Quantstamp

Alchemy - Multisig Plugin

Executive Summary

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

Type	ERC-6900 Plugin
Timeline	2024-03-18 through 2024-03-21
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Multisig Plugin V1 Eng Specs
Source Code	• alchemyplatform/multisig-plugin #70089a2
Auditors	• Ruben Koch Auditing Engineer • Shih-Hung Wang Auditing Engineer • Nikita Belenkov Auditing Engineer • Gereon Mendler Auditing Engineer

Documentation quality	High
Test quality	High
Total Findings	9 Fixed: 6 Acknowledged: 3
High severity findings	0
Medium severity findings	0
Low severity findings	5 Fixed: 5
Undetermined severity findings	1 Acknowledged: 1
Informational findings	3 Fixed: 1 Acknowledged: 2

Summary of Findings

This audit has covered an owner plugin for an ERC-6900 Modular Account, that allows multiple owners to execute a `UserOp` only if a certain number of owners have signed this specific `UserOp` off-chain. The plugin implements k out of n signature threshold, which requires at least k valid owner signatures from a pool of n owners. This plugin is an owner plugin that would protect certain key modular account functionality and session key functionality.

Overall the code is well written and follows very good software development practices.

Throughout the audit, some issues have been identified, ranging from Low to Informational in severity. The key issues revolve around compatibility with ERC-4337, such as ALC-MP-1 and ALC-MP-8, and compatibility with ERC-6900, such as issue ALC-MP-2. It is recommended that all issues be addressed.

The project has a good test suite of 40 tests and a branch coverage of 90%. The test suite consists of unit tests and fuzz tests. We recommend adding integration testing with a bundler, as such testing would help to detect issues like ALC-MP-1.

The audit report also includes an issue reported by the Alchemy team during the audit (ALC-MP-9).

Update Fix-Review

All issues have either been fixed or acknowledged. The test suite has been adequately updated to accommodate the changes.

ID	DESCRIPTION	SEVERITY	STATUS
ALC-MP-1	**Plugin Is Incompatible with the Latest `UserOp` Validation Rules**	● Low	Fixed
ALC-MP-2	**No Reverting Runtime Validation Added for `UserOp` Function Selectors**	● Low	Fixed
ALC-MP-3	**Unsafe Type Casting**	● Low	Fixed

ID	DESCRIPTION	SEVERITY	STATUS
ALC-MP-4	`SIG_VALIDATION_FAILED` Returned in Incorrect Case	● Low	Fixed
ALC-MP-5	The Signature Has No Expiry Time	● Informational	Acknowledged
ALC-MP-6	Contract-Owners Can Reuse Signatures From Other Contract-Owners of Same `UserOp`	● Informational	Acknowledged
ALC-MP-7	Accounts Can Be Created with More than `_MAX_OWNERS_ON_CREATION` Owners	● Informational	Fixed
ALC-MP-8	Potential Size Limitation on `UserOp.signature`	● Undetermined	Acknowledged
ALC-MP-9	Lack Of Constraints On Signature Enable Gas Griefing Attack Vector	● Low	Fixed

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

The following directory was in scope for the `multisig-plugin` repo: `src/*.sol`.

Files Included

The following files were in scope:

1. `src/IMultisigPlugin.sol`
2. `src/MultisigModularAccountFactory.sol`
3. `src/MultisigPlugin.sol`

Files Excluded

Everything outside the included scope.

Findings

ALC-MP-1
Plugin Is Incompatible with the Latest `UserOp` Validation Rules

● Low ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `eb5fa9d4e36d7e04fd89dea97c741c670a1c47aa`. The client provided the following explanation:

> Implemented recommended fix.

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

Description: When validating a `UserOp`, the multisig plugin's `userOpValidationFunction()` function is called with the corresponding parameters. The plugin compares the gas limits specified in the `UserOp` with the upper limits encoded in the signature. If any gas limits differ, it recalculates the `UserOp` hash with the upper limits by calling `ENTRYPOINT.getUserOpHash()`. However, the function call `getUserOpHash()` to `ENTRYPOINT` is not allowed, as specified in ERC-7562 (a draft EIP for ERC-4337 validation rules). According to OP-051 to OP-054, only calls to the `depositTo()` or fallback function are allowed, and any other function calls are forbidden. Our experiments with the Infinitism's bundler showed that the bundler rejects a `UserOp` that calls `ENTRYPOINT.getUserOpHash()` during the validation phase with the following error:

```
Error: processing response error [...]
  reason: 'processing response error',
  code: 'SERVER_ERROR',
  body: '{"jsonrpc":"2.0","id":43,"error":{"message":"illegal call into EntryPoint during validation
getUserOpHash","code":-32502}}',
```

Recommendation: Consider avoiding the `getUserOpHash()` call to `ENTRYPOINT` but calculating the `UserOp` hash directly in the multisig plugin contract. Although the validation rules are subject to change and the banning policies implemented in bundlers may differ (in the case of alt-mempools), we recommend designing the plugin to comply with the ERC-7562 specification to achieve broader compatibility between the plugin and different bundlers.

ALC-MP-2
No Reverting Runtime Validation Added for `UserOp` Function Selectors

● Low ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `85697f423f12e48d73b23de8fc6738f56602e70e`. The client provided the following explanation:

> Added userOp validators pointing to the plugin that always revert as it's unimplemented.

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

Description: According to the plugin manifest, the multisig validation scheme is only expected to be used in the context of a `UserOp`. Therefore, `MultisigPlugin` does not define the runtime validation function for owner-only functions, assuming that any function call in runtime to them will revert due to the lack of validation function.

However, another installed plugin can add a runtime validation function to the owner-only functions, allowing them to be executed in runtime. As a result, one may bypass the multisig verification and execute the owner-only functions as long as the runtime verification allows.

It may be the intended choice if the account owners want to configure two different verification schemes in `UserOp` and runtime contexts, but it may increase the complexity of managing privileged roles.

Recommendation: For simplicity, we recommend disabling runtime validation of owner-only functions if the `MultisigPlugin` is installed. Consider explicitly adding a runtime validation function or `preRuntimeValidationHook` to the owner-only functions and revert any runtime calls.

ALC-MP-3 Unsafe Type Casting

• Low ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `d56ae0478cc7342b4d1532d03233bcc7d86bb34d` and `2585ff2ad6ea22e4f9b25f6fe38429ec1b045770`. The client provided the following explanation:

> Opted to change the arguments to uint128 to get type checking from solidity + viem on the client side.

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

Description: The `updateOwnership()` function allows the account owners to update the multisig threshold, specified as the `newThreshold` parameter. If the parameter's value is zero, it indicates the threshold should not be updated. If not, the parameter will be type-casted from `uint256` to `uint128` and stored in the contract. As a result, the final stored threshold value may become zero if the last 128 bits of a positive `newThreshold` are zero, bypassing the multisig-verification scheme after the change since no signature is required anymore.

Recommendation: Although it is considered more of a user error if `newThreshold` is configured to such large values, we recommend applying safe type-casting to reduce the risks of potential misconfigurations on the user side. A possible solution is to use the `toUint128()` function from OpenZeppelin's `SafeCast` library, which can be applied to every explicit `uint128` type casting in the code.

ALC-MP-4 `SIG_VALIDATION_FAILED` Returned in Incorrect Case

• Low ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `0a7d1193490753a58a967ed43d7d953b5a6d9d55` and `45a0403db1d3c4d75d64a7c4a4b9564094d5da8b`. The client provided the following explanation:

> Implemented recommended fix.

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

Description: The `checkNSignatures()` function returns a signature validation failure if the `digest.tryRecover()` call to verify an EOA's signature fails with an error. As a result, the `RecoverError.InvalidSignatureS` (when the `s` value is in the upper range) will be treated as a signature validation failure. This behavior differs from the Modular Account's `SessionKeyPlugin`, which reverts the transaction if a `RecoverError.InvalidSignatureS` error occurs. See SessionKeyPlugin.sol#L186-L189.

Recommendation: For consistency between the plugins, consider reverting the transaction if the error code returned from `tryRecover()` is not `RecoverError.NoError`.

ALC-MP-5 The Signature Has No Expiry Time

• Informational ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

> Adding expiry timestamps would add overhead and some complexity, we've opted to leave that out for v1.

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

Description: The plugin adds the ability to have multiple owners sign a `UserOp` to be executed. This is done via an ECDSA signature for an EOA or a 1271 signature from a smart contract. The signatures are then assembled together and submitted along with the `UserOp`. In the case of an EOA signature, there currently is no expiry time on the validity of the signature, so if the nonce has not been used yet, this signature is valid for the specific `UserOp` for an indefinite amount of time.

Recommendation: Consider adding an expiry time to the signature. Alternatively, document this aspect to end users.

ALC-MP-6 Contract-Owners Can Reuse Signatures From Other Contract-Owners of Same `UserOp`

• Informational ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

> This design decision is also in Gnosis Safe and should be safe.

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

Description: In the case of a contract owner signature, the `s` value points to the offset of the start of the specific signature of the contract owner. However, unlike the underlying addresses, the offsets are not enforced to be in ascending order. This means that two contract owners can technically share the same signature. It should of course be noted that the underlying contract addresses that those two separate signature validation calls are performed upon still have to differ, but digest and signature can be shared.

Recommendation: Consider enforcing a similar ascending order as in the owner addresses for the offsets of the contract owner signature.

ALC-MP-7 Accounts Can Be Created with More than `_MAX_OWNERS_ON_CREATION` Owners

● Informational ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `207eae4380248b2c570f33b4667119067f7a9762` . The client provided the following explanation:

> Implemented recommended fix.

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

Description: `MultisigModularAccountFactory.getAddress()` reverts in case of `owners.length` exceeds `_MAX_OWNERS_ON_CREATION` . However, `createAccount()` does not perform a similar check, potentially leading to block gas limit-related issues if the number of owners is too high.

Recommendation: Enforce `owners.length` to be smaller than `_MAX_OWNERS_ON_CREATION` in the `createAccount()` function.

ALC-MP-8 Potential Size Limitation on `UserOp.signature`

● Undetermined ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

> Bundlers currently do not have any plans on limiting this, so the upper limit here would be the block gas limit.

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

Description: Currently, all the signatures are encoded into the `userOp.signature` field, including both EOA and 1271 contract-based signatures. We could not find any current limitations on the signature field size at this iteration of bundlers, currently the only limitation is the overall block size limit. It could be possible that bundlers will add a limit on that field, especially given the 1271 contract signatures can be of arbitrary size.

The current ERC-4337 specification states the following:

> When a client receives a `UserOp` , it must first run some basic sanity checks, namely that: The `verificationGasLimit` is sufficiently low (<= `MAX_VERIFICATION_GAS`) ...

The statement is saying there can be a limit on `verificationGasLimit` (to protect bundlers from being DoS'ed). And because the size of `userOp.signature` contributes to the verification gas, we can assume the bundler may drop a `UserOp` with a signature that is too large.

Recommendation: Consider what kind of effects this limitation can have on the system if such a restriction is added in the future.

ALC-MP-9 Lack Of Constraints On Signature Enable Gas Griefing Attack Vector

● Low ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `305f3b3fa1251f97221c30e1d396d96e5cdda87b` and `305f3b3fa1251f97221c30e1d396d96e5cdda87b`.

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

Description: The composite signature doesn't specify a maximum length that valid signatures must have. This opens up a potential vector for griefing by appending bytes to the end of the composite signature. Since the `signature` field in the `UserOp` is inherently malleable, a malicious bundler or middle man could insert additional bytes, maxing out the specified `verificationGasLimit`.

Recommendation: Add additional constraints to the structure of the signature in a way that a bundler can not append additional bytes and can also not insert additional dirty bits into the existing encoding.

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 The `MultisigPlugin.isOwnerOf()` and `ownershipInfoOf()` functions can be marked as `external`.

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

- `975...7fe ./src/MultisigPlugin.sol`
- `672...e01 ./src/IMultisigPlugin.sol`
- `6d5...6eb ./src/MultisigModularAccountFactory.sol`

Tests

- `58b...3c4 ./test/MultisigPlugin.t.sol`
- `77a...1d1 ./test/MultisigMAFactory.t.sol`
- `855...af3 ./test/mocks/MockContractOwner.sol`

Toolset

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

Setup

Tool Setup:

- Slither v0.10.0

Steps taken to run the tools:
1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Automated Analysis

Slither

179 results were found across the codebase. Non-false positive findings have been included in the report.

Test Suite Results

The test suite was obtained through `forge test`.

The test suite has a total of 40 tests of which all pass. The test suite includes fuzz testing and unit testing. It would be a good idea to add integration testing with a bundler.

Update Fix-Review

The test suite has been adequately updated to accommodate the changes.

```
Running 16 tests for test/MultisigMAFactory.t.sol:MultisigModularAccountFactoryTest
[PASS] test_2StepOwnershipTransfer() (gas: 32404)
[PASS] test_addStake() (gas: 80565)
[PASS] test_addressMatch() (gas: 766984)
[PASS] test_badOwnersArray() (gas: 13597)
[PASS] test_badThreshold() (gas: 25120)
[PASS] test_deploy() (gas: 772341)
[PASS] test_deployCollision() (gas: 767598)
[PASS] test_deployWithDuplicateOwners() (gas: 10144)
[PASS] test_deployWithUnsortedOwners() (gas: 10386)
[PASS] test_deployedAccountHasCorrectPlugins() (gas: 763989)
[PASS] test_getAddressWithMaxOwnersAndDeploy() (gas: 3407580)
[PASS] test_getAddressWithTooManyOwners() (gas: 255747)
[PASS] test_getAddressWithUnsortedOwners() (gas: 10258)
[PASS] test_unlockStake() (gas: 87178)
[PASS] test_withdraw() (gas: 80825)
[PASS] test_withdrawStake() (gas: 78280)
Test result: ok. 16 passed; 0 failed; 0 skipped; finished in 672.39ms

Running 24 tests for test/MultisigPlugin.t.sol:MultisigPluginTest
[PASS] testFuzz_isValidSignature_ContractOwner(uint256,bytes32) (runs: 256, μ: 298458, ~: 298458)
[PASS] testFuzz_isValidSignature_EOAOwner(string,bytes32) (runs: 256, μ: 73119, ~: 73052)
[PASS] testFuzz_userOpValidationFunction_ContractOwner(uint256,
(address,uint256,bytes,bytes,uint256,uint256,uint256,uint256,uint256,bytes,bytes)) (runs: 256, μ: 310696,
~: 310680)
[PASS] testFuzz_userOpValidationFunction_EOAOwner(string,
(address,uint256,bytes,bytes,uint256,uint256,uint256,uint256,uint256,bytes,bytes)) (runs: 256, μ: 81681,
~: 81673)
[PASS] testFuzz_userOpValidationFunction_Multisig(uint256,uint256,
(address,uint256,bytes,bytes,uint256,uint256,uint256,uint256,uint256,bytes,bytes)) (runs: 256, μ: 923814,
~: 914366)
[PASS] testFuzz_userOpValidationFunction_Multisig_VariableGas(uint256,uint256,
(address,uint256,bytes,bytes,uint256,uint256,uint256,uint256,uint256,bytes,bytes)) (runs: 256, μ: 929135,
~: 923489)
[PASS] test_eip712Domain() (gas: 10648)
[PASS] test_failUserOpValidationFunction_EOAOwner() (gas: 78616)
[PASS] test_failUserOpValidation_SigLen() (gas: 14205)
[PASS] test_failUserOpValidation_SigOffset() (gas: 305429)
[PASS] test_fuzzFailUserOpValidationFunction_BadGas(string,
(address,uint256,bytes,bytes,uint256,uint256,uint256,uint256,uint256,bytes,bytes)) (runs: 256, μ: 111363,
~: 111099)
[PASS]
test_fuzzFailUserOpValidationFunction_BadGas2((address,uint256,bytes,bytes,uint256,uint256,uint256,uint25
6,uint256,bytes,bytes)) (runs: 256, μ: 156512, ~: 156217)
[PASS] test_multiOwnerPlugin_sentinelIsNotOwner() (gas: 10592)
[PASS] test_onInstall_success() (gas: 94020)
```

```
[PASS] test_onUninstall_success() (gas: 29307)
[PASS] test_pluginInitializeGuards() (gas: 59689)
[PASS] test_pluginManifest() (gas: 34906)
[PASS] test_runtimeValidationFunction_OwnerOrSelf(uint8) (runs: 256, μ: 12086, ~: 12086)
[PASS] test_updateOwners_failExceedThreshold() (gas: 20992)
[PASS] test_updateOwners_failWithDuplicatedAddresses() (gas: 20693)
[PASS] test_updateOwners_failWithEmptyOwners() (gas: 25385)
[PASS] test_updateOwners_failWithNotExist() (gas: 17888)
[PASS] test_updateOwners_failWithZeroAddressOwner() (gas: 15859)
[PASS] test_updateOwners_success() (gas: 84446)
Test result: ok. 24 passed; 0 failed; 0 skipped; finished in 753.39ms

Ran 2 test suites: 40 tests passed, 0 failed, 0 skipped (40 total tests)
```

Code Coverage

Coverage was obtained through `forge coverage --ir-minimum`.

The branch coverage of the system stands at 90%, which means that the test suite is of a sufficient quality. It is always good to attempt to improve the branch coverage to 100%.

Update Fix-Review

With the last commit of the fix-review, branch coverage in the MultisigPlugin has come down from 92.68% to 80.43%, which is still in the acceptable range.

File	% Lines	% Statements	% Branches	% Funcs
src/MultisigModularAccountFactory.sol	88.57% (**31/**35)	91.30% (**42/**46)	80.00% (**16/**20)	85.71% (**6/**7)
src/MultisigPlugin.sol	83.01% (**127/**153)	86.73% (**170/**196)	80.43% (**37/**46)	95.00% (**19/**20)
Total	84.04% (**158/**188)	87.60% (**212/**242)	80.30% (**53/**66)	92.59% (**25/**27)

Changelog

- 2024-03-21 - Initial Report
- 2024-03-29 - 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