Quantstamp Contract Security Certificate

October 15th 2019 — Quantstamp Verified

edgeware-lockdrop

This smart contract audit was prepared by Quantstamp, the protocol for securing smart contracts.

Executive Summary

Type	Smart Contract Audit
Auditors	Ed Zulkoski, Senior Security Engineer Kacper Bąk, Senior Research Engineer Yohei Oka, Forward Deployed Engineer
Timeline	2019-06-18 through 2019-06-19
EVM	Constantinople
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Whitepaper Medium Article on Lockdrop
Source Code	

Repository	Commit
edgeware-lockdrop	6c5692d

Total Issues	0
High Risk Issues	0
Medium Risk Issues	0
Low Risk Issues	0
Informational Risk Issues	0
Undetermined Risk Issues	0

0 issues

Overall Assessment

The contracts are well-written and properly tested. There are no issues to report.

Severity Categories	
High	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low	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.

Goals

- Evaluate `Lockdrop` contracts for any security-related issues.

Changelog

- 2019-06-19 - Initial Report

Quantstamp Audit Breakdown

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

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

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. 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:
   i. 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.
   ii. 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, clarify, 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.

Toolset

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

Setup

Tool Setup:

- Truffle v5.0.22
- Ganache v6.4.4
- Mythril v0.18.9
- MAIAN commit sha: ab387e1
- Securify

Steps taken to run the tools:

1. Installed Truffle: `npm install -g truffle`
2. Installed Ganache: `npm install -g ganache-cli`
3. Installed the Mythril tool from Pypi: `pip3 install mythril`
4. Ran the Mythril tool on each contract: `myth -x path/to/contract`
5. Ran the Securify tool: `java -Xmx6048m -jar securify-0.1.jar -fs contract.sol`
6. Cloned the MAIAN tool: `git clone --depth 1 https://github.com/MAIAN-tool/MAIAN.git maian`
7. Ran the MAIAN tool on each contract: `cd maian/tool/ && python3 maian.py -s path/to/contract contract.sol`

Test Results

Test Suite Results

```
Contract: Lockdrop-1
    ✓ should setup and pull constants (47ms)
    ✓ should lock funds and also be a potential validator (77ms)
    ✓ should unlock the funds after the lock period has ended (148ms)
    ✓ should not allow one to lock before the lock start time (56ms)
    ✓ should not allow one to lock after the lock start time (48ms)
    ✓ should not allow one to lock up any different length than 3,6,12 months
    ✓ should fail to withdraw funds if not enough gas is sent (127ms)
    ✓ should generate the allocation for a substrate genesis spec with THREE_MONTHS term (491ms)
    ✓ should generate the allocation for a substrate genesis spec with SIX_MONTHS term (372ms)
    ✓ should generate the allocation for a substrate genesis spec with TWELVE_MONTHS term (352ms)
    ✓ should aggregate the balances for all non validators and separate for validators (363ms)
    ✓ should turn a lockdrop allocation into the substrate genesis format (359ms)
    ✓ should allow contracts to lock up ETH by signalling (71ms)
    ✓ ensure the contract address matches JS RLP script (39ms)

  Contract: Lockdrop-2
    ✓ should ensure base58 encodings are valid to submit (606ms)
```

Code Coverage

The widely used `solidity-coverage` tool does not yet fully support `Truffle 5` and `solidity ^0.5.0`; as such, coverage results could not be obtained. Manual inspection of the test suites seems to indicate reasonably high coverage.

Automated Analyses

Mythril

Mythril reported three warnings, all of which were classified as false positives. The first indicates that an ether transfer is dependent upon the predictable value `block.timestamp` in the Lock `fallback()` function, however this is intended functionality of the Lock contract. The second warning notes that the Lock `fallback()` function calls an external address which must be trusted, however the external address corresponds to the lock owner, as specified at the time of the Lock's creation. The final warning indicates that the return value of the external `call()` function is not checked, however this check is performed by assembly checks in the following line.

MAIAN

MAIAN has not detected any issues.

Securify

Securify detected several potential issues related to transaction-ordering dependence and locked ether, however manual inspection of the report indicated that these were all false positives.

Adherence to Specification

The code generally meets the requirements of locking ether for various term lengths with the intent of signaling interest in the Edgeware platform. We also confirmed that the `addressFrom()` works as intended based on the semantics defined in the Ethereum yellow paper, as well as through manual testing against various addresses and nonces.

Code Documentation

The code is well-written and properly documented.

Adherence to Best Practices

The code adheres to best practices.

Appendix

File Signatures

The following are the SHA-256 hashes of the audited contracts and/or test files. A smart contract or file with a different SHA-256 hash has been modified, intentionally or otherwise, after the audit. 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 audit.

Contracts

```
7246af5b3f069936754b9e67852d27053def4340c30c7c352dca368457c2db83
./contracts/Migrations.sol

cff41b55fdccd9232e446a345de57e86e31a58b171e02dac91f2b5d0572df7d0
./contracts/Lockdrop.sol
```

Tests

```
fd7cb12fdf1ca02426ee1f14a8d15e2b8e9d6f25572032fe29f7ad4ad1341f05
./test/1-lockdrop.spec.js

8031b911128b24db9a4234cd34e5190f1a00eea97cd45df293c5103c8305c512
./test/2-lockdrop.spec.js

dc979e50b3fb30afe65860b6744c6aa96846860df73b820f60d60f80914eb4d9
./test/3-lockdrop.spec.js
```

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure smart contracts at scale using computer-aided reasoning tools, with a mission to help boost adoption of this exponentially growing technology.

Quantstamp's team boasts decades of combined experience in formal verification, static analysis, and software verification. Collectively, our individuals have over 500 Google scholar citations and numerous published papers. In its mission to proliferate development and adoption of blockchain applications, Quantstamp is also developing a new protocol for smart contract verification to help smart contract developers and projects worldwide to perform cost-effective smart contract security audits.

To date, Quantstamp has helped to secure hundreds of millions of dollars of transaction value in smart contracts and has assisted dozens of blockchain projects globally with its white glove security auditing services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Finally, Quantstamp's dedication to research and development in the form of collaborations with leading academic institutions such as National University of Singapore and MIT (Massachusetts Institute of Technology) reflects Quantstamp's commitment to enable world-class smart contract innovation.