QuillAudits

Audit Report
January, 2022

For

starly

Contents

Contents

Overview

Starly has a utility token that serves as a medium to offer creators, collectors and the surrounding communities the ultimate experience on Starly ecosystem. Through teleportation tokens can be burnt and minted on other chains by the admin e.g between Flow blockchain and Ethereum blockchain. Currently the bridge mechanism is centralized relying on correct information being passed around to allow for correct to and from movements between chains.

Scope of the Audit

The scope of this audit was to analyze Starly smart contract's codebase for quality, security, and correctness.

Starly Contracts: https://github.com/StarlyIO/starly-token-contracts/tree/master/contracts/ethereum/contracts

Time Duration: January 21, 2022 - January 22, 2022

Branch: Master

Commit: 2a8ea29197e51c729f9ac0f73e798501eb22f813

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC20 transfer() does not return boolean
- ERC20 approve() race
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

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

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour 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 analysed 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
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 analysed, 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 behaviour 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
Slither, MythX, Truffle, Remix, Surya.

Issue Categories

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.

Risk-level	Description
High	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	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	Low-level severity issues can cause minor impact and or 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 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

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	0	0	9
Closed	0	0	0	0

Functional Testing Results

Complete functional testing report has been attached below:

Some of the tests performed are mentioned below: Visit Link

Starly Token

- should be able to mint by minter role PASS
- should follow ERC20 specification PASS
- should have a default admin role PASS
- should have a minter role PASS
- anyone without minter role cant mint tokens PASS

Starly TeleportCustody

- should be able to freeze by owner PASS
- should revert teleportIn if frozen PASS
- should revert teleportOut if frozen PASS
- should be able to get Token PASS
- should be able to deposit allowance for any user by admin PASS
- should be able to teleportIn by any user with allowance PASS
- should be able to teleportOut by owner using hash to any valid ethereum address PASS
- should revert revert teleportOut if allowance is insufficient PASS
- should revert teleportOut for flowHash already used PASS

Tests

```
zed@Zeds-MBP starly % truffle test ./test/Token.test.js
Using network 'development'.


Compiling your contracts...
===========================
✔ Fetching solc version list from solc-bin. Attempt #1
✔ Fetching solc version list from solc-bin. Attempt #1
> Compiling ./contracts/TeleportCustody.sol
> Artifacts written to /var/folders/px/cvv0nfkj6_qf16j0rb0d7c4r0000gn/T/test--556-XRq0
P213w5w5
> Compiled successfully using:
   - solc: 0.8.11+commit.d7f03943.Emscripten.clang



  Contract: Token
    deployment
      ✓ tracks the name
      ✓ tracks the symbol
      ✓ tracks the decimals
      ✓ starts with zero supply
0x0000000000000000000000000000000000000000000000000000000000000000
      ✓ deployer has DEFAULT_ADMIN_ROLE, async () (51ms)
      ✓ deployer is admin has MINTER_ROLE, async ()
    mint
      ✓ deployer with MINTER_ROLE can mint tokens (235ms)
      ✓ anyone else without minter role cant mint tokens (411ms)
      ✓ new minter assigned MINTER_ROLE and can mint new tokens (272ms)


  9 passing (3s)
```

```
Contract: TeleportCustody
  depositAllowance()
    ✓ should set allowedAmount of an admin (93ms)
    ✓ should update allowedAmount of an admin (214ms)
    ✓ can only be called by owner (295ms)
    ✓ can have multiple admins (222ms)
  freeze() & unfreeze()
    ✓ should be able to freeze contract (109ms)
    ✓ should be able to unfreeze contract (226ms)
    ✓ can only be called by owner (147ms)
  renounceOwnership()
    ✓ should be rejected
    ✓ can only be called by owner
  teleportIn()
    ✓ can teleport in (burn tokens to other chain and emit TeleportIn event (442ms)
    ✓ should block when teleport service is frozen (210ms)
  teleportOut()
    ✓ should block when teleport service is frozen (118ms)
    ✓ requires sufficient allowance (58ms)
Result {
  '0': '0xf282a5f616fF19D2D74270f37eb22F799265E36D',
  '1': BN {
    negative: 0,
    words: [ 50, <1 empty item> ],
    length: 1,
    red: null
  },
  __length__: 2,
  account: '0xf282a5f616fF19D2D74270f37eb22F799265E36D',
  allowedAmount: BN {
    negative: 0,
    words: [ 50, <1 empty item> ],
    length: 1,
    red: null
  }
}
    ✓ successfully teleports Out and emits TeleportOut event (242ms)
    ✓ rejects flowHash already used  (336ms)


  15 passing (7s)
```

Issues Found

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

No issues were found.

Informational issues

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

  The functions are:
  - mint
  - isFrozen
  - Unfreeze
  - depositAllowance
  - allowedAmount
  - teleportIn
  - teleportOut

  Status: Acknowledged

- Floating pragma ^0.8.0. It is best practice to lock the pragma. Consider using version 0.8.4

  Status: Acknowledged

- Unindexed event parameters. (TeleportCustody.sol)

```
event AdminUpdated(address indexed account, uint256 allowedAmount);
event TeleportOut(
    uint256 amount,
    address indexed ethereumAddress,
    bytes32 indexed flowHash
);
event TeleportIn(uint256 amount, bytes8 indexed flowAddress);
```

Up to 3 parameters can be indexed, no harm in indexing all parameters.

Status: Acknowledged

- Value checks and Missing zero address validation. (TeleportCustody.sol)

```
function teleportOut(
        uint256 amount,
        address ethereumAddress,
        bytes32 flowHash
    ) public notFrozen {
```

Critical address to which funds will be moved, if zero may lead to protocol not working as expected. Values for amount and ethereumAddress need to be checked and verified. At the moment the backend is centralized and works to ensure only valid parameters are used.

Status: Acknowledged

- **Critical address change**
  TeleportCustody which has ownership due to Ownable may have ownership lost if a two step process is not followed. Changing critical addresses in contracts should be a two-step process where the first transaction (from the old/current address) registers the new address (i.e. grants ownership) and the second transaction (from the new address) replaces the old address with the new one (i.e. claims ownership). This gives an opportunity to recover from incorrect addresses mistakenly used in the first step. If not, contract functionality might become inaccessible. Consider using Claimable contracts from OpenZeppelin

  Status: Acknowledged

- Consider using NatSpec and inline code commenting especially for critical functions like teleportIn() and teleportOut()

  Status: Acknowledged

- **Centralization risk for Token**
  The DEFAULT_ADMIN_ROLE and MINTER_ROLE are assigned to the same address.

```
constructor(
      string memory name,
      string memory symbol,
      uint8 decimals_
  ) ERC20(name, symbol) {
      _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
      _setupRole(MINTER_ROLE, msg.sender);

      _decimals = decimals_;
  }
```

However, TeleportCustody will have MINTER_ROLE and through the use of allowance risk of unlimited or inappropriate mints is mitigated.

  Status: Acknowledged

- **Missing test cases**
  It is highly recommended to write test cases for all the functions. Tests will help determine if the code is working in the expected way. Unit tests, functional tests and integration tests should have been performed to achieve good test coverage across the entire codebase.

  Status: Acknowledged

- **Backend and Frontend service**
  Although the team notes that system architecture has hardware restriction via hardware authentication, admin keys are managed by Google KMS, IP whitelisting and such others. It is key to note that correct functioning of the system largely depends on this architecture and its security and the backend correctly extracting parameters and working or providing valid inputs to the smart contracts. Appropriate roles would be provided whilst deploying.

  Status: Acknowledged

Automated Tests

Slither Analysis Results

```
SLITHER
ANALYSIS
  High (0)
  Medium (0)
  Low (5)
    Modifier Migrations.restricted() does not always execute _; or revert
    Reentrancy in TeleportCustody.teleportIn(uint256,bytes8):
    Reentrancy in TeleportCustody.teleportOut(uint256,address,bytes32):
    Token.constructor(string,string,uint8).name shadows:
    Token.constructor(string,string,uint8).symbol shadows:
  Informational (20)
    Different versions of Solidity is used:
    Parameter Migrations.upgrade(address).new_address is not in mixedCase
    Pragma version^0.8.0 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
    Pragma version^0.8.0 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
    Pragma version^0.8.0 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
    Pragma version^0.8.0 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
    Pragma version^0.8.0 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
    Pragma version^0.8.0 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
    Pragma version^0.8.0 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
    Pragma version^0.8.0 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
    Pragma version^0.8.0 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
    Pragma version^0.8.0 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
    Pragma version^0.8.0 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
    Pragma version^0.8.0 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
    Pragma version^0.8.0 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
    Pragma version^0.8.0 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
    Pragma version^0.8.7 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
    Variable Migrations.last_completed_migration is not in mixedCase
    Variable TeleportCustody._teleportOutRecord is not in mixedCase
    solc-0.8.11 is not recommended for deployment
```

MythX Analysis Results

```
TeleportCustody.sol contracts 3
  MythX SWC-103. A floating pragma is set. [2, 1]
  MythX SWC-107. A call to a user-supplied address is executed. [115, 43]
  MythX SWC-107. A call to a user-supplied address is executed. [124, 2]
```

Closing Summary

No issues were found. Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

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

Audit Report
January, 2022

For

starly

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com