Quantstamp

Alchemy Light Account

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	ERC-4337 Account
Timeline	2023-09-11 through 2023-09-18
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	ERC-4337 README.md ERC-1271
Source Code	• https://github.com/alchemyplatform/light-account #aa8196b
Auditors	• Nikita Belenkov Auditing Engineer • Shih-Hung Wang Auditing Engineer • Martinet Lee Senior Research Engineer • Ruben Koch Auditing Engineer

Documentation quality	High
Test quality	Medium
Total Findings	7 **Fixed: 2** **Acknowledged: 3** **Mitigated: 2**
High severity findings	0
Medium severity findings	0
Low severity findings	4 **Acknowledged: 2** **Mitigated: 2**
Undetermined severity findings	0
Informational findings	3 **Fixed: 2** **Acknowledged: 1**

Summary of Findings

Alchemy developed a `LightAccount` contract that is based on `eth-infinitism` 's implementation of `SimpleAccount` . The main modifications include adopting the unstructured storage pattern, the addition of `isValidSignature()` , and the modification of `_validateSignature()` to support `ERC-1271` .

Generally, the code is well written, most of the logic is similar to the original implementation, and the changes were well documented. However, we have found the support of `ERC-1271` may clash with the intention of `ERC-4337` itself in some cases. This clash leads to the incompatibility of certain use cases of `ERC-1271` with the core design of `ERC-4337`, particularly in the validation stage of the bundler. These clashes can be particularly seen in ALC-1 and ALC-2.

The test suite consists of 36 tests, all of which successfully pass and achieve a branch coverage of 100%. A mutation testing tool called SuMo has also been run on the code base. Mutation testing allows us to evaluate the coverage of the test suite, beyond branch coverage. SuMo has highlighted a mutation score of 78.21%, which is good, but testing around enforcing binary comparison conditions and require statements could be improved.

Fix Review

All issues have been either fixed, mitigated, or acknowledged by the Alchemy team in the commit `912340322f7855cbc1d333ddaac2d39c74b4dcc6`.

Disclaimer

It is important to note that this audit was limited to the `LightAccount` itself and the integration with `ERC-4337`. This audit did not cover the `ERC-4337` implementation itself.

The `LightAccount` also featured a basic version of `ERC-1271` signature verification. While the contracts in scope do not include any potential verifying smart contract, we do want to point out that an uncareful implementation of a verifying smart contract could be vulnerable to signature malleability.

ID	DESCRIPTION	SEVERITY	STATUS
ALC-1	**Multiple on-Chain Validations Can Fail Due to Self-Destructed External Contract**	Low	Acknowledged
ALC-2	**Bundler May Drop UserOps if the `owner` of LightAccount Violates ERC-4337's Validation Requirements in `isValidSignature()`**	Low	Mitigated

ID	DESCRIPTION	SEVERITY	STATUS
ALC-3	Potential Security Implication of Indefinite Validity for EOA Owner-Signed User Operations	Low	Acknowledged
ALC-4	Critical Role Transfer Not Following Two-Step Pattern	Low	Mitigated
ALC-5	Improvements for the `CustomSlotInitializable` Contract	Informational	Fixed
ALC-6	Missing Input Validation	Informational	Fixed
ALC-7	Unlocked Pragma	Informational	Acknowledged

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

> **Disclaimer**
>
> Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

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

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

This audit was focused on the implementation of a light account that acts as a wallet/account in the ERC-4337 system. The actual bundler, entry point, or any other aspects of ERC-4337 were out of scope for this audit and are assumed to be working as expected.

Files Included

Repo: https://github.com/alchemyplatform/light-account(aa8196bf925c32aa3fe48b8f582ce399d0ffd73a) Files: `src/CustomSlotInitializable.sol` , `src/LightAccount.sol` , and `src/LightAccountFactory.sol`

Files Excluded

Repo: https://github.com/alchemyplatform/light-account(aa8196bf925c32aa3fe48b8f582ce399d0ffd73a) Files: N/A

Findings

ALC-1

Multiple on-Chain Validations Can Fail Due to Self-Destructed External Contract

• Low Acknowledged

> **Update**
> The client has acknowledged the issue with the following comment: "As stated, this is a risk on the side of the bundlers, and is inherent when dealing with any account that interacts with external contracts during validation, which could become quite common with the introduction of modular accounts and plugins. While a valid concern, it feels more apt to outline these risks elsewhere and not within LightAccount documentation."

File(s) affected: `src/LightAccount.sol`

Description: Before a bundler submits the set of bundled user operations as a transaction to the entry point, they (again) simulate the validation to reduce the likelihood of performing on-chain, failing operation validations.

The integration with `ERC-1271` signature-validating smart contracts creates an (optional) dependency on an external smart contract that can theoretically be shared between many accounts. During the simulation, i.e. during the `validateOps()` function calls, certain opcodes are banned to make validation almost stateless, with the only exception being storage that is directly linked to the validated account's address.

However, during experiments with a local setup including ethinfinitism's bundler implementation, we found that the validation would not revert if these verifying smart contracts make use of the `SELFDESTRUCT` opcode outside of the `validateOp()` function flow. This is important as a self-destruct on such a shared verifying `ERC-1271` smart contract would invalidate the validations of potentially many accounts at once, without any form of on-chain storage manipulation directly related to the addresses of these accounts. So if an attacker would front-run the block inclusion of a bundle with a self destruct of a contract that is used in many of the bundled user operations as the verifying contract, the bundler could have many failing on-chain validations, causing a financial loss to the bundler.

Following the possibility of the `SELFDESTRUCT` opcode, the issues could be made worse in the following scenario. If the contract registered as the `owner` in many accounts was deployed via `CREATE` by an intermediary contract that was instantiated via `CREATE2`, a metamorphic contract attack would be possible, similar to the recent Tornado Cash exploit, where the verifying smart contract is replaced with a contract with arbitrary code. A possible attack vector here would be to replace the `validateSignature()` function with a gas-consuming function that performs a series of costly memory operations, causing all functions to consume the maximum designated `verificationGasLimit` without enabling validation to pass, causing a financial loss in un-reimbursed gas costs to the bundler.

Furthermore, such a metamorphic verifier could also call/re-enter all `onlyOwner()` protected functions of the contract. If `_authorizeUpgrade()` would ever be overwritten to enable upgrades, this would become a concern.

Note that this is a general issue that can occur when interacting with metamorphic contracts, and the associated risk is not limited to the use of `ERC-4337`.

This attack also requires the bundler to include a user operation in a bundle despite the presence of the `SELFDESTRUCT` opcode outside of the validation trace in the simulation.

This is a point that was also raised in a blog post by Vitalik Buterin and is more of an issue in the bundler simulation than in these contracts. We, however, still want to raise awareness that the set of contracts enables a possible function flow that can cause the bundler some losses.

Recommendation: We mainly want to raise awareness of this possibility. Consider adding documentation outlining the potential risk for bundlers.

This risk can also be mitigated by how the ERC-4337 specification outlines the expected behavior for bundlers when running or executing bundles in a transaction. As per the specification, bundlers can act as block builders themselves. Otherwise, they must collaborate with the block-building infrastructure or utilize a PBS (Proposer-Builder Separation) approach.

ALC-2
Bundler May Drop UserOps if the `owner` of LightAccount Violates ERC-4337's Validation Requirements in `isValidSignature()`

Low Mitigated

> **Update**
> The client added a comment to the Light Account documentation explaining the limitations of owners that are contracts in the context of user operation validation sent through a bundler. Mitigated in commit `4a6de2ec330f1b39e7aa5a832619c085ce36865c`.

File(s) affected: `src/LightAccount.sol`

Description: The `LightAccount` wallet supports user operations validation by either EOA or `ERC-1271` signature validation method. It is important to note that the ERC-1271 signature validation only works when the `owner` of the wallet supports `ERC-1271`.

According to the `ERC-1271` standard:

> `isValidSignature` can call arbitrary methods to validate a given signature, which could be context dependent (e.g. time based or state based), EOA dependent (e.g. signers authorization level within smart wallet), signature scheme Dependent (e.g. ECDSA, multisig, BLS), etc.

Therefore, a contract that supports `ERC-1271` may read storage or the current blockchain state to determine the validity of a signature.

However, the usage of `ERC-1271` for the validation of user operations may have specific limitations due to the restrictions of opcode banning and storage access outlined in the `ERC-4337` standard. Specifically, during the validation phase, wallets are not allowed to use any forbidden opcodes, including `TIMESTAMP` and `NUMBER`. Additionally, storage access is restricted to self-storage or storage associated with the wallet's address.

This restriction prevents `ERC-1967` proxies from being an owner of a `LightAccount`, as they access a constant implementation slot not associated with the wallet. These restrictions also mean that violating the storage access rule on the `owner` would lead to bundlers dropping the UserOps that are sent to the `owner`'s light account. While typically, this is recoverable by directly operating on the `owner` smart contract itself, it may not be the expected behavior. There is also the possibility of not being able to recover if the `owner` smart contract is not well designed.

Furthermore, the `LightAccount.isValidSignature()` function cannot be used for user operation validation since `LightAccount` is deployed as an `ERC-1967` proxy and implements a customized storage slot pattern that triggers a storage access violation when the wallet attempts to read its `owner`.

Recommendation: Consider documenting the limitations of validating user operations through contracts that support `ERC-1271` in both user and technical documentation. This documentation should be available for third-party integrations with the `LightAccount` wallet to be aware of such an issue.

ALC-3
Potential Security Implication of Indefinite Validity for EOA Owner-Signed User Operations

• Low ⓘ Acknowledged

Update

The client acknowledged the issue with the following comment: "A similar (but perhaps less severe as there are less intermediaries) issue exists with regular signed transactions, and as noted in the issue, a signed user operation can be invalidated by signing and executing another one with the same nonce. LightAccount is meant to be a minimal implementation of a ERC-4337-compatible smart contract account, so while time-bound signatures (as recommended) make sense, we'll leave it out of this implementation."

File(s) affected: `src/LightAccount.sol`

Description: During the validation process of a user operation, the `LightAccount` wallet initially verifies whether the wallet's `owner` has signed the provided signature. If this verification succeeds, the `_validateSignature()` function returns a result of 0, indicating the signature is correct and valid, without imposing any time-based restrictions.

However, this approach results in the user operation's signature remaining valid indefinitely once the EOA owner has signed it. This indefinite validity could expose users to specific attack vectors. For example, if a bundler behaves maliciously, they could withhold a user operation after receiving it from the user and submit the bundling transaction when there is a favorable profit opportunity. Delays in the bundling transaction, possibly caused by congestion in the bundler's alternate mempool, could create similar issues for users.

It's worth noting that there exists a method for the EOA owner to manually invalidate a signature - by signing another user operation with the same nonce and executing it before the previous one.

Recommendation: Consider evaluating whether the design choice of allowing indefinitely valid EOA signatures might expose users to any potential risks in the intended use cases for `LightAccounts` wallets. If such risks are identified, consider documenting them clearly in publicly accessible documentation.

A possible way to mitigate these risks is using the `validUntil` and `validAfter` fields in the return data to allow `EntryPoint` to enforce time-based restrictions on the validity of signatures.

ALC-4
Critical Role Transfer Not Following Two-Step Pattern

Low Mitigated

Update

Comments to state that ownership transfers are single-step have been added. The client has also provided the following comment: "For users that want this guarantee, they have the option of adding it via the 1271 path provided by isValidSignature and _validateSignature." Mitigated in commit `2eacfa37a317437f2992f049318b2c6198dbdf26`.

File(s) affected: `src/LightAccount.sol`

Description: In the `LightAccount` wallet, the `owner` can transfer their ownership to another address through the `transferOwnership()` function. This transfer of ownership can occur either directly or by `ERC-4337` user operations.

However, the wallet can lose ownership if the `newOwner` address is accidentally provided as an uncontrollable address or a smart contract that does not support `ERC-1271`. As a result, all user operations to the wallet will fail during the validation phase, effectively making the wallet unusable.

Recommendation: Consider clarifying such a risk in user documentation. To mitigate this risk, consider implementing an `acceptOwnership()` function, similar to the approach in OpenZeppelin's `Ownable2Step` contract. This additional step allows the new owner to explicitly accept ownership, reducing the likelihood of accidental ownership loss.

Unlike OpenZeppelin's `Ownable2Step` contract, this implementation should invoke the `isValidSignature()` method at the transfer of ownership, so that if the contract does not support `ERC-1271`, the ownership would not be passed.

ALC-5
Improvements for the `CustomSlotInitializable` Contract

Informational | Fixed

Update

`CustomSlotInitializable` contract and tests have been updated to use uint64 instead of uint8. Fixed in commit `bf99b74d0602de83b4ea737c1dc21a4000d1aed2`.

File(s) affected: `src/CustomSlotInitializable.sol`

Description: A recent version of OpenZeppelin's `Initializable` contract introduced an update to the definition of the `InitializableStorage` structure. The `_initialized` field was modified to be of type `uint64`. As a result, contracts inheriting from `Initializable` can now be initialized or re-initialized up to a maximum of `2 ** 64` times.

However, the `CustomSlotInitializable` contract currently defines the `_initialized` field as type `uint8`, resulting in a discrepancy with OpenZeppelin's contract implementation.

Recommendation: Consider aligning the `CustomSlotInitializableStorage` structure and the `_disableInitializers()` function with the behavior introduced by OpenZeppelin's updated code. Specifically, updating the `_initialized` field to match the `uint64` type used in OpenZeppelin's contracts would ensure consistency and compatibility with the latest developments in the ecosystem.

ALC-6 Missing Input Validation

Informational Fixed

Update

The suggested input validation has been added. Fixed in commit `48e1725d1b0dcbf858df5064d8830d7e71003975`.

File(s) affected: `src/LightAccount.sol`

Related Issue(s): SWC-123

Description: It is important to validate inputs, even if they only come from trusted addresses, to avoid human error. Here is a non-exhaustive list of input validation checks that should be added:

1. `LightAccount._transferOwnership()` should check that `newOwner != oldOwner`.
2. `LightAccount._initialize()` should check that `anOwner` is not the zero address.

Recommendation: We recommend adding the relevant checks.

ALC-7 Unlocked Pragma

Informational | Acknowledged

> **Update**
> The client has acknowledged the issue with the following comment: "In order to allow for the possibility for these contracts to be inherited in other projects which may use future Solidity versions for compilation, we'll keep this as is."

File(s) affected: `src/*`

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.8.*`. The caret (`^`) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, we recommend removing the caret to lock the file onto a specific Solidity version.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.

- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Adherence to Best Practices

1. Fixed Use custom errors in place of require statements for gas savings.
2. Fixed `Address.isContract()` will be removed in the OZ-contracts 5.0 release. Consider replacing it with `(address(this).code.length > 0`
3. Fixed Typo "_getInitialiazableStorage()" ⇒ "_getInitializableStorage()".
4. Acknowledged Global state variables, such as `CustomSlotInitializable._storagePosition` , should not have a leading underscore.
5. Fixed For loops can be gas-optimized by cashing the `array.length` in a `memory` variable and incrementing the iterator via `unchecked {++i;}` .
6. Fixed `LightAccount.transferOwnership()` can be restricted to `external` visibility.

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

- `317...736 ./src/LightAccountFactory.sol`
- `dee...032 ./src/LightAccount.sol`
- `63d...415 ./src/CustomSlotInitializable.sol`

Tests

- `b93...d65 ./test/LightAccount.t.sol`
- `8ec...1ea ./test/LightAccountFactory.t.sol`
- `81c...956 ./test/CustomSlotInitializable.t.sol`

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.8.3
- SuMo 4.3.0

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

SuMo

1. Install SuMo: `npm install @morenabarboni/sumo`
2. Configure SuMo setup sumo-config.js:

```
module.exports = {
    buildDir: 'build',
    contractsDir: 'src',
```

```
    testDir: 'test',
    skipContracts: [],
    skipTests: [],
    testingTimeOutInSec: 300,
    network: "none",
    testingFramework: "forge",
    minimal: false,
    tce: false,
    historyActive: true
  }
```

3. `npx sumo preflight`
4. `npx sumo mutate`
5. `npx sumo test`

Automated Analysis

Slither

Slither found 147 potential issues. Most of them were found in the test files and marked as false positives. All of them were discussed in this report or classified as false positives.

SuMo

Mutation Testing was completed in 87.04 minutes

SuMo generated 202 mutants:

- 39 live;
- 140 killed;
- 23 stillborn;
- 0 equivalent;
- 0 redundant;

- 0 timed-out.

Mutation Score: 78.21 %

Test Suite Results

The test suite consists of 36 tests, of which all pass. SuMo has highlighted a mutation score of 78.21 %, which is good, but testing around enforcing binary comparison conditions and require statements could be improved.

Fix Review

No changes have been made to the test suite.

```
[⠒] Compiling...
[⠒] Compiling 1 files with 0.8.21
[⠒] Solc 0.8.21 finished in 36.70ms
Compiler run successful!

Running 2 tests for test/LightAccountFactory.t.sol:LightAccountTest
[PASS] testGetAddress() (gas: 174448)
[PASS] testReturnsAddressWhenAccountAlreadyExists() (gas: 174330)
Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 8.67ms

Running 27 tests for test/LightAccount.t.sol:LightAccountTest
[PASS] testAddDeposit() (gas: 61762)
[PASS] testCannotTransferOwnershipToLightContractItself() (gas: 19492)
[PASS] testCannotTransferOwnershipToZero() (gas: 19332)
[PASS] testEntryPointCanTransferOwnership() (gas: 153978)
[PASS] testEntryPointGetter() (gas: 12912)
[PASS] testExecuteBatchCalledByOwner() (gas: 48367)
[PASS] testExecuteBatchFailsForUnevenInputArrays() (gas: 22516)
[PASS] testExecuteBatchWithValueCalledByOwner() (gas: 56039)
```

```
[PASS] testExecuteBatchWithValueFailsForUnevenInputArrays() (gas: 23209)
[PASS] testExecuteCanBeCalledByEntryPointWithExternalOwner() (gas: 170286)
[PASS] testExecuteCanBeCalledByOwner() (gas: 47301)
[PASS] testExecuteCannotBeCalledByRandos() (gas: 18882)
[PASS] testExecuteRevertingCallShouldRevertWithSameData() (gas: 98865)
[PASS] testExecuteWithValueCanBeCalledByOwner() (gas: 53698)
[PASS] testExecutedCanBeCalledByEntryPointWithContractOwner() (gas: 182210)
[PASS] testInitialize() (gas: 2413695)
[PASS] testIsValidSignatureForContractOwner() (gas: 33607)
[PASS] testIsValidSignatureForEoaOwner() (gas: 20666)
[PASS] testIsValidSignatureRejectsInvalid() (gas: 24169)
[PASS] testNonOwnerCannotUpgrade() (gas: 1531112)
[PASS] testOwnerCanTransferOwnership() (gas: 26253)
[PASS] testOwnerCanUpgrade() (gas: 1570202)
[PASS] testRandosCannotTransferOwnership() (gas: 16986)
[PASS] testRejectsUserOpsWithInvalidSignature() (gas: 86985)
[PASS] testStorageSlots() (gas: 9495)
[PASS] testWithdrawDepositToCalledByOwner() (gas: 100795)
[PASS] testWithdrawDepositToCannotBeCalledByRandos() (gas: 58610)
Test result: ok. 27 passed; 0 failed; 0 skipped; finished in 8.89ms

Running 7 tests for test/CustomSlotInitializable.t.sol:CustomSlotInitializableTest
[PASS] testCannotCallDisableInitializersInInitializer() (gas: 214849)
[PASS] testCannotReinitialize() (gas: 24416)
[PASS] testCannotUpgradeBackwards() (gas: 41167)
[PASS] testDisableInitializers() (gas: 34643)
[PASS] testIsInitializing() (gas: 235207)
[PASS] testSimpleInitialization() (gas: 824863)
[PASS] testUpgrade() (gas: 35264)
Test result: ok. 7 passed; 0 failed; 0 skipped; finished in 8.92ms

Ran 3 test suites: 36 tests passed, 0 failed, 0 skipped (36 total tests)
```

Code Coverage

The branch coverage is 100%, which is on par with the recommended 90%+. Whilst contract level tests have good coverage, it is unclear on the level of the integration tests, as they are very important for correct functioning with the ERC–4337 .

Fix Review

No changes have been made to the test suite.

File	% Lines	% Statements	% Branches	% Funcs
src/CustomSlotInitializable.sol	100.00% (**9/**9)	100.00% (**10/**10)	100.00% (**4/**4)	100.00% (**4/**4)
src/LightAccount.sol	100.00% (**44/**44)	100.00% (**54/**54)	100.00% (**18/**18)	100.00% (**19/**19)
src/LightAccountFactory.sol	100.00% (**6/**6)	100.00% (**9/**9)	100.00% (**2/**2)	100.00% (**2/**2)
test/CustomSlotInitializable.t.sol	100.00% (**6/**6)	100.00% (**9/**9)	100.00% (**0/**0)	100.00% (**10/**10)
test/LightAccount.t.sol	83.33% (**5/**6)	90.00% (**9/**10)	50.00% (**1/**2)	100.00% (**4/**4)
Total	98.59% (**70/**71)	98.91% (**91/**92)	96.15% (**25/**26)	100.00% (**39/**39)

Changelog

- 2023-09-18 - Initial report
- 2023-09-27 - Final report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy,

timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication or other making available of the report to you by Quantstamp.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites&aspo; owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that your access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. The

report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

Quantstamp

Alchemy Light Account