SmartDec

Faireum Smart Contracts Security Analysis

This report is public.

Published: January 24, 2019

Abstract

In this report, we consider the security of the Faireum project. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

The audit does not give any warranties on the security of the code. One audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, security audit is not an investment advice.

Summary

In this report, we considered the security of Faireum smart contracts. We performed our audit according to the procedure described below.

The audit showed no critical issues. However, one medium severity and a number of low severity issues were found. They do not endanger project security. Nevertheless, we highly recommend addressing them.

General recommendations

The contracts code is of good quality. The contracts code does not contain issues that endanger project security. However, we recommend covering Code with tests, fixing Documentation mismatch, and removing Redundant code.

In addition, if the developers decide to improve the code, we recommend following best practices for Pragmas version. However, these issues are minor and do not influence code operation.

Checklist

Smart Contracts Security

The audit showed no vulnerabilities.
Here by vulnerabilities we mean security issues that can be exploited by an external attacker. This does not include low severity issues, documentation mismatches, overpowered contract owner, and some other kinds of bugs. ✓

Compliance with the documentation

The audit showed no discrepancies between the code and the provided documentation. ✓

ERC20 compliance

We checked ERC20 compliance during the audit. The audit showed that **FaireumToken** contract was fully ERC20 compliant.

ERC20 MUST

The audit showed no ERC20 "MUST" requirements violations. ✓

ERC20 SHOULD

The audit showed no ERC20 "SHOULD" requirements violations. ✓

Tests

All the tests passed successfully. ✓

The text below is for technical use; it details the statements made in Summary and General recommendations.

Procedure

In our audit, we consider the following crucial features of the smart contract code:

1. Whether the code is secure.
2. Whether the code corresponds to the documentation (including whitepaper).
3. Whether the code meets best practices in efficient use of gas, code readability, etc.

We perform our audit according to the following procedure:

- automated analysis
    - we scan project's smart contracts with our own Solidity static code analyzer SmartCheck
    - we scan project's smart contracts with several publicly available automated Solidity analysis tools such as Remix, and Solhint
    - we manually verify (reject or confirm) all the issues found by tools
- manual audit
    - we manually analyze smart contracts for security vulnerabilities
    - we check smart contracts logic and compare it with the one described in the whitepaper
    - we check ERC20 compliance
    - we run tests and check code coverage
- report
    - we reflect all the gathered information in the report

Checked vulnerabilities

We have scanned Faireum smart contracts for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered (the full list includes them but is not limited to them):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level
- Address hardcoded
- Using delete for arrays
- Integer overflow/underflow
- Locked money
- Private modifier
- Revert/require functions
- Using var
- Visibility
- Using blockhash
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Project overview

Project description

In our analysis we consider Faireum whitepaper ("Faireum_whitepaper2.0_En.pdf", sha1sum: 630f7cd253fa9e61f56e0b51e777857bb943b6da) and smart contracts code ("FaireumToken.sol", sha1sum: 637e2aa0797f95d5853cf4f3c692a36853977c03).

The latest version of the code

After initial audit, the customer applied fixes and updated the code to the latest version("FaireumToken-v1.1.zip", sha1sum: 8712ae10b21faff29f8435de7e28dcda8dcc2e03).

Project architecture

For the audit, we were provided with the truffle project. The project is an `npm` package and includes tests.

- The project successfully compiles with `truffle compile` command.
- The project successfully passes all the tests (see Tests output).

The total LOC of audited Solidity sources is 273.

Automated analysis

We used several publicly available automated Solidity analysis tools. Here are the combined results of SmartCheck, Solhint, and Remix. All the issues found by tools were manually checked (rejected or confirmed).

True positives are constructions that were discovered by the tools as vulnerabilities and can actually be exploited by attackers or lead to incorrect contracts operation.

False positives are constructions that were discovered by the tools as vulnerabilities but do not consist a security threat.

Cases when these issues lead to actual bugs or vulnerabilities are described in the next section.

Tool	Rule	False positives	True positives
Remix	Gas requirement of function high: infinite	24	
	Should be constant but is not	2	
	Use of "now"	3	
	Use of "this" for local functions	1	
Total Remix		29	
SmartCheck	Erc20 Approve		2
	Pragmas Version		1
	Private Modifier Don’t Hide Data	4	
	Safemath	2	
Total SmartCheck		6	3

Solhint	Avoid to make time-based decisions in your business logic	3	
	Compiler version must be fixed		1
	Event and function names must be different	3	
Total Solhint		6	1
Total Overall		**42**	**4**

Manual analysis

The contracts were completely manually analyzed, their logic was checked and compared with the one described in the documentation. Besides, the results of the automated analysis were manually verified. All confirmed issues are described below.

Critical issues (not found)

Critical issues seriously endanger smart contracts security. We highly recommend fixing them.

The audit showed no critical issues.

Medium severity issues (fixed)

Medium issues can influence smart contracts operation in current implementation. We highly recommend addressing them.

No tests and deployment script (fixed)

The provided code does not contain tests. Testing is crucial for code security and audit does not replace tests in any way.
We highly recommend both covering the code with tests and making sure that the test coverage is sufficient.

There is also no deployment script. However, the contracts deployment does not seem trivial. Bugs and vulnerabilities often appear in deployment scripts and severely endanger system's security.
We highly recommend developing and testing deployment scripts very carefully.
The issue has been fixed by the developers and is not present in the latest version of the code.

Low severity issues (fixed)

Low severity issues can influence smart contracts operation in future versions of code. We recommend taking them into account.

Documentation mismatch (fixed)

There is a discrepancy between the smart contracts code and the whitepaper:
According to the documentation (“Faireum_whitepaper2.0_En.pdf”, page 35):

```
Team/Developer/Advisor: Locked for a minimum of 12 months and
followed by 1 to 2 years vesting schedule to the long-term
benefit of the team/developer/advisor.
```

However, half of the tokens locked for 6 months and another half for a year in the code.

```
function lockTeamTokens(address _beneficiary, uint256
_tokensAmount) public onlyAdmin {
    uint256 _half = _tokensAmount.div(2);
    _lockTokens(address(teamAdvisorsTokensVault), false,
_beneficiary, _half);
    _lockTokens(address(teamAdvisorsTokensVault), true,
_beneficiary, _half);
}
```

We recommend either changing contracts’ functionality so that it matches the whitepaper or modifying the whitepaper in order to avoid any discrepancies.
The issue has been fixed by the developers and is not present in the latest version of the code.

Redundant code (fixed)

SafeERC20 library at **FaireumToken.sol**, lines 357-369 is redundant, as it is not used in the project.

We highly recommend removing redundant code in order to improve code readability and transparency and decrease cost of deployment and execution.
The issue has been fixed by the developers and is not present in the latest version of the code.

Pragma version (fixed)

Solidity source files indicate the versions of the compiler they can be compiled with.
Example:

```
pragma solidity ^0.4.24; // bad: compiles w 0.4.24 and above
pragma solidity 0.4.24; // good: compiles w 0.4.24 only
```

We recommend following the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee. Besides, we recommend using compiler version 0.4.25. Moreover, new version of compiler (0.5.2 at the moment) is

available and contains several important changes. In order to update your contracts to compiler version 0.5.2 the developers should follow solidity documentation.
The issue has been fixed by the developers and is not present in the latest version of the code.

Notes

ERC20 approve

There is ERC20 approve issue: changing the approved amount from a nonzero value to another nonzero value allows a double spending with a front-running attack.

We recommend instructing users to follow one of two ways:

- not to use `approve()` function directly and to use `increaseApproval()/decreaseApproval()` functions instead
- to change the approved amount to 0, wait for the transaction to be mined, and then to change the approved amount to the desired value.

This analysis was performed by SmartDec.

Boris Nikashin, Project Manger
Alexander Drygin, Analyst
Pavel Kondratenkov, Analyst
Kirill Gorshkov, Analyst

Sergey Pavlin, Chief Operating Officer

January 24, 2019

Appendix

Code coverage

```
-------------------|----------|----------|----------|----------|----------------|
File               |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
-------------------|----------|----------|----------|----------|----------------|
 contracts/        |    91.38 |    63.89 |    95.92 |    93.44 |                |
 FaireumToken.sol  |    91.38 |    63.89 |    95.92 |    93.44 |... 344,345,346 |
-------------------|----------|----------|----------|----------|----------------|
 All files         |    91.38 |    63.89 |    95.92 |    93.44 |                |
-------------------|----------|----------|----------|----------|----------------|
```

Test output

```
Compiling ./contracts/FaireumToken.sol...
Compiling ./contracts/Migrations.sol...

  Contract: FaireumToken
    ✓ has a name (38ms)
    ✓ has a symbol
    ✓ has 18 decimals
    ✓ lock start time is Mon Mar 11 2019 00:00:00 GMT+0000 (83ms)
    total supply
      ✓ the initial total supply should be zero
      ✓ the default const of total supply should be (1200000000 *
10**18)
      ✓ the total supply after createTokensVaults should be
(1200000000 * 10**18) (224ms)
    onlyAdmin
      ✓ allows the admin account to call onlyAdmin functions (259ms)
      ✓ reverts when anyone calls onlyAdmin functions (55ms)
    token vaults
      the TokensVaults before the createTokensVaults called
        ✓ teamAdvisorsTokensVault should not be 0x
        ✓ rewardPoolTokensVault should not be 0x
        ✓ foundersTokensVault should not be 0x
        ✓ marketingAirdropTokensVault should not be 0x
        ✓ saleTokensVault should not be 0x
      the TokensVaults after the createTokensVaults function called
        ✓ teamAdvisorsTokensVault should be 0x
        ✓ rewardPoolTokensVault should be 0x
        ✓ foundersTokensVault should be 0x
        ✓ marketingAirdropTokensVault should be 0x
        ✓ saleTokensVault should be 0x
      the initial TokensVaults balance after the createTokensVaults
function called
        ✓ createTokensVaults function cant be called twice
        ✓ teamAdvisorsTokensVault initial balance is correct
        ✓ rewardPoolTokensVault initial balance is correct (38ms)
        ✓ foundersTokensVault initial balance is correct
        ✓ marketingAirdropTokensVault initial balance is correct
        ✓ saleTokensVault initial balance is correct
    balanceOf
      the contract creator account should no tokens at the beginning
        ✓ returns zero
      the contract creator account should no tokens after
createTokensVaults function called
        ✓ returns zero
    lock balance
      lock all of the balance
        ✓ lock teamAdvisorsTokensVault with a odd number (76ms)
        all of the rewardPoolTokensVault tokens can be locked
correctly
          ✓ locked balance correct
          ✓ the holder's balance correct
          ✓ teamAdvisorsTokensVault balance correct
          ✓ emits Approval event
          ✓ emits Transfer event
        all of the foundersTokensVault tokens can be locked
correctly
          ✓ locked balance correct
          ✓ the holder's balance correct
          ✓ teamAdvisorsTokensVault balance correct
          ✓ emits Approval event
          ✓ emits Transfer event
        all of the teamAdvisorsTokensVault tokens can be locked
correctly with a even number
          ✓ locked balance correct
          ✓ the holder's balance correct
          ✓ teamAdvisorsTokensVault balance correct
          ✓ emits Approval event of the 1st half part
          ✓ emits Transfer event of the 1st half part
      can't be locked over balance
```

```
         ✓ rewardPoolTokensVault (82ms)
         ✓ foundersTokensVault (77ms)
         ✓ teamAdvisorsTokensVault (102ms)
       can lock the tokens multi-twice
         ✓ rewardPoolTokensVault (152ms)
         ✓ foundersTokensVault (137ms)
         ✓ teamAdvisorsTokensVault (170ms)
     unlock tokens
       all of the tokens can be approved correctly
         ✓ saleTokensVault (86ms)
         ✓ marketingAirdropTokensVault (94ms)
       approved holder can transfer from the tokens
         saleTokensVault balance
           ✓ should set to zero (121ms)
           ✓ approved account balance should be transferred (118ms)
         marketingAirdropTokensVault balance
           ✓ marketingAirdropTokensVault balance should set to zero
(134ms)
           ✓ marketingAirdropTokensVault approved account balance
should be transferred (116ms)
       others can't transfer from the approved tokens
         ✓ saleTokensVault (105ms)
         ✓ marketingAirdropTokensVault (121ms)
     the basic ERC20 testing
       transfer
         when the recipient is not the zero address
           when the sender does not have enough balance
             ✓ reverts (40ms)
           when the sender has enough balance
             ✓ transfers the requested amount (71ms)
             ✓ emits a transfer event
         when the recipient is the zero address
           ✓ reverts (41ms)
       approve
         when the spender is not the zero address
           when the sender has enough balance
             ✓ emits an approval event
             when there was no approved amount before
               ✓ approves the requested amount (53ms)
             when the spender had an approved amount
               ✓ approves the requested amount and replaces the
previous one (118ms)
           when the sender does not have enough balance
```

```
             ✓ emits an approval event
             when there was no approved amount before
               ✓ approves the requested amount (50ms)
             when the spender had an approved amount
               ✓ approves the requested amount and replaces the
previous one (52ms)
        when the spender is the zero address
          ✓ reverts
      transfer from
        when the recipient is not the zero address
          when the spender has enough approved balance
            when the initial holder has enough balance
              ✓ transfers the requested amount (134ms)
              ✓ decreases the spender allowance (56ms)
              ✓ emits a transfer event (45ms)
              ✓ emits an approval event (60ms)
            when the initial holder does not have enough balance
              ✓ reverts (42ms)
          when the spender does not have enough approved balance
            when the initial holder has enough balance
              ✓ reverts (42ms)
            when the initial holder does not have enough balance
              ✓ reverts (38ms)
        when the recipient is the zero address
          ✓ reverts (45ms)
      decrease allowance
        when the spender is not the zero address
          when the sender has enough balance
            when there was no approved amount before
              ✓ reverts
            when the spender had an approved amount
              ✓ emits an approval event (39ms)
              ✓ decreases the spender allowance subtracting the
requested amount (51ms)
              ✓ sets the allowance to zero when all allowance is
removed (53ms)
              ✓ reverts when more than the full allowance is removed
(40ms)
          when the sender does not have enough balance
            when there was no approved amount before
              ✓ reverts
            when the spender had an approved amount
              ✓ emits an approval event
              ✓ decreases the spender allowance subtracting the
requested amount (75ms)
              ✓ sets the allowance to zero when all allowance is
removed (48ms)
              ✓ reverts when more than the full allowance is removed
        when the spender is the zero address
          ✓ reverts
      increase allowance
        when the spender is not the zero address
          when the sender has enough balance
            ✓ emits an approval event
            when there was no approved amount before
              ✓ approves the requested amount (52ms)
            when the spender had an approved amount
              ✓ increases the spender allowance adding the requested
amount (50ms)
          when the sender does not have enough balance
            ✓ emits an approval event
            when there was no approved amount before
              ✓ approves the requested amount (43ms)
            when the spender had an approved amount
              ✓ increases the spender allowance adding the requested
amount (47ms)
        when the spender is the zero address
          ✓ reverts
      _burn
        for a non null account
          ✓ rejects burning more than balance (39ms)
          for entire balance
            ✓ decrements totalSupply
            ✓ decrements holderMarketing balance
            ✓ emits Transfer event
          for less amount than balance
            ✓ decrements totalSupply
            ✓ decrements holderMarketing balance
            ✓ emits Transfer event
    the admin role allocation
      ✓ the contract creator is the beginning admin
      ✓ the others is not a admin
      ✓ the admin can add another admin (50ms)
      ✓ only admin can add another admin
      ✓ the admin can be renounced by himself (42ms)
    recoverERC20Tokens
```

```
      ✓ the contract holds a correct value of erc20 tokens
      ✓ the erc20 in the contract can be recovered by admin (65ms)
      ✓ the erc20 in the contract can't be recovered by non-admin
(39ms)
    the locked balance can be transferred time points
      ✓ locked balance can't be transferred before unlock time point
(45ms)
      ✓ locked balance can't be transferred over a half after 6
months(182 days) later (59ms)
      ✓ locked balance can be transferred a half after 6 months(182
days) later (57ms)
      ✓ locked balance can be transferred after 1 year(365 days)
later  (66ms)

  113 passing (36s)
```