Quantstamp Smart Contract Audit

edge-lockdrop

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

Quantstamp helps to secure blockchain applications such as smart contracts.
We are developing a new protocol for smart contract verification, performing professional audits and consultations, and developing security tools. Quantstamp also has expertise in application security and secure software development.

Executive Summary

Type	Smart Contract Audit
Auditors	Ed Zulkoski, Senior Security Engineer Kacper Bąk, Senior Research Engineer Yohei Oka, Forward Deployed Engineer
Timeline	2019-03-28 through 2019-04-08
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
edge-lockdrop	d07c11a

Total Issues	1 (1 Resolved)
High Risk Issues	0
Medium Risk Issues	1 (1 Resolved)
Low Risk Issues	0
Informational Risk Issues	0
Undetermined Risk Issues	0

1 issue

Overall Assessment

The smart contracts are overall well-written. Although a few issues were detected during the audit, they are easily addressable, and the core functionality of the smart contracts works as expected.

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

Changelog

- 2019-04-01 - Initial Report
- 2019-04-08 - Reviewed report based on commit `51478f1`.

Quantstamp Audit Breakdown

This report focused on evaluating security of the Lockdrop smart contracts, as requested by the Commonwealth Labs team.

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

Testing Setup:

- Truffle v5.0.9
- Ganache v6.1.0
- Mythril v0.20.2
- MAIAN commit sha: ab387e1

Steps taken to run the full test suite:

1. Installed Truffle: `npm install -g truffle`
2. Installed Ganache: `npm install -g ganache-cli`
3. Flattened the source code using `truffle-flattener` to accommodate the auditing tools.
4. Installed the Mythril tool from Pypi: `pip3 install mythril`
5. Ran the Mythril tool on each contract: `myth -x path/to/contract`
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`

Assessment

Findings

Unexpected Ether

Severity: Medium

Status: Fixed

Contract(s) affected: `Lockdrop`

Description: Smart contracts, though they may not expect it, can receive ether forcibly. This may affect the operation of the smart contract in unpredictable ways. For example, let's say a smart contract can only pay out when the balance matches some number exactly and takes in preset amounts of ether to prevent cheating. However, an attacker can force the contract to receive a small amount of ether and may be able to forcibly withdraw the jackpot.

Exploit Scenario: In the function `lock()`, the assertion `address(this).balance == 0` may fail if the contract is forcibly sent ether through a `selfdestruct`. If this occurs, all subsequent `lock()` calls will fail.

Recommendation: Remove the require statement.

Test Results

Test Suite Results

```
Contract: Lockdrop
    ✓ should setup and pull constants (38ms)
    ✓ should lock funds and also be a potential validator (61ms)
    ✓ should unlock the funds after the lock period has ended (123ms)
    ✓ should not allow one to lock before the lock start time (38ms)
    ✓ should not allow one to lock after the lock start time
    ✓ 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 (138ms)
    ✓ should generate the allocation for a substrate genesis spec with THREE_MONTHS term (293ms)
    ✓ should generate the allocation for a substrate genesis spec with SIX_MONTHS term (259ms)
    ✓ should generate the allocation for a substrate genesis spec with TWELVE_MONTHS term (269ms)
    ✓ should aggregate the balances for all non validators and separate for validators (293ms)
    ✓ should turn a lockdrop allocation into the substrate genesis format (301ms)
    ✓ should allow contracts to lock up ETH by signalling (67ms)
    ✓ ensure the contract address matches JS RLP script

  14 passing (3s)
```

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

Since current versions of `mythril` only support `0.4.*` contracts, after flattening all contracts, the `pragma` was modified from `^0.5.0` to `0.4.25`. The tool only produced false positives, as explained below:

- In the `Lockdrop` constructor, a warning indicates that the expression `startTime + LOCK_DROP_PERIOD` may overflow. Since the `startTime` is specified by the contract owner, this is not an issue.
- In `unlockTimeForTerm()`, a warning states that a reachable exception may occur. Since the function is `internal` and only called with valid `Term` values, this is a false positive.
- Mythril reports integer overflows in `SafeMath`, however these are handled by the following require-statements and are therefore false positives.

MAIAN

MAIAN has not detected any issues.

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.

There is a minor discrepancy between the whitepaper and the contract in terms of when users may invoke `signal()`. In the whitepaper description of the Lockdrop contract, it is mentioned that "Following the initial two week period, lock signals will be accepted". However the `signal()` function only allows signals during the two-week period (due to the modifiers `didStart` and `didNotEnd`).

Code Documentation

The code is well-written and documented. Library dependencies are properly managed through `npm`. Here are a few minor suggestions:

- While the assembly-based implementation of Lock works as expected, unless optimized gas efficiency is needed for these functions, it may be favorable to simply use the non-assembly equivalent code. If users of Lockdrop wish to inspect the smart contracts before utilizing them, they may be more easily convinced of its correctness with typical approaches, rather than dropping into assembly code.
- On Line 67: `assert(address(lockAddr).balance == msg.value)`, since `eth` has already been defined above, it can be reused instead of `msg.value` if desired.

Adherence to Best Practices

The code adheres to best practices. Further, it mitigates any potential issues by deploying new Lock contracts for every deposit.

Appendix

File Signatures

The following are the SHA-256 hashes of the audited contracts and 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	Tests
`dc6f82756719ac8e27529b61201f13101197442798e1418cc56843702b64fbee` `./contracts/Migrations.sol`	`a671757733f713d56f006dbec8977422b37f7ae636b5ba421f85e1b019324a46` `./test/lockdrop.spec.js`
`b3089fc52a09660f632e308089af690aee2475e1ecfaec53301e886214f71251` `./contracts/Lockdrop.sol`	

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.