



Premium OWASP Penetration Testing Report for [REDACTED] Web Platform

Testers

1. [REDACTED]
2. [REDACTED]

Table of Contents

Table of Contents	2
Management Summary	3
Vulnerabilities by Severity	4
Approach	5
Introduction	5
Scope Overview	6
Technical Overview	6
Scope Validation	6
Threat Model	6
Security Evaluation Methodology	7
Security Assessment	8
Issue Table Description	9
Security Assessment Findings	14
Stored (Dom-based) Cross-Site Scripting	15
Credentials Spraying Attack	16
CloudFront Bypass	18
Absence of Rate Limit - Brute Forcing Verification Code	20
Blocking New Users Registration	21
Access Control - Private Information Leakage	22
Session Management - Session Hijacking	24
Improper Error Handling - Backend Communication Leakage	25
Appendix A: Security Evaluation Fixes	26

Management Summary

██████████ contacted Sayfer Security in order to perform full black-box penetration testing on the

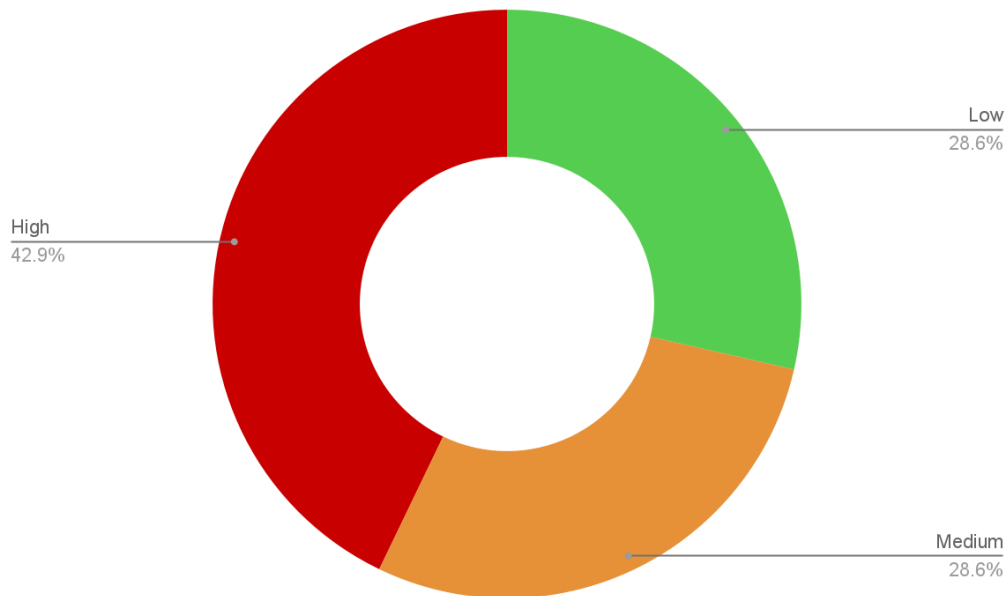
Before assessing these services we held a kickoff meeting with the technical team and received an overview of the system and the goals for this assessment.

Over the assessment, we discovered 8 vulnerabilities in the system, 2 of which are at high risk. The first vulnerability lets an anonymous attacker enumerate archive file names and get access to these files.

The second vulnerability lets an anonymous attacker to DoS the ██████████ API by abusing the rate limit mechanism.

Other vulnerabilities do not have a high impact on the system but should be fixed and these are described in this report in more technical details.

Vulnerabilities by Severity



Severity	Low	Medium	High	Informational
# of issues	2	2	3	1

- **High** – Direct threat to key business processes.
- **Medium** – Indirect threat to key business processes or partial threat to business processes.
- **Low** – No direct threat exists. The vulnerability may be exploited using other vulnerabilities.
- **Informational** – This finding does not indicate vulnerability, but states a comment that notifies about design flaws and improper implementation that might cause a problem in the long run.

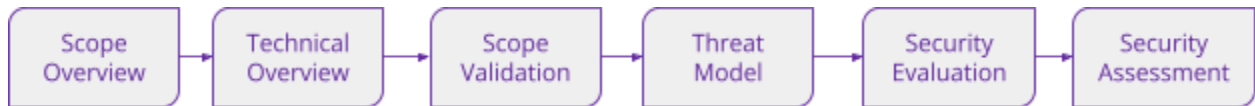
Approach

Introduction

██████████ contacted Sayfer Security in order to perform full black-box penetration testing for ██████████

This report documents a security assessment carried out by Sayfer targeting both applications. Particularly, this report displays the security posture review for both applications and their surrounding infrastructure and process implementations.

Our penetration testing project life cycle:



Scope Overview

During our first meeting with [REDACTED], after understanding the company's needs, we defined the application's scope that resides at the following URLs as the scope of the project:

- [REDACTED]
- [REDACTED]
- [REDACTED]

Our tests were performed between [REDACTED]

Technical Overview

[REDACTED]

[REDACTED]

Scope Validation

Before starting looking for security vulnerabilities we made sure the scope defined to us by the client was technically logical.

it is a very common mistake to forget an old server or account connected to the internet with permissions to access or control the system that is being audited under the defined scope.

By performing the scope validation we made sure that there are no unknown risks to the tested system.

Threat Model

During our kickoff meetings with the client, we defined the most important assets the application possesses.

[REDACTED]

Security Evaluation Methodology

Sayfer uses [OWASP WSTG](#) as our technical standard when reviewing web applications.

After gaining a thorough understanding of the system we decided which OWASP tests are required to evaluate the system.

Security Assessment

After understanding and defining the scope, performing threat modeling, and evaluating the correct tests required in order to fully check the application for security flaws, we performed our security assessment.

Issue Table Description

Issue title

ID	The OWASP ID of the issue. Additional tests that we conduct and are not included in the OWASP table will have Sayfer ID. Example ID: WSTG-INFO-002 WSTG - Web Security Test Guide. INFO - A shorthand for the topic to which the issue belongs. 002 - Issue number.
Risk	Represents the risk factor of the issue. For further description refer to the Vulnerabilities by Risk section.
Required Skill	Describes the skill level required to conduct successful exploitation. The lower the skill level the easier the exploitation process.
OWASP Reference	A link to the relevant OWASP page for further knowledge.
Location	The URL in which this issue was detected. Issues with no location have no particular location and refer to the product as a whole.
Tools	The tools used to detect the issue.
Description	Here we provide a brief description of the issue and how it formed, the steps we made to find or exploit it, along with proof of concept (if present), and how this issue can affect the product or its users.
Mitigation	Suggested resolving options for this issue and links to advised sites for further remediation.

Security Evaluation

The following tests were conducted while auditing the system:

Information Gathering	Test Name	Status
WSTG-INFO-01	Conduct Search Engine Discovery Reconnaissance for Information Leakage	Pass
WSTG-INFO-02	Fingerprint Web Server	Pass
WSTG-INFO-03	Review Webserver Metafiles for Information Leakage	Pass
WSTG-INFO-04	Enumerate Applications on Webserver	Pass
WSTG-INFO-05	Review Webpage Content for Information Leakage	Pass
WSTG-INFO-06	Identify application entry points	Pass
WSTG-INFO-07	Map execution paths through application	Pass
WSTG-INFO-08	Fingerprint Web Application Framework	Pass
WSTG-INFO-09	Fingerprint Web Application	Pass
WSTG-INFO-10	Map Application Architecture	Pass

Configuration and Deploy Management Testing	Test Name	Status
WSTG-CONF-01	Test Network Infrastructure Configuration	Pass
WSTG-CONF-02	Test Application Platform Configuration	Pass
WSTG-CONF-03	Test File Extensions Handling for Sensitive Information	Pass
WSTG-CONF-04	Review Old Backup and Unreferenced Files for Sensitive Information	Pass
WSTG-CONF-05	Enumerate Infrastructure and Application Admin Interfaces	Pass
WSTG-CONF-06	Test HTTP Methods	Pass
WSTG-CONF-07	Test HTTP Strict Transport Security	Pass
WSTG-CONF-08	Test RIA cross domain policy	Pass
WSTG-CONF-09	Test File Permission	Pass
WSTG-CONF-10	Test for Subdomain Takeover	Pass
WSTG-CONF-11	Test Cloud Storage	Pass

Identity Management Testing	Test Name	Status
WSTG-IDNT-01	Test Role Definitions	Pass
WSTG-IDNT-02	Test User Registration Process	Pass
WSTG-IDNT-03	Test Account Provisioning Process	Pass

WSTG-IDNT-04	Testing for Account Enumeration and Guessable User Account	Pass
WSTG-IDNT-05	Testing for Weak or unenforced username policy	Pass

Authentication Testing	Test Name	Status
WSTG-ATHN-01	Testing for Credentials Transported over an Encrypted Channel	Pass
WSTG-ATHN-02	Testing for Default Credentials	Pass
WSTG-ATHN-03	Testing for Weak Lock Out Mechanism	Pass
WSTG-ATHN-04	Testing for Bypassing Authentication Schema	Pass
WSTG-ATHN-05	Testing for Vulnerable Remember Password	Pass
WSTG-ATHN-06	Testing for Browser Cache Weaknesses	Pass
WSTG-ATHN-07	Testing for Weak Password Policy	Pass
WSTG-ATHN-08	Testing for Weak Security Question Answer	Pass
WSTG-ATHN-09	Testing for Weak Password Change or Reset Functionalities	Pass
WSTG-ATHN-10	Testing for Weaker Authentication in Alternative Channel	Pass

Authorization Testing	Test Name	Status
WSTG-ATHZ-01	Testing Directory Traversal File Include	Pass
WSTG-ATHZ-02	Testing for Bypassing Authorization Schema	Pass
WSTG-ATHZ-03	Testing for Privilege Escalation	Pass
WSTG-ATHZ-04	Testing for Insecure Direct Object References	Pass

Session Management Testing	Test Name	Status
WSTG-SESS-01	Testing for Session Management Schema	Pass
WSTG-SESS-02	Testing for Cookies Attributes	Pass
WSTG-SESS-03	Testing for Session Fixation	Pass
WSTG-SESS-04	Testing for Exposed Session Variables	Pass
WSTG-SESS-05	Testing for Cross Site Request Forgery	Pass
WSTG-SESS-06	Testing for Logout Functionality	Pass
WSTG-SESS-07	Testing Session Timeout	Pass
WSTG-SESS-08	Testing for Session Puzzling	Pass
WSTG-SESS-09	Testing for Session Hijacking	Pass

Data Validation Testing	Test Name	Status
WSTG-INPV-01	Testing for Reflected Cross Site Scripting	Pass
WSTG-INPV-02	Testing for Stored Cross Site Scripting	Pass

WSTG-INPV-03	Testing for HTTP Verb Tampering	Pass
WSTG-INPV-04	Testing for HTTP Parameter Pollution	Pass
WSTG-INPV-05	Testing for SQL Injection	Pass
WSTG-INPV-06	Testing for LDAP Injection	Pass
WSTG-INPV-07	Testing for XML Injection	Pass
WSTG-INPV-08	Testing for SSI Injection	Pass
WSTG-INPV-09	Testing for XPath Injection	Pass
WSTG-INPV-10	Testing for IMAP SMTP Injection	Pass
WSTG-INPV-11	Testing for Code Injection	Pass
WSTG-INPV-12	Testing for Command Injection	Pass
WSTG-INPV-13	Testing for Format String Injection	Pass
WSTG-INPV-14	Testing for Incubated Vulnerability	Pass
WSTG-INPV-15	Testing for HTTP Splitting Smuggling	Pass
WSTG-INPV-16	Testing for HTTP Incoming Requests	Pass
WSTG-INPV-17	Testing for Host Header Injection	Pass
WSTG-INPV-18	Testing for Server-side Template Injection	Pass
WSTG-INPV-19	Testing for Server-Side Request Forgery	Pass

Error Handling	Test Name	Status
WSTG-ERRH-01	Testing for Improper Error Handling	Pass
WSTG-ERRH-02	Testing for Stack Traces	Pass

Cryptography	Test Name	Status
WSTG-CRYP-01	Testing for Weak Transport Layer Security	Pass
WSTG-CRYP-02	Testing for Padding Oracle	Pass
WSTG-CRYP-03	Testing for Sensitive Information Sent via Unencrypted Channels	Pass
WSTG-CRYP-04	Testing for Weak Encryption	Pass

Business logic Testing	Test Name	Status
WSTG-BUSL-01	Test Business Logic Data Validation	Pass
WSTG-BUSL-02	Test Ability to Forge Requests	Pass
WSTG-BUSL-03	Test Integrity Checks	Pass
WSTG-BUSL-04	Test for Process Timing	Pass
WSTG-BUSL-05	Test Number of Times a Function Can be Used Limits	Pass
WSTG-BUSL-06	Testing for the Circumvention of Work Flows	Pass
WSTG-BUSL-07	Test Defenses Against Application Mis-use	Pass
WSTG-BUSL-08	Test Upload of Unexpected File Types	Pass
WSTG-BUSL-09	Test Upload of Malicious Files	Pass

Client Side Testing	Test Name	Status
WSTG-CLNT-01	Testing for DOM-Based Cross Site Scripting	Pass
WSTG-CLNT-02	Testing for JavaScript Execution	Pass
WSTG-CLNT-03	Testing for HTML Injection	Pass
WSTG-CLNT-04	Testing for Client Side URL Redirect	Pass
WSTG-CLNT-05	Testing for CSS Injection	Pass
WSTG-CLNT-06	Testing for Client Side Resource Manipulation	Pass
WSTG-CLNT-07	Test Cross Origin Resource Sharing	Pass
WSTG-CLNT-08	Testing for Cross Site Flashing	Pass
WSTG-CLNT-09	Testing for Clickjacking	Pass
WSTG-CLNT-10	Testing WebSockets	Pass
WSTG-CLNT-11	Test Web Messaging	Pass
WSTG-CLNT-12	Testing Browser Storage	Pass
WSTG-CLNT-13	Testing for Cross Site Script Inclusion	Pass

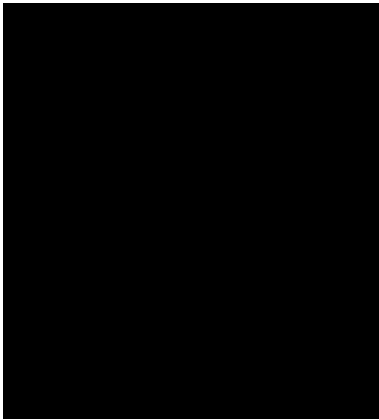
API Testing	Test Name	Status
WSTG-APIT-01	Testing GraphQL	Pass

Security Assessment Findings

Stored (Dom-based) Cross-Site Scripting

ID	WSTG-INPV-02
Risk	High
Required Skill	Low
OWASP Reference	Link
Location	██████████
Tools	Burp repeater, Web browser
Description	It is possible to modify the HTML content of the templates and include an XSS payload within it, which is stored on the server and accessed when users view the page. Stored XSS occurs when a web application gathers input from a user which might be malicious, and then stores that input in a data store for later use. The input that is stored is not correctly filtered. As a consequence, the malicious data will appear to be part of the website and run within the user's browser under the privileges of the web application. Assuming the payload successfully bypasses Cloudflare / WAF (which we were able to do) it is possible to save it. From that point, anyone "previewing" the email would instantly trigger the XSS, in addition to any users using an insecure mail client who received this email. An attacker with access to the portal can target other users in the same tenant account, running malicious javascript code on their browser targeting to steal cookies, keylogging, etc.
Mitigations	Preventing cross-site scripting is trivial in some cases but can be much harder depending on the complexity of the application and the ways it handles user-controllable data. In general, effectively preventing XSS vulnerabilities is likely to involve a combination of the following measures: • Filter input. At the point where user input is received, filter as strictly as possible based on what is expected or valid input. • Encode output. At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding. • Content Security Policy. As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur.

Credentials Spraying Attack

ID	WSTG-BUSL-05
Risk	High
Required Skill	Low
OWASP Reference	Link
Location	██████████
Tools	Burp Repeater, Burp Intruder, Web browser
Description	Brute force prevention mechanisms are not effective against credential spraying attacks. This vulnerability mostly affects the fields that are accessible by unauthenticated users as it allows brute-forcing user names or spamming and possibly performing DOS (Denial of Service) attacks. Many of the problems that applications solve require limits to the number of times a function can be used or action can be executed. Applications must be “smart enough” to not allow the user to exceed their limit on the use of these functions, since in many cases each time the function is used the user may gain some type of benefit that must be accounted for to properly compensate the owner. It was noticed that on any input fields designed to be used by unauthenticated users (login, registration, and forgotten password) the application tends to prevent brute force attacks after 20 sequent requests. However, if the requests are made each containing different credentials the control does not stop the attacker and permits sending the request an unlimited amount of times. • We successfully made 1000 requests to ██████████ with a Password Spraying attack, using different usernames for each request.  For example, it is possible to abuse the send SMS functionality to send numerous/continuous SMS messages to clients and random people by making

thousands of requests for either “new registration” or “forgotten password”, using different phone numbers for each request.

The anti-automation control is ineffective against credential spraying attacks like the one described above.

In addition, this could be an inconvenience for both the users/clients (i.e. reputation damage) as well as it will most likely enquire cost to [REDACTED] for all the SMS being sent out.

Mitigations

The application should set hard controls to prevent limit abuse. This can be achieved by setting a rate limit for the requests each IP can make per minute, a CAPTCHA on multiple requests in a short amount of time, set a counter limit per user on the back end or database level, as all users should be identified through a session, whichever is better for the business requirement.

CloudFront Bypass

ID	-
Risk	Medium
Required Skill	Low
OWASP Reference	Link
Location	██████████
Tools	DnsSpy
Description	We successfully bypassed CloudFront by finding the IP of the server and requesting it directly. During the assessment we looked for bypassing Cloudflare and Cloudfront and obtaining the real IP of the web application. We used different online tools for searching the DNS history of the domain. The generated DNS report from DnsSpy leaked important IP addresses and could be seen here ██████████. The results showed the IP address of ██████████ & ██████████. We used this information to bypass the Cloudfront protection, by redirecting all requests to ██████████ to this IP. We confirmed the bypass by comparing two responses, one from the IP and another from the actual URL • Response to IP request ██████████ • Response to URL request contains the following ██████████ In addition the scan provided one more IP address that looks like an deprecated web site

This application was not behind any WAF and it had references to 2021 in the footer (indicating that it hasn't been updated recently). This could either be the previous version of the www website, or the new one under development.

With this an attacker can

1. Bypass logging which cloudfront offers
2. Bypass potential WAF rules that cloudfront could be enforcing
3. Potentially bypass any access controls which are placed on traffic coming from cloudfront (and not from anywhere else).

Mitigations

Encapsulate all subdomains to one alias that is behind CloudFront to mitigate this issue.

Absence of Rate Limit - Brute Forcing Verification Code

ID	WSTG-BUSL-05
Risk	Medium
Required Skill	Low
OWASP Reference	Link
Location	██████████
Tools	Burp intruder
Description	The 6-digit code used in the reset password functionality is vulnerable to brute-force attacks as it is not invalidated after an unsuccessful attempt. As part of the email change functionality within the user profile settings, a user can update their email by first making a request to get a code via ██████████, and then verifying the ownership of the email by providing a 6-digit verification code which is sent over the email. It was found that this code does not get invalidated after a series of unsuccessful attempts. We were able to brute force the code with over 2,000 requests and reset users password using this code.  Considering the token is valid for 10 minutes (before you can request a new code and try again) it is very likely that an attacker can easily brute-force the code and verify an email address they don't own or control. This could be used as part of a phishing attack where the user can then attempt to represent an organization that they are actually not a part of (for example having @apple.com email address saying they represent Apple).
Mitigation	It is important to set limits on the usage of functionalities as part of brute force prevention, denial of service protection, enumeration etc.

Blocking New Users Registration

ID	WSTG-BUSL-07
Risk	High
Required Skill	Low
OWASP Reference	Link
Location	██████████
Tools	Browser
Description	Using the registration page logic flaw, any user can potentially block other new users from registering to the site. The misuse and invalid use of valid functionality can identify attacks attempting to enumerate the web application, identify weaknesses, and exploit vulnerabilities. Validation mechanisms should be implemented in such cases to prevent misuse and restrict unwanted behavior. There is a logical bug in the registration process of the application. Whenever a user attempts to register, they provide a phone number to which they should receive a code to activate the account. However, it was found that if a phone number is "registered" but never activated, this number can no longer be used to register a new user. - Attacker registers with random mobile number: ██████████ - Server response: ██████████ A new user with the same phone number tries to register, but the phone number is blocked by the attacker. - Server response to user registration request: ██████████ This vulnerability can be exploited by innocent users who mistyped their phone numbers potentially blocking the registration of another future user. The lack of active defenses allows an attacker to perform a Denial of Service attack without any recourse. The application's owner will thus not know their application is under attack.
Mitigations	The solution is to block the phone number after the activation after the registration process finishes.

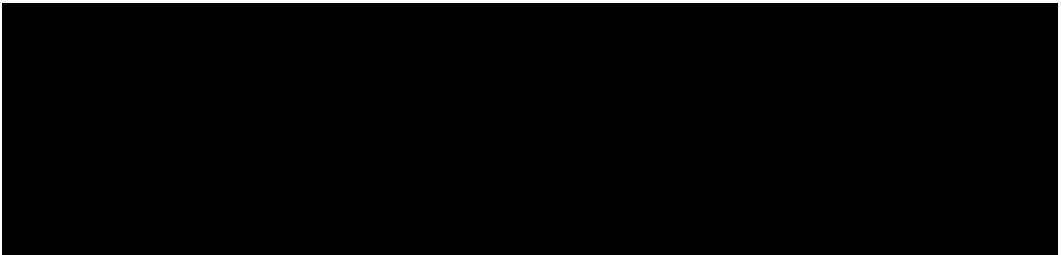
Access Control - Private Information Leakage

ID	WSTG-ATHZ-02
Risk	Medium
Required Skill	Low
OWASP Reference	Link
Location	██████████
Tools	Burp repeater
Description	Accounts can view private information of others by cycling the numbers of the ██████████ parameter. We confirmed that the merchant account we were given can view the information of other accounts. ██████████ parameter represents the merchants accounts registered in the application. Its value is numeric and the ids are ordered sequentially. By passing any numeric value to this parameter, the user can view the account profile of another merchant thus gaining access to their personal information, resulting an IDOR vulnerability. • ██████████ -IMAGE REDACTED- This was confirmed to be a vulnerability as the accounts that should be available to our user were given by making a different request. • A request to an endpoint returning a list of available accounts • ██████████ -IMAGE REDACTED-
Mitigation	Access control should be applied to the merchantid parameter as its available values should be different to each user.

Session Management - Session Hijacking

ID	WSTG-SESS-07
Risk	Medium
Required Skill	Medium
OWASP Reference	Link
Location	-
Tools	Burp
Description	The session (i.e JWT) token is valid for 24 hours, which is problematic considering the inefficient logout mechanism and the fact it is possible to use the same token more than once resulting in concurrent sessions. The JWT token is responsible to keep track of the user session and contains a timestamp to limit its validity. The application sets the timestamp of the JWT to 24 hours, making the token and thus the session valid even after the idle timeout is reached. Another misconfiguration is the unrestricted amount of sessions that can run simultaneously. Although there is control over the maximum amount of concurrent sessions, it is still possible to establish more than one concurrent connection. Hijacking the session can be done in numerous ways where in the end the attacker possesses the session token and uses it to impersonate a valid user. The timestamp and the concurrent logins limit should make it harder to use these tokens and create new sessions.
Mitigation	JWT timestamps can be updated by the server at any moment and should be as close to the session timeout as possible so no leakage of valid session tokens will accrue. In addition, if no more than one person should use an account, there is no need for multiple sessions running simultaneously. Otherwise, the restrictions should work as intended.

Improper Error Handling - Backend Communication Leakage

ID	WSTG-ERRH-02
Risk	Informational
Required Skill	Low
OWASP Reference	Link
Location	██████████
Tools	Browser
Description	Error message with information about a failed request is being shown in the UI. All types of applications (web apps, web servers, databases, etc.) will generate errors for various reasons. Developers often ignore handling these errors or push away the idea that a user will ever try to trigger an error purposefully (e.g. sending a string where an integer is expected). During the assessment of the application, we used the XSS finding from the above section and made a further investigation of the "locations" GET parameter. The payload we used was as follows: • ██████████ In response we got an error message appearing for a short time after clicking the refresh:  The message leak what kind of request is being made on the backend, and appears to have translated to the following GET request: • ██████████ One of the conclusions of this leakage is that we can see that the connection is made through unsafe HTTP leading us to a new man in the middle attack vector.
Mitigations	Considering that this cluster doesn't appear to be on localhost, but rather a remote service, this cleartext traffic is not advised. For remediation of the error handling, check out the Proactive Controls C10 and the Error Handling Cheat Sheet.

Appendix A: Security Evaluation Fixes



We found that all vulnerabilities found in our initial report were fixed during our second iteration.

 systems that were evaluated in the scope of this penetration testing are certified by OWASP WSGT version 4.2 standard.