OctoFi Web Application

Penetration Test

Initial Report: April 9th, 2021
Revision: May 6th, 2021

For :
OctoFi

By :
Peiyu Wang @ CertiK
peiyu.wang@certik.org

Minzhi He @ CertiK
minzhi.he@certik.org

Confidentiality Statement

All information contained in this document is provided in confidence for the sole purpose of adjudication of the document and shall not be published or disclosed wholly or in part to any other party without CertiK's prior permission in writing and shall be held in safe custody. These obligations shall not apply to information that is published or becomes known legitimately from some source other than CertiK.
All transactions are subject to the appropriate CertiK Standard Terms and Conditions. Certain information given in connection with this proposal is marked “In Commercial Confidence”. That information is communicated in confidence, and disclosure of it to any person other than with CertiK’s consent will be a breach of confidence actionable on the part of CertiK.

Disclaimer

This document is provided for information purposes only. CertiK accepts no responsibility for any errors or omissions that it may contain.
This document is provided without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and non-infringement. In no event shall CertiK be liable for any claim, damages or other liability (either direct or indirect or consequential), whether in an action of contract, tort or otherwise, arising from, out of or in connection with this document or the contents thereof.
This document represents our budgetary price proposal for the solution further described in this herein and is provided for information and evaluation purposes only and is not currently a formal offer capable of acceptance.

Overview

Scope

At the start of the engagement, CertiK worked with OctoFi to identify the target and set the limits on the scope of the test. A White Box type of testing approach was done where CertiK performed the test with the source code available from the public GitHub repository.

Application Name	OctoFi Web Application
Hostname	app.octo.fi
Environment	Production
Codebase	https://github.com/octofi/octofi-app-aquafarm/tree/v4.0
Commit Hash	fc1d5d0ecf951c47edbca62b14bde381d8fe08b9

Audit Summary

Delivery Date	May. 6, 2021
Method of Audit	Static Code Review, Dynamic Testing
Consultants Engaged	2
Initial Test	Apr. 5, 2021 - Apr. 9, 2021
Re-test	May. 6, 2021

Vulnerability Summary

Total Issues	4
Total Low	3
Total Informational	1

Executive Summary

OctoFi engaged CertiK to perform an application penetration test for their web application. OctoFi Dapp is an all-in-one decentralized application serving up direct access to a broad range of DeFi and NFT marketplaces from one convenient location. It integrates a variety of third-party APIs to obtains data to power up the application. Users will select one of the supported wallets and connect it with the application to sign generated transactions.

The main objective of the engagement is to test the overall resiliency of the application to various real-world attacks against the application's controls and functions, and thereby be able to identify its weaknesses and provide recommendations to fix and improve its overall security posture. Additionally, we aim to identify if the application and any third-party APIs track users' actualities.

Two members of the CertiK team were involved in completing the engagement which took place over the course of 4 days in April 2021 and yielded four security-relevant findings. Given the severity of the vulnerabilities on the application, it is unlikely that the application will be directly compromised.

The most significant vulnerability is the repeated use of "dangerouslySetInnerHTML" to insert data from untrusted sources into DOM. An attacker can perform a cross-site scripting (XSS) attack if he can find a way to inject arbitrary data into the third-party APIs' response. Other weaknesses were also found and are detailed on the Findings section of the report.

Regarding the review on user activity tracking, we didn't find the application purposely track its users. The only requests that involve tracking belong to the embedded third-party application "Bitrefill". The detail is included in the tracking section below. Note that the issue has been resolved in the latest commit.

It is recommended that OctoFi works on remediating the findings to raise the security posture of the application.

Tracking[Resolved]

Page

https://app.octo.fi/#/fiat/off-cards

Track by

Bitrefill

Endpoints

https://www.google-analytics.com/, https://googleads.g.doubleclick.net/, https://api-js.mixpanel.com/track/, https://px.ads.linkedin.com/

Information collected

User OS, Browser, IP address, User event, Page visited, Visit time

Alleviation

The application removed the embedded "Bitrefill" integration. The browser will open "Bitrefill.com" on a new tab when users click the "Go to Bitrefill" button.

Findings

ID	Title	Severity	Vulnerability Class	Status
OTF-01	Use of dangerously SetInnerHTML	Low	Injection	Acknowledged
OTF-02	Heavy use of third-party APIs	Low	Application Resource Handling	Acknowledged
OTF-03	Clickjacking	Low	Security Misconfiguration	Acknowledged
OTF-04	Reconstruct source code with source map	Informational	Security Misconfiguration	Acknowledged

OTF-01: Use of dangerouslySetInnerHTML

Severity: Low

Description:

"dangerouslySetInnerHTML" is React's replacement for using innerHTML in the browser DOM. In general, setting HTML from code is risky because it's easy to inadvertently expose users to a cross-site scripting (XSS) attack.

We identify the use of "dangerouslySetInnerHTML" in multiple locations within the application. After analyzing the use cases, we observe the value passes into the dangerouslySetInnerHTML comes from third-party APIs including api.transak.com, api.coingecko.com, and hub.snapshot.page. It's risky to insert data from untrusted sources into DOM via dangerouslySetInnerHTML without sanitization, as an attacker might be able to find a way to inject JavaScript code into data return from the third-party APIs.

Location

src/pages/Vote/index.js, L40
src/pages/CoinDetails/index.js, L512
src/pages/ContractDetails/index.js, L316
src/components/CryptoInput/index.js, L157, L204
src/components/CurrencySelectModal/CurrencyList.js, L73

Exploit Scenario

An attacker can perform a cross-site scripting (XSS) attack if he can find a way to inject arbitrary data into the third-party APIs' response. For example, the "Page/CoinDetails/index.js" passes token description from "api.coingecko.com/api/v3/coins/$token_name". into `dangerouslySetInnerHTML`. The attacker can potentially register a new token in coingecko and include malicious JavaScript code into its token description.

Recommendation

It's not recommended to use `dangerouslySetInnerHTML` to render data from untrust sources. If there isn't a workaround, use HTML sanitizer such as DOMPurify to sanitize the data before inserting it in the DOM via `dangerouslySetInnerHTML`.

OTF-02: Heavy use of third-party APIs

Severity: Low

Description:

The tested application is an all-in-one decentralized application serving up direct access to a broad range of DeFi and NFT marketplaces from one location. The application doesn't have its own backend database, and it entirely relies on data that comes from third-party APIs. The integrity of third-party APIs is out of the control of the team, and the heavy use of them inevitably enlarges the attack surface of the application.

Impact

This doesn't introduce immediate threats; however, the security and integrity of the application rely on the third-party APIs it connects to. If an attacker fully or partially compromises the coingecko's API, he can manipulate the vast majority of data in the application. Even if there isn't an attack, any APIs server outage can cause the application to become unusable.

Recommendation

Although it's considered a feature for a Dapp not to have a centralized database and obtain data from multiple sources, the following items can be considered to enhance the security of the system.

- Utilize APIs from more established companies and evaluate them carefully.
- Establish a system to monitor and detect abnormal behavior of third-party APIs.
- React and alert the community if any incident occurs

OTF-03: Clickjacking

Severity: Low

Introduction:

Clickjacking is an interface-based attack in which a user is tricked into clicking on actionable content on a hidden website by clicking on some other content on a decoy website. The impact of the clickjacking depends on the action that the user performed on the target application. Clickjacking can turn system features on and off, tricked users into download malware, and steal sensitive information such as password and credit card number.
For more information about Clickjacking, please visit https://en.wikipedia.org/wiki/Clickjacking

Description:

The following screenshot shows the HTTP response of the GET request to https://app.octo.fi. The response doesn't contain any security headers to prevent the website from loading in an iframe.

Request | Response

Raw | Headers | Hex | HTML | Render

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 6789
Server: GitHub.com
Content-Type: text/html; charset=utf-8
Strict-Transport-Security: max-age=31556952
Last-Modified: Sun, 28 Mar 2021 13:30:23 GMT
Access-Control-Allow-Origin: *
ETag: W/"606084ef-1a85"
expires: Fri, 09 Apr 2021 01:06:56 GMT
Cache-Control: max-age=600
x-proxy-cache: MISS
X-GitHub-Request-Id: 7C88:098E:2F2492:6315C8:606FA658
Accept-Ranges: bytes
Date: Fri, 09 Apr 2021 00:56:56 GMT
Via: 1.1 varnish
Age: 0
X-Served-By: cache-ewr18149-EWR
X-Cache: MISS
X-Cache-Hits: 0
X-Timer: S1617929817.638826,VS0,VE15
Vary: Accept-Encoding
X-Fastly-Request-ID: 6af68027070b551941f957b04f246237c7ddc217
```

Location

https://app.octo.fi

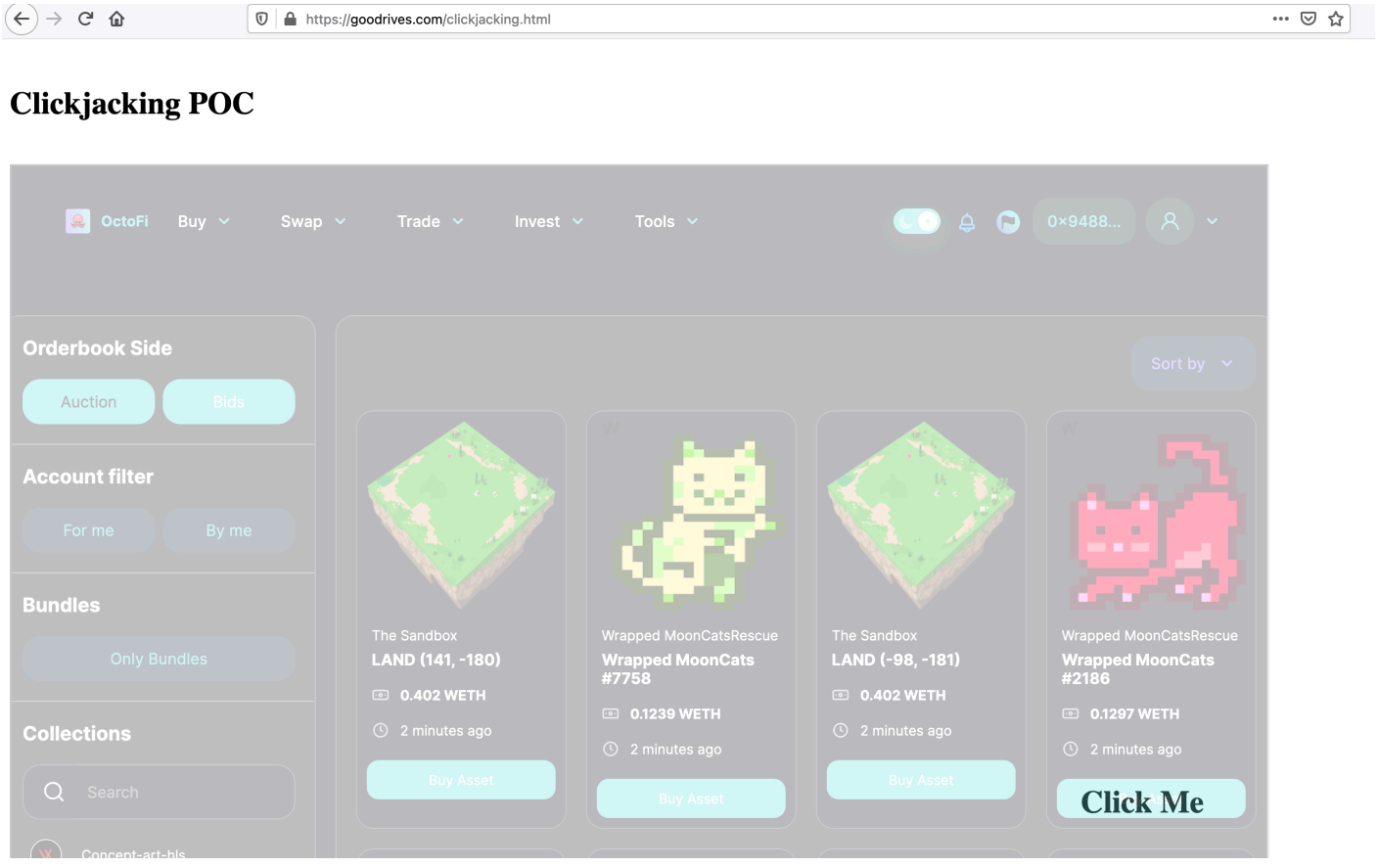
Impact

An attacker can trick the victim into clicking on actionable content in the Octofi application by loading it in a hidden iframe and cover it with decoy contents. The Proof of concept HTML file demonstrates an attacker can trick a user to purchase assets represented by NFT.

Step to Reproduce

1. Visit the website https://app.octo.fi with a browser and connect to the MetaMask wallet.
2. Save the HTML code snippet in the Proof of Concept section into a .html file.
3. Open the saved HTML file in the same browser in step 1.
4. Notice the application loaded in the iframe.

Evidence



Proof of Concept

```
<head>
  <style>
    #target_website {
      position:relative;;
      width:1280px;
      height:1280px;
      opacity:0.3;
      z-index:2;
      }
    #decoy_website {
      position:absolute;
      width:300px;
      height:610px;
      top:740px;
      left:1100px;
      z-index:1;
      }
  </style>
</head>
<body>
<br>
<h1>Clickjacking POC</h1>
<br>
  <div id="decoy_website">
    <h1>Click Me</h1>
  </div>
  <iframe id="target_website" src="https://app.octo.fi/#/invest/nft">
  </iframe>
</body>
```

Recommendation

Setting "X-Frame-Options" and "Content Security Policy" headers in the HTTP response is the most effective way to protect the application from clickjacking.

With X-Frame-Options:

Deny the website loading in any iframe with the "deny" directive:
X-Frame-Options: deny
Restricted to the same origin as the website using the sameorigin directive:
X-Frame-Options: sameorigin
Restricted to a specific website using the allow-from directive:
X-Frame-Options: allow-from https://example.com

With Content Security Policy:

Deny the website loading in any iframe with the "none" directive:
Content-Security-Policy: frame-ancestors 'none'
CSP whitelists frames to the same domain only:
Content-Security-Policy: frame-ancestors 'self'
Restricted to a specific website with:
Content-Security-Policy: frame-ancestors example.com

OTF-04: Reconstruct source code with source map

Severity: Informational

Introduction:

JavaScript running in a page is often machine-generated, as when compiled from a language like CoffeeScript or TypeScript. The JavaScript sources executed by the browser are often transformed in some way from the original sources created by a developer. Sources are often combined and minified to make delivering them from the server more efficient. In these situations, it's much easier to debug the original source, rather than the source in the transformed state that the browser has downloaded. A "source map" is a file that maps from the transformed source to the original source, enabling the browser to reconstruct the original source and present the reconstructed original in the debugger.

Description:

The web application front-end is written in React. In the compile-time, react converts source files into a minified JavaScript file. We were able to recover the original React source code in Chrome and Firefox with the JavaScript source and the source map file.

Location

https://app.octo.fi/static/js/main.99d98cdc.chunk.js.map
https://app.octo.fi/static/js/9.93104ce0.chunk.js.map
https://app.octo.fi/static/js/10.2a29b67d.chunk.js.map
https://app.octo.fi/static/js/29.45cabc44.chunk.js.map

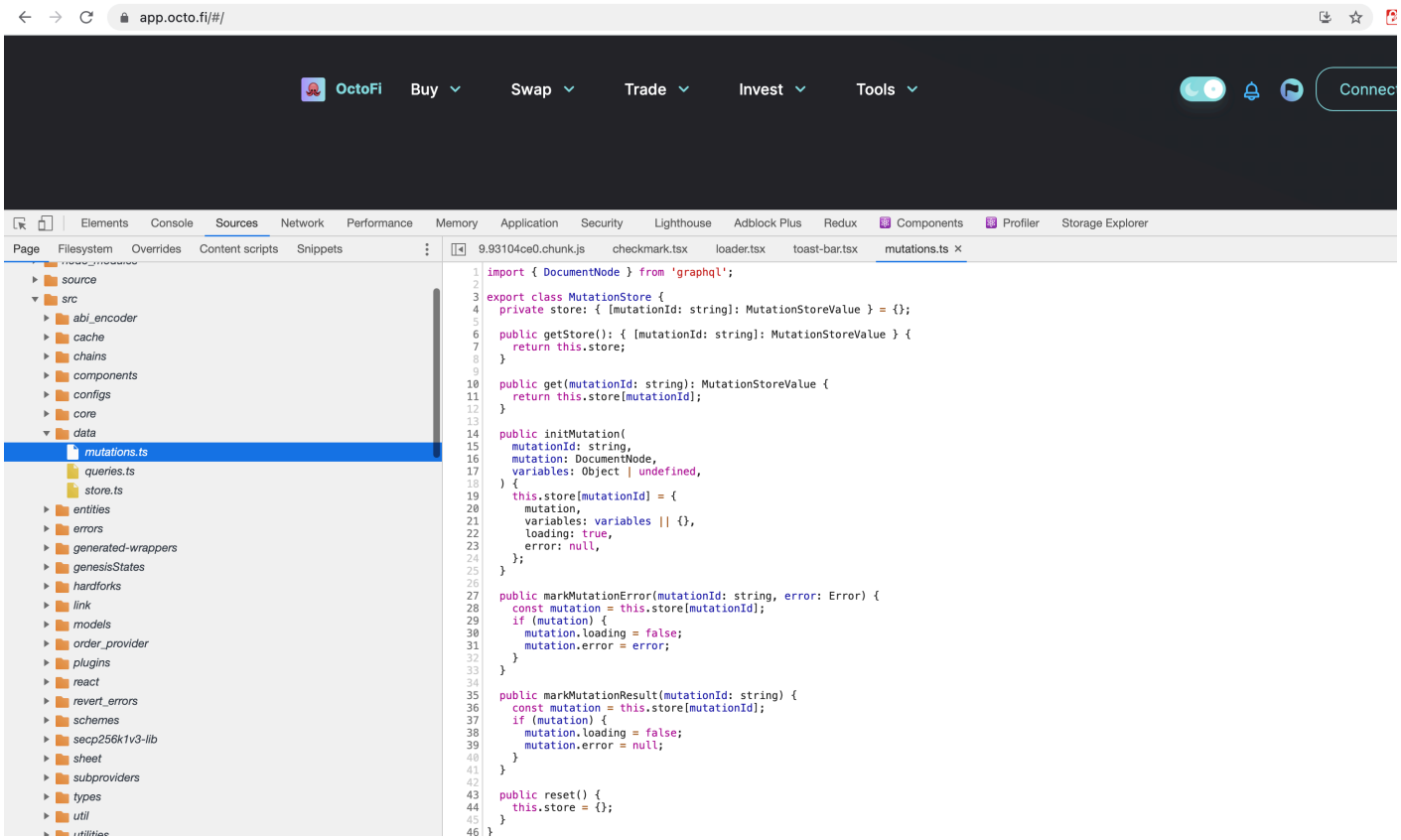
Impact

An attacker can gain important detail on code structure or potential flaws in the application by analyzing the front-end source code. This information can potentially be used by an attacker to perform more sophisticated attacks. Although the attacker can still obtain the JavaScript source without the source map, it will be much harder for them to analyze the minified version of the JavaScript code.

Step to Reproduce

1. Visit the application https://app.octo.fi/ with Chrome or Firefox.
2. In Chrome, right-click the web page, select "Inspect" and navigate to the "sources" tab. In Firefox, right-click the web page, select "Inspect Element" and navigate to the "Debugger" tab.
3. The Front-end source should present under the "app.octo.fi" section.

Evidence



Recommendation

Reconstruct program code with the source map file makes the debugging process much easier. However, the source map file should be removed in the production environment.

Two recommend ways to remove source map are:

- Add "GENERATE_SOURCEMAP=false" in the build script in the "package.json" file. Specifically, change line 204 in package.json to the following:

```
"build": "react-scripts  --max_old_space_size=4096 build
&&GENERATE_SOURCEMAP=false"
```

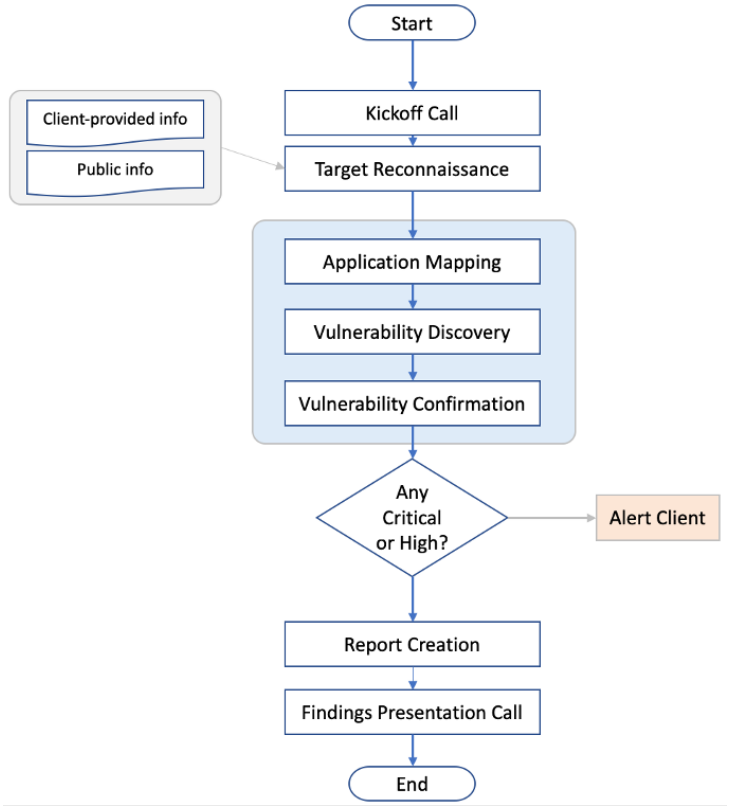
- Remove source map files after building the application.

Reference:
https://stackoverflow.com/questions/51984146/how-to-disable-source-maps-for-react-js-application
https://github.com/facebook/create-react-app/issues/1341

Appendix – Methodology

CertiK uses a comprehensive penetration testing methodology which adheres to industry best practices and standards in security assessments including from OWASP (Open Web Application Security Project), NIST, PTES (Penetration Testing Execution Standard).

Below is a flowchart of our assessment process:



Coverage and Prioritization

As many components as possible will be tested manually. Priority is generally based on three factors: critical security controls, sensitive data, and the likelihood of vulnerability.

Critical security controls will always receive the top priority in the test. If a vulnerability is discovered in the critical security control, the entire application is likely to be compromised, resulting in a critical-risk to the business. For most applications, critical controls will include the login page, but it could also include major workflows such as the checkout function in an online store.

The Second priority is given to application components that handle sensitive data. This is dependent on business priorities, but common examples include payment card data, financial data, or authentication credentials.

Final priority includes areas of the application that are most likely to be vulnerable. This is based on CertiK' experience with similar applications developed using the same technology or with other applications that fit the same business role. For example, large applications will often have older sections that are less likely to utilize modern security techniques.

Reconnaissance

CertiK gathers information about the target application from various sources depending on the type of test being performed. CertiK obtains whatever information that is possible and appropriate from the client during scoping and supplements it with relevant information that can be gathered from public sources. This helps provide a better overall picture and understanding of the target.

Application Mapping

CertiK examines the application, reviewing its contents, and mapping out all its functionalities and components. CertiK makes use of different tools and techniques to traverse the entire application and document all input areas and processes. Automated tools are used to scan the application and it is then manually examined for all its parameters and functionalities. With this, CertiK creates and widens the overall attack surface of the target application.

Vulnerability Discovery

Using the information that is gathered, CertiK comes up with various attack vectors to test against the application. CertiK uses a combination of automated tools and manual techniques to identify vulnerabilities and weaknesses. Industry-recognized testing tools will be used, including Burp Suite, Nikto, Metasploit, and Kali. Furthermore, any controls in place that would inhibit the successful exploitation of a particular system will be noted.

Vulnerability Confirmation

After discovering vulnerabilities in the application, CertiK validates the vulnerabilities and assesses its overall impact. To validate, CertiK performs a Proof-of-Concept of an attack on the vulnerability, simulating real world scenarios to prove the risk and overall impact of the vulnerability.

Through CertiK' knowledge and experience on attacks and exploitation techniques, CertiK is able to process all weaknesses and examine how they can be combined to compromise the application. CertiK may use different attack chains, leveraging different weaknesses to escalate and gain a more significant compromise.

To minimize any potential negative impact, vulnerability exploitation was only attempted when it would not adversely affect production applications and systems, and then only to confirm the presence of a specific vulnerability. Any attack with the potential to cause system downtime or seriously impact business continuity was not performed. Vulnerabilities were never exploited to delete or modify data; only read-level access was attempted. If it appeared possible to modify data, this was noted in the list of vulnerabilities below.

Immediate escalation of High or Critical Findings

If critical or high findings are found whereby application elements are compromised, client’s key security contacts will be notified immediately.

Vulnerability Classes

Security Misconfiguration	• Missing Security Headers • Debugging Enabled
Information Disclosure	• Directory Indexing • Verbose Error Messages • HTML Comments
Account Policy	• Default / Weak Passwords • Unlimited Login Attempts • Password Reset • Insufficient Session Expiration
Session Management	• Session Identifier Prediction • Session Hijacking • Cross-Site Request Forgery • Insufficient Session Expiration
Injection	• SQL Injection • Cross-Site Scripting • HTML Injection • XML Injection • OS Command Injection
Broken Access Control	• Authentication Bypass • Authorization Bypass • Privilege Escalation
Application Resource Handling	• Path Traversal • Predictable Object Identifiers • XML External Entity Expansion • Local & Remote File Inclusion
Logic Flaws	• Abuse of Functionality • Workflow Bypass
Insufficient Cryptography	• Weak Hashing Algorithms • Weak Encryption Algorithms • Hard Coded Cryptographic Key
Denial of Service	• Server-side Denial of service • Client-side Denial of service

Risk Assessment

The following risk levels categorize the risk level of issues presented in the report:

Risk Level	CVSS Score	Impact	Exploitability
Critical	9.0-10.0	Root-level or full-system compromise, large-scale data breach	Trivial and straightforward
High	7.0-8.9	Elevated privilege access, significant data loss or downtime	Easy, vulnerability details or exploit code are publicly available, but may need additional attack vectors (e.g., social engineering)
Medium	4.0-6.9	Limited access but can still cause loss of tangible assets, which may violate, harm, or impede the org's mission, reputation, or interests.	Difficult, requires a skilled attacker, needs additional attack vectors, attacker must reside on the same network, requires user privileges
Low	0.1-3.9	Very little impact on an org’s business	Extremely difficult, requires local or physical system access
Informational	0.0	Discloses information that may be of interest to an attacker.	Not exploitable but rather is a weakness that may be useful to an attacker should a higher risk issue be found that allows for a system exploit