SOLIDIFIED

Audit Report for Morpher Protocol - April 12, 2021

Summary

Audit Report prepared by Solidified covering the Hop Protocol smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on April 12, 2021, and the results are presented here.

Audited Files

The source code has been supplied in the form of a GitHub repository:

https://github.com/Morpher-io/MorpherProtocol

Commit number: `de6b56ec5bf44c096ddfd56fcaa9072f1380005f`

The scope of the audit was limited to the following files:

```
contracts
├── IERC20.sol
├── IMorpherStaking.sol
├── IMorpherState.sol
├── IMorpherToken.sol
├── MerkleProof.sol
├── Migrations.sol
├── MorpherAdmin.sol
├── MorpherAirdrop.sol
├── MorpherBridge.sol
├── MorpherEscrow.sol
├── MorpherFaucet.sol
├── MorpherGovernance.sol
├── MorpherOracle.sol
├── MorpherStaking.sol
├── MorpherState.sol
├── MorpherToken.sol
├── MorpherTradeEngine.sol
├── Ownable.sol
└── SafeMath.sol
```

Intended Behavior

The smart contracts implement an order book exchange with accompanying staking and governance functionality. The protocol is essentially a permissioned exchange with an on-chain order book for transparency.

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	High	-
Level of Documentation	High	-
Test Coverage	Medium	-

Issues Found

Solidified found that the Morpher protocol contracts contain no critical issue, 5 major issues, 2 minor issues, in addition to 4 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	MorpherState.sol: setSideChainMerkleRoot() does not pay the operating reward	Major	Pending
2	MorpherGovernance.sol: Erroneous vote reset	Major	Pending
3	MorpherToken.sol: Contract owner has control over balances	Major	Pending
4	MorpherGovernance.sol: Ineffective governance system	Major	Pending
5	MorpherState.sol: Renouncing Ownership breaks functionality	Major	Pending
6	MorpherToken.sol: Restriction on token transfers erroneous when using transferFrom()	Minor	Pending
7	Integer Arithmetic Precision: Divisions before multiplications throughout the codebase	Minor	Pending
8	MorpherBridge.sol: Tautology in transferToSideChain()	Note	-
9	MorpherBridge.sol: Unused private function	Note	-
10	MorpherBridge.sol: Wrong comment	Note	-
11	Incorrect Documentation on Ganache port	Note	-

Critical Issues

No critical issues have been identified.

Major Issues

1. `MorpherState.sol:` `setSideChainMerkleRoot()` does not pay the operating reward

The function `setSideChainMerkleRoot()` fails to invoke the function `payOperatingReward()` due to missing function call parenthesises in the statement:

```
payOperatingReward;
```

Recommendation
Change the statement to:

```
payOperatingReward();
```

2. `MorpherGovernance.sol:` Erroneous vote reset

The statement in line 184 erroneously resets the administrator vote count:

```
countVotes[administratorVote[validatorAddress[i]]] = 0;
```

The statement should, instead, reset the oracle vote count.

Recommendation
Change the statement to:

```
countVotes[OracleVote[validatorAddress[i]]] = 0;
```

3. `MorpherToken.sol:` Contract owner has control over balances

The overall architecture of the protocol relies on separating state from implementation. However, this is not achieved through a proxy pattern (eternal storage) but by a separate state contract that can also be upgraded. It is clear that the protocol is meant to implement a trusted exchange with an on-chain component for transparency. However, in the case of the token, allowing the state to be replaced by the token owner, essentially gives the Morpher team full control over all user balances. In extension this allows provides control of the governance system.

Recommendation

Consider separating token balances state from the rest of the state and treating this part of the system as non-upgradable.

4. MorpherGovernance.sol: Ineffective governance system

The governance system allows users to become validators. However, there are several flaws in the system:

- There are 21 slots available and once these are filled, there is no way to replace a validator. It is not possible to overbid existing validators to replace them.
- There is no real incentive to become a validator. Validators simply stake their tokens. Unstaking results in fewer tokens being returned.
- There seems to be no advantage to the 7-day delay when a validator joins. It simply blocks voting for a while.
- The cool-down period could be taken advantage of for DoS-style attacks.

Recommendation

Consider revising the governance protocol or disabling governance in the early releases of the protocol.

5. MorpherState.sol: Renouncing Ownership breaks functionality

The documentation states that the team plans to renounce ownership of the state contract at some point. However, a number of functions in the protocol are only executable by the state contract owner. Thus, renouncing ownership would break the protocol.

Recommendation

Change the modifier and governance contracts to allow governance to access these functions.

Minor Issues

6. MorpherToken.sol: Restriction on token transfers erroneous when using transferFrom()

The modifier `canTransfer()` enables transfer operations for authorized addresses. However, this checks for `msg.sender` being transfer-enabled. In the case of `transferFrom()`, the sender account should be checked instead. The current implementation would allow users to have unauthorized transfers executed on their behalf by other whitelisted accounts.

Recommendation
Change the modifier to use the origin address instead of `msg.sender`.

7. Integer Arithmetic Precision: Divisions before multiplications throughout the codebase

Throughout the codebase, integer arithmetic precision could be improved by switching the order of operations. It is generally best to execute divisions on the result of multiplication than vice versa.

Recommendation
Change the order of operations.

Informative Notes

8. MorpherBridge.sol: Tautology in transferToSideChain()

The function `transferToSideChain()` includes the following pre-condition check:

```
require(_tokens >= 0, "MorpherBridge: Amount of tokens must be positive.");
```

However, the argument `_tokens` is an unsigned integer variable and, therefore must always be greater or equal to 0.

Recommendation
Remove the unnecessary check.

9. `MorpherBridge.sol:` Unused private function

The function `setInactivityPeriod()` is a private function but is not used anywhere in the contract.

Recommendation
Remove unused code.

10. `MorpherBridge.sol:` Wrong comment

The comment documentation of function `trustlessTransferFromLinkedChain()` erroneously refers to `trustlessTransferFromSideChain()`.

Recommendation
Correct the comment.

11. Incorrect Documentation on Ganache port

The port number in the documentation (7545) differs from the `truffle-config.js` file (8545).

Recommendation
Correct the documentation.

Disclaimer