SOLIDIFIED

Audit Report for Gath3r - July 8, 2021

Summary

Audit Report prepared by Solidified covering the Gath3r smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code. The debrief took place on 9 July 2021.

Audited Files

The source code has been supplied in the form zip file repository snapshot (tag id: `bc8a7e369590`)

Scope limited to the following files:

```
contracts/
├── Distribution.sol
├── DistributionMultiowned.sol
├── DistributionMultiownedV2.sol
├── DistributionV2.sol
└── Multiowned.sol
```

Intended Behavior

The smart contracts implement a fund distribution contract with a multi sig model.

Code Complexity and Test Coverage

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does equate to a higher risk. Certain bugs are more easily detected in unit testing than a security audit and vice versa. It is, therefore, more likely that undetected issues remain if the test coverage is low or non-existent.

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium	-
Level of Documentation	Medium	-
Test Coverage	Medium	-

SOLIDIFIED

Audit Report for Gath3r - July 8, 2021

Issues Found

Solidified found that the Gath3r contracts contain no critical issues, 1 major issue, 3 minor issues, 1 Warning in addition to 3 informational notes.

We recommend all issues are amended, while the notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	Multiowned.sol: removeOwner breaks contract if not called on last owner	Major	Pending
2	Multiowned.sol: Missing emit keyword for events	Minor	Pending
3	Multiowned.sol: Edge case in confirmAndCheckForAlmostAll()	Minor	Pending
4	Multiowned.sol: One malicious owner can cause DoS	Minor	Pending
5	Multiowned.sol: Consider reusing existing multisig model	Warning	Pending
6	Wide Solidity compiler target	Note	-
7	Code cleanup	Note	-
8	Multiowned.sol: Missing return value for all code paths	Note	-

SOLIDIFIED

Audit Report for Gath3r - July 8, 2021

Critical Issues

No critical issues have been found.

Major Issues

1. Multiowned.sol: removeOwner breaks contract if not called on last owner

The only circumstance in which `m_numOwners` is reduced is if the removed owner maps to the last spot in the `m_owners` map. If it doesn't, then an owner will be removed without decreasing the `m_numOwners` counter, breaking the "All Owners" and "Almost All Owners" checks in the contract.

Recommendation
Replace the custom implementation of indexable map with something known to work and well tested like EnumerableMap from OpenZeppelin.

Minor Issues

2. Multiowned.sol: Missing emit keyword for events

Most of the events do not use the `emit` keyword while emitting an event. By default, the compiler prevents the code from compiling, but there is a solidity bug in the compiler version which allows events without the `emit` keyword.

Recommendation
We recommend adding the `emit` keyword to all the events emitted to avoid compatibility issues.

3. Multiowned.sol: Edge case in confirmAndCheckForAlmostAll()

The function `confirmAndCheckForAlmostAll()` will fail if there is only one owner address. Since the function `addOwner()` utilizes the `onlyalmostallowners` modifier, this situation is not recoverable.

Recommendation
Consider either fixing the code in `confirmAndCheckForAlmostAll()` function or checking that the number of owners is greater than one when initializing the `Multiowned`.

4. Multiowned.sol: One malicious owner can cause DoS

A single malicious owner could always sabotage pending actions by issuing a number of pending transactions, so that the `512 == m_multiOwnedPendingIndex.length`, and the pending list is cleared.
The malicious address can prevent itself from being removed as well.

Recommendation
Consider implementing a different method to handle the pending queue to prevent this.

Warning

5. Multiowned.sol: Consider reusing existing multisig model

The smart contract tries to implement a custom multisig approval model for transactions. Several reliable existing solutions exist and are generally preferable to custom original implementations, since multsig contracts are critical components.

Recommendation
Consider using an existing multisig system like Gnosis Safe to implement the functionality.

Informational Notes

6. Wide Solidity compiler target

The contracts use different compiler versions defined by pragmas. It is considered best practice to stick to a single compiler version throughout the codebase.

Recommendation
Choose a single compiler version.

7. Code cleanup

It is recommended to fix the following code cleanup notes.

1. The contracts contain a lot of duplicate code and consider using reusable functions and modifiers to increase the readability and reduce the overall size.
2. `Multiowned.sol`: Consider using a library like `EnumerableSet` from OpenZeppelin rather than hand rolling the indexable owner and pending map.
3. `Multiowned.sol`: Consider adding curly braces for all conditions for better readability.
4. `Multiowned.sol`: The modifier `onlyallowners` never used.
5. Consider using `require()` instead of `assert()` to save gas for failed transactions.

8. Multiowned.sol: Missing return value for all code paths

In the functions `confirmAndCheck()`, `confirmAndCheckForAll()`, `confirmAndCheckForAlmostAll()` not all code paths return a result.

Recommendation
Consider returning a result for all code paths.

Audit Report for Gath3r - July 8, 2021

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Gath3r its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Solidified Technologies Inc.