QuillAudits

Audit Report, June, 2024

For

EVM

Table of Content

Executive Summary

Project Name	EVM.INK
Overview	ERC20S is an extension of ERC20 token with the functionality of adding and removing liquidity with eth.
Timeline	12th June 2024 - 21st June 2024
Updated Code Received	22nd June 2024
Second Review	24th June 2024
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Audit Scope	This audit aimed to analyze the ERC20S.sol Codebase for quality, security, and correctness. 1. ERC20S.sol
Source code	https://github.com/undefy-io/erc20s/blob/main/contracts/Attack.sol
Branch	Main
Commit Hash	b6ce3c181f7ca1414b5e6f9f8708d1337a026d08
Fixed in	b1597ec68b3551d466b30d4f3aa3957441321631

Number of Issues per Severity

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	1	1	1	1

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- Transaction-Ordering Dependence
- ERC20 API violation
- Compiler version not fixed
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level
- Redundant fallback function

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Hardhat, Foundry.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

High Severity Issues

1. Wrong input parameters passed in _addLiquidity function will result in incorrect token amount calculation

Path

ERC20S.sol

Function

_addLiquidity():

https://github.com/undefy-io/erc20s/blob/b6ce3c181f7ca1414b5e6f9f8708d1337a026d08/contracts/ERC20S.sol#L170-L206

Description

The addLiquidity() function internally calls _addLiquidity() which uses quote() function to calculate the optimal amount of tokens to be added.

The function quote(uint amountA, uint reserveA, uint reserveB)
calculates the output amount using this formula

amountB = (amountA * reserveB) / reserveA;

This means that if we want to calculate tokenAmountOptimal (amount B) we have to pass these input parameters in the exact order

- nativeAmountDesired(amount A)
- _nativeReserve(reserve A)
- _tokenReserve(reserve B)

Similarly, to calculate nativeAmountOptimal (amount B) we have to pass these input parameters in the exact order

- tokenAmountDesired(amount A),
- _tokenReserve(reserve A),
- _nativeResever(amount B)

However, the input parameters passed are in the wrong order here.

```
uint nativeAmountOptimal = quote(
    tokenAmountDesired,
    _nativeReserve,
    _tokenReserve
);
```

https://github.com/undefy-io/erc20s/blob/b6ce3c181f7ca1414b5e6f9f8708d1337a026d08/contracts/ERC20S.sol#L195-L199

As a result, nativeAmountOptimal will be wrongly calculated, resulting in imbalanced liquidity.

Since this logic is identical to Uniswap, we can also compare how uniswap handles input parameters which verifies the our statement.

https://github.com/Uniswap/v2-periphery/blob/0335e8f7e1bd1e8d8329fd300aea2ef2f36dd19f/contracts/UniswapV2Router01.sol#L46-L51

Recommendation

Change the order of input parameters during the calculation of nativeAmountOptimal

Status

Resolved

Medium Severity Issues

2. Extra eth during liquidity addition was never returned

Path

ERC20S.sol

Function

addLiquidity:

https://github.com/undefy-io/erc20s/blob/b6ce3c181f7ca1414b5e6f9f8708d1337a026d08/contracts/ERC20S.sol#L208-L246

Description

The user needs to pass msg.value with tokens in order to add liquidity to the pair. However, any extra msg.value was never returned to the user after the execution.

This is problematic since users often provide more eth than required to compensate for volatile gas prices.

Recommendation

Add extra logic to check if msg.value > amountNative and transfer back extra msg.value if this is true.

Status

Resolved

Low Severity Issues

3. Lack of address(0) check while minting can result in loss of funds

Files

ERC20S.sol

Function

mint():

https://github.com/undefy-io/erc20s/blob/b6ce3c181f7ca1414b5e6f9f8708d1337a026d08/contracts/ERC20S.sol#L293-L320

Description

There is no input validation to ensure that minting does not happen on address(0).

Since mint is used in crucial functions such as addLiquidity(), this can cause user funds loss if they accidentally pass null address as an input parameter.

Recommendation

Add a zero address check.

Status

Resolved

Informational Issues

4. Lack of address(0) check while minting can result in loss of funds

Path

ERC20S.sol

Function

mint()

https://github.com/undefy-io/erc20s/blob/b6ce3c181f7ca1414b5e6f9f8708d1337a026d08/contracts/ERC20S.sol#L475-L477

Description

Owner has the ability to mint unlimited amounts of tokens to any address which can become a central point of faliure if compromised or if owner role is untrusted.

Recommendation

Ensure that owner address is a multisig wallet if trusted.

Status

Resolved

Functional Tests

Some of the tests performed are mentioned below:

- Should add liquidity to the pair
- Should remove liquidity from the pair
- Should not add and remove liquidity past the deadline
- Should correctly swap based on the given inputs
- Should not be able to mint if not owner
- Should correctly update reserves after adding and removing liquidity

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the EVM.INK contract. We performed our audit according to the procedure described above.

Some issues of High, Medium, Low severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.In the End, EVM.INK Tea

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in EVM.INK smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of EVM.INK smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the EVM.INK to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
June, 2024

For

EVM

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com