Cyberscope

Audit Report

LETSTOP

October 2024

Commit 8c9a1c91d07868abf36937fb288e43df7127ae96

Audited by © cyberscope

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

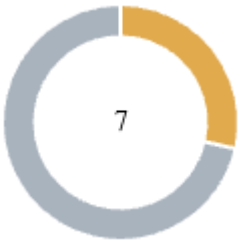
Audit Updates

Initial Audit	24 Oct 2024

Source Files

Filename	SHA256
auditPartsWallet.ts	4bee6d2d599f2078e8d168b28f4d8f1aa4cd24b31a74c1c64e7985f05568998f

Findings Breakdown



Critical	0
Medium	2
Minor / Informative	5

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	2	0	0	0
Minor / Informative	5	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Medium	FDS	Function Declaration Syntax	Unresolved
Medium	MI	Missing Imports	Unresolved
Minor / Informative	MMN	Misleading Method Naming	Unresolved
Minor / Informative	MDCF	Missing Dependency Configuration Files	Unresolved
Minor / Informative	MEC	Missing ESLint Configuration	Unresolved
Minor / Informative	MORH	Missing Operation Result Handling	Unresolved
Minor / Informative	RWGL	Redundant Wallet Generation Logic	Unresolved

FDS - Function Declaration Syntax

Criticality	Medium
Location	auditPartsWallet.ts#L15,24,32,46
Status	Unresolved

Description

Several functions in the repository that handle cryptographic operations are declared without the function keyword, which is required for function declarations in TypeScript and JavaScript. This omission causes runtime errors when these functions are invoked, interrupting execution and potentially compromising the application's cryptographic processes.

```
generateMnemonics() { /* ... */ }
getSeedFromMnemonics(mnemonics: string, password = '') { /* ... */ }
createWallet(seed: Buffer) { /* ... */ }
createWallet(seed: string) { /* ... */ }
```

Recommendation

To mitigate this issue, the team is advised to add the `function` keyword to each function declaration to prevent runtime errors and ensure consistent syntax across the codebase. Additionally, the team could implement runtime testing for each cryptographic function to verify they execute as expected without syntax or type errors.

MI - Missing Imports

Criticality	Medium
Location	auditPartsWallet.ts#L18,19,25,26,35,36,47
Status	Unresolved

Description

The project references several functions and classes which are presumably part of external libraries used for cryptographic operations. However, these dependencies have not been imported in the module. Consequently, attempts to invoke these functions will lead to runtime exceptions, as they are undefined in the current scope.

```
LangEn
Mnemonic
toUtf8Bytes
Keypair
PublicKey
HDNodeWallet
```

Recommendation

The team is advised to import the required functions and classes at the beginning of the file to ensure they are accessible at runtime. The team should ensure all cryptographic functions and dependencies are imported correctly to prevent runtime issues and maintain functionality.

MMN - Misleading Method Naming

Criticality	Minor / Informative
Location	auditPartsWallet.ts#L25
Status	Unresolved

Description

Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The project uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand.

```
getSeedFromMnemonics(mnemonics: string, password = '') {
  const normalizedMnemonic = toUtf8Bytes(mnemonics, 'NFKD');
  const salt = toUtf8Bytes('mnemonic' + password, 'NFKD');
  const seed = Crypto.pbkdf2Sync(normalizedMnemonic, salt, 2048, 64,
'sha512');
  return `0x${Buffer.from(seed).toString('hex')}`;
}
```

Recommendation

It's always a good practice for the project to contain method names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

MDCF - Missing Dependency Configuration Files

Criticality	Minor / Informative
Status	Unresolved

Description

The repository does not contain essential dependency configuration files, specifically `package.json` and `package-lock.json`. This absence requires team members and contributors to manually generate a package.json file each time the repository is cloned. This practice can lead to inconsistencies in the dependencies used across different environments, as there are no fixed versions or specific packages defined. Such inconsistencies may introduce unexpected behavior and potential vulnerabilities due to mismatched or outdated package versions.

Recommendation

The team is strongly recommended to add `package.json` and `package-lock.json` files to the repository to standardize the dependency configuration. These files should list all project dependencies along with their specific versions, ensuring consistency across all environments and preventing potential security risks associated with version discrepancies.

MEC - Missing ESLint Configuration

Criticality	Minor / Informative
Status	Unresolved

Description

The codebase lacks an ESLint configuration, which is a valuable tool for identifying and fixing issues in JavaScript code. ESLint not only helps catch errors but also enforces a consistent code style and promotes best practices. It can significantly enhance code quality and maintainability by providing a standardized approach to coding conventions and identifying potential problems.

Recommendation

The team is strongly advised to integrate ESLint into the project by creating an ESLint configuration file (e.g., `.eslintrc.js` or `.eslintrc.json`) and defining rules that align with the team's coding standards. Consider using popular ESLint configurations, such as Airbnb, Standard, or your own customized set of rules. By incorporating ESLint into the project, the team ensures consistent code quality, catches potential problems early in the development process, and establishes a foundation for collaborative and maintainable code.

MORH - Missing Operation Result Handling

Criticality	Minor / Informative
Location	auditPartsWallet.ts#L59,68
Status	Unresolved

Description

The `setSecret` and `deleteSecret` methods in the SecretsManager class invoke `Keychain.setGenericPassword` and `Keychain.resetGenericPassword`, respectively, to manage credentials. However, neither of these methods checks the success of these operations. If Keychain.setGenericPassword or Keychain.resetGenericPassword fails, they return false, indicating an unsuccessful operation. Since this status is not captured or returned by setSecret and deleteSecret, any errors or failures will go unhandled, potentially resulting in incorrect assumptions about the success of credential operations.

```
static async setSecret(key: string, secret: string): Promise<void> {
    await Keychain.setGenericPassword(key, secret, {service: key});
}

static async deleteSecret(key: string): Promise<void> {
    await Keychain.resetGenericPassword({service: key});
}
```

Recommendation

To mitigate this issue, the team could modify `setSecret` and `deleteSecret` to handle the return status of Keychain.setGenericPassword and Keychain.resetGenericPassword. This could involve logging an error or throwing an exception if the operation is unsuccessful. Implementing error handling ensures that failures are detected and can be addressed promptly, enhancing reliability and robustness in managing sensitive credentials.

RWGL - Redundant Wallet Generation Logic

Criticality	Minor / Informative
Location	auditPartsWallet.ts#L15,24,46
Status	Unresolved

Description

The functions `generateMnemonics` , `getSeedFromMnemonics` , and `createWallet` implement custom logic to generate mnemonics, derive seeds, and create wallets. However, this functionality is already available in the ethers.js library through `Wallet.createRandom()` , which provides a tested, well-maintained, and community-accepted approach to wallet generation. Reimplementing this functionality increases the likelihood of inconsistencies, bugs, and deviations from best practices, especially as libraries like ethers.js undergo updates and improvements.

```
generateMnemonics() { /* ... */ }
getSeedFromMnemonics(mnemonics: string, password = '') { /* ... */ }
createWallet(seed: string) { /* ... */ }
```

Recommendation

The team is advised to take these segments into consideration and refactor them to leverage the existing `Wallet.createRandom()` method in ethers.js for wallet creation. By utilizing ethers.js's built-in functionality, the project benefits from an established implementation, reducing maintenance overhead and minimizing the risk of security vulnerabilities or inconsistencies with standards. This approach simplifies the codebase, aligns with best practices, and ensures compatibility with future updates in ethers.js.

Summary

LETSTOP implements a utility mechanism. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

cyberscope.io