Morphose
Smart Contracts Audit

SMCAuditors

https://smcauditors.com / info@smcauditors.com

26th of March 2021

This document is the audit of Morphose smart contracts performed by SMCAuditors.

1. Executive Summary

This report was written to provide a security audit for Morphose (https://morphose.cash) smart contracts. Morphose is a Zk-Snark based cryptocurrency mixer. SMCAuditors conducted the audit focusing on whether Morphose smart contracts and circuits are designed and implemented without any security vulnerabilities or cryptographic flaws. The contracts and circuits as stated are listed below with their links.

- MorphoseAdmin.sol
    - https://github.com/morphosecash/morphose-core/contracts/blob/main/MorphoseAdmin.sol
- Morphose.sol
    - https://github.com/morphosecash/morphose-core/contracts/blob/main/Morphose.sol
- MembershipVerifier.sol
    - https://github.com/morphosecash/morphose-core/contracts/blob/main/MembershipVerifier.sol

- MorphoseToken.sol
    - https://github.com/morphosecash/morphose-core/contracts/blob/main/MorphoseToken.sol
- Membership.circom
    - https://github.com/morphosecash/morphose-core/circuits/blob/main/Membership.circom

Morphose team requested rigorous review of their code. We have run extensive static analysis of the codebase as well as standard security assessment utilising industry approved tools. There are no critical/high level issues with the currently deployed contracts. Our medium and low level findings are available in the next section.

2. Audit Findings

MorphoseAdmin.sol

Low Level Findings

- ❖ Contract owner has no other controls except creating mixers. We recommend a more fine-grained permissions system.
    - ➢ Team's Response: we use mixer creation on deployment and after that we transfer ownership to zero-address. This is why we don't need anymore owner permissions.
- ❖ There is no function to remove or edit a mixer. Necessity of updates on existing mixers have to be carefully reviewed considering this issue.
    - ➢ Team's Response: Mixers won't be edited or removed in their lifecycle. We will carefully review their constructor variables before deployment.
- ❖ Solidity Compiler Version should be locked. Pragma should be simplified and the version should be locked for published contracts, ideally using the most recent Solidity compiler version supported by the build framework.
    - ➢ Team's Response: Solidity compiler has been updated to the recent 0.8.0

MembershipVerifier.sol

Low Level Findings

- Multiple hardcoded keys on *verifyingKey and verify* functions. Hard coded values can lead to vulnerabilities. Create functions to update these values or make sure you are using them right.
    - Team's Response: we use the standard zk-snark verifier which is also used in the solc compiler. The same verifier code is used in several other widely used repos as well. This is why we didn't update anything from the original code and won't be updating.

Morphose.sol

Low Level Findings

- Hardcoded address on *withdraw* function. Since we cannot modify smart contracts after deploying them, hard coded addresses can lead to vulnerabilities. Create a function to update the address.
    - Team's Response: Since we will transfer ownership of all contacts, creating a function to update the treasury address won't matter. This is an address that won't be changed and it can stay as it is.

Medium Level Findings

- No mechanism for handling addresses that reject cryptocurrency transfer. Using the relayer, if the person who withdraws the deposit sends the funds to an address that rejects the transfer, the key will be used but the transaction won't be rolled back. You can consider adding mechanisms for rolling back these kinds of transactions and holding balance for each user in the smart contract.
    - Team's Response: If we had used a relayer node that spends gas every time a user makes a transaction we would need to add this mechanism since a malicious user would take advantage of this situation to make relayer spend gas fees repeatedly. However we use an integrated relayer that the user himself pays for relayer every time, so we assume he won't send the transactions to an address that rejects transfers to pay for gas fees every time. Standard procedures for reverting standard transactions are applied to contracts but we don't find necessary to implement additional mechanisms for this case.

MorphoseToken.sol

Medium Level Findings

- Insecure approve() Implementation. The ERC-20(BEP-20) standard has a security vulnerability related to the approve() method being vulnerable to race-condition. The current implementation does not mitigate this. Consider adding functionality to increase or decrease allowances, such as implemented by the Openzeppelin ERC-20 implementation.
    - Team's Response: approve() implementation updated to OpenZeppelin's standard.

3. Conclusion

In this audit, we thoroughly analyzed the Morphose smart contracts. Overall, the smart contracts were well written and common security standards were used by the team. Our identified issues are promptly confirmed, taken into consideration and resolved accordingly.

4. Disclaimer

This report is not advice on investment, nor does it guarantee adequacy of a business model and/or a bug-free code. This report should be used only to discuss known technical problems. It will be necessary to resolve addressed issues and conduct thorough tests to ensure the safety of the smart contract.