SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: FIDOMETA
Date: May 17th, 2022

Document

Name	Smart Contract Code Review and Security Analysis Report for FIDOMETA.
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type of Contracts	ERC20 token
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://fidometa.io
Timeline	25.03.2022 - 17.05.2022
Changelog	12.04.2022 - Initial Review 21.04.2022 - Revision 17.05.2022 - Revision

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by FIDOMETA (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is smart contracts in the repository:
Repository: https://github.com/fidometa/Smart_Contract
Commit:
ad56b0ce678f77d162752dbb5d987d2f4a29fddf
66f43e8644a20ed2ebbc4db94d6b2dcc797ac1c7 (revision)
7bb7d6d237f8a6ff97470a8969f42a9bb78a6db5 (revision)
Technical Documentation: Yes
(https://github.com/fidometa/Smart_Contract/blob/main/FidometaDocument.pdf)
JS tests: Yes
Contract: FidoMeta.sol

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ EIP standards violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency
Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency ▪ Kill-Switch Mechanism

Executive Summary

The score measurement details can be found in the corresponding section of the methodology.

Documentation quality

The Customer provided functional but no technical requirements. The total Documentation Quality score is **5** out of **10**.

Code quality

The total CodeQuality score is **5** out of **10**. Code duplications. Superficial unit tests were provided.

Architecture quality

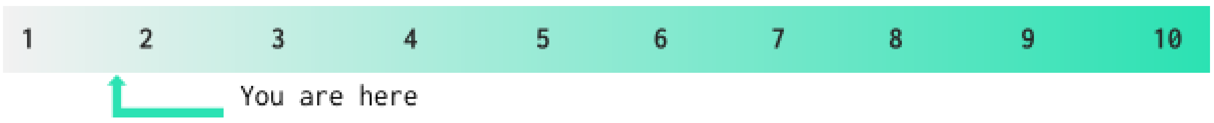
The architecture quality score is **8** out of **10**. Functions are overwhelmed with template code that could be moved to separate functions and be reused.

Security score

As a result of the audit, security engineers found **2** high, **2** medium, and **9** low severity issues. The security score is **0** out of **10**. All found issues are displayed in the "Issues overview" section.

Summary

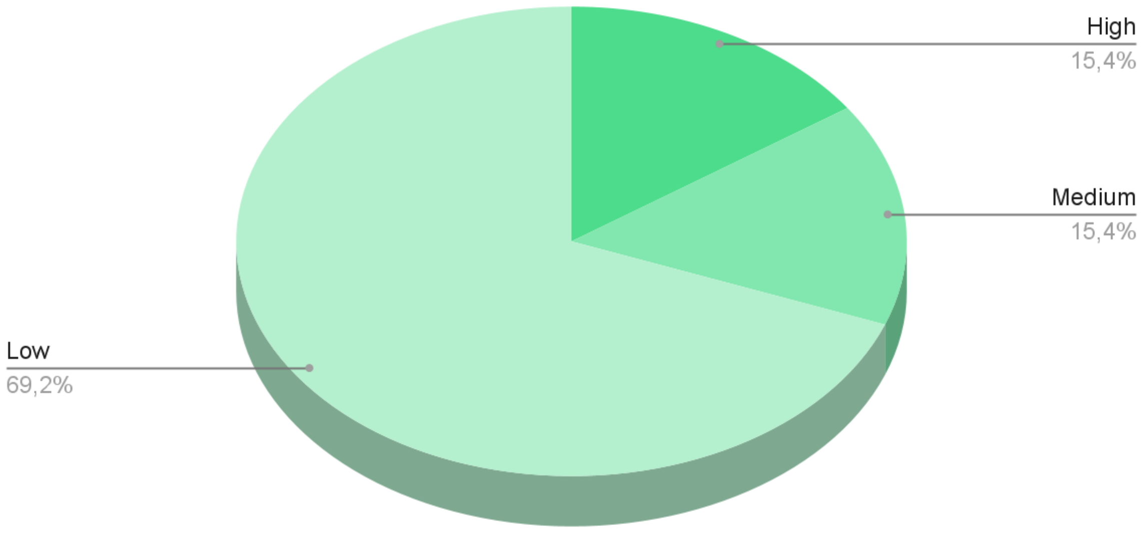
According to the assessment, the Customer's smart contract has the following score: **1.8**



Notices

1. The owner can exclude users from getting or paying any rewards.
2. The owner can change the max transfer amount and, in such a way, prevent transferring for everyone except him.

Graph 1. The distribution of vulnerabilities after the audit.



Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution

Findings

■■■■ Critical

1. **Missing functionality**

   The functions emit *Transfer* event but do not change any balances. This could lead to possible double-spending or reduction of users' rewards.

   Contract: FidoMeta.sol

   Functions: _burn, _mint, constructor

   Recommendation: review and fix the logic.

   Status: Fixed (Revised Commit: 66f43e8)

2. **Excluding from receiving rewards logic is corrupted**

   After excluding the user from rewards (*excludeFromReward*), the user should not get part of the community charge (and the user sees it so). However, his reflection balance updates along with his prime balance, and on including this user back (*includeInReward*), his balance is updated according to reflection tokens (*_rOwned*), and all hidden community rewards will be received.

   Contract: FidoMeta.sol

   Recommendation: review and fix the logic.

   Status: Fixed (Revised Commit: 66f43e8)

3. **Operation between different time units**

   block.timestamp may be bigger than *startTime* plus *initialLock* in seconds, but *initialLock* is given in days. This could lead to unexpected unlocking of funds or reverting the transaction according to internal reasons.

   Contract: FidoMeta.sol

   Functions: unlock

   Recommendation: multiply *initialLock* by 1 days.

   Status: Fixed (Revised Commit: 66f43e8)

4. **Significant impact on market price**

   The functions change the balances of all users at one moment. This could be used for changing market price significantly, predicting exchange rates, and as a result, looting untrained users.

   This may lead to total user discontent.

   Contract: FidoMeta.sol

   Functions: burn, mint

Recommendation: minting and burning functionality should change the balance of one user, provide this functionality or remove the functions.

Status: Fixed (Revised Commit: 7bb7d6d)

■■■ High

1. A new vesting period may not be created

locks mapping is not cleared on finishing of the vesting period, so the owner will not be able to set new vesting for the same user.

Contract: FidoMeta.sol

Functions: transferWithLock, unlock

Recommendation: delete *locks'* item on the last unlocking.

Status: Fixed (Revised Commit: 66f43e8)

2. No checks to prevent percentage overflow

Fees may become more than 100%. Every fee could be up to 100%, and their sum is 500%.

This could lead to reverting of all transactions.

Contract: FidoMeta.sol

Function: setCommunityCharge, setEcoSysFee, setSurcharge1, setSurcharge2, setSurcharge3

Recommendation: implement checks to prevent the scenario, validate that sum of all fees does not exceed 100%.

Status: Fixed (Revised Commit: 7bb7d6d)

3. User balance may exceed the total supply

If the owner excludes himself from rewards and then burns all tokens, the total supply will be lower than the sum of user balances.

This may lead to wrong rates calculation and unexpected behavior of taking charges logic.

Contract: FidoMeta.sol

Functions: burn

Recommendation: check this case and fix the corresponding logic.

Status: Fixed (Revised Commit: 7bb7d6d)

4. Integer Underflow

If a user unlocks vesting firstly in the first four months and then in the sixth month or later, the transaction will be reverted due to underflow.

This may lead to total unavailability to unlock vestings.

Contract: FidoMeta.sol

Functions: unlock

Recommendation: rewrite the code avoiding recalculations, and structure actions sequentially, processing this case.

Status: Fixed (Revised Commit: 7bb7d6d)

5. **Missing management of excluded accounts**

According to the current algorithm, all excluded accounts should be added to *_excluded* array and *_isExcludedFromReward* mapping, but in the function adding to *_excluded* is missed.

This could lead to wrong rates calculation, unexpected contract behavior, and possible double spending.

Contract: FidoMeta.sol

Function: setServiceWallet

Recommendation: carefully manage all excluded accounts.

Status: New

6. **A new vesting period may not be created**

locks mapping may not be cleared on finishing of the vesting period because of rounding division to the floor, so the owner will not be able to set new vesting for the same user. Example:

- transaction_1: admin do transferWithLock 4 tokens
- transaction_2 (on 4th vesting month): user does unlock
- transaction_3 (on 5th vesting month): user does unlock
- transaction_4: admin tries transferWithLock any amount
- result: transaction_4 will fail

This could lead to no possibility of creating new vesting for the user.

Contract: FidoMeta.sol

Functions: transferWithLock, unlock

Recommendation: delete *locks'* item on the last unlocking, check the case.

Status: New

■■ Medium

1. **Old wallets are not included back to changes and rewards**

These functions set new wallets and exclude them from charging but do not include back the old ones.

Contract: FidoMeta.sol

Functions: setEcoSysWallet, setSurcharge_1_Wallet, setSurcharge_2_Wallet, setSurcharge_3_Wallet

Recommendation: use excluding from charges and rewards only for service accounts.

Status: Fixed (Revised Commit: 66f43e8)

2. **Possible Gas exceeding**

   The contract can exceed the gas limit in long cycles.

   Contract: FidoMeta.sol

   Functions: includeInReward

   Recommendation: update logic to prevent situations when *_excludedFromReward* is too long.

   Status: Fixed (Revised Commit: 66f43e8)

3. **SafeMath is unneeded after 0.8.0**

   To save Gas and make code transparent, it is better not to use the SafeMath library. Solidity >= 0.8.0 version has built-in math checks.

   Contract: FidoMeta.sol

   Recommendation: get rid of using SafeMath.

   Status: Fixed (Revised Commit: 7bb7d6d)

4. **Initial minting event skipped**

   In the constructor, if all tokens are assigned to the owner, the *Transfer* event should be emitted.

   Contract: FidoMeta.sol

   Recommendation: emit the event.

   Status: Fixed (Revised Commit: 7bb7d6d)

5. **Burn wallet could be changed**

   The burn wallet (surcharge1) should never be changed. As documentation provides info that this wallet is only for unused fees, the owner should not have the ability to change it and claim the next fees that should be burned. Moreover, to save Gas, it is better to burn assets by decreasing user balance and changing the supply of tokens.

   Contract: FidoMeta.sol

   Recommendation: implement burning as decreasing of *_rTotal* and *_tTotal* or mention in documentation that this charge may change its purpose in future.

   Status: Fixed (Revised Commit: 7bb7d6d)

6. **Vested assets should be unlocked automatically**

The function should calculate the actual withdrawable amount for the user. To improve user experience, it is better to implement automatic unlock of the corresponding vesting if assets are ready for unlocking.

Contract: FidoMeta.sol

Function: _transfer

Recommendation: check if the users may unlock tokens and unlock them.

Status: Mitigated (with customer notice)

7. **Possible Gas exceeding**

   The contract can exceed the gas limit in long cycles.

   As accounts could be only excluded, it is better to update the logic of rates calculation and separate excluded accounts from r-space. In such a way, a significant amount of Gas could be saved at recalculation of total supply and actual rates.

   Contract: FidoMeta.sol

   Function: _getCurrentSupply

   Recommendation: update logic to prevent situations when *_excluded* array is too long, keep actual information about supply for not recalculating it each time.

   Status: Acknowledged

8. **Violation of ERC-20 standard**

   transferFrom should be possible if the amount is allowed by the owner of the assets. If a user tries to transfer all the allowance size, it fails.

   Contract: FidoMeta.sol

   Function: transferFrom

   Recommendation: fix this case.

   Status: New

Low

1. **Unused library**

   Using the *Address* library is not needed.

   Contract: FidoMeta.sol

   Recommendation: remove the library.

   Status: Fixed (Revised Commit: 66f43e8)

2. **Modification of well-known contract**

Imported or copy-pasted contracts (like *SafeMath*, *Context*, *Ownable*, etc.) should not be modified to keep the code clear. *Ownable* contract was modified with a freezing feature and several unused variables.

Contract: FidoMeta.sol

Recommendation: move this feature to the *FidoMeta* contract body.

Status: Fixed (Revised Commit: 66f43e8)

3. **Floating pragma**

The contracts use floating pragma ^0.8.11.

Contract: FidoMeta.sol

Recommendation: consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

Status: Fixed (Revised Commit: 66f43e8)

4. **Violation of IERC-20 standard**

According to the IERC-20 standard, the functions should be external to save Gas.

Contract: FidoMeta.sol

Function: transfer, allowance, approve, transferFrom

Recommendation: these functions should be declared as external.

Status: Fixed (Revised Commit: 66f43e8)

5. **Unused functions**

Unused code overloads the code and takes additional Gas for deploying. It is better to use the *public* keyword in the struct definition for automatic generation of view methods.

Contract: FidoMeta.sol

Functions: isExcludedFromCommunityCharge, isExcludedFromEcoSysFee, isExcludedFromSurcharge1, isExcludedFromSurcharge2, isExcludedFromSurcharge3, viewSurcharge_1_Wallet, viewSurcharge_2_Wallet, viewSurcharge_3_Wallet, viewEcoSysWallet, isExcludedFromReward, totalCommunityCharge, name, symbol, decimals

Recommendation: remove these functions and declare corresponding structs as public.

Status: Fixed (Revised Commit: 66f43e8)

6. **Unused functions**

Unused code overloads the code and takes additional Gas for deploying.

Contract: FidoMeta.sol

Function: reflectionFromToken

Recommendation: remove this function.

Status: Fixed (Revised Commit: 66f43e8)

7. **Similar functions**

   Functions that do mostly the same things may be united.

   Contract: FidoMeta.sol

   Function: setCommunityCharge, setEcoSysFee, setSurcharge1, setSurcharge2, setSurcharge3

   Recommendation: unite these functions into one that sets all charges.

   Status: Fixed (Revised Commit: 66f43e8)

8. **Similar functions**

   Functions that do mostly the same things may be united.

   Contract: FidoMeta.sol

   Function: excludeFromCommunityCharge, includeInCommunityCharge, excludedFromEcoSysFee, includeInEcoSysFee, excludedFromSurcharge1, includeInSurcharge1, excludedFromSurcharge2, includeInSurcharge2, excludedFromSurcharge3, includeInSurcharge3

   Recommendation: unite these functions in one that sets all includes/excludes.

   Status: Fixed (Revised Commit: 66f43e8)

9. **Template code**

   It is better to create a *modifier* to exclude the account from all charges.

   Contract: FidoMeta.sol

   Function: setEcoSysWallet, setSurcharge_1_Wallet, setSurcharge_2_Wallet, setSurcharge_3_Wallet

   Recommendation: rid of template code.

   Status: Fixed (Revised Commit: 66f43e8)

10. **Template code**

    The functions do not compose calculations and are used only in one place, so code separation is useless.

    Contract: FidoMeta.sol

    Function: calculateCommunityCharge, calculateSurcharge1, calculateSurcharge2

    Recommendation: the functions could be deleted or replaced with a function that receives the fee and amount and returns the calculated value.

Status: Fixed (Revised Commit: 66f43e8)

11. **Template code**

It is better to create a *modifier* to disable necessary changes.

Contract: FidoMeta.sol

Function: removeEcosysFee, removeSurcharge1, removeSurcharge2, removeSurcharge3, removeCommunityCharge, restoreCommunityCharge, restoreEcosysFee, restoreSurcharge1, restoreSurcharge2, restoreSurcharge3

Recommendation: remove these functions and implement a modifier that unites all the logic.

Status: Fixed (Revised Commit: 66f43e8)

12. **Similar functions**

Functions that do mostly the same things may be united.

Contract: FidoMeta.sol

Function: _takeEcoSysCharge, _takeSurcharge1, _takeSurcharge2, _takeSurcharge3

Recommendation: unite these functions into one that takes all charges.

Status: Fixed (Revised Commit: 66f43e8)

13. **Functions that can be declared as external**

To save Gas, public functions that are never called in the contract should be declared as *external*.

Contract: FidoMeta.sol

Function: setCommunityCharge, setEcoSysFee, setSurcharge1, setSurcharge2, setSurcharge3, setEcoSysWallet, setSurcharge_1_Wallet, setSurcharge_2_Wallet, setSurcharge_3_Wallet, burn, mint, name, symbol, decimals, increaseAllowance, decreaseAllowance, excludeFromReward, excludeFromCommunityCharge, includeInCommunityCharge, excludedFromEcoSysFee, includeInEcoSysFee,

Recommendation: aforementioned should be declared as *external*.

Status: Fixed (Revised Commit: 66f43e8)

14. **Requires that would never revert transaction**

uint256 will always be *>= 0*, the requires are useless

Contract: FidoMeta.sol

Functions: transferWithLock

Recommendation: review what the requirements should prevent and fix them.

Status: Fixed (Revised Commit: 7bb7d6d)

15. **Modification of well-known contract**

    Imported or copy-pasted contracts (like *SafeMath*, *Context*, *Ownable*, etc.) should not be modified to keep the code clear. *Ownable* contract is modified with a *_lockTime* variable that is never used.

    Contract: FidoMeta.sol

    Recommendation: remove unused variable.

    Status: Fixed (Revised Commit: 7bb7d6d)

16. **Local variable shadowing**

    Local variable shadowing may lead to unexpected code behavior in future development.

    Contract: FidoMeta.sol

    Functions: _approve:(owner)

    Contract: BreederDaoTokenLockWallet.sol

    Recommendation: rename the local variables that shadow other components.

    Status: Fixed (Revised Commit: 7bb7d6d)

17. **State variables that could be declared constant**

    To save Gas, constant state variables should be declared *constant*.

    Contract: FidoMeta.sol

    Recommendation: add the *constant* attribute to state variables that never change.

    Status: Fixed (Revised Commit: 7bb7d6d)

18. **Functions that can be declared as external**

    To save Gas, public functions that are never called in the contract should be declared as *external*.

    Contract: FidoMeta.sol

    Function: excludeFromReward

    Recommendation: use the *external* attribute for functions never called from the contract.

    Status: Fixed (Revised Commit: 7bb7d6d)

19. **Template variables**

    The code should not be overwhelmed with copy-pasted variables, functions, and other structures. Variables that mean mostly the same and are used in the same places may be united into a structure.

Contract: FidoMeta.sol

Variables: _community_charge, _ecoSysFee, _surcharge1, _surcharge2, _surcharge3

Variables: _previousCommunityCharge, _previousEcoSysFee, _previousSurcharge1, _previousSurcharge2, _previousSurcharge3

Variables: _ecoSysWallet, _surcharge_1_Wallet, _surcharge_2_Wallet, _surcharge_3_Wallet

Variables: takeCommunityCharge, takeEcosysFee, takeSurcharge1, takeSurcharge2, takeSurcharge3

Recommendation: unite the variables into structures.

Status: Acknowledged

20. Template code

To save Gas costly reassignment operations should not be used. Changing and reverting values for special calls is not a good way to solve this problem. It is better to process special arguments to the final destination where the changes are needed.

Contract: FidoMeta.sol

Function: _tokenTransfer

Recommendation: get rid of template code and *previous* value variables, structure actions sequentially, and process all options on the corresponding level of abstractions.

Status: Acknowledged

21. Overwhelmed code

Unneeded reassignment is provided. *take* variables could be defined already with actual values. Moreover, as they are unused there directly, it is better to use them in *_getTValues* function where their sense is needed.

Contract: FidoMeta.sol

Function: _transfer

Recommendation: update the code and use values exactly where they are needed.

Status: Acknowledged

22. Repeated code

The functions include the same operation (updating r-space balance of the sender) that could be moved to *_doTransfer* function.

Contract: FidoMeta.sol

Functions: _transferFromExcluded, _transferToExcluded, _transferStandard, _transferBothExcluded

Recommendation: move the operation to *_doTransfer* function.

Status: New

23. **Unnecessary code**

Any unnecessary code should be removed in order to save Gas. There could be only 4 types of transfer, so the *else* case is not needed.

Contract: FidoMeta.sol

Function: _tokenTransfer

Recommendation: remove unnecessary code.

Status: New

24. **Functions that can be declared as external**

To save Gas, public functions that are never called in the contract should be declared as *external*.

Contract: FidoMeta.sol

Function: totalSupply

Recommendation: aforementioned should be declared as *external*.

Status: New

25. **Unnecessary checks**

Any unnecessary code should be removed in order to save Gas. Checks for each fee separately are not needed because the check for the sum is stronger.

Contract: FidoMeta.sol

Function: setCharges

Recommendation: remove unnecessary code.

Status: New

26. **Modification of well-known contract**

Imported or copy-pasted contracts (like *SafeMath*, *Context*, *Ownable*, etc.) should not be modified to keep the code clear. *Ownable* contract was modified with a *_previousOwner* unused variable.

Contract: FidoMeta.sol

Recommendation: import the contract from public repositories or copy it from a trusted source and do not modify it anymore.

Status: New

27. **Hardcoded values**

Hardcoded values make development harder and the code less readable. *decimals* value should be defined in one place, and the constant should be used to introduce it.

Contract: FidoMeta.sol

Function: _getTValues

Variables: _ecoSysFee, _surcharge1

Recommendation: use *decimals +(-) value* to introduce needed value dependent on *decimals* value.

Status: New

Recommendations

1. Rename variables or write explaining comments to make it easy to understand what they are responsible for.

   Contracts: FidoMeta.sol

2. Provide technical documentation that answers how to build the project, run tests for it, etc.

   Contracts: FidoMeta.sol

3. Provide conscious abstraction levels and restructure the code sequentially, disposing of actions and checks.

   Contracts: FidoMeta.sol

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit cannot guarantee the explicit security of the audited smart contracts.