39:24 AM |

Method express at line 11 of app/server.js gets a parameter from a user request from app. This parameter value flows through the code and is eventually used to access application state altering functionality. This may enable Cross-Site Request Forgery (CSRF).

| | Source | Destination |
|---|---|---|
| File | app/server.js | app/server.js |
| Line | 11 | 11 |
| Object | app | app |

Code Snippet
File Name app/server.js
Method const app = express();

```
....
11.  const app = express();
```

## Use Of Hardcoded Password

Query Path:
JavaScript\Cx\JavaScript Server Side Vulnerabilities\Use Of Hardcoded Password Version:6

### Categories

FISMA 2014: Identification And Authentication
NIST SP 800-53: SC-28 Protection of Information at Rest (P1)
OWASP Top 10 2017: A3-Sensitive Data Exposure
ASD STIG 4.10: APSC-DV-001740 - CAT I The application must only store cryptographic representations of passwords.
OWASP Top 10 2021: A7-Identification and Authentication Failures

*Description*

**Use Of Hardcoded Password\Path 1:**

| | |
|---|---|
| Severity | Low |
| Result State | To Verify |
| Online Results | https://cxmanager.pnl.gov/CxWebClient/ViewerMain.aspx?scanid=1017574&projectid=1025&pathid=15 |
| Status | New |
| Detection Date | 3/11/2022 11:39:24 AM |

The application uses the hard-coded password "postgres" for authentication purposes, either using it to verify users' identities, or to access another remote system. This password at line 7 of app/server/postgresQueries.js appears in the code, implying it is accessible to anyone with source code access, and cannot be changed without rebuilding the application.

| | Source | Destination |
|---|---|---|
| File | app/server/postgresQueries.js | app/server/postgresQueries.js |
| Line | 7 | 7 |
| Object | "postgres" | password |

Code Snippet
File Name app/server/postgresQueries.js
Method password: process.env.POSTGRES_PASSWORD || 'postgres',

```
....
7.      password: process.env.POSTGRES_PASSWORD || 'postgres',
```

# SQL Injection

## Risk

**What might happen**

An attacker could directly access all of the system's data. The attacker would likely be able to steal any sensitive information stored by the system, including private user information, credit card details, proprietary business data, and any other secret data. Likewise, the attacker could possibly modify or erase existing data, or even add new bogus data. In some scenarios, it may even be possible to execute code on the database.
In addition to disclosing or altering confidential information directly, this vulnerability might also be used to achieve secondary effects, such as bypassing authentication, subverting security checks, or forging a data trail.
Further increasing the likelihood of exploit is the fact that this flaw is easy for attackers to find, and easy to exploit.

---

## Cause

**How does it happen**

The application stores and manages data in a database, by submitting a textual SQL query to the database engine for processing. The application creates the query by simple string concatenation, embedding untrusted data. However, there is no separation between data and code; furthermore, the embedded data is neither checked for data type validity nor subsequently sanitized. Thus, the untrusted data could contain SQL commands, or modify the intended query. The database would interpret the altered query and commands as if they originated from the application, and execute them accordingly.
Note that an attacker can exploit this vulnerability either by modifying the URL, or by submitting malicious data in the user input or other request fields.

---

## General Recommendations

**How to avoid it**

- Validate all untrusted data, regardless of source. Validation should be based on a whitelist: accept only data fitting a specified structure, rather than reject bad patterns.
- In particular, check for:
  - Data type
  - Size
  - Range
  - Format
  - Expected values.
- Restrict access to database objects and functionality, according to the Principle of Least Privilege.
- Do not use dynamically concatenate strings to construct SQL queries.
- Prefer using DB Stored Procedures for all data access, instead of ad-hoc dynamic queries.
- Instead of unsafe string concatenation, use secure database components such as parameterized queries and object bindings (for example, commands and parameters).
- Alternatively, an even better solution is to use an ORM library, in order to pre-define and encapsulate the allowed commands enabled for the application, instead of dynamically accessing the database directly. In this way the code plane and data plane should be isolated from each other.

---

## Source Code Examples

**JavaScript**
**SQL Injection in "id" Parameter**

```
app.get('/profile/address', function(req, res) {
      var id = req.query.id;
      connection.query('SELECT * FROM users WHERE id=' + id, function(err,results) {
            var user = results[0];
```

```
            if (user)
                  res.render('address', {address: user.address})
            else
            res.render('addressNotFoundErrorPage');
      });
});
```

**SQL Query Uses Parameterized Queries to Avoid SQL Injection**

```
app.get('/profile/address', function(req, res) {
      var id = req.query.id;
      connection.query('SELECT * FROM users WHERE id=?',[id], function(err,results) {
            var user = results[0];
            if (user)
                  res.render('address', {address: user.address})
            else
            res.render('addressNotFoundErrorPage');
      });
});
```

# Missing HSTS Header

## Risk

**What might happen**

Failure to set an HSTS header and provide it with a reasonable "max-age" value of at least one year may leave users vulnerable to Man-in-the-Middle attacks.

---

## Cause

**How does it happen**

Many users browse to websites by simply typing the domain name into the address bar, without the protocol prefix. The browser will automatically assume that the user's intended protocol is HTTP, instead of the encrypted HTTPS protocol.
When this initial request is made, an attacker can perform a Man-in-the-Middle attack and manipulate it to redirect users to a malicious web-site of the attacker's choosing. To protect the user from such an occurence, the HTTP Strict Transport Security (HSTS) header instructs the user's browser to disallow use of an unsecure HTTP connection to the the domain associated with the HSTS header.
Once a browser that supports the HSTS feature has visited a web-site and the header was set, it will no longer allow communicating with the domain over an HTTP connection.
Once an HSTS header was issued for a specific website, the browser is also instructed to prevent users from manually overriding and accepting an untrusted SSL certificate for as long as the "max-age" value still applies. The recommended "max-age" value is for at least one year in seconds, or 31536000.

---

## General Recommendations

**How to avoid it**

- Before setting the HSTS header - consider the implications it may have:
  - Forcing HTTPS will prevent any future use of HTTP, which could hinder some testing
  - Disabling HSTS is not trivial, as once it is disabled on the site, it must also be disabled on the browser
- Set the HSTS header either explicitly within application code, or using web-server configurations.
- Ensure the "max-age" value for HSTS headers is set to 31536000 to ensure HSTS is strictly enforced for at least one year.
- Include the "includeSubDomains" to maximize HSTS coverage, and ensure HSTS is enforced on all sub-domains under the current domain
  - Note that this may prevent secure browser access to any sub-domains that utilize HTTP; however, use of HTTP is very severe and highly discouraged, even for websites that do not contain any sensitive information, as their contents can still be tampered via Man-in-the-Middle attacks to phish users under the HTTP domain.
- Once HSTS has been enforced, submit the web-application's address to an HSTS preload list - this will ensure that, even if a client is accessing the web-application for the first time (implying HSTS has not yet been set by the web-application), a browser that respects the HSTS preload list would still treat the web-application as if it had already issued an HSTS header. Note that this requires the server to have a trusted SSL certificate, and issue an HSTS header with a maxAge of 1 year (31536000)
- Note that this query is designed to return one result per application. This means that if more than one vulnerable response without an HSTS header is identified, only the first identified instance of this issue will be highlighted as a result. If a misconfigured instance of HSTS is identified (has a short lifespan, or is missing the "includeSubDomains" flag), that result will be flagged. Since HSTS is required to be enforced across the entire application to be considered a secure deployment of HSTS functionality, fixing this issue only where the query highlights this result is likely to produce subsequent results in other sections of the application; therefore, when adding this header via code, ensure it is uniformly deployed across the entire application. If this header is added via configuration, ensure that this configuration applies to the entire application.
- Note that misconfigured HSTS headers that do not contain the recommended max-age value of at least one year or the "includeSubDomains" flag will still return a result for a missing HSTS header.

---

## Source Code Examples

**JavaScript**
**Using Helmet with Express**

```
var express = require('express')
var helmet = require('helmet') // Helmet includes HSTS, defined to one year and with
```

```
"includeSubDomains", as a built-in header

var app = express()
app.use(helmet())
```

**Using Explicit HSTS Package - Built into Helmet, So Either 'HSTS' or 'Helmet' Can Be Used**

```
var hsts = require('hsts')

app.use(hsts({
  maxAge: 31536000,
  includeSubDomains: true // Also enabled by default
}))
```

**Explicitly Setting HSTS Header in Code**

```
res.setHeader("Strict-Transport-Security", "max-age=31536000; includeSubDomains");
```

# Client Insecure Randomness

## Risk

**What might happen**

Random values are often used as a mechanism to prevent malicious users from knowing or predicting a given value, such as a password, encryption key, or session identifier. Depending on what this random value is used for, an attacker would be able to predict the next numbers generated, or previously generated values, based on sources often used to derive certain randomness; however, while they may seem random, large statistical samples would demonstrate that they are insufficiently random, producing a much smaller space of possible "random" values than a truly random sample would. This could enable an attacker to derive or guess this value, and thus hijack another user's session, impersonate another user, or crack an encryption key (depending on what the pseudo-random value was used for).

---

## Cause

**How does it happen**

The application uses a weak method of generating pseudo-random values, such that other numbers could be determined from a relatively small sample size. Since the pseudo-random number generator used is designed for statistically uniform distribution of values, it is approximately deterministic. Thus, after collecting a few generated values, it would be possible for an attacker to calculate past or future values.
Specifically, if this pseudo-random value is used in any security context, such as one-time passwords, keys, secret identifiers or salts - an attacker would likely be able to predict the next value generated and steal it, or guess a previously generated value and spoof its original intent.

---

## General Recommendations

**How to avoid it**

- Always use a cryptographically secure pseudo-random number generator, instead of basic random methods, particularly when dealing with a security context
- Use the cryptorandom generator that is built-in to your language or platform, and ensure it is securely seeded. Do not seed the generator with a weak, non-random seed. (In most cases, the default is securely random).
- Ensure you use a long enough random value, thus making brute-force attacks unfeasible.

---

## Source Code Examples

**JavaScript**

**Use of Math.Random() To Generate A Random Number Between MIN And MAX**

```
var randomValue = Math.floor(Math.Random() * MAX_RANGE - MIN_RANGE + 1) + MIN_RANGE;
```

**Use of CSPRNG To Generate a Random Uint32Array**

```
// Assuming window.crypto.getRandomValues is available

var sessionId = new Uint32Array(10);
window.crypto.getRandomValues(sessionId);
```

# Client Hardcoded Domain

## Risk

**What might happen**

An externally imported Javascript file may leave users vulnerable to attack - if the Javascript's host is compromised, if communications with the host are intercepted or if the host itself is not trustworthy, then the contents of the Javascript file may change to have malicious code, which could result in a Cross-Site Scripting (XSS) attack.

---

## Cause

**How does it happen**

Javascript files can be imported dynamically from remote hosts when they are embedded into HTML. However, this reliance on a remote host for these scripts may diminish security, as web-application's users are only ever as secure as the remote host serving these Javascript files.

---

## General Recommendations

**How to avoid it**

Where possible, host all script files locally, rather than remotely. Ensure that locally hosted 3rd party script files are constantly updated and maintained.

---

## Source Code Examples

**JavaScript**

**Remote Importation of A Script File**

```
<script src="https://example.com/scripts/jquery.js" />
```

**Local Importation of A Script File**

```
<script src="/scripts/jquery.js" />
```

# React Deprecated

## Risk

**What might happen**

Referencing deprecated modules can cause an application to be exposed to known vulnerabilities, that have been publicly reported and already fixed. A common attack technique is to scan applications for these known vulnerabilities, and then exploit the application through these deprecated versions. However, even if deprecated code is used in a way that is completely secure, its very use and inclusion in the code base would encourage developers to re-use the deprecated element in the future, potentially leaving the application vulnerable to attack, which is why deprecated code should be eliminated from the code-base as a matter of practice. Note that the actual risk involved depends on the specifics of any known vulnerabilities in older versions.
Use of a deprecated API on client code may leave users vulnerable to browser-based attacks; this is exacerbated by the fact client-side code is available to any attacker with client access, who may be able to trivially detect use of this deprecated API.

---

## Cause

**How does it happen**

The application references code elements that have been declared as deprecated. This could include classes, functions, methods, properties, modules, or obsolete library versions that are either out of date by version, or have been entirely deprecated. It is likely that the code that references the obsolete element was developed before it was declared as obsolete, and in the meantime the referenced code was updated.

---

## General Recommendations

**How to avoid it**

- Always prefer to use the most updated versions of libraries, packages, and other dependancies.
- Do not use or reference any class, method, function, property, or other element that has been declared deprecated.

---

## Source Code Examples

**JavaScript**
**ReactJS - Using a Deprecated Method to Interact with DOM**

```
// Using findDOMNode to access a component is highly discouraged, because it breaks React component abstraction by treating it like a normal Javascript DOM object; this may result in unexpected or dangerous behavior
ReactDOM.findDOMNode(component);
```

**Obtain Year via Deprecated JavaScript Method**

```
var d = new Date();
var year = d.getYear(); // getYear() is deprecated and affected by Y2K; for a given year, 20xx, it will return 1xx.
```

**Obtain Year via a Supported JavaScript Method**

```
var d = new Date();
var year = d.getFullYear();
```

**Invoking a Deprecated Function, Denoted Using JSDoc**

```
/** @deprecated */
function myOldFunction() {
      /* Code that is deprecated */
}

myOldFunction();
```

# Potential Clickjacking on Legacy Browsers

## Risk

**What might happen**

Clickjacking attacks allow an attacker to "hijack" a user's mouse clicks on a webpage, by invisibly framing the application, and superimposing it in front of a bogus site. When the user is convinced to click on the bogus website, e.g. on a link or a button, the user's mouse is actually clicking on the target webpage, despite being invisible.
This could allow the attacker to craft an overlay that, when clicked, would lead the user to perform undesirable actions in the vulnerable application, e.g. enabling the user's webcam, deleting all the user's records, changing the user's settings, or causing clickfraud.

---

## Cause

**How does it happen**

The root cause of vulnerability to a clickjacking attack, is that the application's web pages can be loaded into a frame of another website. The application does not implement a proper frame-busting script, that would prevent the page from being loaded into another frame. Note that there are many types of simplistic redirection scripts that still leave the application vulnerable to clickjacking techniques, and should not be used.
When dealing with modern browsers, applications mitigate this vulnerability by issuing appropriate Content-Security-Policy or X-Frame-Options headers to indicate to the browser to disallow framing. However, many legacy browsers do not support this feature, and require a more manual approach by implementing a mitigation in Javascript. To ensure legacy support, a framebusting script is required.

---

## General Recommendations

**How to avoid it**

Generic Guidance:

- Define and implement a a Content Security Policy (CSP) on the server side, including a frame-ancestors directive. Enforce the CSP on all relevant webpages.
- If certain webpages are required to be loaded into a frame, define a specific, whitelisted target URL.
- Alternatively, return a "X-Frame-Options" header on all HTTP responses. If it is necessary to allow a particular webpage to be loaded into a frame, define a specific, whitelisted target URL.
- For legacy support, implement framebusting code using Javascript and CSS to ensure that, if a page is framed, it is never displayed, and attempt to navigate into the frame to prevent attack. Even if navigation fails, the page is not displayed and is therefore not interactive, mitigating potential clickjacking attacks.

Specific Recommendations:

- Implement a proper framebuster script on the client, that is not vulnerable to frame-buster-busting attacks.
    - Code should first disable the UI, such that even if frame-busting is successfully evaded, the UI cannot be clicked. This can be done by setting the CSS value of the "display" attribute to "none" on either the "body" or "html" tags. This is done because, if a frame attempts to redirect and become the parent, the malicious parent can still prevent redirection via various techniques.
    - Code should then determine whether no framing occurs by comparing self === top; if the result is true, can the UI be enabled. If it is false, attempt to navigate away from the framing page by setting the top.location attribute to self.location.

---

## Source Code Examples

**JavaScript**
**Clickjackable Webpage**

```
<html>
    <body>

     <button onclick="clicked();">
            Click here if you love ducks
        </button>
    </body>
```

```
</html>
```

**Bustable Framebuster**

```
<html>
    <head>
     <script>
            if ( window.self.location != window.top.location ) {
                  window.top.location = window.self.location;
            }
        </script>
    </head>

    <body>
     <button onclick="clicked();">
            Click here if you love ducks
        </button>
    </body>
</html>
```

**Proper Framebusterbusterbusting**

```
<html>
    <head>
     <style> html {display : none; } </style>
        <script>
            if ( self === top ) {
                  document.documentElement.style.display = 'block';
            }
            else {
                  top.location = self.location;
            }
        </script>
    </head>

    <body>
     <button onclick="clicked();">
            Click here if you love ducks
        </button>
    </body>
</html>
```

# Log Forging

## Risk

**What might happen**

An attacker could engineer audit logs of security-sensitive actions and lay a false audit trail, potentially implicating an innocent user or hiding an incident.

---

## Cause

**How does it happen**

The application writes audit logs upon security-sensitive actions. Since the audit log includes user input that is neither checked for data type validity nor subsequently sanitized, the input could contain false information made to look like legitimate audit log data,

---

## General Recommendations

**How to avoid it**

1. Validate all input, regardless of source. Validation should be based on a whitelist: accept only data fitting a specified structure, rather than reject bad patterns. Check for:
    - Data type
    - Size
    - Range
    - Format
    - Expected values
2. Validation is not a replacement for encoding. Fully encode all dynamic data, regardless of source, before embedding it in logs.
3. Use a secure logging mechanism.

---

## Source Code Examples

**JavaScript**

**Passing Unsanitized Values to HAPI server.log()**

```
var id = request.query["id"];
try {
    var val = tryGetById(id); // Assume this throws an exception if "id" is not found
      // Handle val
}
catch(err) {
    server.log(['error','id'],id); // Log unsanitized values, which could also not be
sanitized downstream, and could contain CRLF
}
```

**Passing Sanitized Values to HAPI server.log()**

```
var id = request.query["id"];
try {
    var val = tryGetById(id); // Assume this throws an exception if "id" is not found
      // Handle val
}
catch(err) {
    server.log(['error','id'],encodeURI(id)); // encodeURI() is a sufficient sanitizer for
CRLF, as it URL-encodes the line break characters
}
```

# Missing CSP Header

## Risk

**What might happen**

The Content-Security-Policy header enforces that the source of content, such as the origin of a script, embedded (child) frame, embedding (parent) frame or image, are trusted and allowed by the current web-page; if, within the web-page, a content's source does not adhere to a strict Content Security Policy, it is promptly rejected by the browser. Failure to define a policy may leave the application's users exposed to Cross-Site Scripting (XSS) attacks, Clickjacking attacks, content forgery and more.

---

## Cause

**How does it happen**

The Content-Security-Policy header is used by modern browsers as an indicator for trusted sources of content, including media, images, scripts, frames and more. If these policies are not explicitly defined, default browser behavior would allow untrusted content. The application creates web responses, but does not properly set a Content-Security-Policy header.

---

## General Recommendations

**How to avoid it**

Explicitly set the Content-Security-Policy headers for all applicable policy types (frame, script, form, script, media, img etc.) according to business requirements and deployment layout of external file hosting services. Specifically, do not use a wildcard, '*', to specify these policies, as this would allow content from any external resource.
The Content-Security-Policy can be explicitly defined within web-application code, as a header managed by web-server configurations, or within `<meta>` tags in the HTML `<head>` section.

---

## Source Code Examples

**JavaScript**
**Setting The CSP Header Explicitly**

```
app.use(function(req, res, next) {
    res.setHeader("Content-Security-Policy", "script-src 'self'");
    return next();
});
```

# Potentially Vulnerable To Csrf

## Risk

**What might happen**

An attacker could cause the victim to perform any action for which the victim is authorized, such as transferring funds from the victim's account to the attacker's. The action will be logged as being performed by the victim, in the context of their account, and potentially without their knowledge that this action has occurred.

---

## Cause

**How does it happen**

The application performs some action that modifies database contents, based purely on HTTP request content, and does not require per-request renewed authentication (such as transaction authentication or a synchronizer token), instead relying solely on session authentication. This means that an attacker could use social engineering to cause a victim to browse to a link which contains a transaction request to the vulnerable application, submitting that request from the user's browser. Once the application receives the request, it would trust the victim's session, and would perform the action. This type of attack is known as Cross-Site Request Forgery (CSRF).

A Cross-Site Request Forgery attack relies on the trust between a server and an authenticated client. By only validating the session, the server ensures that a request has emerged from a client's web-browser. However, any website may submit GET and POST requests to other websites, to which the browser will automatically add the session token if it is in a cookie. This cross-site request can then be trusted as arriving from the user's browser, but does not validate that it was their intent was to make this request.

In some cases, XSRF protection functionality exists in the application, but is either not implemented, or explicitly disabled.

---

## General Recommendations

**How to avoid it**

Mitigating CSRF requires an additional layer of authentication that is built into the request validation mechanism. This mechanism would attach an additional token that only applies to the given user; this token would be available within the user's web-page, but will not be attached automatically to a request from a different website (e.g. not stored in a cookie). Since the token is not automatically attached to the request, and is not available to the attacker, and is required by the server to process the request, it would be completely impossible for the attacker to fill in a valid cross-site form that contains this token.

Many platforms offer built-in CSRF mitigation functionality which should be used, and perform this type of token management under the hood. Alternatively, use a known or trusted library which adds this functionality.

If implementing CSRF protection is required, this protection should adhere to the following rules:

- Any state altering form (Create, Update, Delete operations) should enforce CSRF protection, by adding an CSRF token to every state altering form submission on the client.
- An CSRF token should be generated, and be unique per-user per-session (and, preferably, per request).
- The CSRF token should be inserted into the client side form, and be submitted to the server as part of the form request. For example, it could be a hidden field in an HTML form, or a custom header added by a Javascript request.
- The CSRF token in the request body or custom header must then be verified as belonging to the current user by the server, before a request is authorized and processed as valid.

Always rely on best practices when using XSRF protection - always enable built-in functionality or available libraries where possible.

When using application-wide XSRF protection, never explicitly disable or subvert XSRF protection for specific functionality unless said functionality has been thoroughly verified to not require XSRF protection.

---

## Source Code Examples

**JavaScript**

**Applying 'csurf' Library to HTML Forms in Express**

```
app.get('/profile/email/edit', (req, res) => {
  // Add csrfToken() as a value inside the form, so that the token is submitted in the
request
  res.send(`

  <h1>Change E-Mail Form</h1>
```

```
  <form action="/profile/email/edit" method="POST">
    <input id="email" name="email" type="text" />
        <input id="email" name="verify_email" type="text" />
    <input type="submit" value="Ok" />
    <input type="hidden" name="_csrf" value="${req.csrfToken()}" />
     </form>
  `);
});

// If the _csrf token in the request body is not equal to the synchronizer token, the POST
request will be rejected, and this method will not trigger
app.post('/profile/email/edit', (req, res) => {
      // Authenticate user session
      // Update email address
}
```

# Use Of Hardcoded Password

## Risk

**What might happen**

Hardcoded passwords expose the application to password leakage. If an attacker gains access to the source code, she will be able to steal the embedded passwords, and use them to impersonate a valid user. This could include impersonating end users to the application, or impersonating the application to a remote system, such as a database or a remote web service.
Once the attacker succeeds in impersonating the user or application, she will have full access to the system, and be able to do anything the impersonated identity could do.

---

## Cause

**How does it happen**

The application codebase has string literal passwords embedded in the source code. This hardcoded value is used either to compare to user-provided credentials, or to authenticate downstream to a remote system (such as a database or a remote web service).
An attacker only needs to gain access to the source code to reveal the hardcoded password. Likewise, the attacker can reverse engineer the compiled application binaries, and easily retrieve the embedded password. Once found, the attacker can easily use the password in impersonation attacks, either directly on the application or to the remote system.
Furthermore, once stolen, this password cannot be easily changed to prevent further misuse, unless a new version of the application is compiled. Moreover, if this application is distributed to numerous systems, stealing the password from one system automatically allows a class break in to all the deployed systems.

---

## General Recommendations

**How to avoid it**

- Do not hardcode any secret data in source code, especially not passwords.
- In particular, user passwords should be stored in a database or directory service, and protected with a strong password hash (e.g. bcrypt, scrypt, PBKDF2, or Argon2). Do not compare user passwords with a hardcoded value.
- Sytem passwords should be stored in a configuration file or the database, and protected with strong encryption (e.g. AES-256). Encryption keys should be securely managed, and not hardcoded.

---

## Source Code Examples

**JavaScript**
**Hardcoded Account Password**

```
var username = request.body.username;
var password = request.body.password;
var admin_username = "admin";
var admin_password = "5up3r53cr3t";
if (username == admin_username && password == admin_password) {
      // Authenticate
}
else {
      // Reject
}
```

**Authenticating by Querying the Database with Credentials**

```
var username = request.body.username;
var password = secureHashImplementation(request.body.password);

connection.query('SELECT * FROM users WHERE name=? AND hashed_password=?',[username,
password], function(err,results) {
      if (error) {
```

```
            // handle error
    }
      if (results.length == 1) {
      // Authenticate
    }
});
```

## Scanned Languages

| Language | Hash Number | Change Date |
|---|---|---|
| JavaScript | 9095271965336651 | 1/14/2022 |
| VbScript | 0386000544005133 | 12/9/2021 |
| Common | 0318477963775793 | 1/14/2022 |

# Pacific Northwest National Laboratory

902 Battelle Boulevard
P.O. Box 999
Richland, WA 99354
1-888-375-PNNL (7665)

***www.pnnl.gov***