HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Millix Foundation
Date: March 1, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Millix Foundation
Approved By	Noah Jelich | Lead Solidity SC Auditor at Hacken OU
Type	ERC20 token
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://millix.org/
Changelog	14.02.2023 - Initial Review 1.03.2023 - Second Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Introduction **4**

Scope **4**

Severity Definitions **5**

Executive Summary **6**

Checked Items **7**

System Overview **10**

Findings **11**

Critical 11

High 11

H01. Requirements Violation **11**

H02. Highly Permissive Role Access **11**

Medium 11

M01. Contradiction **11**

M02. Tautology **12**

M03. Best Practice Violation **12**

M04. Missing Events **12**

M05. Best Practice Violation **12**

Low 13

L01. Floating Pragma **13**

L02. Missing Empty String Check **13**

Disclaimers **14**

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Millix Foundation (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:

Initial review scope

Repository	https://github.com/millix/millix-bridge-contract
Commit	76f8469c97c0052a7217cb29e42d80dbc4806e52
Whitepaper	Link
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/WrappedMillix.sol SHA3: 80156cb39d46faff08ccfddb654bf87bebabfd4a28d36229bcbcd7572dbc046d

Second review scope

Repository	https://github.com/millix/millix-bridge-contract
Commit	76a2dc84776e9881423bde9033f6f03fad385518
Whitepaper	Link
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/interfaces/IMillixBridge.sol SHA3: c617e2c1f1bef7ed99493a2ba63f30d78659e43dbeeb0ab2739ee232b97ff4c2 File: ./contracts/WrappedMillix.sol SHA3: e4fd1f9a2220126802ec4d5698288dac573b0e804f4723dbca21c7db558760cb

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation by external or internal actors.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor Gas optimization. These issues won't have a significant impact on code execution but affect code quality

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

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

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are provided.
- Technical description is provided.

Code quality

The total Code Quality score is **10** out of **10**.

- The development environment is configured.
- The code follows the Solidity style guides.

Test coverage

Code coverage of the project is **66.67%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Interactions by several users are not tested thoroughly.
- Negative cases coverage is missed.

Security score

As a result of the audit, the code does not contain issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **10.0**.

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
14 February 2023	2	5	2	0
1 March 2023	0	0	0	0

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Not Relevant
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155 EIP-712	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of Unused Variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP Standards Violation	EIP	EIP standards should not be violated.	Passed
Assets Integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed
User Balances Manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply Manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed

Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Style Guide Violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, which may be changed in the future.	Passed

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

System Overview

Millix is a ERC20 token with the following contract:

- *WrappedMillix* — simple ERC-20 token with additional features, including pause and resume, minting and burning, and vesting restrictions.
  It has the following attributes:
  - Name: WrappedMillix
  - Symbol: WMLX
  - Decimals: 1
  - Total supply: 9,000,000,000,000,000 WMLX (nine quadrillion) tokens.

Privileged roles

- The owner of the *WrappedMillix* contract can pause, unpause the contract, and stop transfer, burn, mint processes for specific address or for all addresses.

Findings

Critical

No critical severity issues were found.

High

H01. Requirements Violation

It is possible that the user may inadvertently pay a higher burn fee than the current *_burnFees* amount. This can occur as a result of the use of the *>=* operator.

This can lead to users to pay more fees than required.

Path: ./contracts/WrappedMillix.sol : unwrap()

Recommendation: Either return the excessive amount to the user or use a strict equals for fees.

Status: Fixed (Revised commit: 76a2dc84776e9881423bde9033f6f03fad385518)

H02. Highly Permissive Role Access

The owner can stop the token transfers at any time via the *setVestingState* function for any specific address or all addresses via *pause* function. Owners should not have an access to funds that belongs to users.

Path: ./contracts/WrappedMillix.sol : pause(), setVestingState()

Recommendation: In the public documentation, mention the access privileges associated with the owner role for the users.

Status: Fixed (Revised commit: 76a2dc84776e9881423bde9033f6f03fad385518)

Medium

M01. Contradiction

The functions NatSpecs, the *param* statement is not followed by function parameter name.

Path: ./contracts/WrappedMillix.sol : mint(), setBurnFees(), burnFees(), isVested(), setVestingState(), unwrap()

Recommendation: According to the Solidity documents, after the param statement function parameter name should be followed.

Status: Fixed (Revised commit: 76a2dc84776e9881423bde9033f6f03fad385518)

M02. Tautology

The *setBurnFees()* function has a requirement that contains a tautology. Specifically, the requirement that *fees >= 0* is in conflict with the definition of the *fees* variable as a *uint*. By definition, variables of type uint are always equal to or greater than zero.

Path: ./contracts/WrappedMillix.sol : setBurnFees()

Recommendation: Remove related *require* statement.

Status: Fixed (Revised commit: 76a2dc84776e9881423bde9033f6f03fad385518)

M03. Best Practice Violation

The built-in transfer and send functions process hard-coded amount of Gas. In case of receiver is a contract with receive or fallback function, the transfer may fail due to the “out of Gas” exception.

Path: ./contracts/WrappedMillix.sol : unwrap()

Recommendation: Replace transfer and send functions with call or provide special mechanism for interacting with a smart contract.

Status: Fixed (Revised commit: 76a2dc84776e9881423bde9033f6f03fad385518)

M04. Missing Events

The functions do not emit events on change of important values.

Path: ./contracts/WrappedMillix.sol : setBurnFees(), setVestingState()

Recommendation: Emit events on critical state changes.

Status: Fixed (Revised commit: 76a2dc84776e9881423bde9033f6f03fad385518)

M05. Best Practice Violation

The *unwrap()* function deducts a **fixed** fee from the user, regardless of the token amount of burn.

Path: ./contracts/WrappedMillix.sol : unwrap()

Recommendation: Either explain the functionality and inform the users in the public documentation or implement different fee logic which accounts percentage of the amount to be burned.

Status: Fixed (Revised commit: 76a2dc84776e9881423bde9033f6f03fad385518)

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Low

L01. Floating Pragma

The project uses floating pragmas ^0.8.9

Path: ./contracts/WrappedMillix.sol

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

Status: Fixed (Revised commit: 76a2dc84776e9881423bde9033f6f03fad385518)

L02. Missing Empty String Check

The mint functions emit events according to the input parameters, it can be given as empty. This may lead to empty event emitting, which can lead to unnecessary Gas consumption.

Path: ./contracts/WrappedMillix.sol: mint(), unwrap()

Recommendation: Implement empty string checks.

Status: Fixed (Revised commit: 76a2dc84776e9881423bde9033f6f03fad385518)

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing 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 report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and 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.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

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 Consultant cannot guarantee the explicit security of the audited smart contracts.