Penetration Testing Report for Centralized Exchange

Testers

1. Or Duan
2. Omri Shdaimah

Table of Contents

Management Summary

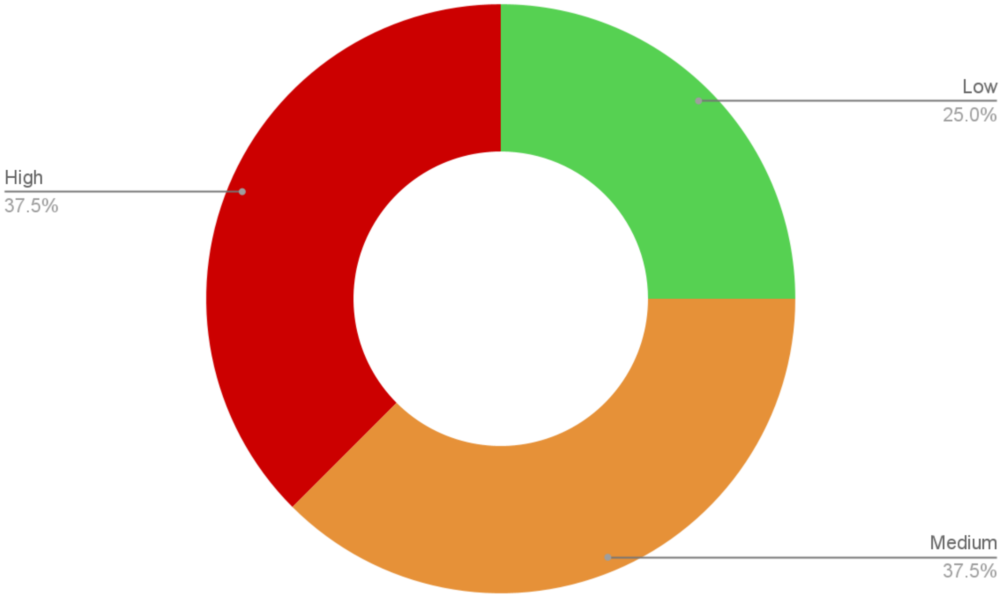
Centralized Exchange team contacted Sayfer in order to perform full blackbox penetration testing on their web application and whitebox review for their crypto architecture in December 2021.

Before assessing the above services, we held a kickoff meeting with the Centralized Exchange technical team and received an overview of the system and the goals for this research.

Over the research period of 4 weeks, we discovered 10 vulnerabilities in the system. The most dangerous vulnerabilities were SQL injection and flaws in business logic.

The impact on the system is critical as a malicious attacker could exploit some of these vulnerabilities to take advantage of the system, either by changing his user role to “super_user” via the SQL injection or by abusing the system and stealing money from the Centralized Exchange using the 30s system update mechanism.

Vulnerabilities by Risk



Risk	Low	Medium	High	Informational
# of issues	2	3	3	0

- **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.

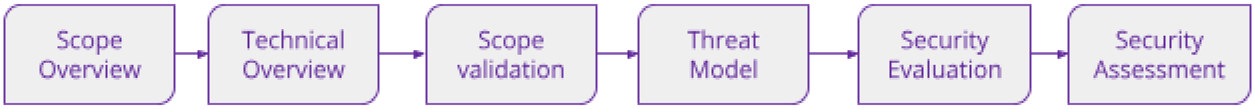
Approach

Introduction

The Centralized Exchange team contacted Sayfer in order to perform full grey-box penetration testing on the Centralized Exchange application, and to perform white-box security auditing of the Centralized Exchange business logic and code from a cryptocurrency point of view.

This report documents the research carried out by Sayfer targeting the selected resources defined under the research scope. Particularly, this report displays the security posture review for the Centralized Exchange application and code, and its surrounding infrastructure and process implementations.

Our penetration testing project life cycle:



Scope Overview

During our first meeting and 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]
- [redacted]

Our tests were performed between 21/12/2021 to 17/01/2022

Scope Validation

We began by ensuring that the scope defined to us by the client was technically logical. Deciding what scope is right for a given system is part of the initial discussion. Getting the scope right is key to deriving maximum business value from the research.

Threat Model

During our kickoff meetings with the client we defined the most important assets the application possesses.
We defined that the largest current threat to the system is manipulating the users and [redacted] financial assets.

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.

Security Evaluation

The following tests were conducted while auditing the system

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

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

Identity Management Testing	Test Name
WSTG-IDNT-01	Test Role Definitions
WSTG-IDNT-02	Test User Registration Process
WSTG-IDNT-03	Test Account Provisioning Process
WSTG-IDNT-04	Testing for Account Enumeration and Guessable User Account
WSTG-IDNT-05	Testing for Weak or unenforced username policy

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

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

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

Data Validation Testing	Test Name
WSTG-INPV-01	Testing for Reflected Cross Site Scripting
WSTG-INPV-02	Testing for Stored Cross Site Scripting
WSTG-INPV-03	Testing for HTTP Verb Tampering
WSTG-INPV-04	Testing for HTTP Parameter Pollution
WSTG-INPV-05	Testing for SQL Injection
WSTG-INPV-06	Testing for LDAP Injection
WSTG-INPV-07	Testing for XML Injection
WSTG-INPV-08	Testing for SSI Injection

WSTG-INPV-09	Testing for XPath Injection
WSTG-INPV-10	Testing for IMAP SMTP Injection
WSTG-INPV-11	Testing for Code Injection
WSTG-INPV-12	Testing for Command Injection
WSTG-INPV-13	Testing for Format String Injection
WSTG-INPV-14	Testing for Incubated Vulnerability
WSTG-INPV-15	Testing for HTTP Splitting Smuggling
WSTG-INPV-16	Testing for HTTP Incoming Requests
WSTG-INPV-17	Testing for Host Header Injection
WSTG-INPV-18	Testing for Server-side Template Injection
WSTG-INPV-19	Testing for Server-Side Request Forgery

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

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

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

Client Side Testing	Test Name
WSTG-CLNT-01	Testing for DOM-Based Cross Site Scripting
WSTG-CLNT-02	Testing for JavaScript Execution
WSTG-CLNT-03	Testing for HTML Injection
WSTG-CLNT-04	Testing for Client Side URL Redirect
WSTG-CLNT-05	Testing for CSS Injection
WSTG-CLNT-06	Testing for Client Side Resource Manipulation
WSTG-CLNT-07	Test Cross Origin Resource Sharing
WSTG-CLNT-08	Testing for Cross Site Flashing
WSTG-CLNT-09	Testing for Clickjacking

WSTG-CLNT-10	Testing WebSockets
WSTG-CLNT-11	Test Web Messaging
WSTG-CLNT-12	Testing Browser Storage
WSTG-CLNT-13	Testing for Cross Site Script Inclusion

API Testing	**Test Name**
WSTG-APIT-01	Testing GraphQL

Crypto Wallet Review	**Test Name**
SAYFER-CRPTW-01	Test Trade Business Logic
SAYFER-CRPTW-03	Test UTXO-based cryptocurrency node configurations
SAYFER-CRPTW-04	Test account-based cryptocurrency code configurations
SAYFER-CRPTW-05	Test transaction confirmation critical code
SAYFER-CRPTW-06	Test TAPROOT support
SAYFER-CRPTW-07	Test private key storage

Security Assessment Findings

Saving Private MPC Keys Insecurely

ID	SAYFER-CRPTW-07
Risk	High
Required Skill	High
OWASP Reference	-
Location	-
Tools	Configuration Audit
Description	Centralized exchanges suffer from low-quality key management practices. There are many examples of such cases where the keys were lost or stolen causing the service to lose all the wallet funds or lock the funds completely. During our configuration files audit, we went over the key management storage. We found that the keys that are being used for the cold multi-sig wallet are not stored in distributed enough places. There are 3 keys that are being used within the MPC key signing. 1 is stored in a physical protected machine. The other 2 are stored within the same dedicated machine in GCP. There are a couple of security measurements taken to secure these machines but the problem relies on the distribution, if the machine deployed on GCP gets compromised, an attacker can sign any transaction from the cold wallet. This is a high-risk and sensitive place where many have failed in the past, best practices should be strictly followed.
Mitigations	Use 3rd party custodian service to manage hot wallets and vaults. We will be happy to recommend one of our partners. These services handle MPC and key management for you, with other added security layers making the use of such services the best choice for centralized exchanges.

SQL Injection

ID	WSTG-INPV-05
Risk	High
Required Skill	Medium
OWASP Reference	- Link
Location	- [redacted]
Tools	Burp Repeater, sqlmap, PayloadAllTheThings
Description	An SQL injection attack involves inserting or "injecting" a partial or complete SQL query into the data input that is transmitted from the client to the web application. A successful SQL injection attack can read sensitive data from the database, modify database data (insert/update/delete), perform database administration operations (such as shutting down the DBMS), recover the content of a given file on the DBMS file system or write files into the file system, and, in some cases, issue commands to the operating system. Using the `transaction` endpoint we were able to abuse the `more` URL query parameter to injection malicious SQL payload: `/api/transactions?size=10&more=`**`te');INJECTION_PAYLOAD`** The payload we used was: `/api/transactions?size=10&sort=time,DESC&more=te');SELECT+CASE+WHEN+(substring(version(),12,2)+%3d+'10')+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END%3b` - Which in this case, checks if the running instance of Postgres is version 10 or not. An attacker that exploits this vulnerability could take over the system. We were able to extract the table schemas, update our own user's role or dump any information saved on the DB and even changed our user's balance on the DB.
Mitigations	SQL injection vulnerability is easy to fix but hard to mitigate. Strong linting or implementation of compiling rules that enforce future change are needed. Mitigation of SQL injection vulnerabilities is usually done by following a framework of choice, which means that the developer should never concatenate strings into a full SQL statement. Every user input should be sanitized into an SQL executor rather than being used as a simple SQL query string. For more information about SQL injection perfection please refer to the SQL Injection Prevention CheatSheet.

Insecure Direct Object References

ID	WSTG-ATHZ-04
Risk	High
Required Skill	Medium
OWASP Reference	LInk
Location	- [redacted]/dashboad/{DASHBOARD_ID}
Tools	Burp Repeater, DevTools

Description	Insecure direct object references (IDOR) are a type of access control vulnerability that arises when an application uses user-supplied input to access objects directly. As a result of this vulnerability, attackers can bypass authorization and access resources in the system directly, for example, database records or files.
	We found that the [redacted] API lets an attacker view other users' dashboard information, including all the financial data of this user.
	The vulnerability relies on the dashboard id parameter which is a guessable integer.
	Example request for a single dashboard (for a dashboard that **is not owned** by the current user):
	`GET [redacted]/dashboard/827371 HTTP/1.1`
	`Host: [redacted]`
	`api-key: [redacted]`
	`...`
	That indicates that the `/dashboard/DASHBOARD_ID` endpoint does not check for authorization for the requested resource. An authenticated attacker could scrape every single dashboard which holds information about the user funds and past transactions.

Mitigation	There are multiple ways to mitigate IDOR vulnerabilities, for this case it seems the solution might be to check for authorization for each and every request. This means that every request key will be able to fetch only its account’s dashboards

Weaker Authentication in Alternative Channel

ID	WSTG-ATHN-10
Risk	High
Required Skill	High
OWASP Reference	Link
Location	● [redacted]
Tools	Google Chrome, DevTools, amass, ffuf
Description	Even if the primary authentication mechanisms does not include any vulnerabilities, it may be that vulnerabilities exist in alternative legitimate authentication user channels for the same user accounts. This vulnerability is part of a chain of 2 vulnerabilities that enabled us to take over any account with just an email address. As part of our reconnaissance phase where we try to find a wider attack vector by enumerating the main target subsystems, we found an admin interface under the subdomain [redacted]. It is possible to login into the admin interface using a normal app user, but for almost all the network requests we inspected during the loading of the main page, the server returns a 401 error. *[IMAGE_REDACTED]* We reverse-engineered the main.js bundle file which has the front-end code for the app and found all the potential endpoints an admin can interact with. We could exploit only the endpoint of `api/updateUser`. The endpoint enabled us to edit any user email, and by doing so we were able to reset the victim's password and take over the account *[IMAGE_REDACTED]*
Mitigations	It is highly recommended to make an authentication mechanism or a VPN for debugging or for the administrative services of the system to prevent the presence of unsecured public applications that can be exploited by an attacker. In addition, there is an authorization mechanism in the admin interface, but this is out of the scope of this project.

Review Webserver Metafiles for Information Leakage

ID	WSTG-INFO-03
Risk	Medium
Required Skill	Low
OWASP Reference	Link
Location	• [redacted] • [redacted] • [redacted]
Tools	Chrome, go buster
Description	As part of our research about the target and its sub-domains we found some metafiles that should not be public, or at least not without the proper authentication mechanism. • [redacted].gitignore • [redacted]/docker-compose.yml • [redacted]/swagger-ui.html *[IMAGE_REDACTED]* *[IMAGE_REDACTED]* *[IMAGE_REDACTED]* We found three kinds of files that may harm [redacted] services, .gitignore, swagger-ui and the docker-compose.yml file. These three files reveal sensitive data about the service architecture. A malicious actor can use this information to increase his attack vector on the target.
Mitigations	If possible, remove these files from the public service or implement an authorization mechanism that grants access only to privileged users.

Missing Content Security Policy header

ID	SAYFER-CONFIG-008
Risk	Medium
Required Skill	High
OWASP Reference	-
Location	-
Tools	Burp, Web browser
Description	Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross-Site Scripting (XSS) and data injection attacks. We didn't find a CSP header in any of the server's responses. *[IMAGE_REDACTED]* By using CSP website administrators add another line of defense against XSS or clickjacking attacks, by doing so the system will be safe even if future unsecured changes are made to the source code. A basic CSP policy should at least describe the default whitelisted domains for static files (like scripts, images, and CSS). And `frame-ancestors` to prevent clicking-jacking attacks. More info: 1. https://cspvalidator.org/ 2. https://csp-evaluator.withgoogle.com/
Mitigations	Adding the `Content-Security-Policy: [policy]` on every response where loading external resources could be dangerous We highly recommend using it and testing it first with the "Report-Only" variation to test your policy before releasing it to production: `Content-Security-Policy-Report-Only: [policy]`

Testing for Security Headers

ID	SAYFER-CONFIG-009
Risk	Medium
Required Skill	High
OWASP Reference	-
Location	- [redacted]
Tools	Burp, Web browser
Description	Browsers support many HTTP headers that can improve applications security to protect against a variety of common attacks, the headers are exchanged between a web client (usually a browser) and a server to specify the security-related details of HTTP communication. When looking at [redacted] security headers the following are missing: ● X-Content-Type-Options Setting this header will prevent the browser from interpreting files as something other than what is declared by the content type in the HTTP headers. ● Strict-Transport-Security HSTS is a web security policy mechanism that helps to protect websites against protocol downgrade attacks and cookie hijacking. It allows web servers to declare that web browsers should only interact with it using secure HTTPS connections, and never via the insecure HTTP protocol. ● Referrer-Policy The Referer header is a request header that indicates the site from which the traffic originated. If there is no adequate prevention in place, the URL itself, and even sensitive information contained in the URL will be leaked to the cross-origin site. ● Access-Control-Allow-Origin The header has the value of “*” which exposes the API for every website, this might not be the desired outcome.
Mitigations	Adding the headers mentioned above to all back-end services.

Review Webserver Metafiles for Information Leakage

ID	WSTG-INFO-03
Risk	Low
Required Skill	Medium
OWASP Reference	Link
Location	-
Tools	DevTools
Description	While researching the target with DevTool we were able to view the frontend source code without any obfuscation. This vulnerability occurs because the JS bundles are shipped with sourcemaps to production, which make it possible to read the original source code with comments that might reveal information, for instance, the following paths.ts file: [redacted]/paths.ts [IMAGE_REDACTED] By having the sourcemap, an attacker can learn about the code base, read comments, and find deprecated code parts which later can be used to find vulnerabilities.
Mitigations	Do not ship sourcemaps to production, most logging and error tracing systems have an opinion to upload the sourcemaps to a back-office system. Another approach would be to serve the sourcemaps to only authenticated users via VPN or other mechanisms.

Fingerprint Web Server

ID	WSTG-INFO-002
Risk	Low
Required Skill	Medium
OWASP Reference	LInk
Location	- [redacted]
Tools	Burp
Description	While exposed server information in itself is not necessarily a vulnerability, it is information that can assist attackers in exploiting other vulnerabilities that may exist. Most of the endpoints are not disclosing any information about the server through HTTP headers or error pages. Using the following mal-formed HTTP request we were able to fingerprint an Nginx server through a 400 response GET /v2 **HTTPMALFORMED**/1.1 Host: [redacted] Accept: */* The response body is: <html> <head><title>400 Bad Request</title></head> <body bgcolor="white"> <center><h1>400 Bad Request</h1></center> <hr><center>**nginx 1.14.0**</center> </body> </html>
Mitigation	There are different ways to obscure web server headers, the most commonly used methods are: 1. Reverse proxy servers that stand between the global internet and the internal servers. 2. Configure each web server to strip these headers.

Appendix A: Security Evaluation Fixes

Will be updated by the Sayfer team after the first revision.