FANTOM

Fantom Multisig Contract Review

Version: 1.0

July, 2018

Contents

Introduction

Sigma Prime was commercially engaged by Fantom [1] to perform a time-boxed security review of the smart contract `MultiSigWallet` which implements a mechanism allowing multiple parties to agree upon a transaction prior to its execution. The review focuses solely on the security aspects of the Solidity implementation of the contract, however general recommendations and informational comments are also offered.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review regarding, the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the contract contained in the scope of the security review (`MultiSigWallet`). A summary of the discovered vulnerabilities is then followed by a detailed review, where a severity rating is assigned to each vulnerability (see Vulnerability Severity Classification), along with an open/closed status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as "informational". Outputs of automated testing that were developed during this assessment are also included for reference (see the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities in the `MultiSigWallet` contract.

Overview

The `MultiSigWallet` contract serves multiple purposes, namely it:

- Specifies a set of `owners` and a `required` number of owners that must reach agreement on a transaction prior to its execution.
- Includes an `enterRecoveryMode` function allowing an `owner` to progressively decrease `required` by `1` if the elapsed time since the last transaction exceeds the `recoveryModeTriggerTime` .
- Provides functionality allowing an `owner` to submit a transaction. If the `required` number of owners confirms the transaction, the transaction may be executed by an `owner` .
- Allows an `owner` to revoke their confirmation of a transaction, if desired.
- Provides a `public` function (`getTransactionIds`) that returns an array containing all transactions IDs less than an input value `_number` .

It should be noted that this multisig is a one-time use multi-sig contract. Owners cannot be added or removed and if the recovery mode is activated it cannot be inactivated.

Note that `MultiSigWallet` does not contain any development relating to Fantom's DAG-based platform. The contract relates solely to transactions performed on the Ethereum blockchain and does not dictate the Fantom platforms' functionality. Additional details regarding Fantom's DAG-based platform can be found in the Fantom Whitepaper [1].

Audit Summary

This review was initially conducted on commit ac15ce9, which includes the file MultiSigWallet.sol. This file contains a single contract (`MultiSigWallet`) and a single library (`SafeMath`).

The final version of this review targets commit 48c51ba.

Per-Contract Vulnerability Summary

SafeMath (`MultiSigWallet.sol`)

No potential vulnerabilities have been identified.

MultiSigWallet (`MultiSigWallet.sol`)

Some gas-saving modifications are suggested.
All discovered vulnerabilities have been addressed by the authors.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the `MultiSigWallet` contract. The severity classification assigned to each vulnerability is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contract, including gas optimisations, are also described in this section and are labelled as “informational”.

Summary of Findings

ID	Description	Severity	Status
FMS-01	A set of owners (less than `required`) can force arbitrary transactions	Medium	Resolved
FMS-02	Redundant function logic	Informational	Resolved
FMS-03	Miscellaneous comments, gas savings and suggestions	Informational	Resolved

FMS-01	A set of owners (less than `required`) can force arbitrary transactions		
Asset	MultiSigWallet.sol		
Status	Closed: Resolved in commit 48c51ba.		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

Conventionally, Multisignature wallets require p signatures of n total users in order to execute a transaction. If $n-p+1$ users disagree with the transaction, the transaction should not be accepted. This multi-signature wallet, allows the collusion of $n-p+1$ (or more) to force an arbitrary transaction to be accepted.

To clarify, consider the case $n = 10$, $p = 7$. That is, there are 10 owners and 7 are required to confirm a transaction. $n - p + 1 = 4$, so consider 4 owners wish to steal the funds. Typically 4 owners would not be enough to confirm a transaction which takes the funds. In this case, the 4 owners can perform an attack where they never confirm a transaction, which means the remaining 6 owners can never submit a transaction. This can eventually force the recovery mode by making the contract stagnant. Once the specified time has elapsed, there is a race condition for the 4 malicious owners to `enterRecoveryMode()` then steal the funds. Thus $n - p + 1$ owners is sufficient force arbitrary transactions, compared to the traditional, required p owners.

Recommendations

There are a number of implementations that can address this issue. One example would be to allow an owner to increment `lastTransactionTime`. Thus if all keys are not lost, an owner can prevent a subset of malicious users from calling `enterRecoveryMode()`. This solution only forces a stalemate between malicious users preventing anyone from withdrawing funds.

Another possibility is to change the functionality of `enterRecoveryMode()` such that it reduces the value of `required` by one. In each recovery periodn, p will be reduced which therefore increases the number of required malicious users ($n-p+1$) to perform this attack. In the example $n = 10, p = 7$, the 6 good owners would then be able to perform a transaction after one iteration of `enterRecoveryMode` and the 4 malicious actors still cannot confirm a transaction.

Resolution

The second recommendation was opted for by the authors whereby the recoveryMode function now reduces `required` by one each call.

FMS-02	Redundant function logic
Asset	MultiSigWallet.sol
Status	Closed: Removed from codebase in commit d2da277.
Rating	Informational

Description

The function `getTransactionIds()` contains redundant logic.

This function will always return an array of integers from 0 to `number` if `number` is less than `transactionCount`. As `count` will always be equal to `i`, the resulting array will always be of the form `[0,1,2,3,4,5...]`.

Furthermore, if `_number` is greater than `transactionCount` the for loop will attempt to assign a value to the array, greater than the arrays length.

The entire function appears to be redundant, which may be a bi-product from simplifying the original contract.

Recommendations

Either remove this function, unless it has some purpose, i.e backwards compatibility or if it is necessary, on line [284] replace `transactionCount` with `_number` to ensure the `_transactionIds` array's referenced index is not greater than the array's length.

Resolution

This function was removed from the codebase.

FMS-03	Miscellaneous comments, gas savings and suggestions
Asset	MultiSigWallet.sol
Status	Closed: See inline comments
Rating	Informational

Description

This section describes some non-security related comments and potential gas-saving optimisations that were discovered in the process of this audit.

- [209] - The `require` on this line is redundant as this is checked in the `notNull` modifier.
  ✓ Resolved in commit [24f133a].
- Lines [187, 257, 276] initialise a variable to it's default value. These are memory variables and cost minimal gas, however not initialising these variables can save gas in unoptimized code.
  ✓ Resolved in commit [24f133a].
- [99] - The `_recoveryModeTriggerTime` is only validated to be greater than 0. There is no minimum time, thus multisigs of this form can be set with very low `recoveryModeTriggerTime` for example, 1, defeating the purpose of a multisig contract.
  ✓ This has been acknowledged by the authors.
- Once the requisite time has expired and an owner activates the recovery mode, the contract cannot be return to its original state. It will forever have `required = 1`, indicating that a new multisig contract will need to be deployed.
  ✓ This has been acknowledged by the authors.

Recommendations

Ensure these are as expected.

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are given along with this document. The `pytest` framework was used to perform these tests and the output of `pytest -v` is given below.

```
test_deployment.py::test_deploys PASSED                                  [  2%]
test_deployment.py::test_params_instantiated PASSED                      [  4%]
test_deployment.py::test_wont_deploy_with_invalid_required_count PASSED  [  6%]
test_execute_transaction.py::test_execute_transaction[8-5-5-True] PASSED [  9%]
test_execute_transaction.py::test_execute_transaction[2-1-1-True] PASSED [ 11%]
test_execute_transaction.py::test_execute_transaction[4-4-4-True] PASSED [ 13%]
test_execute_transaction.py::test_execute_transaction[1-1-1-True] PASSED [ 15%]
test_execute_transaction.py::test_execute_transaction[20-10-10-True] PASSED [ 18%]
test_execute_transaction.py::test_execute_transaction[20-10-1-False] PASSED [ 20%]
test_execute_transaction.py::test_execute_transaction[8-5-4-False] PASSED [ 22%]
test_execute_transaction.py::test_execute_transaction[2-2-1-False] PASSED [ 25%]
test_recovery_mode.py::test_updates_last_tx_time PASSED                  [ 27%]
test_recovery_mode.py::test_enters_recover[100-50-False] PASSED          [ 29%]
test_recovery_mode.py::test_enters_recover[100000-50-False] PASSED       [ 31%]
test_recovery_mode.py::test_enters_recover[937502340-100-False] PASSED   [ 34%]
test_recovery_mode.py::test_enters_recover[100-100-True] PASSED          [ 36%]
test_recovery_mode.py::test_enters_recover[1-100-True] PASSED            [ 38%]
test_recovery_mode.py::test_enters_recover[100-101-True] PASSED          [ 40%]
test_recovery_mode.py::test_enters_recover[100-10000-True] PASSED        [ 43%]
test_recovery_mode.py::test_recovery_iterations[3-2-1-True] PASSED       [ 45%]
test_recovery_mode.py::test_recovery_iterations[5-2-1-False] PASSED      [ 47%]
test_recovery_mode.py::test_recovery_iterations[5-2-3-True] PASSED       [ 50%]
test_recovery_mode.py::test_recovery_iterations[5-3-2-True] PASSED       [ 52%]
test_recovery_mode.py::test_recovery_iterations[5-1-2-False] PASSED      [ 54%]
test_recovery_mode.py::test_recovery_iterations[5-1-3-False] PASSED      [ 56%]
test_recovery_mode.py::test_recovery_iterations[5-1-4-True] PASSED       [ 59%]
test_recovery_mode.py::test_recovery_iterations[5-3-1-False] PASSED      [ 61%]
test_revoke_confirmation.py::test_confirmation_revocation PASSED         [ 63%]
test_scenario.py::test_scenarios[10-10-10-5-50-1-1-True] PASSED          [ 65%]
test_scenario.py::test_scenarios[10-1-5-2-30-1-1-True] PASSED            [ 68%]
test_scenario.py::test_scenarios[10-1-1-0-1-1-1-True] PASSED             [ 70%]
test_scenario.py::test_scenarios[1-1-1-0-1-1-1-True] PASSED              [ 72%]
test_scenario.py::test_scenarios[10-5-5-9-50-50-1-True] PASSED           [ 75%]
test_scenario.py::test_scenarios[10-10-4-5-50-1-2-False] PASSED          [ 77%]
test_scenario.py::test_scenarios[10-1-10-2-30-1-2-True] PASSED           [ 79%]
test_scenario.py::test_scenarios[10-1-1-0-1-1-2-True] PASSED             [ 81%]
test_scenario.py::test_scenarios[1-1-1-0-1-1-2-True] PASSED              [ 84%]
test_scenario.py::test_scenarios[10-5-5-9-50-50-2-True] PASSED           [ 86%]
test_scenario.py::test_scenarios[10-10-4-5-50-1-3-False] PASSED          [ 88%]
test_scenario.py::test_scenarios[10-1-10-2-30-1-3-False] PASSED          [ 90%]
test_scenario.py::test_scenarios[10-1-1-0-1-1-3-False] PASSED            [ 93%]
test_scenario.py::test_scenarios[1-1-1-0-1-1-3-False] PASSED             [ 95%]
test_scenario.py::test_scenarios[10-5-1-9-50-50-3-False] PASSED          [ 97%]
test_token_transfer.py::test_token_transfer PASSED                       [100%]

============================ 44 passed in 16.21 seconds ============================
```

Appendix B Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

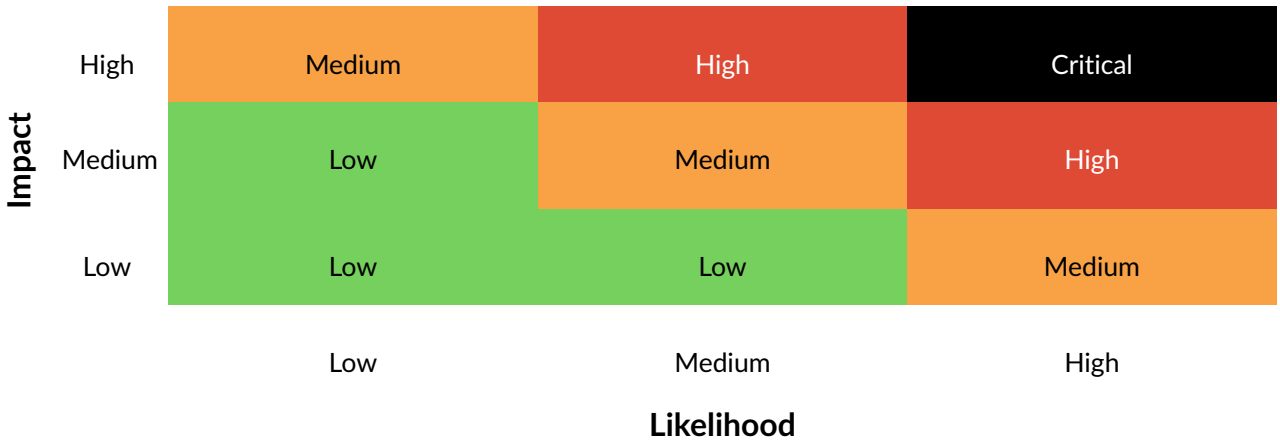

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] FANTOM - Whitepaper v1.3. Website, May 2018, Available: http://www.fantom.foundation/data/FANTOM%20Whitepaper%20English%20v1.3.pdf.