QuillAudits

Audit Report
June, 2021

DIGIPHARM

Contents

Audit Details and Target	01
Scope of Audit	02
Techniques and Methods	04
Issue Categories	05
Issues Found – Code Review/Manual Testing	06
Summary	09
Disclaimer	10

Audit Details and Target

1. Contract

https://ropsten.etherscan.io/address/0x1147779 56c274A0Ed8E23297bF160C9363f49b1A

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

- migrateFromETH()
- getRevision()
- migrationStarted()
- getRevision()

Overview of The Contract

DigipharmToTerraMigrator contract is responsible for locking DPH ERC20 tokens. The user accepts allowance for this contract, and the locks inside the contract amount of tokens they are willing to migrate to another blockchain. Once locked, it emits an event DPHMigrated which is used by a bridge to send tokens to the generated address.
The contract uses safeMath, VersionedInitializable, and standard erc functions for development. The contract is developed for token migration from one blockchain platform to another.

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.

Scope of Audit

The scope of this audit was to analyse DigipharmToTerraMigrator 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
- https://docs.google.com/document/d/1fivCY-
- lb1DLj6GErriFP9R5DteBSTpPg2TuJ3pfcJyY/edit?usp=sharingChecks 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	2	0	3

Issues Found – Code Review / Manual Testing

High severity issues

No issues found in this category

Medium severity issues

1. Code does not perform the business logic as per the description or expected events.
(Contract: DigipharmToTerraMigrator .sol)
Logic: migrateFromEth()
Line: 55 - 61

Suggestion: One function will be created on one chain, which will initiate the burn function. Burn event will be handled by the backend script. Popup will be there on another blockchain where the token is burned. After that event, the event handler will notify the other blockchain, and the token will be minted on the other blockchain.

Status: Resolved

2. Code comments and business logic are not matching. There is no description of the actual logic in the comment. (LEND to AAVE) But contract functionality is different.
Line: 55 - 61 (Contract: DigipharmToTerraMigrator .sol)

```
49
50      /**
51       * @dev executes the migration from LEND to AAVE. Users need to give allowance to this contract to transfer LEND before executing
52       * this transaction.
53       * @param amount the amount of LEND to be migrated
54       */
55      function migrateFromETH(uint256 amount, bytes32 to) external {
56          require(lastInitializedRevision != 0, "MIGRATION_NOT_STARTED");
57
58          _totalDPHMigrated = _totalDPHMigrated.add(amount);
59          DPH.transferFrom(msg.sender, address(this), amount);
60          emit DPHMigrated(msg.sender, to, amount);
61      }
```

Status: Resolved

Low level severity issues

No issues found in this category

Informational

Comment: Code is written in a very concise way which can be more informative and secure by more checks and modifier use. Gas use is proper and optimised as per the testing on the Ropsten test network and checking of the old transaction from this contract.

1. The use of variables and naming conventions can be minimised.

   ### Status: Resolved

2. Burn of transfer of tokens missing, which needs to be defined to the user for the actual event on the tokens.
   Line: 55 - 61 (Contract : DigipharmToTerraMigrator .sol)

   ### Status: Resolved

3. Functional logic failed on the implementation on the test network for multiple transaction testing scripts.
   Line: 55 - 61 (Contract : DigipharmToTerraMigrator .sol)

Status: Resolved

Functional Test Table

Function Names	Technical results	Logical results	Overall
migrateFromETH()	Pass	Pass	Changes Done
getRevision()	Pass	Pass	Pass
migrateFromETH()	Pass	Pass	Pass

Closing Summary

All the suggestions and issues are fixed and reviewed. The contract is working as per the logic and outputs are reviewed. Comments are well explained and informational regarding the logic of the contract. Security of the contract is checked, tested, and reported in this report.

Disclaimer

QuillHash audit is not a security warranty, investment advice, or an endorsement of the DigipharmToTerraMigrator 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 DigipharmToTerraMigrator Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

DIGIPHARM

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

audits@quillhash.com