SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Etherlin

Date: Nov 24th, 2022

This document may contain confidential information about IT systems and the intellectual property of the Customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities are fixed — upon a decision of the Customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for Etherlin
Approved By	Panner S Vignesh | SC Audits Department Head at Maco Inc
Type	BEP20 token; Vesting
Platform	BNB Smart Chain
Language	Solidity
Methods	Manual Review, Automated Review, Architecture review
Website	https://ethrlin.com/
Timeline	13.11.2022 – 24.11.2022
Changelog	17.11.2022 – Initial Review 24.11.2022 – Second Review

Table of contents

Introduction

Maco Inc (Consultant) was contracted by Etherlin (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/Ethrlin
Commit:
0xE4EF56e4C87eeef60ec5Ae5EFb5eaA99548e96AF
Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
Link

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/Ethrlin.sol
SHA3: 1c0f63b2f02e4eef2b97ca72ca8078a6a40b03c318f8b8dbf12b8de675a6c864

File: ./contracts/Ethrlin.sol
SHA3: 36aec4934ff56f367a331372a895641fbc82e1da20edf3faff3a668b1c330b91

Second review scope

Repository:
https://github.com/Ethrlin
Commit:
0xE4EF56e4C87eeef60ec5Ae5EFb5eaA99548e96AF
Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
Link

Integration and Unit Tests: Yes
Deployed Contracts Addresses:
https://bscscan.com/token/0xe4ef56e4c87eeef60ec5ae5efb5eaa99548e96af#code
Contracts:
File: ./contracts/Ethrlin.sol
SHA3: d31f3bf3ed516554a0fd34d457405182068f7ec44cb8f1daf5715df9409fe034

File: ./contracts/TokenVesting.sol
SHA3: 14402732e859e3736c6b9a2536a0b5d6a2ca62fb7c6f443e69c56f467f6cc8a7

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.

Executive Summary

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

Documentation quality

The total Documentation Quality score is 10 out of 10. The whitepaper is provided, and vesting periods are well-described.

Code quality

The total CodeQuality score is 8 out of 10. Deployment and user interactions are covered with tests. Missing some validations, a redundant cast exists.

Architecture quality

The architecture quality score is 10 out of 10. Code follows the single responsibility principle, is well structured, and is well commented.

Security score

As a result of the audit, the code contains 1 medium severity issue. The security score is 9 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: 9.1.

Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are 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.	Passed
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.	Passed
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless it is required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization	SWC-115	tx.origin should not be used for	Passed

through tx.origin		authorization.	
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	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifier should always be used.	Passed
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.	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	Passed

		fails due to the block Gas limit.	
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
Tests Coverage	Custom	The code should be covered with unit tests. Test coverage should be 100%, 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, that may be changed in the future.	Passed

System Overview

Etherlin is a cryptocurrency-powered By Its own Exchange Kryptrx. ETHERLIN, a BEP-20 token and abbreviated as "ETL," will be the official token of the exchange platform performing services and utilities which will impact the token holder's pockets.

The system consists of the following contracts:

- *EtherlinToken* — a simple BEP 20 token that mints all initial supply to a deployer. Additional minting is not allowed.
  It has the following attributes:
  - Name: Etherlin Token
  - Symbol: ETL
  - Decimals: 14
  - Total supply: 1b tokens.
- *TokenVesting* — a contract that is responsible for the control of vesting periods and unlocks.

Privileged roles

- The owner of the *EtherlinToken* contract can pause all token transactions.
- The owner of the *TokenVesting* contract can revoke the vesting schedule for a specific identifier.
- The owner of the *TokenVesting* contract can withdraw excess tokens from the vesting pool.
- The owner of the *TokenVesting* contract can create a vesting schedule for a beneficiary.
- The owner of the *TokenVesting* contract can withdraw ERC20 tokens from the contract.

Risks

- In case of an admin keys leak, an attacker can lock all token transactions or change vestings.
- Vesting addresses and periods will be configured after the contract deployment. It is impossible to verify that all vestings would be declared as per provided documentation.

Findings

Critical

No critical severity issues were found.

High

No high severity issues were found.

Medium

1. Unnecessary SafeMath usage.

   Solidity >= 0.8.0 provides errors for buffer overflow and underflow. No need to use SafeMath anymore.

   Contracts: TokenVesting.sol

   Recommendation: Do not use SafeMath.

   Status: Fixed (899eb5e454c1b2863f5a31d423b3bc3f5bbe0a90)

2. Stucked funds in the contract.

   The contract contains payable functions to receive native tokens, but there are no methods to withdraw them from the contract. As a result
   - all sent native tokens to the contract would be stuck.

   Contracts: TokenVesting.sol

   Recommendation: Remove *receive* and *fallback* functions to forbid accidental native coin transfers to the contract.

   Status: Reported

Low

1. Floating Pragma.

   Contracts should be deployed with the same compiler version and flags that have been tested thoroughly. Locking the Pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

   Contracts: EthrlinToken.sol

   Recommendation: Use a fixed version of the compiler (^ symbol should be removed from Pragma). Consider using the same compiler version for all contracts.

   Status: Fixed (899eb5e454c1b2863f5a31d423b3bc3f5bbe0a90)

2. The public function could be declared external.

   Public functions that are never called by the contract should be declared external to save Gas.

Contracts: EthrlinToken.sol, TokenVesting.sol

Functions: pause, un pause, create Vesting Schedule, revoke, withdraw, compute Release Amount, get Withdrawable Amount, compute Next Vesting Scheduled For Holder, get Last Vesting Schedule For Holder

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

Status: Fixed (899eb5e454c1b2863f5a31d423b3bc3f5bbe0a90)

3. Zero address is allowed.

The new address for the service signer does not check if it is a zero address, which could be sent as a default value.

Contracts: TokenVesting.sol

Functions: createVestingSchedule

Recommendation: Add check for zero address for *_beneficiary*

Status: Fixed (899eb5e454c1b2863f5a31d423b3bc3f5bbe0a90)

4. Redundant payable address cast.

Release function casts beneficiary address to payable, which is redundant, as contract transfer ERC20 token, not the network native token.

Contracts: TokenVesting.sol

Functions: release

Recommendation: Remove *payable* cast before the transfer.

Status: Fixed (899eb5e454c1b2863f5a31d423b3bc3f5bbe0a90)

5. Missing vesting validation.

createVestingSchedule function does not validate if the cliff period is less than the vesting duration. If the cliff is bigger than the duration - nothing would be released to the beneficiary before the cliff is ended.

Recommendation: Validate cliff duration when creating a vesting schedule.

Status: Fixed (899eb5e454c1b2863f5a31d423b3bc3f5bbe0a90)

Disclaimers

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.