Source Code Review

Prepared for Coinfabrik• January 2018

V1.1

1. Table of Contents

2. Executive Summary

In December 2017, Coinfabrik engaged Coinspect to perform a security audit of the Syndicate contract. The contract was received in the following repository branch:

https://github.com/dggventures/syndicate commit
cae07e171ee237739263221c9e64e6fed6005e1d

The objective of the audit was to evaluate the security of the smart contract. During the assessment, Coinspect identified the following issues:

High-Risk	Medium Risk	Low Risk	No Risk
0	2	1	0

After notifying Coinfabrik of these issues, Coinfabrik fixed every one of the them.

All issues were fixed in commit
833a406779829e629100c42d1a6a7461981500db

3. Introduction

The contract Syndicate token comprises the following contracts:

- EIP20Token.sol
- Ownable.sol
- SafeMath.sol
- Syndicate.sol

The first three corresponds to variants of the standard OpenZeppelin contracts. The Syndicate.sol contract, which inherits from Ownable, is the one that implements the crowdfund functionality.

A whitebox security audit was conducted on these smart contracts.
The present report was completed on January 7th, 2017, by Coinspect. The report includes all issues identified in the audit.

The following checks, related to best practices, were performed:

- Confusion of the different method calling possibilities: send(), transfer(), and call.value()

- Missing error handling of external calls
- Erroneous control flow assumptions after external calls
- The use of push over pull for external calls
- Lack of enforcement of invariants with assert() and require()
- Rounding errors in integer division
- Fallback functions with higher gas limit than 2300
- Functions and state variables without explicitly visibility
- Missing pragmas to for compiler version
- Race conditions, such as contract Reentrancy
- Transaction front running
- Timestamp dependence
- Integer overflow and underflow
- Code blocks that consumes a non-constant amount of gas, that grows over block gas limit.
- Denial of Service attacks
- Suspicious code or underhanded code.
- Error prone code constructs
- Race conditions

4. Findings

The list of issues found follows.

4.1. Potential Reentrancy problem in start()

Low Risk, FIXED

The start() function of the Syndicate.sol contract transfers ethers to the admins. However, the state of the contract is unchanged before transfers occur, meaning that a malicious admin in collusion with the contract owner could potentially re-enter start() with a recursive call. Because start() uses the transfer() method, which limits the amount of gas transferred to the callee, and because start() always perform the fee payments, this defect cannot be exploited to double-pay the admins. However it's a good practice to first change the state of the contract before calling external functions. In this case, executing "state = State.Started" before performing the fee payments protects the method from reentrancy. If this contract is re-used in a Solidity-compatible platform that subsidizes funds transfers or has different semantics regarding for CALL gas passed, this flaw could become exploitable.

Recommendation

Execute "state = State.Started" before executing transfers,

4.2. A malicious admin can block the token investment

Medium Risk, FIXED

The method start() pays to the admins using the transfer() method. This method fails if the called contract fails. As the callee is controlled by an admin, the admin can execute a REVERT instruction when receiving the fees and cause the start() method to revert, blocking the investment. This can also happen because of a bug in the admin contract that receives the payment.

Recommendation

Use the send() function instead of transfer(). Check the return value, and transfer the fee to a designated third party if a transfers fails to accept a payment.

4.3. Token address update during lock period may be used to steal tokens by contract owner

Medium Risk, FIXED

The update_token() method allows the owner to change the token contract during the lock period. This has two consequences. First, a malicious owner can use it to change the token

contract unilaterally and give the investors fake tokens. Second, a malicious owner can use update_token() in conjunction with approve_unwanted_tokens() to move out of the Syndicate contract the tokens before the lock period has expired. The attack can be carried out in a single transaction by executing the following sequence:

- update_token(fake_token)
- approve_unwanted_tokens(real_token_address, attackers_address,total_nr_of_tokens)
- update_token(real_token_address)

Recommendation

If the update_token() method is mandatory then the change of the token contract should be an operation approved by all the admins or by a certain percentage of them.. We suggest that the owner of the contract be a Gnosis multi-signature wallet, that allows to call contract functions based on a signature threshold.

5.Closing Remarks

It has been a pleasure to work with Coinfabrik. We believe that the Coinfabrik Syndicate contracts are free from critical defects. The scope of the present security audit is limited to smart contract code. It does not cover the technologies and designs related to these smart contracts, nor the frameworks and wallets that communicate with the contracts, nor the operational security of the company that created and will deploy the audited contracts. This document should not be read as investment or legal advice.