Web Application Security

Assessment Report

For [REPLACED\_COMPANY] Inc

Report date: 2/14/2020

# Table of Contents

Table of Contents 2

Executive Summary 3

Assumptions and Constraints 3

Objectives & Scope 4

Major Findings 4

Strategic Recommendations 4

Technology Recommendations 5

Conclusion 5

Current State Analysis 6

Web Application Assessment 6

Summary of Weaknesses 6

HIGH risk findings 8

MEDIUM risk findings 14

LOW risk findings 18

Informational Findings 30

OWASP TOP 10 COMPARISON 33

SANS Top 25 Comparison 34

APPENDIX 1: OWASP ASVS LEVEL 1 CHECKLIST FOR WEB APPLICATION 36

# Executive Summary

[REPLACED\_COMPANY] engaged our company to perform a security assessment to enumerate possible attack vectors, evaluate existing security controls, and provide recommendations for improvement. Based on the evidence collected from the security assessment, our company has concluded that gaps in security coverage do exist for [REPLACED\_COMPANY]. Specifically, our company obtained access to technical information that wasn’t sufficiently protected. This technical information could be used to open up other doors for hackers to get inside [REPLACED\_COMPANY] System in the future. [REPLACED\_COMPANY] have remediated the high-risk vulnerability as indicated in this report. Areas of improvement have been identified, and [REPLACED\_COMPANY] should formulate a remediation plan for mitigating the rest of the findings discovered during the security assessment. Our company would like to thank [REPLACED\_COMPANY] for this opportunity to help the organization evaluate its current security posture.

## Assumptions and Constraints

[REPLACED\_COMPANY] application security assessment was performed in accordance with the OWASP ASVS Level 1 standard. The test falls into the category of Black Box test where testers have little to no information about the software and technology behind the platform and try different methods that real attackers would use. The security assessment was conducted in a manner designed to be as thorough as possible without causing interruption of services. Our company has performed security assessment on the following network endpoints:

* www.[REPLACED\_COMPANY].com – Web Application

[REPLACED\_COMPANY] helped our company to register a Payer account through the Web Application endpoint. [REPLACED\_COMPANY] also whitelisted our company’s IPs and provided credentials to authenticate into SFTP endpoint above. Other endpoints were tested without any assistance from [REPLACED\_COMPANY].

Manual penetration testing was performed to provide in-depth analysis and validate automated scan results. Existing vulnerabilities may not have been reported due to limitations in testing tools, and the necessity of a time-boxed testing approach.

Our company believes that the statements made in this document provide an accurate assessment in line with the Statement of Work. As the environment changes, and new vulnerabilities and risks are discovered and made public, an organization’s overall security posture will change. Such changes may affect the validity of this assessment report. Therefore the findings described in this report represent only a “snapshot” in time.

Our company’s tactical and strategic recommendations reflect best practices that will work for most organizations. Any change to a system, application, or process can have unintended consequences and lead to service disruptions. Changes should always be rolled out carefully, in small batches and tested thoroughly before full deployment. Special attention should be paid to unique use cases or circumstances of your organization that might be impacted by the included recommendations.

## Objectives & Scope

The assessment began on Feb 3rd, 2020 and concluded on Feb 12th, 2020. The security assessment followed our company’s comprehensive methodology to provide a thorough analysis of the [REPLACED\_COMPANY] web application and other endpoints. The shortcomings identified during the assessment were used to formulate recommendations and mitigation strategies for improving the overall security posture of [REPLACED\_COMPANY]. Specific scoping parameters for each phase of the assessment are described in detail under the Current State Analysis Section.

## Major Findings

Major findings represent a high-level overview of systemic issues identified within the environment and/or high risk vulnerabilities that warrant expedited attention. Our company believes that major findings represent the most important issues and that [REPLACED\_COMPANY] should prioritize its remediation efforts accordingly

**Technical Information exposed.** Our company was able to access the error logs for the [REPLACED\_COMPANY] web application that included following sensitive information:

* Session Ids
* Partial Private keys
* Account login emails
* Customer Bank Account and Routing Numbers

The finding was reported and remediated by [REPLACED\_COMPANY] within 24 hours.

**Account Enumeration Possible.** Our company was able to enumerate 9000 email addresses within few minutes to determine if they can be used as valid login credentials.

## Strategic Recommendations

Strategic initiatives are actions that should be taken to secure [REPLACED\_COMPANY]’s environment that can have wide-reaching effects.

**Perform Perimeter Audit.** Our company discovered that [REPLACED\_COMPANY] runs multiple systems for development, performance testing, execution of scheduled processes, privileged access and reporting. Keeping track of all publicly accessible endpoints is a crucial step in maintaining control over risk exposure of the organization. Our company recommends that full network perimeter audit to be performed to:

* Identify any publicly accessible endpoints that could be used as entry doors by hackers
* Perform a network scan on each discovered endpoint
* Perform periodic Dynamic Application Security Testing (DAST) on all public endpoints.

[REPLACED\_COMPANY] needs to maintain an inventory of all publicly exposed endpoints and perform vulnerability analysis for them on a regular basis.

## Technology Recommendations

**Avoid Logging Sensitive Information.** Sensitive information that could be used to gain access to [REPLACED\_COMPANY] account or perform financial transactions should not be logged. Our company recommends re-configuring [REPLACED\_COMPANY] Logging modules to avoid storing or transferring this information outside of [REPLACED\_COMPANY] application.

**Strengthen Authentication Controls.** Our company has identified multiple gaps in authentication controls that could lead to the compromise of individual customer accounts. [REPLACED\_COMPANY] should take measures to implement stronger authentication and session management controls to mitigate this risk.

## Conclusion

Our has concluded that gaps in security coverage exist for the [REPLACED\_COMPANY] web application and other endpoints in scope. [REPLACED\_COMPANY] has remediated high risk finding and only medium and low risk findings are outstanding. The OWASP Checklist at the end of this report, OWASP Top 10 and SANS Top 25 comparison as well as recommendations above are based on the re-tested state after the high-risk finding in this report has been remediated.

# Current State Analysis

Our company analyzed the current state of the security of the environment between the dates of February 3rd, 2020, and February 12, 2020. During the assessment, our company identified 1 high-risk issue, 1 medium-risk issues, 5 low-risk issues and 3 informational findings. The table below shows each finding grouped by risk and attack category.

| Category | Critical | High | Med | Low | Total |
| --- | --- | --- | --- | --- | --- |
| Malicious Input Handling | 0 | 0 | 0 | 0 | 0 |
| Broken Access Control | 0 | 1 | 0 | 2 | 3 |
| Authentication | 0 | 0 | 0 | 0 | 0 |
| Session Management | 0 | 0 | 0 | 1 | 1 |
| File and Resources | 0 | 0 | 0 | 0 | 0 |
| Configuration | 0 | 0 | 1 | 2 | 3 |
| Communications | 0 | 0 | 0 | 0 | 6 |

## Web Application Assessment

During the web application security assessment, our company identified vulnerabilities within the application. Our company examined all identified vulnerabilities to determine whether they can be exploited by an attacker to compromise targeted systems and/or used to gain access to sensitive information.

## Summary of Weaknesses

The detailed findings section described potential vulnerabilities, the likelihood or difficulty of exploitation, the relative level of impact of Company’s business, and recommendations. Vulnerabilities are arranged in order of business impact, with high-impact issues appearing first. The following findings have the potential to impact the confidentiality, integrity, and availability of Company’s assets.

**High Risk Findings**

* **Sentry Application Allows Public Access**

**Medium Risk Findings**

* **Account Enumeration Possible through Registration**

**Low Risk Findings**

* **Insufficient Password Policy**
* **Insertion of Sensitive Information into Log File**
* **API Auth token accessible to users with revoked permission**
* **"View Upload History" permission has no effect**
* **Uploaded Files accessible to limited account users**
* **No Session management capabilities**

**Informational Risk Findings**

* **OpenVPN Upgrade**
* **Python Dependency Upgrades**
* **SFTP Endpoint**

## HIGH risk findings

**Se****ntry Application Allows Public Access**



**Vulnerability description**![](data:None;base64...)

Sentry Application allows anyone to register and self-join a project without administrator approval. Hence allowing public access to detailed error logs and snippets of application code in stack traces. Our company was able to register 2 accounts and self-join **[REPLACED\_COMPANY]-app** team without being noticed by administrators, download **9GB** of error log information dating back to 2016, extract **5189** valid [REPLACED\_COMPANY] application usernames and **6 customer bank account numbers**. Our company was able to obtain partial private keys and individual **Session IDs**.

**Impact**

An attacker could gain un-obstructed access to application error logs and stack traces and use this information to understand the internal structure of the application and create further attack vectors that could affect integrity of the app. Attacker could use the session IDs obtained through logs to access customer accounts. Attacker could use discovered bank account numbers to initiate financial transactions (ACH) to/from said bank accounts.

**Pages Impacted**

https://sentry.[REPLACED\_COMPANY].com

**Verification and Attack Information**

Step 1: Register as new user

**[SCREENSHOT\_REPLACED\_FOR\_SECURITY\_REASON]**

Step 2: Join [REPLACED\_COMPANY]-app Team

**[SCREENSHOT\_REPLACED\_FOR\_SECURITY\_REASON]**

Step 3: Pick a Project: Production

**[SCREENSHOT\_REPLACED\_FOR\_SECURITY\_REASON]**

Step 4: Browse Error logs and stack traces

**[SCREENSHOT\_REPLACED\_FOR\_SECURITY\_REASON]**

Step 5: Request Auth Token

**[SCREENSHOT\_REPLACED\_FOR\_SECURITY\_REASON]**

Step 6: Get Auth Token

**[SCREENSHOT\_REPLACED\_FOR\_SECURITY\_REASON]**

Step 7: Get detailed event information through the API

**[SCREENSHOT\_REPLACED\_FOR\_SECURITY\_REASON]**

**Recommendations**

* Disable Self-Registration on Sentry Server **SENTRY\_FEATURES['auth:register'] = False**

**References**

* https://docs.sentry.io/server/config/#authentication

**Category**

| Category | CWE/SANS Top 25 | OWASP Top 10 |
| --- | --- | --- |
| Broken Access Control | CWE-284 | OWASP-A5 |

## MEDIUM risk findings

**Account Enumeration Possible through Registration**

****

**Vulnerability description**

HTTP POST submissions to the Payee / Payers registration pages could be used to enumerate user account emails for [REPLACED\_COMPANY] application. Our company was able to enumerate about 9000 emails retrieved from Sentry Vulnerability in 7 minutes and identify admin, payer and payee login emails.

**Impact**

Posting data to the registration pages, an attacker, not authenticated as a user, can test if an email address is being used to authenticate into [REPLACED\_COMPANY] application. Also, it is possible to establish what type of account linked to the email (payee/payer/admin). There is no lock-out or throttling mechanism in place.

**Pages Impacted**

https://www.[REPLACED\_COMPANY] .com/accounts/register-payee/

https://www.[REPLACED\_COMPANY] .com/accounts/register-payer/

**Verification and Attack Information**

Our company noticed when a site visitor trying to register as a Payee or as a Payer, his choice of email/username being checked against a list of existing users upon submission and if the username already used,

displays an error:

* "A user with this email is already registered as a payer" (payee registration)
* "A user with this email is already registered as a payee" (payer registration)
* Returns 500 Error code (admin registration)

or returns a 50X HTTP status code if a user has administrative privileges, because something is breaking on the back-end in this case.

Application Response:

**[SCREENSHOT\_REPLACED\_FOR\_SECURITY\_REASON]**

Single Request Payload:

**[SCREENSHOT\_REPLACED\_FOR\_SECURITY\_REASON]**

User Enumeration:

**[SCREENSHOT\_REPLACED\_FOR\_SECURITY\_REASON]**

Admin User Identification:

**[SCREENSHOT\_REPLACED\_FOR\_SECURITY\_REASON]**

**Recommendations**

* Fix 500 error when registering with already existing admin user email. Return a vague 200 response that account could not be registered.
* When account email already exists on registration, provide a vague response that account could not be registered
* Implement Throttling of requests (preferably by IP address) to avoid mass enumeration attempts demonstrated above

**References**

* https://www.django-rest-framework.org/api-guide/throttling/

**Category**

| Category | CWE/SANS Top 25 | OWASP Top 10 |
| --- | --- | --- |
| Authentication | CWE-203 |  |

## LOW risk findings

**Insufficient Password Policy**

****

**Vulnerability description**

Our company was able to get unauthorized access to 2 [REPLACED\_COMPANY] user accounts by using compromised passwords database. The personal passwords of individuals who happen to be customers of [REPLACED\_COMPANY] were compromised and became public information due to one of the recent breaches, not related to [REPLACED\_COMPANY] .

**Impact**

Attacker could use compromised passwords database to get access to individual [REPLACED\_COMPANY] accounts.

**Pages Impacted**

https://www.[REPLACED\_COMPANY] .com/

**Verification and Attack Information**

Our company was able to gain unauthorized access to [EMAIL\_REPLACED] account and get their API Token.

**[SCREENSHOT\_REPLACED\_FOR\_SECURITY\_REASON]**

Our company was able to get unauthorized access to [EMAIL\_REPLACED] account and view their vendor invitations

**[SCREENSHOT\_REPLACED\_FOR\_SECURITY\_REASON]**

**Recommendations**

Our company recommends implementing a compromised password check mechanism to check new passwords against compromised database on registration.

* Free Password Check API is available for the public: https://haveibeenpwned.com/API/v3#PwnedPasswords
* Sample API Request for password SHA1 hash beginning with 21BD1 https://api.pwnedpasswords.com/range/21BD1
* A JavaScript checker is available to check passwords on login/registration forms: https://github.com/OktaSecurityLabs/passprotect-js

**References**

NIST Special Publication 800-63 Recommends Checking passwords against compromised databases: https://pages.nist.gov/800-63-3/sp800-63b.html#memsecret

**Category**

| Category | CWE/SANS Top 25 | OWASP Top 10 |
| --- | --- | --- |
| Authentication | CWE-521 |  |

**Insertion of Sensitive Information into Log File**



**Vulnerability description**

Our company was able to download 9GB of error log information dating back to 2016, extract 5189 valid

[REPLACED\_COMPANY] application usernames and 6 customer bank account numbers. Our company was able to

obtain partial private keys and individual Session IDs.

**Impact**

Attacker could use the session IDs obtained through logs to access customer accounts. Attacker could use discovered bank account numbers to initiate financial transactions (ACH) to/from said bank accounts.

**Pages Impacted**

* https://sentry.[REPLACED\_COMPANY] .com

**Verification and Attack Information**

Our companylabs could register as new user at sentry.[REPLACED\_COMPANY] .com and join [REPLACED\_COMPANY] -app team. As result it was possible to browse error logs and stack traces. Analysying log files Our companylabs extracted 5189 valid [REPLACED\_COMPANY] application usernames, 6 customer bank account numbers and other private information.

**[SCREENSHOT\_REPLACED\_FOR\_SECURITY\_REASON]**

Extracted user session\_key

**[SCREENSHOT\_REPLACED\_FOR\_SECURITY\_REASON]**

Extracted PRIVATE KEY

**[SCREENSHOT\_REPLACED\_FOR\_SECURITY\_REASON]**

**Recommendations**

* Remove user SessionIDs, Private Keys, contents of forms that contain Bank Account numbers and other sensitive information from being logged.

**References**

* https://cwe.mitre.org/data/definitions/532.html

**Category**

| Category | CWE/SANS Top 25 | OWASP Top 10 |
| --- | --- | --- |
| Configuration | CWE-532 |  |

**API Auth token accessible to users with revoked permission**

****

**Vulnerability description**

API Auth token represents an alternative authentication mechanism for a user to be able to control his own account. Our company has discovered that Auth Token is accessible at /api-docs page to a user after his "View Auth Token" permission was first granted and revoked later.

**Impact**

A user with a revoked token permission could still access API without the authorization for doing so.

**Pages Impacted**

https://www.[REPLACED\_COMPANY] .com/api-docs

**Verification and Attack Information**

A new user with no permission for "View Auth Token" in his role is unable to view his token at API documentation page.

**[SCREENSHOT\_REPLACED\_FOR\_SECURITY\_REASON]**

The user is able to see the token after that permission was granted, but after removing the permission, user can still see the token

**[SCREENSHOT\_REPLACED\_FOR\_SECURITY\_REASON]**

**Recommendations**

Our company recommends taking the following measures to protect API Token from un-authorized use:

* Invalidate / erase Auth Token upon revocation of "View Auth Token Permission"
* Remove access to /api-docs to users who do not have permission to view their token

**Category**

| Category | CWE/SANS Top 25 | OWASP Top 10 |
| --- | --- | --- |
| Broken Access Control | CWE-284 | OWASP-A5 |

**"View Upload History" permission has no effect**

****

**Vulnerability description**

View Upload History permission supposed to control if a user is authorized or not to view / inspect past uploaded files. This permission has no effect - if it is not granted, the user still can see the last 5 uploaded files.

**Impact**

A user with limited permission can view the past uploaded files which he might be not authorized to view.

**Pages Impacted**

https://www.[REPLACED\_COMPANY] .com/app/setup/admin/upload-data

**Verification and Attack Information**

A user with permission "Upload Data" but without "View Upload History" permission is still able to view

the last 5 uploaded suppliers files and the last 5 uploaded invoice files.

No “View Upload History” permission

**[SCREENSHOT\_REPLACED\_FOR\_SECURITY\_REASON]**

User can still see the files:

**[SCREENSHOT\_REPLACED\_FOR\_SECURITY\_REASON]**

**Recommendations**

* Enforce "View Upload History" permission check at /app/setup/admin/upload-data page.

**Category**

| Category | CWE/SANS Top 25 | OWASP Top 10 |
| --- | --- | --- |
| Broken Access Control | CWE-284 | OWASP-A5 |

**Uploaded Files accessible to limited account users**

****

**Vulnerability description**

A limited user within a company, who does not possess "Upload Data" and "View Upload History" permissions is able to access uploaded files by guessing a filename.

**Impact**

A user with limited permissions can access the uploaded files which he might be not authorized to view.

**Pages Impacted**

https://www.[REPLACED\_COMPANY] .com/api/files/<company\_slug>/private/uploads/invoices/processed/...

https://www.[REPLACED\_COMPANY] .com/api/files/<company\_slug>/private/uploads/suppliers/processed/...

**Verification and Attack Information**

A file uploaded to the server is saved under auto-generated filename. The saved filename is based on the original filename a user uploaded prefixed with Unix timestamp and a dash (-).

**[SCREENSHOT\_REPLACED\_FOR\_SECURITY\_REASON]**

This pattern can be guessed and retrieval of the files can be attempted using multiple combinations by iterating Unix timestamp part of the file.

A user within the same company but with limited permissions

(not having "View Upload History" bit) can retrieve the file by knowing (guessing) it's filename.

**[SCREENSHOT\_REPLACED\_FOR\_SECURITY\_REASON]**

**Recommendations**

* Enforce "View Upload History" permission check for retrieving company files or introduce a new “View Uploaded Files” permission
* Use hashes for filenames stored on the server, et: cc2bd8f09bb88b5dd20f9b432631b8ca.csv

**Category**

| Category | CWE/SANS Top 25 | OWASP Top 10 |
| --- | --- | --- |
| Broken Access Control | CWE-284 | OWASP-A5 |

**No Session management capabilities**

****

**Vulnerability description**

Application doesn’t have an ability to tell users about all users that are currently logged in under the user’s credentials. It doesn’t have an ability to terminate all user sessions under user’s account. Same applies to API Token Authenticated requests. There is no way to track them.

**Impact**

User has no ability to tell if his account has been compromised. User has no controls for remediating against an attack. Our company’s compromise of user accounts went completely undetected due to absence of those controls.

**System Impacted**

https://www.[REPLACED\_COMPANY] .com/

https://www.[REPLACED\_COMPANY] .com/api

**Recommendations**

Our company recommends to create an interface for the user in which he can see all active sessions and requests authenticated with API Token of his account and terminate sessions, if necessary.

**Category**

| Category | CWE/SANS Top 25 | OWASP Top 10 |
| --- | --- | --- |
| Session Management | - | - |

## Informational Findings

**OpenVPN Upgrade**

**Description**

Our company was able to fingerprint OpenVPN Servers and identify its version:

OpenVPN 2.3.10 x86\_64-pc-linux-gnu

This version contains a number of vulnerabilities described in OpenVPN Security Advisory:

https://community.openvpn.net/openvpn/wiki/VulnerabilitiesFixedInOpenVPN243

**System Impacted**

https://vpn1.[REPLACED\_COMPANY] .com/

**Recommendations**

OpenVPN recommends upgrading to **v2.3.17**

**References**

https://nvd.nist.gov/vuln/detail/CVE-2017-7508

**Category**

| Category | CWE/SANS Top 25 | OWASP Top 10 |
| --- | --- | --- |
| Configuration | - | OWASP-A9 |

**Python Dependency Upgrades** 

**Description**

After exploiting Sentry vulnerability Our company was able to retrieve [REPLACED\_COMPANY] application python dependencies and their versions. Our company was able to find publicly known vulnerabilities for some of the python dependencies of [REPLACED\_COMPANY] application:

| rest\_framework | 3.5.4 | https://snyk.io/vuln/SNYK-PYTHON-DJANGORESTFRAMEWORK-450194 |
| --- | --- | --- |
| storages | 1.5.2 | https://snyk.io/vuln/SNYK-PYTHON-DJANGOSTORAGES-72415 |

Our company was able to build a full dependency tree of the python packages, however it is still unknown whether that tree is identical to that of [REPLACED\_COMPANY] application. We have identified some potential issues with dependencies based on the tree we were able to construct:

| PIL/Pillow | https://seclists.org/oss-sec/2016/q1/269 |
| --- | --- |
| Jinja2 | https://snyk.io/vuln/SNYK-PYTHON-JINJA2-455616 |
| lxml | https://snyk.io/vuln/SNYK-PYTHON-LXML-72651 |
| pycryptodome | https://snyk.io/vuln/SNYK-PYTHON-PYCRYPTODOME-42184 |
| pyyaml | https://snyk.io/vuln/SNYK-PYTHON-PYYAML-42159 |
| simplejson | https://www.cvedetails.com/cve/CVE-2014-4616/ |
| urllib3 | https://snyk.io/vuln/pip:urllib3 |
| python-saml | https://snyk.io/vuln/SNYK-PYTHON-PYTHONSAML-40774 |

**Recommendation**

* Upgrade packages as advised in each referenced link above if current version is vulnerable

**Category**

| Category | CWE/SANS Top 25 | OWASP Top 10 |
| --- | --- | --- |
| Configuration | - | OWASP-A9 |

**SFTP Endpoint**

**Description**

SFTP endpoint was scanned by Our company from the IP address that wasn’t white-listed. Our company performed SYN, ACK and Fragmented packet scans on the endpoint as displayed below:

**[SCREENSHOT\_REPLACED\_FOR\_SECURITY\_REASON]**

Our company then performed a Scan from a white-listed IP address to fingerprint the SSHD Service and attempted to perform username enumeration which wasn’t successful. Our company has validated that SFTP endpoint is not susceptible to CVE-2018-15473

**[SCREENSHOT\_REPLACED\_FOR\_SECURITY\_REASON]**

## OWASP TOP 10 COMPARISON

The following table represents the most prevalent and critical vulnerabilities identified in OWASP’s 2017 Top 10 List.

| # | Description | Reference | Verification |
| --- | --- | --- | --- |
| 1 | Injection | OWASP-A1 | Not Identified |
| 2 | Broken Authentication | OWASP-A2 | Not Identified |
| 3 | Sensitive Data Exposure | OWASP-A3 | Not Identified |
| 4 | XML External Entities | OWASP-A4 | Not Identified |
| 5 | Broken Access Control | OWASP-A5 | **Identified** |
| 6 | Security Misconfiguration | OWASP-A6 | Not Identified |
| 7 | Cross-Site Scripting | OWASP-A7 | Not Identified |
| 8 | Insecure Deserialization | OWASP-A8 | Not Identified |
| 9 | Using Components with Known Vulnerabilities | OWASP-A9 | **Identified** |
| 10 | Insufficient Logging & Monitoring | OWASP-A10 | Not Identified |

**Identified** – Indicates that this type of vulnerability was identified during Our company’ssecurity assessment of the application

Not Identified – Indicates that this type of vulnerability was NOT identified during Our company’ssecurity assessment of the application

## SANS Top 25 Comparison

The following table represents the most prevalent and critical vulnerabilities identified in SANS’s 2011 Top 25 List. This list was last updated by SANS on June 29, 2011 but remains the most recent update from SANS as of June 08, 2018.

**Insecure Interaction Between Components**

These weaknesses are related to insecure ways in which data is sent and received between separate components, modules, programs, processes, threads, or systems

| # | Description | Reference | Verification |
| --- | --- | --- | --- |
| 1 | Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') | CWE-89 | Not Identified |
| 2 | Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') | CWE-78 | Not Identified |
| 4 | Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') | CWE-79 | Not Identified |
| 9 | Unrestricted Upload of File with Dangerous Type | CWE-434 | Not Identified |
| 12 | Cross-Site Request Forgery (CSRF) | CWE-352 | Not Identified |
| 22 | URL Redirection to Untrusted Site ('Open Redirect') | CWE-601 | Not Identified |

**Risky Resource Management**

The weaknesses in this category are related to ways in which software does not properly manage the creation, usage, transfer, or destruction of important system resources

| # | Description | Reference | Verification |
| --- | --- | --- | --- |
| 3 | Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') | CWE-120 | Not Identified |
| 13 | Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') | CWE-22 | Not Identified |
| 14 | Download of Code Without Integrity Check | CWE-494 | Not Identified |
| 16 | Inclusion of Functionality from Untrusted Control Sphere | CWE-829 | Not Identified |
| 18 | Use of Potentially Dangerous Function | CWE-676 | Not Identified |
| 20 | Incorrect Calculation of Buffer Size | CWE-131 | Not Identified |
| 23 | Uncontrolled Format String | CWE-134 | Not Identified |
| 24 | Integer Overflow or Wraparound | CWE-190  | Not Identified |

**Porous Defenses**

The weaknesses in this category are related to defensive techniques that are often misused, abused, or just plain ignored.

| # | Description | Reference | Verification |
| --- | --- | --- | --- |
| 5 | Missing Authentication for Critical Function | CWE-306 | Not Identified |
| 6 | Missing Authorization | CWE-862 | Not Identified |
| 7 | Use of Hard-coded Credentials | CWE-798 | Not Identified |
| 8 | Missing Encryption of Sensitive Data | CWE-311 | Not Identified |
| 10 | Reliance on Untrusted Inputs in a Security Decision | CWE-807 | Not Identified |
| 11 | Execution with Unnecessary Privileges | CWE-250 | Not Identified |
| 15 | Incorrect Authorization | CWE-863 | Not Identified |
| 17 | Incorrect Permission Assignment for Critical Resource | CWE-732 | Not Identified |
| 19 | Use of a Broken or Risky Cryptographic Algorithm | CWE-327 | Not Identified |
| 21 | Improper Restriction of Excessive Authentication Attempts | CWE-307  | Not Identified |
| 25 | Use of a One-Way Hash without a Salt | CWE-759 | Not Identified |

**Identified** – Indicates that this type of vulnerability was identified during Our company’ssecurity assessment of the application

Not Identified – Indicates that this type of vulnerability was NOT identified during Our company’ssecurity assessment of the application

# APPENDIX 1: OWASP ASVS LEVEL 1 CHECKLIST FOR WEB APPLICATION

| # | Description | Verified | Comments |
| --- | --- | --- | --- |
| V1. Architecture, design and threat modelling |  |  |  |
| 1.1 | Verify that all application components are identified and are known to be needed. | ✔ |  |
| V2. Authentication |  |  |  |
| 2.1 | Verify all pages and resources by default require authentication except those specifically intended to be public (Principle of complete mediation). | ✔ |  |
| 2.2 | Verify that all password fields do not echo the user's password when it is entered. | ✔ |  |
| 2.4 | Verify all authentication controls are enforced on the server side. | ✔ |  |
| 2.6 | Verify all authentication controls fail securely to ensure attackers cannot log in. | ✔ |  |
| 2.7 | Verify password entry fields allow, or encourage, the use of passphrases, and do not prevent long passphrases/highly complex passwords being entered. | ✔ |  |
| 2.8 | Verify all account identity authentication functions (such as update profile, forgot password, disabled / lost token, help desk or IVR) that might regain access to the account are at least as resistant to attack as the primary authentication mechanism. | ✔ |  |
| 2.9 | Verify that the changing password functionality includes the old password, the new password, and a password confirmation. | ✔ |  |
| 2.16 | Verify that credentials are transported using a suitable encrypted link and that all pages/functions that require a user to enter credentials are done so using an encrypted link. | ✔ |  |
| 2.17 | Verify that the forgotten password function and other recovery paths do not reveal the current password and that the new password is not sent in clear text to the user. | ✔ |  |
| 2.18 | Verify that information enumeration is not possible via login, password reset, or forgot account functionality. | ✔ |  |
| 2.19 | Verify there are no default passwords in use for the application framework or any components used by the application (such as "admin/password"). | ✔ |  |
| 2.2 | Verify that request throttling is in place to prevent automated attacks against common authentication attacks such as brute force attacks or denial of service attacks. | ✔ |  |
| 2.22 | Verify that forgotten password and other recovery paths use a soft token, mobile push, or an offline recovery mechanism. | ✔ |  |
| 2.24 | Verify that if knowledge based questions (also known as "secret questions") are required, the questions should be strong enough to protect the application. | ✔ |  |
| 2.27 | Verify that measures are in place to block the use of commonly chosen passwords and weak passphrases. |  |  |
| 2.3 | Verify that if an application allows users to authenticate, they use a proven secure authentication mechanism. | ✔ |  |
| 2.32 | Verify that administrative interfaces are not accessible to untrusted parties | ✔ |  |
| V3. Session management |  |  |  |
| 3.1 | Verify that there is no custom session manager, or that the custom session manager is resistant against all common session management attacks. | ✔ |  |
| 3.2 | Verify that sessions are invalidated when the user logs out. | ✔ |  |
| 3.3 | Verify that sessions timeout after a specified period of inactivity. | ✔ |  |
| 3.5 | Verify that all pages that require authentication have easy and visible access to logout functionality. | ✔ |  |
| 3.6 | Verify that the session id is never disclosed in URLs, error messages, or logs. This includes verifying that the application does not support URL rewriting of session cookies. | x | Session IDs exposed in logs |
| 3.7 | Verify that all successful authentication and re-authentication generates a new session and session id. | ✔ |  |
| 3.11 | Verify that session ids are sufficiently long, random and unique across the correct active session base. | ✔ |  |
| 3.12 | Verify that session ids stored in cookies have their path set to an appropriately restrictive value for the application, and authentication session tokens additionally set the "HttpOnly" and "secure" attributes | ✔ |  |
| 3.16 | Verify that there is no custom session manager, or that the custom session manager is resistant against all common session management attacks. | ✔ |  |
| 3.17 | Verify that an active session list is displayed in the account profile or similar of each user. The user should be able to terminate any active session. | x |  |
| 3.18 | Verify the user is prompted with the option to terminate all other active sessions after a successful change password process. | x |  |
| V4. Access control |  |  |  |
| 4.1 | Verify that the principle of least privilege exists - users should only be able to access functions, data files, URLs, controllers, services, and other resources, for which they possess specific authorization. This implies protection against spoofing and elevation of privilege. | ✔ |  |
| 4.4 | Verify that access to sensitive records is protected, such that only authorized objects or data is accessible to each user (for example, protect against users tampering with a parameter to see or alter another user's account). | ✔ |  |
| 4.5 | Verify that directory browsing is disabled unless deliberately desired. Additionally, applications should not allow discovery or disclosure of file or directory metadata, such as Thumbs.db, .DS\_Store, .git or .svn folders. | ✔ |  |
| 4.8 | Verify that access controls fail securely. | ✔ |  |
| 4.9 | Verify that the same access control rules implied by the presentation layer are enforced on the server side. | ✔ |  |
| 4.13 | Verify that the application or framework uses strong random anti-CSRF tokens or has another transaction protection mechanism. | ✔ |  |
| 4.16 | Verify that the application correctly enforces context-sensitive authorisation so as to not allow unauthorised manipulation by means of parameter tampering. | ✔ |  |
| V5. Malicious input handling |  |  |  |
| 5.1 | Verify that the runtime environment is not susceptible to buffer overflows, or that security controls prevent buffer overflows. | ✔ |  |
| 5.3 | Verify that server side input validation failures result in request rejection and are logged. | ✔ |  |
| 5.5 | Verify that input validation routines are enforced on the server side. | ✔ |  |
| 5.1 | Verify that all SQL queries, HQL, OSQL, NOSQL and stored procedures, calling of stored procedures are protected by the use of prepared statements or query parameterization, and thus not susceptible to SQL injection | ✔ |  |
| 5.11 | Verify that the application is not susceptible to LDAP Injection, or that security controls prevent LDAP Injection. | ✔ |  |
| 5.12 | Verify that the application is not susceptible to OS Command Injection, or that security controls prevent OS Command Injection. | ✔ |  |
| 5.13 | Verify that the application is not susceptible to Remote File Inclusion (RFI) or Local File Inclusion (LFI) when content is used that is a path to a file. | ✔ |  |
| 5.14 | Verify that the application is not susceptible to common XML attacks, such as XPath query tampering, XML External Entity attacks, and XML injection attacks. | ✔ |  |
| 5.15 | Ensure that all string variables placed into HTML or other web client code is either properly contextually encoded manually, or utilize templates that automatically encode contextually to ensure the application is not susceptible to reflected, stored and DOM Cross-Site Scripting (XSS) attacks. | ✔ |  |
| 5.22 | Make sure untrusted HTML from WYSIWYG editors or similar are properly sanitized with an HTML sanitizer and handle it appropriately according to the input validation task and encoding task. | ✔ |  |
| V7. Cryptography at rest |  |  |  |
| 7.2 | Verify that all cryptographic modules fail securely, and errors are handled in a way that does not enable oracle padding. | ✔ |  |
| 7.7 | Verify that cryptographic algorithms used by the application have been validated against FIPS 140-2 or an equivalent standard. | ✔ |  |
| V8. Error handling and logging |  |  |  |
| 8.1 | Verify that the application does not output error messages or stack traces containing sensitive data that could assist an attacker, including session id, software/framework versions and personal information | ✔ |  |
| V9. Data protection |  |  |  |
| 9.1 | Verify that all forms containing sensitive information have disabled client side caching, including autocomplete features. | ✔ |  |
| 9.3 | Verify that all sensitive data is sent to the server in the HTTP message body or headers (i.e., URL parameters are never used to send sensitive data). | ✔ |  |
| 9.4 | Verify that the application sets appropriate anti-caching headers as per the risk of the application, such as the following: Expires: Tue, 03 Jul 2001 06:00:00 GMTLast-Modified: {now} GMTCache-Control: no-store, no-cache, must-revalidate, max-age=0Cache-Control: post-check=0, pre-check=0Pragma: no-cache | ✔ |  |
| 9.9 | Verify that data stored in client side storage - such as HTML5 local storage, session storage, IndexedDB, regular cookies or Flash cookies - does not contain sensitive or PII). | ✔ |  |
| V10. Communications |  |  |  |
| 10.1 | Verify that a path can be built from a trusted CA to each Transport Layer Security (TLS) server certificate, and that each server certificate is valid. | ✔ |  |
| 10.3 | Verify that TLS is used for all connections (including both external and backend connections) that are authenticated or that involve sensitive data or functions, and does not fall back to insecure or unencrypted protocols. Ensure the strongest alternative is the preferred algorithm. | ✔ |  |
| 10.11 | Verify that HTTP Strict Transport Security headers are included on all requests and for all subdomains, such as Strict-Transport-Security: max-age=15724800; includeSubdomains | ✔ |  |
| 10.13 | Ensure forward secrecy ciphers are in use to mitigate passive attackers recording traffic. | ✔ |  |
| 10.14 | Verify that proper certification revocation, such as Online Certificate Status Protocol (OCSP) Stapling, is enabled and configured. | ✔ |  |
| 10.15 | Verify that only strong algorithms, ciphers, and protocols are used, through all the certificate hierarchy, including root and intermediary certificates of your selected certifying authority. | ✔ |  |
| 10.16 | Verify that the TLS settings are in line with current leading practice, particularly as common configurations, ciphers, and algorithms become insecure. | ✔ |  |
| V11. HTTP security configuration |  |  |  |
| 11.1 | Verify that the application accepts only a defined set of required HTTP request methods, such as GET and POST are accepted, and unused methods (e.g. TRACE, PUT, and DELETE) are explicitly blocked. | ✔ |  |
| 11.2 | Verify that every HTTP response contains a content type header specifying a safe character set (e.g., UTF-8, ISO 8859-1). | ✔ |  |
| 11.5 | Verify that the HTTP headers or any part of the HTTP response do not expose detailed version information of system components. | ✔ |  |
| 11.6 | Verify that all API responses contain X-Content-Type-Options: nosniff and Content-Disposition: attachment; filename="api.json" (or other appropriate filename for the content type). | ✔ |  |
| 11.7 | Verify that the Content Security Policy V2 (CSP) is in use in a way that either disables inline JavaScript or provides an integrity check on inline JavaScript with CSP noncing or hashing. | ✔ |  |
| 11.8 | Verify that the X-XSS-Protection: 1; mode=block header is in place. | ✔ |  |
| V16. File and resources |  |  |  |
| 16.1 | Verify that URL redirects and forwards only allow whitelisted destinations, or show a warning when redirecting to potentially untrusted content. | ✔ |  |
| 16.2 | Verify that untrusted file data submitted to the application is not used directly with file I/O commands, particularly to protect against path traversal, local file include, file mime type, and OS command injection vulnerabilities. | ✔ |  |
| 16.3 | Verify that files obtained from untrusted sources are validated to be of expected type and scanned by antivirus scanners to prevent upload of known malicious content. | ✔ |  |
| 16.4 | Verify that untrusted data is not used within inclusion, class loader, or reflection capabilities to prevent remote/local file inclusion vulnerabilities. | ✔ |  |
| 16.5 | Verify that untrusted data is not used within cross-domain resource sharing (CORS) to protect against arbitrary remote content. | ✔ |  |
| 16.8 | Verify the application code does not execute uploaded data obtained from untrusted sources. | ✔ |  |
| 16.9 | Do not use Flash, Active-X, Silverlight, NACL, client-side Java or other client side technologies not supported natively via W3C browser standards. | ✔ |  |
| V18. Web Services |  |  |  |
| 18.1 | Verify that the same encoding style is used between the client and the server. | ✔ |  |
| 18.2 | Verify that access to administration and management functions within the Web Service Application is limited to web service administrators. | ✔ |  |
| 18.3 | Verify that XML or JSON schema is in place and verified before accepting input. | ✔ |  |
| 18.4 | Verify that all input is limited to an appropriate size limit. | ✔ |  |
| 18.5 | Verify that SOAP based web services are compliant with Web Services-Interoperability (WS-I) Basic Profile at minimum. | ✔ |  |
| 18.6 | Verify the use of session-based authentication and authorization. Please refer to sections 2, 3 and 4 for further guidance. Avoid the use of static "API keys" and similar. | X | API Auth token is used that doesn't expire |
| 18.7 | Verify that the REST service is protected from Cross-Site Request Forgery. | ✔ |  |
| V19. Configuration |  |  |  |
| 19.1 | All components should be up to date with proper security configuration(s) and version(s). This should include removal of unneeded configurations and folders such as sample applications, platform documentation, and default or example users. | ✔ |  |