FIN & NOM TOKEN SMART CONTRACT AUDIT RESULTS

FOR FINOM AG

05/08/2018

Made in Germany by chainsulting.de

Change history

Version	Date	Author	Changes
1.0	07.05.2018	Y. Heinze	Audit created
1.5	08.05.2018	Y. Heinze	Vulnerability check
2.0	09.05.2018	Y. Heinze	Executive Summary

Smart Contract Audit FINOM ICO

Table of Contents

1. Disclaimer

The audit makes no statements or warrantees about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of FINOM AG. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

2. About the Project and Company

Company address:

Finom AG
Alte Haslenstrasse 5
9053 Teufen Switzerland

Company Check: https://www.easymonitoring.ch/handelsregister/finom-ag-357643

Project **Overview:**

Finom is a Blockchain company founded after the merger of market leaders - trading app TabTrader, mining multipool Nanopool, exchange Cryptonit - along with mining farm Cryptal and brokerage app Beetle.io. Finom is building the genetic code of the future economic and financial system.

Guided by the principles of transparency and security, they create one universal and low cost tool for an easy management and access to finance.

They use Blockchain technology to create trustworthy and convenient financial services. That allow users all over the world to become more independent, to improve their material prosperity and to be sure of safety of their investments.

Finom AG is offering two kind of Tokens, security token (FIN) and utility token (NOM).
Whitepaper::https://finom.io/files/whitepaper_eng.pdf

3. Vulnerability Level

0-Informational severity – A vulnerability that have informational character but is not effecting any of the code.

1-Low severity - A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.

2-Medium severity – A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.

3-High severity – A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.

4-Critical severity – A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.

4. Overview of the audit

The project has two tokens, the FIN Token which contains 290 lines of Solidity code and NOM Token which contains 285 lines of Solidity code. All the functions and state variables are well commented using the natspec documentation for the functions which is good to understand quickly how everything is supposed to work.

Etherscan:

FIN Token address:
https://etherscan.io/address/0xef6efb3fc5b9aba75af7250989db7974fd6361ba#code

NOM Token address:
https://etherscan.io/addres/0x23ab81fd565d49259675eb87209d6899bf2e814d#code

Fast overview about the spec.

Token Standard	**Token burning mechanics**	**Solidity Version**	**Token minting mechanics**
ERC20	Not supported	FIN v0.4.21 NOM v0.4.21	NOM Token
Min/max contribution	**Token bonus structure**	**Number of Tokens**	
Not supported	Not supported	FIN 2,623,304 NOM 5,650,000,000	

Verification used (FIN Token)

Verifiers can be added or removed by the owner of the smart contract. These Verifiers can approve the addresses. Transactions can only be performed between such approved addresses. Approved addresses are addresses, whose users have passed the KYC procedure.

Mintable function (NOM Token)

The first release of tokens (April 28, 2013)	3,390,000,000 (three billion 390 million)
The second additional release (July 28, 2018)	565 million (565 million)
The third additional release (October 28, 2018)	565 million (565 million)
The fourth additional release (January 28, 2019)	565 million (565 million)
The fifth additional release (April 28, 2019)	565 million (565 million)

Only the owner of a smart contract can issue tokens.

Used Code from other Smart Contracts

1. SafeMath (Math operations with safety checks that throw on error)
https://github.com/OpenZeppelin/zeppelin-solidity/tree/master/contracts/math

2. ERC20Basic (Simpler version of ERC20 interface)
https://github.com/ethereum/EIPs/issues/179
https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/token/ERC20/ERC20Basic.sol

3. Standard ERC20 (Based on code by FirstBlood)
https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/token/ERC20/StandardToken.sol

4. Mintable Token
https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/token/ERC20/MintableToken.sol

5. Attack made to the contract

Security Report

Attack: Using the approve function of the ERC-20 standard
The approve function of ERC-20 might lead to vulnerabilities.

FIN Token # 157-161
NOM Token #157-161

```
function approve(address _spender, uint256 _value) public returns(bool) {
    allowed[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value);
    return true;
}
```

Severity: 2

Result / Recommendation:
Only use the approve function of the ERC-20 standard to change allowed amount to 0 or from 0 (wait till transaction is mined and approved).

https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/edit

Attack: Functions transfer and transferFrom of ERC-20 Token should throw.
Functions of ERC-20 Token Standard should throw in special cases:

- transfer should throw if the _from account balance does not have enough tokens to spend
- transferFrom should throw unless the _from account has deliberately authorized the sender of the message via some mechanism

FIN Token # 283-289

```
function transfer(address _to, uint256 _value) public onlyVerified(msg.sender, _to) returns(bool) {
    super.transfer(_to, _value);
}

function transferFrom(address _from, address _to, uint256 _value) public onlyVerified(_from, _to) returns(bool) {
    super.transferFrom(_from, _to, _value);
}
```

Severity: 1

Result / Recommendation:
The ERC20 standard recommends throwing exceptions in functions transfer and transferFrom.

Attack: Compiler version not fixed
Solidity source files indicate the versions of the compiler they can be compiled with.

pragma solidity ^0.4.21; // bad: compiles w 0.4.21 and above
pragma solidity 0.4.21; // good: compiles w 0.4.21 only

It is recommended to follow the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee.

Severity: 2

Result / Recommendation:
Specify the exact compiler version (pragma solidity x.y.z;).

You're specifying a pragma version with the caret symbol (^) up front which tells the compiler to use any version of solidity bigger than 0.4.21 .

This is not a good practice since there could be major changes between versions that would make your code unstable. That's why I recommend to set a fixed version without the caret like 0.4.21.

Attack: Unchecked math
Solidity is prone to integer over- and underflow. Overflow leads to unexpected effects and can lead to loss of funds if exploited by a malicious account.

FIN Token # 219

```
uint256 public constant INITIAL_SUPPLY = 2623304 * (10 * * uint256(decimals));
```

Severity: 2

Result / Recommendation:
Check against over- and underflow (use the SafeMath library).
SafeMath is already used in both contracts.

Attack: Unhandled Exception

A **call**/**send** instruction returns a non-zero value if an exception occurs during the execution of the instruction (e.g., out-of-gas). A contract must check the return value of these instructions and throw an exception.

Severity: 0

Result / **Recommendation:**

Catching exceptions is not yet possible.

Attack: Transactions May Affect Ether Receiver

A contract is exposed to this vulnerability if a miner (who executes and validates transactions) can reorder the transactions within a block in a way that affects the receiver of ether.

Severity: 1

Result / **Recommendation:**

Both contracts are not vulnerable to this vulnerability as the receiver of ether is **msg.sender**, which cannot be modified by previously executed transactions

6. Executive Summary

A majority of the code was standard and copied from widely-used and reviewed contracts and as a result, a lot of the code was reviewed before. It correctly implemented widely-used and reviewed contracts for safe mathematical operations. The audit identified no major security vulnerabilities, at the moment of audit.

7. General Summary

The issues identified were minor in nature, and do not affect the security of the contract. The code specifies Solidity version 0.4.21, which has only recently had a newer version of 0.4.24 released. As a result, FINOM AG should consider updating the pragma statements to require the latest version of Solidity.

Additionally, the code implements and uses a SafeMath contract, which defines functions for safe math operations that will throw errors in the cases of integer overflow or underflows. The simplicity of the audited contracts contributed greatly to their security. The minimalist approach in choosing which pieces of functionality to implement meant there was very little attack surface available.

Solidity Version Updates

Solidity 0.4.24 will add several features which could be useful in these contracts:

- Type Checker: Improve error message for failed function overload resolution.
- Type Checker: Do not complain about new-style constructor and fallback function to have the same name.
- Type Checker: Detect multiple constructor declarations in the new syntax and old syntax.
- Type Checker: Explicit conversion of bytesXX to contract is properly disallowed.

Also recommended is to Update the etherscan.io information with Logo/Website for example. That gives buyers more transparency.

NOM Token
https://etherscan.io/tokenupdate?a=0x23ab81fd565d49259675eb87209d6899bf2e814d

FIN Token
https://etherscan.io/tokenupdate?a=0xef6efb3fc5b9aba75af7250989db7974fd6361ba