QuillAudits

Audit Report
June, 2021

UniFarm
by OroPocket

Contents

Audit Details and Target

1. Contract

https://github.com/themohitmadan/unifarm_timelock

Commit Id - 9fa2bc5d7e107ee7aa7336663ee2861e5a366a4c

2. Audit Target

- To find the security bugs and issues regarding security, potential risks, and critical bugs.
- Check gas optimisation and check gas consumption.
- Check function reusability and code optimisation.
- Test the limit for token transfer and check the accuracy in decimals.
- Check the functions and naming conventions.
- Check the code for proper checks before every function call.
- Event trigger checks for security and logs.
- Checks for constant and function visibility and dependencies.
- Validate the standard functions and checks.
- Check the business logic and correct implementation.
- Automated script testing for multiple use cases including transfers, including values and multi transfer check.
- Automated script testing to check the validations and limit tests before every function call.
- Check the use of data type and storage optimisation.
- Calculation checks for different use cases based on the transaction, calculations and reflected values.

Functions list and audit details

Major Functions

- lockToken()
- safeWithdraw()
- updateUnlockTime()
- getNow()

Overview of The Contract

UniFarm_TimeLock contract is designed and developed to perform staking based on the block time and using standard locking functionality of open zeppelin. The contract has all major standard functions like safe math, erc20 safe, Address, context, and ownable. Users can send the token, and this can be locked based on the time and the values added by the contract address.

Tokenomics

As per the information provided, the tokens generated will be initially transferred to the contract owner, and then further division will be done based on the business logic of the application. Tokens cannot be directly purchased from the smart contract, so there will be an additional or third-party platform that will help users to purchase the tokens. Tokens can be held, transferred, and delegated freely. Tokens are generated based on the fixed flow in the supply, which can be checked by total supply.

Scope of Audit

The scope of this audit was to analyze UniFarm_TimeLock smart contracts codebase for quality, security, and correctness.

Checked Vulnerabilities

The smart contract is scanned and checked for multiple types of possible bugs and issues. This mainly focuses on issues regarding security, attacks, mathematical errors, logical and business logic issues. Here are some of the commonly known vulnerabilities that are considered:

- TimeStamp dependencies.
- Variable and overflow
- Calculations and checks
- SHA values checks
- Vulnerabilities check for use case
- Standard function checks
- Checks for functions and required checks
- Gas optimisations and utilisation
- Check for token values after transfer
- Proper value updates for decimals
- Array checks
- Safemath checks
- Variable visibility and checks
- Error handling and crash issues
- Code length and function failure check
- Check for negative cases
- D-DOS attacks
- Comments and relevance
- Address hardcoded
- Modifiers check
- Library function use check
- Throw and inline assembly functions
- Locking and unlocking (if any)
- Ownable functions and transfer ownership
- checksArray and integer overflow possibility checks
- Revert or Rollback transactions check

Techniques and Methods

- Manual testing for each and every test cases for all functions.
- Running the functions, getting the outputs and verifying manually for multiple test cases.
- Automated script to check the values and functions based on automated test cases written in JS frameworks.
- Checking standard function and check compatibility with multiple wallets and platforms
- Checks with negative and positive test cases.
- Checks for multiple transactions at the same time and checks d-dos attacks.
- Validating multiple addresses for transactions and validating the results in managed excel.
- Get the details of failed and success cases and compare them with the expected output.
- Verifying gas usage and consumption and comparing with other standard token platforms and optimizing the results.
- Validate the transactions before sending and test the possibilities of attacks.

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.
SmartCheck.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability 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 automated analysis were manually verified.

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

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

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Ganache, Solhint, Mythril, Slither, SmartCheck.

Issue Categories

High severity issues

Issues that must be fixed before deployment else they can create major issues.

Medium level severity issues

These issues will not create major issues in working but affect the performance of the smart contract.

Low level severity issues

These issues are more suggestions that should be implemented to refine the code in terms of gas, fees, speed and code accuracy

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.

Number of issues per severity

	High	Medium	Low	Informational
Open	0	0	0	0
Closed	0	0	2	4

Issues Found – Code Review / Manual Testing

High severity issues

No issues found under this category.

Medium severity issues

No issues found under this category.

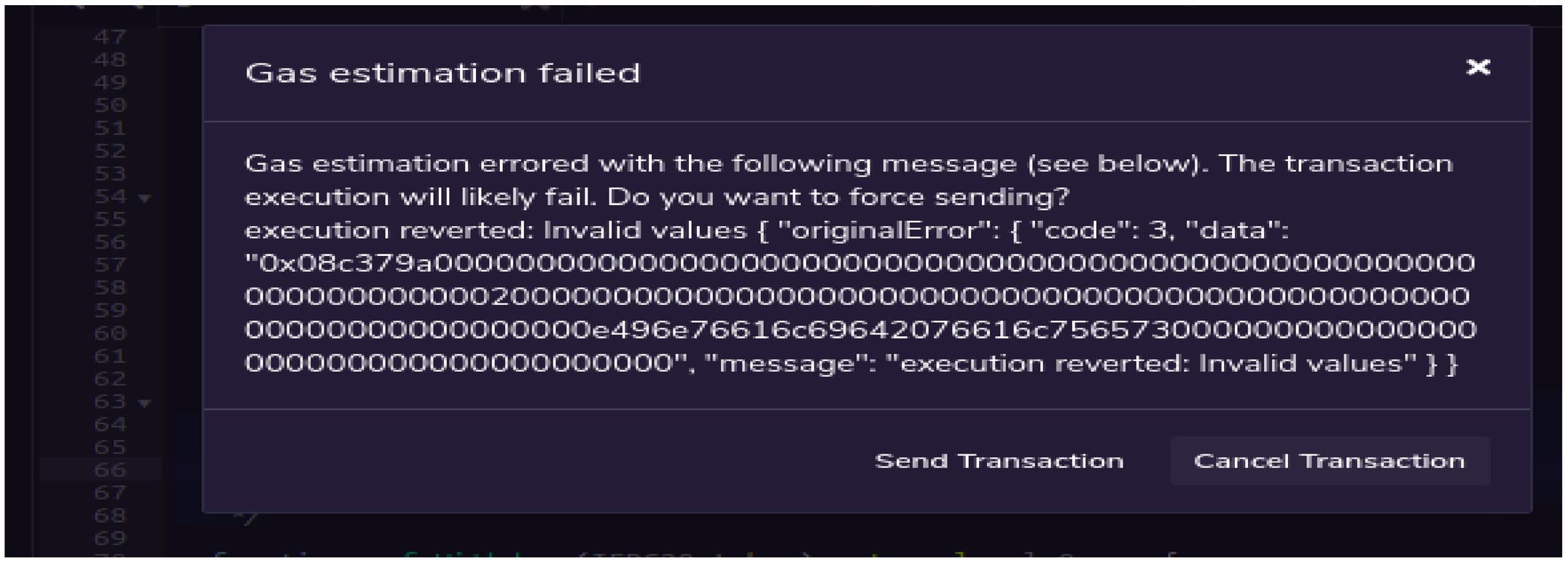
Low level severity issues

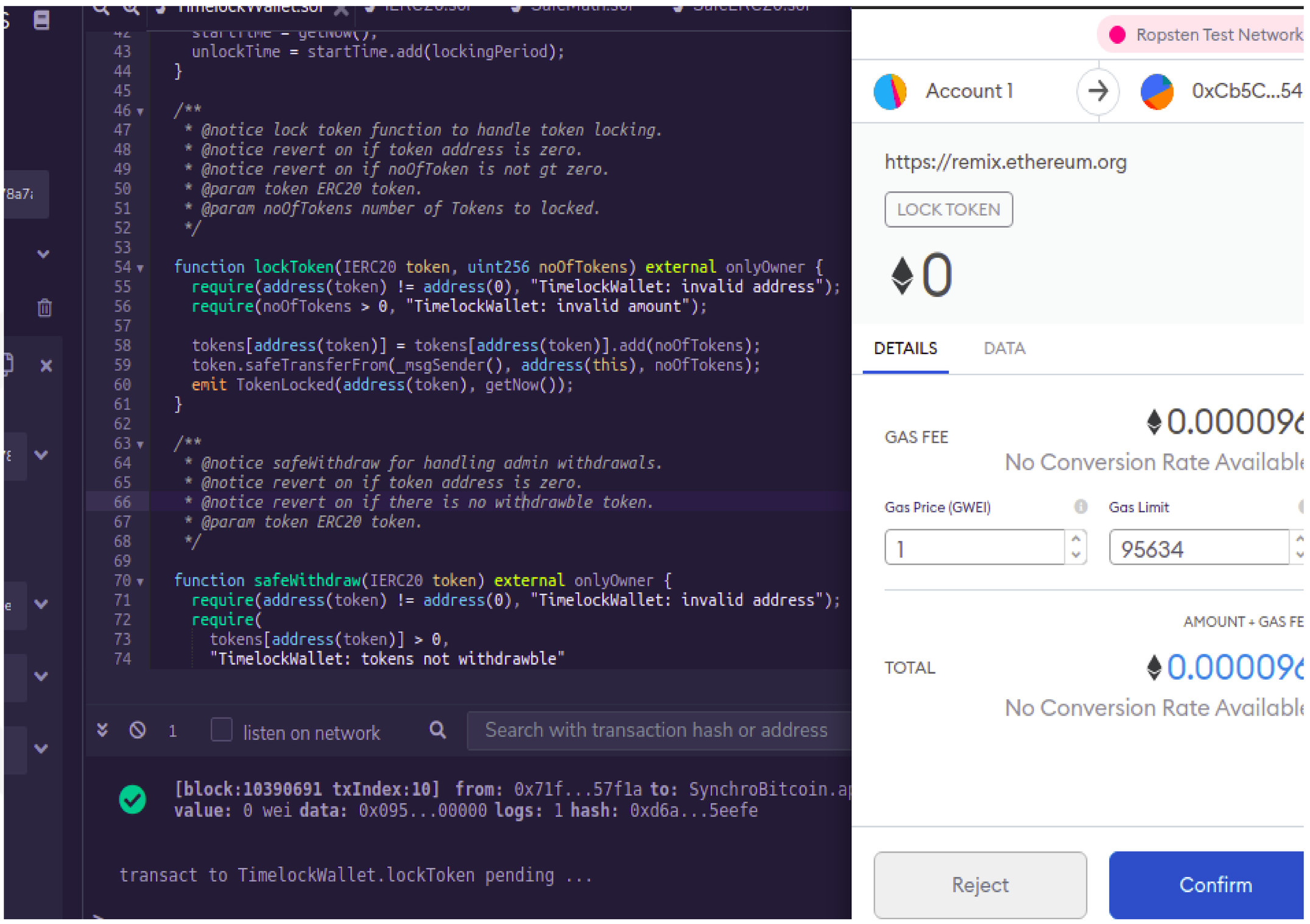
1. **Contract:** TimeLockWallet.sol
   Line: 34
   Issues: Comment 2 doesn't match the implementation or need to check.
   Reasons: Checked with the comment value but failed on 0

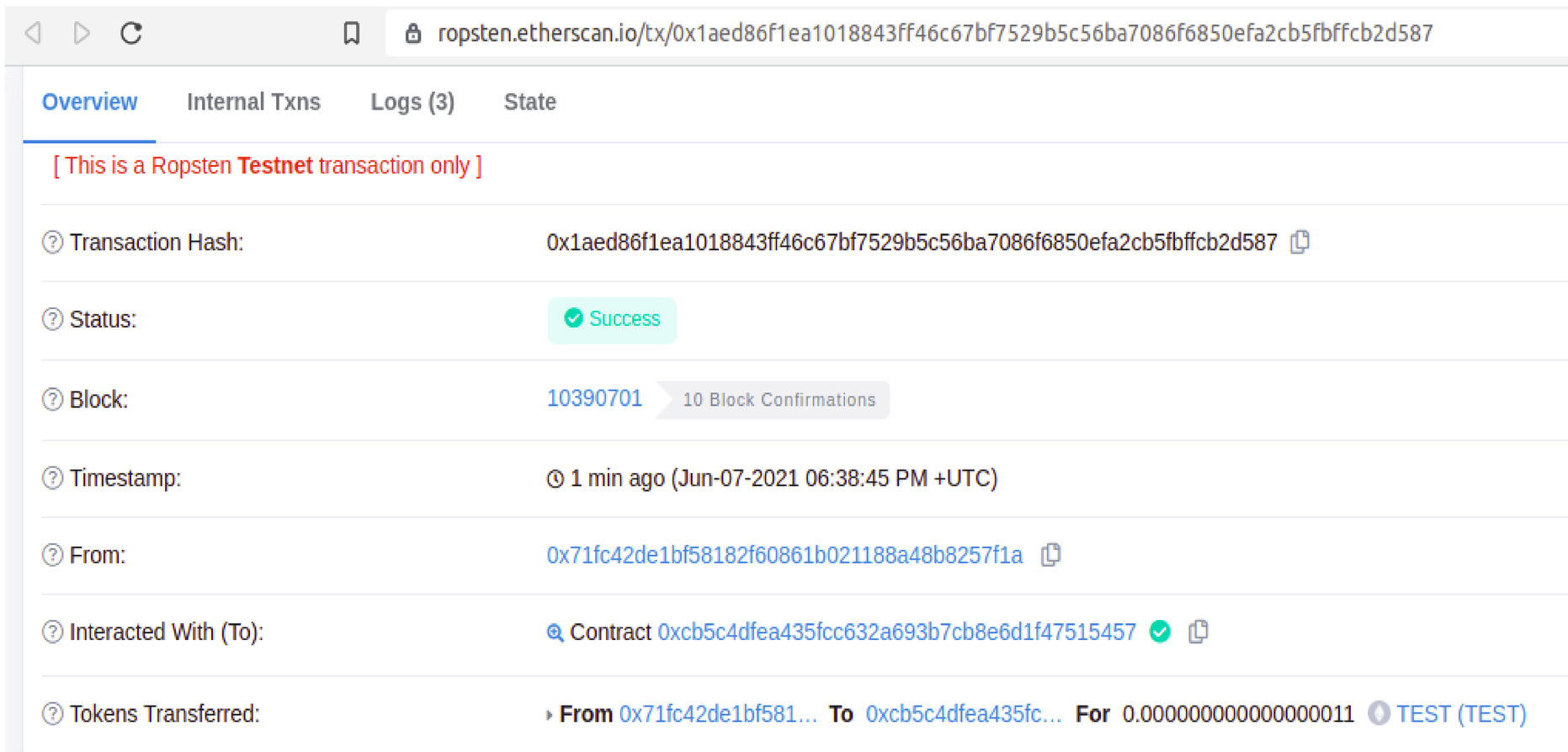
```
32 ▾   /**
33      * @notice construst a TimelockWallet contract.
34      * @notice deployment will failed when lockingPeriod > 0.
35      * @param lockingPeriod locking period eg 90 days, 180 days in seconds.
36      */
```

   Status: Fixed and Closed

2. **Contract:** TimeLockWallet.sol
   Line: 54
   Issues: Operation failed with invalid value error.
   Reasons: Unable to make any transaction via this contract for the lockTokens Functions. (Need to first manually approve the contract)







Status: Fixed and Closed

Informational

1. **Approve Function:** The above failure caused due to approve issue, need approve in the same contract or mention in the comment before function usage.
   Status: Acknowledged by the developer and Closed.

2. **OnlyOwner:** The major functions are only accessible by the contract owner, and the tokens are also transacted from the owner wallet only. If this is the correct implementation, then acknowledge this informational point.
   Status: Acknowledged by the developer and Closed.

3. **Unused functions or interfaces:** Some interfaces are functions that are unused from the owner, context contract which can be ignored.
   Status: Acknowledged by the developer, fixed and Closed.

4. **Review Function:** Amount needed to be specified in the token function safeWithdraw().
   Status: Acknowledged by the developer, fixed and Closed.

Functional Test Table

Function Names	Technical results	Logical results	Overall
lockToken()	Pass	Pass	Pass
safeWithdraw()	Pass	Pass	Pass
updateUnlockTime()	Pass	Pass	Pass
getNow()	Pass	Pass	Pass

Closing Summary

All the major/minor issues are resolved and other suggestions are acknowledged by the developer. Code is clean with proper description in comments for use and transparency. Checked on all major token security parameters and reports generated here. Closed issues are either fixed or acknowledged as per the mutual discussion and agreement on the solutions. Code is clean and applicable to the successful implementation of the business logic.

Disclaimer

QuillHash audit is not a security warranty, investment advice, or an endorsement of the UniFarm_TimeLock platform. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the UniFarm_TimeLock Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

UniFarm

by OroPocket

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

audits@quillhash.com