CERTIK

Standard Hashrate Token

Security Assessment

January 6th, 2021

For :

Standard Hashrate Group

By :

Guilong Li @ CertiK

guilong.li@certik.org

Bryan Xu @ CertiK

buyun.xu@certik.org

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product’s IT infrastructure and or source code.

Overview

Project Summary

Project Name	Standard Hashrate Token
Description	An ERC20 token implementation with an linear release mechanism
Platform	Ethereum; Solidity
Codebase	GitHub Repository
Commit	1c767c5f5e2ab8fc9d6bef3649a2c43b150b7ad6

Audit Summary

Delivery Date	Jan 6th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Dec. 14, 2020 - Dec. 18, 2020

Vulnerability Summary

Total Issues	11
Total Critical	0
Total Major	0
Total Minor	1
Total Informational	10

Executive Summary

This report has been prepared for **Standard Hashrate Token** Portocol to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Documentation

The sources of truth regarding the operation of the contracts in scope were lackluster and are something we advise to be enriched to aid in the legibility of the codebase as well as project. To help aid our understanding of each contract's functionality we referred to in-line comments and naming conventions.

These were considered the specification, and when discrepancies arose with the actual code behaviour, we consulted with the **Standard Hashrate Group** or reported an issue.

File in Scope

Contract	SHA-256 Checksum
BTCST.sol	b933da93e42acfaf3c4a96975d08bc572058fb92943e1951cdfff3d1e4b25be0
ETHST.sol	fd019661bd309e9d3be357b1297caca02533e21c990be04a02ee6e7df2487944
MockERC20.sol	a88a83ac1d2492dc054d0d636b70746cb1c42142fc65166dddb69d345ce4dd26
StandardHashrateToken.sol	9545518a34857f0961cc7e6c443fe45d63aa7a07bc6d03c34328e9195a451e82
LinearReleaseToken.sol	a36e544bcf8f8bf45166c0409ab8316259f2799861da9124799f61c0b96ccb94
OwnableContract.sol	8e49f05681eb4790dc28617183abb30780c70fb2c9b3bd6d169a8e8f5197b339
PeggyToken.sol	34b52647eb4e7dd11ed9057177c16e29f32facfd8cd6f2bddeb0b5ff409ae2fa
TokenUtility.sol	2f1a5b3b32c66ef18893707e42c7c8d29f3dd004bd02c00b34a4e1a5b7d4bee1

System Overview

The **Standard Hashrate Token** protocol creates an efficient market for Bitcoin's mining power. By staking BTCSTs, holders of the tokens will receive daily Bitcoin distributions that correspond to the mining power staked.

AdminUpgradeabilityProxy, which is deployed on 0x78650B139471520656b9E7aA7A5e9276814a38e9 on Binance Smart Chain, serves as the entry of the protocol and brings the upgradeability to allow administrator to improve the quality of the protocol by redirecting the transaction to the **BTCST.sol**, which is deployed on 0xe28c4b5ca0d6cf41e5af4fca9a19b548bf3b0def.

BTCST.sol is the core implementation of the protocol following formal Upgradeable ERC20 interface, which includes significant functions, such as mint, burn and transfer. All these significant functions can be invoked in delegate method through **AdminUpgradeabilityProxy**.

The advantage of taking delegate method in protocol is that administrator reserves the ability to improve the quality and fix the runtime issues of the project. It is also worthy of note the down side of delegate method, where the point to the core implementation in **AdminUpgradeabilityProxy** could be modified.

In order to improve the trustworthy of the project, any dynamic runtime changes on **AdminUpgradeabilityProxy** should be notified to clients. Any modified version of core implementation which is pointed by **AdminUpgradeabilityProxy** may be beyond the scope of this audit.

Review Notes

Certain optimization steps that we pinpointed in the source code mostly referred to coding standards and inefficiencies, however 1 minor vulnerability was identified during our audit that solely concerns the specification.

Certain discrepancies between the expected specification and the implementation of it were identified and were relayed to the team, however they pose no type of vulnerability and concern an optional code path that was unaccounted for.

Recommendations

Overall, the codebase of the contracts should be refactored to assimilate the findings of this report, enforce linters and / or coding styles as well as correct any spelling errors and mistakes that appear throughout the code to achieve a high standard of code quality and security.

Findings

ID	Title	Type	Severity
Exhibit-01	Unlocked Compiler Version Declaration	Language Sepcific	Informational
Exhibit-02	Incorrect Naming Convention Utilization	Coding Style	Informational
Exhibit-03	Proper Imports	Dead Code	Informational
Exhibit-04	Too Many Digits	Coding Style	Informational
Exhibit-05	Unused State Variables	Dead Code	Informational
Exhibit-06	Divide before Multiply	Mathematical Operations	Informational
Exhibit-07	Missing Emit Events	Optimization	Minor
Exhibit-08	Misleading Error Message	Optimization	Informational
Exhibit-09	Missing Checks of Parameters	Gas Consumption	Informational
Exhibit-10	Redundant Codes	Dead Code	Informational
Exhibit-11	Use SafeMath	Mathematical Operations	Informational

Exhibit-01: Unlocked Compiler Version Declaration

Type	Severity	Location
Language Sepcific	Informational	StandardHashrateToken.sol, LinearReleaseToken.sol, OwnableContract.sol, PeggyToken.sol, TokenUtility.sol, MockERC20.sol, ETHST.sol, BTCST.sol

Description:

The compiler version utilized throughout the project uses the ”>=0.4.22 <0.8.0” specifier, denoting that a compiler version which is greater than the version 0.4.22 and smaller than 0.8.0 will be used to compile the contracts. Recommend the compiler version should be consistent throughout the codebase.

Recommendation:

It is a general practice to instead lock the compiler at a specific version rather than allow a range of compiler versions to be utilized to avoid compiler-specific bugs and be able to identify ones more easily. We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project utilizes a compiler version that has been in use for the longest time and as such is less likely to contain yet-undiscovered bugs.

Alleviation:

The team heeded our advice and locked the version of their contracts at version 0.6.9, ensuring that compiler-related bugs can easily be narrowed down should they occur.
The recommendations were applied in commit fe4c51420106d67b63598d76165974cfd0745774.

Exhibit-02: Incorrect Naming Convention Utilization

Type	Severity	Location
Coding Style	Informational	OwnableContract.sol L38,L43 PeggyToken L16,L18,L19,L73

Description:

Solidity defines a naming convention that should be followed. In general, parameters should use mixedCase, refer to: https://solidity.readthedocs.io/en/v0.6.12/style-guide.html#naming-conventions

Function arguments should use mixedCase.
Examples:
Parameters like: `_devaddr`

Constands should use UPPER_CASE_WITH_UNDERSCORES.
Examples:
Parameters like: `_lockMagicNum` , `_unLockMagicNum`

Inside each contract, library or interface, use the following order:
Type declarations
State variables
Events
Functions

refer to: https://docs.soliditylang.org/en/v0.6.12/style-guide.html?highlight=layout#order-of-layout

Examples:

```
    event Lock(address indexed account,uint256 amount);
    event UnLock(address indexed account,uint256 amount);
    uint internal constant  _lockMagicNum = 16;
    uint internal constant  _unLockMagicNum = 0;
    ...
```

Recommendation:

The recommendations outlined here are intended to improve the readability, and thus they are not rules, but rather guidelines to try and help convey the most information through the names of things.

Exhibit-03: Proper Imports

Type	Severity	Location
Dead Code	Informational	StandardHashrateToken.sol L4, PeggyToken.sol L4

Description:

There are some imported files not used in the contract `StandardHashrateToken`.

```
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
```

There are some imported files not used in the contract `PeggyToken`.

```
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
```

There are some OpenZeppelin libraries are imported by copying to the project.
Better import libraries from github rather than copy it to directory.

Recommendation:

We recommend to remove the unused imports, and import neccessary libraries from github.

Alleviation:

This issue was addressed in commit 0817f4f1eb6fa27ad2549b8b44e5d816e5033986.

Exhibit-04: Too Many Digits

Type	Severity	Location
Coding Style	Informational	LinearReleaseToken.sol L211

Description:

Literals with many digits are difficult to read and review.

```
    require(nval < 864000000,"LockTimeUnitPerSeconds should less than 10000
days");
```

Recommendation:

Consider to use Ether suffix.

```
    uint256 private constant TEN_THOUSAND_DAYS = 864*1e6;
    require(nval < TEN_THOUSAND_DAYS,"LockTimeUnitPerSeconds should less than
10000 days");
```

Exhibit-05: Unused State Variables and Functions

Type	Severity	Location
Dead Code	Informational	OwnableContract.sol L125

Description:

Unused state variable.

```
    uint256[49] private __gap;
```

Recommendation:

We recommend to remove unused state variables.

Exhibit-06: Divide before Multiply

Type	Severity	Location
Mathematical Operations	Informational	TokenUtility.sol L91 LinearReleaseToken.sol L164,L276

Description:

Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.

```
    uint round = time.sub(_farmStartedTime).div(_miniStakePeriodInSeconds);
    uint end = _farmStartedTime.add(round.mul(_miniStakePeriodInSeconds));
```

```
    uint256 timePerRound = _lockTime.div(_lockRounds);
    ...
    uint passedRound = passed.div(timePerRound * _lockTimeUnitPerSeconds);
```

```
    freeAmount = records[keys[ii]].mul(
    (now - (keys[ii] - _lockTime * _lockTimeUnitPerSeconds))
    .div(_lockTime.div(_lockRounds) *
_lockTimeUnitPerSeconds)).div(_lockRounds);
```

Recommendation:

We recommend ordering multiplication before division or multiply 1e18 on the division results, then divide 1e18 on the final results.

Exhibit-07: Missing Emit Events

Type	Severity	Location
Optimization	Minor	OwnableContract.sol L87 PeggyToken.sol L53,L56,L73 LinearReleaseToken.sol L202,L206,L211

Description:

Several sensitive actions are defined without event declarations.

Examples:
`transferOwnership()` in `OwnableContract` contract.
`changeIcon()`, `changeMeta()`, `dev()` in `PeggyToken` contract.
`changeLockTime()`,`changeLockRounds()`,`changeLockTimeUnitPerSeconds()` in `LinearReleaseToken` contract.

Recommendation:

Consider adding events for sensitive actions, and emit it in the function like below.

```
    function transferOwnership(address newOwner) public onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero 
address");
        emit OwnershipTransferred(_owner, newOwner);
        pendingOwner = newOwner;
    }
```

Exhibit-08: Misleading Error Message

Type	Severity	Location
Optimization	Informational	PeggyToken.sol L74

Description:

The error message below is misleading.

`require(msg.sender == devaddr, "dev: wtf?");`

Recommendation:

We recommend changing it as follows
`require(msg.sender == devaddr, "PeggyToken: Not devaddr");`

Exhibit-09: Missing Checks of Parameters

Type	Severity	Location
Gas Optimization	Informational	PeggyToken.sol L60

Description:

Better to check parameter value does not equals to zero in function `burn()`.
Better to check mapping _lockMap[account] does not equals to zero in function `lockAccount()`.
Better to check mapping _lockMap[account] is greater than zero in function `unLockAccount()`.

Recommendation:

Consider to add checks for parameter values.

```
    function burn(uint value) override public onlyOwner {
        require (value != 0 , "Value equals to zero");
        super.burn(value);
    }
```

```
    function lockAccount(address account) public onlyOwner {
        require(_lockMap[account] != 0,"Account has been locked");
        uint256 bal = balanceOf(account);
        _totalSupplyLocked = _totalSupplyLocked.add(bal);
        _lockMap[account] = _lockMagicNum;
        emit Lock(account,bal);
    }
```

```
    function unLockAccount(address account) public onlyOwner {
        require(_lockMap[account] > 0,"Account is not locked;
        uint256 bal = balanceOf(account);
        _totalSupplyLocked =
_totalSupplyLocked.sub(bal,"bal>_totalSupplyLocked");
        _lockMap[account] = _unLockMagicNum;
        emit UnLock(account,bal);
    }
}
```

Exhibit-10: Redundant Codes

Type	Severity	Location
Dead Code	Informational	PeggyToken.sol L65

Description:

The below codes are reduntant:

```
    function finishMinting() public view onlyOwner returns (bool) {
        return false;
    }
```

This function can only return false.

Recommendation:

We recommend removing the redundant codes.

Exhibit-11: Use SafeMath

Type	Severity	Location
Mathematical Operations	Informational	LinearReleaseToken.sol L111

Description:

Below codes in function `mintWithTimeLock` did not use SafeMath.

```
    if (_lockTime>0){
        uint freeTime = now + _lockTime * _lockTimeUnitPerSeconds;
        _timeKeysPush(account, freeTime);
        ...
    }
```

Recommendation:

We recommend to use SafeMath for calculations.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an instorage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete` .

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.