DeHacker

Code Security Assessment

ATLAS WALLET

October 5st, 2024

Contents

Contents

Summary

DeHacker's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best prac tices. Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow/underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service/logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Issue Categories

Every issue in this report was assigned a severity level from the following:

Critical severity issues

A vulnerability that can disrupt the contract functioning in a number of scenarios or creates a risk that the contract may be broken.

Major severity issues

A vulnerability that affects the desired outcome when using a contract or provides the opportunity to use a contract in an unintended way.

Medium severity issues

A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.

Minor severity issues

A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.

Informational

A vulnerability that has informational character but is not affecting any of the code.

Overview

Project Summary

Project Name	ATLAS WALLET
Platform	Extension/ Mobile Application
Website	atlaswallet.com
Type	DeFi
Language	Pentest
Codebase	

Vulnerability Summary

Vulnerability Level	Total	Mitigated	Declined	Acknowledged	Partially Resolved	Resolved
Critical	2	0	0	0	0	2
High	1	0	0	0	0	1
Medium	4	0	0	0	0	4
Low	7	0	0	1	0	6
Informational	0	0	0	0	0	0
Discussion	0	0	0	0	0	0

Audit scope

Mobile App	Android
Mobile App	iOS
Extension	Google Chrome
API	dev-api.atlaswallet.com

Findings

ID	Title	Severity	Status
GLOBAL-03	Broken Access Control Throughout The Application	Critical	Resolved
GLOBAL-06	SQL Injection	Critical	Resolved
GLOBAL-07	Verbose SQL Error Message	High	Resolved
GLOBAL-08	Lack Of "Attestation" Origin VerificationDuring WalletConnect Connection	Medium	Resolved
GLOBAL-09	Insecure Fetch Design	Medium	Resolved
GLOBAL-13	AES Encryption Without IV	Medium	Resolved
GLOBAL-14	Insecure Database Credentials	Medium	Resolved
GLOBAL-01	ATS Misconfiguration	Low	Resolved
GLOBAL-02	Sensitive Information Disclosed InApplication Memory	Low	Resolved
GLOBAL-04	Lack Of Certificate Pinning	Low	Resolved
GLOBAL-05	Source Map Files Not Removed	Low	Resolved
GLOBAL-10	Public Swagger Documentation	Low	Resolved
GLOBAL-11	Using Components With KnownVulnerabilities	Low	Acknowledged
GLOBAL-12	Insecure Android Configuration	Low	Resolved

CRITICAL

GLOBAL-03| Broken Access Control throughout the application

Issue	Severity	Location	Status
Broken Access Control	Critical	https://dev-api.atlaswallet.com/	Resolved

Description

The authentication mechanism of the wallet relies on an authorization token transmitted via the Authorization HTTP header. However, the backend does not check if the authorization token belongs to the wallet address. An attacker only needs to know the user's wallet address to access and modify its content.

Several endpoints are susceptible to this vulnerability. It is recommended to review the entire application. The following sections will detail possible exploitation.

Recommendation

Ensure that strong access controls are in place to prevent unauthorized access to sensitive information. Authorization checks should be performed prior to providing users access to information within the application. Any part of the application that relies on user input to return data should perform server-side authorization checks. Do not rely on authorization checks that are performed client-side, as the client may be able to manipulate and bypass these checks. Ensuring that server-side authorization checks are in place will help prevent users from gaining unauthorized access to sensitive data within the application

CRITICAL

Recommendation

In addition to implementing proper access controls and authorization mechanisms for the API endpoint, it is recommended that a new parameter called "signature" be introduced. This parameter would be obtained from the user's wallet and could be used further to enhance the security and integrity of the order-closing process. By requiring a valid signature from the user's wallet, the application can confirm the authenticity of the request and ensure that it is being made by the user who is authorized to perform the action. This can help to prevent malicious actors from exploiting the API endpoint and further mitigate the risk of unauthorized access or fraudulent activity.

MAJOR

GLOBAL-06| SQL injection

Issue	Severity	Location	Status
Injection	Critical	https://dev-api.atlaswallet.com/dev/api/v1/wallet/transaction/list	Resolved

Description

The backend uses sequelize ORM (v6.35.1) to communicate with the database. This ORM includes the Sequelize.literal() function, which allows inserting of SQL statements without performing the security checks implemented by the ORM.

During the code review, the use of the Sequelize.literal() function in various endpoints was identified. Additionally, it has been possible to exploit SQL injection due to unfiltered user input data is concatenated in literal function.

The insecure functions is used in the following files:

- atlas_backend-master/src/admin_module/wallet/wallet_controllers.ts
- atlas_backend-master/src/modules/extension/extension.controller.ts
- atlas_backend-master/src/modules/on_off_ramp/on_off_ramp.controller.ts
- atlas_backend-master/src/modules/swft/swft.controller.ts
- atlas_backend-master/src/modules/wallet/wallet.controller.ts
- atlas_backend-master/src/modules/wallet/wallet.helper.ts
- atlas_processes-master/crons/src/modules/common/common.controller.ts
- atlas_processes-master/crons/src/modules/prices/chain_link_prices.controller.ts
- atlas_processes-master/crons/src/modules/prices/dex_prices.controller.ts
- atlas_processes-master/crons/src/modules/prices/prod_dex_prices.controller.ts

Recommendation

The requested item pointed out may not be the only one with SQL injection vulnerability. It is recommended that the team thoroughly examine all server-side code to replace the use of the Sequelize.literal() function with secure functions from the ORM.

Additionally, ensure that proper server-side input validation is performed on all sources of user input. Various protections should be implemented using the following in order of effectiveness:

- Errors: Ensure that SQL errors are turned off and not reflected back to a user when an error occurs as to not expose valuable information to an attacker.
- Parameterize Queries: Ensure that when a user's input is added to a backend SQL query, it is not string appended but placed into the specific SQL parameter. The method to perform this varies from language to language.
- Server-Side Input Length: Limit the length of each field depending on its type. For example, a name should be less than 16 characters long, and an ID should be less than 5 characters long.
- Whitelist: Create character ranges (ie. Numeric, alpha, alphanumeric, alphanumeric with specific characters) and ensure that each input is restricted to the minimum length whitelist necessary.
- Blacklist: Disallow common injection characters such as "<>\/?*()&, SQL and SCRIPT commands such as SELECT, INSERT, UPDATE, DROP, and SCRIPT, newlines %0A, carriage returns %0D, null characters %00 and unnecessary or bad encoding schemas (malformed ASCII, UTF-7, UTF-8, UTF-16, Unicode, etc.).
- Logging and Web Specific IDS/IPS (Intrusion Detection/Prevention System): Ensure that proper logging is taking place and is being reviewed, and any malicious traffic which generates an alert is promptly throttled and eventually blacklisted

HIGH

GLOBAL-07 | Verbose SQL error message

Issue	Severity	Location	Status
Information Disclosure	High		Resolved

Description

Improper handling of errors can introduce a variety of security problems for an application server. The most common is the application returning detailed stack traces, database dumps, and source code to the user when the server encounters an error. These messages reveal implementation details that should never be revealed.

The application returns a verbose error message with a SQL error with part of the statement. This issue allows for the successful exploitation of the finding GLOBAL-05: SQL Injection.

The application use the handle_catch function to handle errors. However, when the error is not undefined the function return the original error message. The function is located in the ./atlas_backend-master/src/helpers/global.helper.ts.

```
static handle_catch = (res: Response, error: any) => {
        console.log("handle_catch error", error)
        let message = "Something went wrong";
        if (error !== undefined) {
            message = Object.keys(error).length > 0 &&
error?.message != undefined ? error.message : error;
                 if(error?.error == "Returned error: insufficient
funds for gas * price + value"){
                     message = "Insufficient funds.";
                 }
            }
```

Recommendation

To mitigate this issue, the application server should handle server error properly and return a more generic error message.

HIGH

GLOBAL-08 | Lack of "Attestation" Origin Verification During WalletConnect Connection

Issue	Severity	Location	Status
Security Misconfiguration	High		Resolved

Description

The WalletConnect protocol recently launched a Verify API, which is used to confirm whether the connected origin matches the registered origin on the WalletConnect site. This verification occurs during the connection confirmation in the user's wallet. However, the wallet failed to ensure that the returned origin from the "Attestation" API matches the original origin from the QR code.

Recommendation

It is recommended that the wallet verifies the returned "origin" value of the "attestation" request against the website the user is attempting to connect to. For more information regarding the Verify API, please refer to:

- https://medium.com/walletconnect/unlocking-the-power-of-verify-api-a-step-by-step-guide-for-wallets-4e939a273d9a
- https://docs.walletconnect.com/web3wallet/verify

MEDIUM

GLOBAL-09| Insecure fetch design

Issue	Severity	Location	Status
Logic Flaws	Medium	https://dev-api.atlaswallet.com/dev/api/v1/extension/fetch_price	Resolved

Description

The API includes some endpoints that are responsible for fetching data from third parties. The implementation of the fetch functionality is considered insecure as it receives part of the URL as a parameter from client side.

Recommendation

It is recommended to abstract fetching functionalities so that the final URL cannot be modified on the client side. If it is necessary to include client-side information, it is crucial to filter data inputs before performing the data fetch.

Additionally, it is recommended to review all the fetch functions and enhance them in accordance with the recommendation.

MEDIUM

GLOBAL-13| AES encryption without IV

Issue	Severity	Location	Status
Insufficient Cryptography	High	Google Chrome Extension	Resolved

Description

The extension store encrypts the mnemonic using the AES algorithm from crypto-js module. "However, it is being done insecurely as an Initialization Vector (IV) is not being used.

File: security.helper.js

```
export class SecurityHelper {
    static encrypt = (payload, password) => {
        const data = AES.encrypt(JSON.stringify(payload),
password);
       return data.toString();
    };
```

The standard practice in AES encryption involves using an IV to introduce randomness, ensuring that the same plaintext input does not result in identical ciphertext output when encrypted with the same key.

Recommendation

It is recommended to use the AES algorithm with Initialization Vector (IV).

MEDIUM

GLOBAL-14| Insecure database credentials

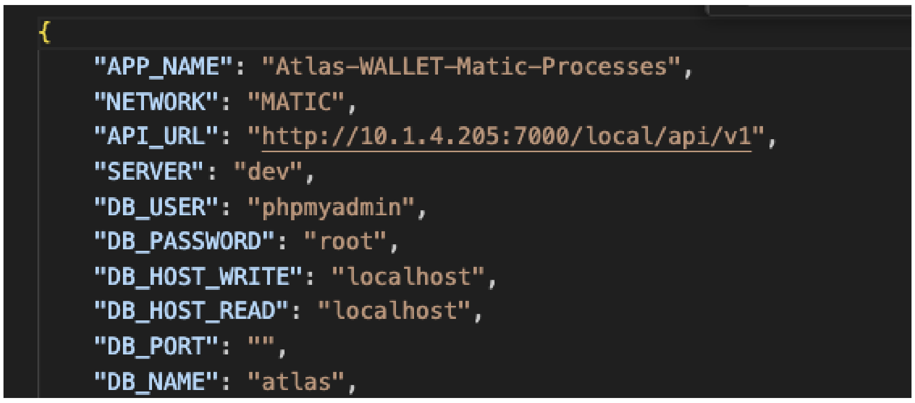
Issue	Severity	Location	Status
Security Misconfiguration	Medium	atlas_processes-master/*/src/config/zookeeper.config.json	Resolved

Description

An authentication mechanism is only as strong as its credentials. For this reason, it is important to use strong passwords. Lack of password complexity significantly reduces the search space when trying to guess the passwords, making brute-force attacks easier.

When reviewing the application's source code, the tester found the database's password. The password is considered insecure.

Database's credential



```
{
    "APP_NAME": "Atlas-WALLET-Matic-Processes",
    "NETWORK": "MATIC",
    "API_URL": "http://10.1.4.205:7000/local/api/v1",
    "SERVER": "dev",
    "DB_USER": "phpmyadmin",
    "DB_PASSWORD": "root",
    "DB_HOST_WRITE": "localhost",
    "DB_HOST_READ": "localhost",
    "DB_PORT": "",
    "DB_NAME": "atlas",
```

Recommendation

Ensure that the application uses a strong password for the credentials used to access its infrastructure. Strong password policies should include the following rules:

- Require at least eight characters with letter, number, and special character
- Do not allow usernames to be included in the password
- Set a maximum age for the password
- Disallow password reuse

LOW

GLOBAL-01 | ATS Misconfiguration

Issue	Severity	Location	Status
Security Misconfiguration	Low	iOS application	Resolved

Description

App Transport Security (ATS) is a set of security checks that the operating system enforces when making connections with NSURLConnection, NSURLSession and CFURL to public hostnames. ATS is enabled by default for applications build on iOS SDK 9 and above.

During the pentest, it is observed that on the iOS App, the App Transport Security restrictions are disabled for all network connections.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>NSAllowsArbitraryLoads</key>
        <true/>
        <key>NSExceptionDomains</key>
        <dict>
            <key>New Exception Domain</key>
            <dict>
<key>NSExceptionAllowsInsecureHTTPLoads</key>
                    <true/>
                    <key>NSIncludesSubdomains</key>
                    <true/>
              </dict>
              <key>localhost</key>
              <dict>
<key>NSExceptionAllowsInsecureHTTPLoads</key>
                   <true/>
              </dict>
          </dict>
     </dict>
```

Description

Disabling ATS means that insecure HTTP connections are allowed. HTTPS connections are also allowed and are still subject to default server trust evaluation. However, extended security checks like requiring a minimum Transport Layer Security (TLS) protocol version are disabled.

Additionally, the configuration includes a key that is not correct. The <key>New Exception Domain</key> key should be the domain to exclude.

- https://developer.apple.com/documentation/bundleresources/information_property_list/nsapptransportsecurity/

Recommendation

Set the NSAllowArbitraryLoads key value to False in info.plist file. If it's required to be set to Yes for some functionalities then Implement the following list of App Transport Security Requirements;

- Enable additional security features like Certificate Transparency using the NSRequiresCertificateTransparency key.
- Reduce or remove security requirements for communication with particular servers using the NSExceptionDomains key.
- Ensure that the X.509 Certificate has a SHA256 fingerprint and must be signed with at least a 2048-bit RSA key or a 256-bit Elliptic-Curve Cryptography (ECC) key.
- Transport Layer Security (TLS) version must be 1.2 or above and must support Perfect Forward Secrecy (PFS) through Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) key exchange and AES-256 symmetric ciphers.
- If the application connects to a defined number of domains that the application owner controls, then configure the servers to support the ATS requirements and opt-in for the ATS requirements within the application according to best practices defined by Apple.

LOW

GLOBAL-02| Sensitive Information Disclosed in Application Memory

Issue	Severity	Location	Status
Information Disclosure	Low	Android applicationi, iOS application, Browser extension	Resolved

Description

Several credentials are available in the memory of the application after they are no longer used.

- Mnemonic
- PIN code
- Private key

This allows for an attacker with physical access to the user's system to access the memory and steal this information. The clear text-sensitive information in memory should be reset after its use, or after the user logs out of the application.

The finding affects both mobile applications and the browser extension.

Recommendation

It is recommended to clear sensitive values from application memory after they are used.

Do not use immutable structures (e.g., String and BigInteger) to represent secrets. Nullifying these structures will be ineffective: the garbage collector may collect them, but they may remain on the heap after garbage collection. Nevertheless, you should ask for garbage collection after every critical operation (e.g., encryption, parsing server responses that contain sensitive information). When copies of the information have not been properly cleaned (as explained below), your request will help reduce the length of time for which these copies are available in memory.

To properly clean sensitive information from memory, store it in primitive data types, such as byte-arrays (byte[]) and char-arrays (char[]).

Reference

- https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05d-Testing-Data-Storage.md#checking-memory-for-sensitive-data-mstg-storage-10
- https://books.nowsecure.com/secure-mobile-development/en/coding-practices/securely-store-sensitive-data-in-ram.html
- https://developer.android.com/training/articles/security-tips#UserData

LOW

GLOBAL-04| Lack of Certificate Pinning

Issue	Severity	Location	Status
Security Misconfiguration	Low	iOS application	Resolved

Description

The application does not implement certificate pinning. Certificate pinning is the act of associating a host with its expected certificate within the application. When the application connects to the host, the stored certificate is compared to the certificate held by the remote host. If the two certificates do not match, the request is dropped. Currently, the application only verifies that the server presents a TLS certificate that is trusted by the Android or iOS trust stores, not validating that the TLS certificate is in fact the one known to be deployed on the servers.

Recommendation

Mobile applications should use certificate pinning to verify the identity of the remote host communicating with the application. Certificate pinning verifies that the client application is connecting to the designated server and not an intermediary attacker.

For more information about certificate pinning and how to implement it in the respective operating systems, see
https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning.

LOW

GLOBAL-05| Source Map Files Not Removed

Issue	Severity	Location	Status
Security Misconfiguration	Low		Resolved

Description

The extension includes JavaScript Source Map files in its distribution package. The source map files are used for debugging and development purposes. However, including these files in the extension's distribution could pose security and privacy concerns.

Recommendation

It is recommended to remove the source map files in the production version of the extension.

LOW

GLOBAL-10| Public Swagger documentation

Issue	Severity	Location	Status
Information Disclosure	Low	https://dev-api.atlaswallet.com/docs/	Resolved

Description

The swagger documentation is publicly accessible.

Recommendation

It is recommended to remove the page if it does not fulfill any business/development requirements. Otherwise, it is recommended to restrict the access to the Swagger docs.

LOW

GLOBAL-11| Using components with known vulnerabilities

Issue	Severity	Location	Status
Security Misconfiguration	Low		Resolved

Description

The result of the npm audit command shows the application is using outdated versions of npm packages.

- Atlas_MobileApp-security: 70 vulnerabilities (1 low, 39 moderate, 26 high, 4 critical
- atlas_backend-master: 27 vulnerabilities (19 moderate, 7 high, 1 critical)

Recommendation

There should be a patch management process in place to:

- Remove unused dependencies, unnecessary features, components, files, and documentation.
- Continuously inventory the versions of both client-side and server-side components (e.g. frameworks, libraries) and their dependencies using tools like versions, DependencyCheck, retire.js, etc.
- Continuously monitor sources like CVE and NVD for vulnerabilities in the components. Use software composition analysis tools to automate the process. Subscribe to email alerts for security vulnerabilities related to components you use.
- Only obtain components from official sources over secure links. Prefer signed packages to reduce the chance of including a modified, malicious component.
- Monitor for libraries and components that are unmaintained or do not create security patches for older versions. If patching is not possible, consider deploying a virtual patch to monitor, detect, or protect against the discovered issue.

Organizations should ensure that there is an ongoing plan for monitoring, triaging, and applying updates or configuration changes for the lifetime of the application or portfolio.

LOW

GLOBAL-12| Insecure Android Configuration

Issue	Severity	Location	Status
Security Misconfiguration	Low	Android application	Resolved

Description

When analyzing the AndroidManifest.xml file, it was found that the following configuration are consider insecure

- android:usesCleartextTraffic="true"

When the attribute is set to "true", platform components (for example, HTTP and FTP stacks, DownloadManager, and MediaPlayer) will accept the app's requests to use cleartext traffic.

Recommendation

It is recommended to set the "usesCleartextTraffic" to false in the AndroidManifest.xml file.

Disclaimer

Appendix

Finding Categories

Centralization / Privilege
Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Coding Style
Coding Style findings usually do not affect the generated bytecode but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Volatile Code
Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Logical Issue
Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block. timestamp works.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux “sha256sum" command against the target file.

About

DeHacker is a team of auditors and white hat hackers who perform security audits and assessments. With decades of experience in security and distributed systems, our experts focus on the ins and outs of system security. Our services follow clear and prudent industry standards. Whether it's reviewing the smallest modifications or a new platform, we'll provide an in-depth security survey at every stage of your company's project. We provide comprehensive vulnerability reports and identify structural inefficiencies in smart contract code, combining high-end security research with a real-world attacker mindset to reduce risk and harden code.

BLOCKCHAIINS

- Ethereum
- Cosmos
- Eos
- Substrate

TECH STACK

- Python
- Solidity
- Rust
- c++

CONTACTS

- https://dehacker.io
- https://twitter.com/dehackerio
- https://github.com/dehacker/audits_public
- https://t.me/dehackerio
- https://blog.dehacker.io/

DeHacker

October 5st 2024