Preliminary Comments

Security Assessment

March 4, 2021

Preliminary Report

For :
Standard Hashrate Token team @ Standard Hashrate Group

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	3478e6a86ca97f6ae8b267157fd8d92fc80ace11

Audit Summary

Delivery Date	March 4, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Mar. 1, 2021 - Mar. 4, 2021

Vulnerability Summary

Total Issues	10
Total Critical	0
Total Major	0
Total Minor	1
Total Informational	8
Total Discussion	1

Executive Summary

This report has been prepared for **Standard Hashrate Token** smart contract 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.

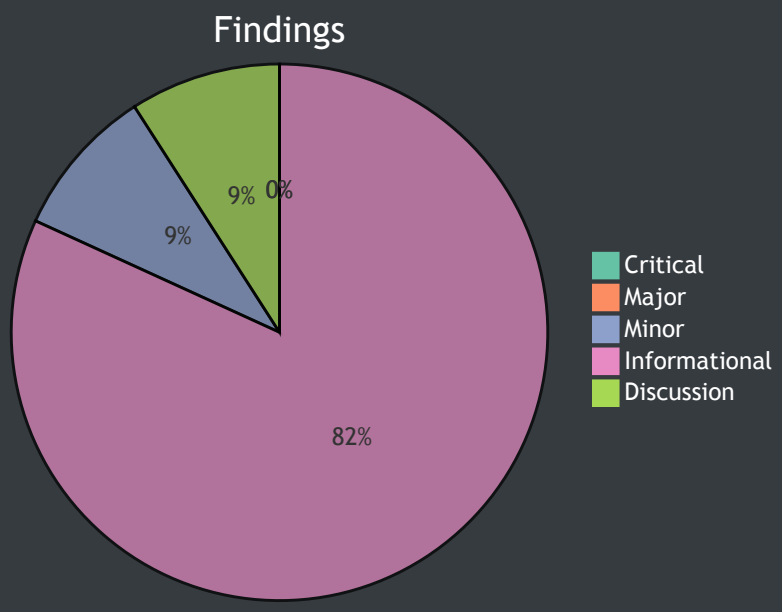
The auditing process pays special attention to the following considerations:

- 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.

File in Scope

ID	Contract	SHA-256 Checksum
PT	**libraries/PeggyToken.sol**	cdeb49f26ffca5b9a19741c184fbf1f135c9a72b178d132ba5e69b84281b7a8d
TU	**libraries/TokenUtility.sol**	481bd2cedec89030be731966ee0e74ecf387c1315968713548e308960cb4823c
BT	**token/BTCSTV2.sol**	62ab66d74f32f9cab28f84aceb71ed8f75b3f846d1c4527a36c7e3404d7514b1
LR	**token/LinearReleaseTokenV2.sol**	905f7d8f4713ae6f4ea018e36120e1f35847eaa1e277ace70e95a54fdea153de
SH	**token/StandardHashrateTokenV2.sol**	dd12c70f6862711453f1caa51d6ee255c698e4c1b32df7b58db752f8f43df131

Findings



ID	Title	Type	Severity
PT-01	Missing Emit Events	Optimization	Informational
PT-02	Missing Check Zero Address	Optimization	Informational
PT-03	Function State Mutability	Optimization	Informational
BT-01	Economy Model	Logical	Discussion
LR-01	Usage of `_balanceFreeTimeKeysIndex`	Logical	Minor
LR-02	Similar Functions	Logical	Informational
LR-03	Optimalization of If Condition	Optimization	Informational
LR-04	Missing Check Zero Value	Logical	Informational
SH-01	Usage of Local Variables	Optimization	Informational
SH-02	Initialization of `_farmContract`	Optimization	Informational

PT-01: Missing Emit Events

Type	Severity	Location
Optimization	Informational	PeggyToken.sol, LinearReleaseTokenV2.sol, StandardHashrateTokenV2.sol

Description:

Several sensitive actions are defined without event declarations.

Examples:
Functions `changeIcon()`, `changeMeta()`, and `dev()` in contract `PeggyToken.sol`.

Functions `changeLockTime()`, `changeLockRounds()`, and `changeLockTimeUnitPerSeconds` in contract `LinearReleaseTokenV2.sol`.

Function `changeFarmContract()` in contract `StandardHashrateTokenV2.sol`.

Recommendation:

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

```
event Lock(address indexed devAddress);
function dev(address _devaddr) public {
    ......
    emit Dev(_devaddr);
}
```

PT-02: Missing Check Zero Address

Type	Severity	Location
Optimization	Informational	PeggyToken.sol, BTCSTV2.sol LinearReleaseTokenV2.sol, StandardHashrateTokenV2.sol

Description:

Functions `lockAccount()` and `unLockAccount()` in contract `PeggyToken.sol` are missing check zero address.

Function `adminUpgradeDecimal` in contract `BTCSTV2.sol` .

Functions `allowanceLocked()` , `linearLockedBalanceOf` , `_linearLockedBalanceOf` , `getFreeToTransferAmount` , `transferLockedFrom` , `transferLockedTo` , and `approveLocked` in contract `LinearReleaseTokenV2.sol` are missing check zero address.

Functions `transferLockedTo` and `transfer` in contract `StandardHashrateTokenV2.sol` are missing check zero address.

Recommendation:

Consider adding neccessary check, for example:

```
function lockAccount(address account) public onlyOwner {
    require(account != address(0), "account is address(0)")
    ...
}
```

PT-03: Function State Mutability

Type	Severity	Location
Optimization	Informational	PeggyToken.sol

Description:

Function `renounceOwnership()` in contract `PeggyToken.sol`, does not change the state.

Recommendation:

Consider restricting the state mutability of the function to be `view`.

BT-01 : Economy Model

Type	Severity	Location
Logical	Discussion	BTCSTV2.sol

Description:

The contract `BTCSTV2` will create a new token but has the same name `BTCST` with the before listed token. What's your solution to make the new token compatible with the old BTCST token under the function `adminUpgradeDecimal` ? What's your solution to migrate user data from the old token to new token?

LR-01: Usage of `_balanceFreeTimeKeysIndex`

Type	Severity	Location
Logical	Minor	LinearReleaseTokenV2.sol

Description:

The below statement is unnecessary and has a mistake:

```
function _timeKeysRemove(address account,uint timeKey)internal returns(bool){
  ...
  _balanceFreeTimeKeysIndex[account][bytes32(lastvalue)] = toDeleteIndex+1;
  ...
}
```

Value `bytes32(timeKey)` insteads of `bytes32(lastvalue)` should be the index of `_balanceFreeTimeKeysIndex[account][]`, based its usage in the functions `_timeKeysPush()` and `_timeKeysRemove()`.

Recommendation:

Consider removing this useless statement.

LR-02: Useless Functions

Type	Severity	Location
Logical	Informational	LinearReleaseTokenV2.sol

Description:

There are two similar funtions, `linearLockedBalanceOf()` and `_linearLockedBalanceOf()`, in contract `LinearReleaseTokenV2.sol`.

Function `decreaseGasConsumptionByClearExpiredRecordst` only returns a zero and is useless.

Recommendation:

Consider removing one of functions: `linearLockedBalanceOf()` and `_linearLockedBalanceOf()`.

Consider removing the function `decreaseGasConsumptionByClearExpiredRecordst()`.

LR-03: Optimalization of If Condition

Type	Severity	Location
Optimization	Informational	LinearReleaseTokenV2.sol

Description:

In the last if branch in the function `getFreeToTransferAmount` of contract `LinearReleaseTokenV2.sol` , once `allFreed` equals `lockedBalance` . the last if condition will lead more opcodes to be executed since statement `lockedBalance.sub(allFreed,"allFreed>lockedBalance")` returns zero.

Recommendation:

Consider changing the condition like below without any side effects:

```
        if (allFreed < lockedBalance){
            return balance.sub(lockedBalance.sub(allFreed,"allFreed>lockedBalance"),"balance
    limited");
        }
        return balance;
```

LR-04: Missing Check Zero Value

Type	Severity	Location
Logical	Informational	LinearReleaseTokenV2.sol

Description:

Function `changeLockTime` and `changeLockTimeUnitPerSeconds` missing check zero value for parameter `nLockTime` and `nval` respectively. Function `calculateFreeAmount` in contract will be reverted once `nLockTime` or `_lockTimeUnitPerSeconds` is zero since they are factors of denominator in the function `calculateFreeAmount`.

Recommendation:

Consider checking zero value for the parameter `nLockTime` and `_lockTimeUnitPerSeconds`, like:

```
function changeLockTime(uint256 nLockTime) public onlyOwner{
    require(nLockTime > 0,"nLockTime should greater than 0");
    _lockTime = nLockTime;
}
```

SH-01: Usage of Local Variables

Type	Severity	Location
Optimalization	Informational	StandardHashrateTokenV2.sol

Description:

There are local variables are declared and only used once in the functions of contract `StandardHashrateTokenV2.sol` , like `address owner` in the function `initialize()` , `address farm` in the function `onlyFarm` .

Recommendation:

Consider removing those local variables, like:

```
    function initialize(string memory name, string memory symbol) public override
initializer{
        super.initialize(name,symbol,msg.sender,25*7,25);
    }
```

SH-02: Initialization of `_farmContract`

Type	Severity	Location
Optimalization	Informational	StandardHashrateTokenV2.sol

Description:

Important contract `_farmContract` does not be initialized in function `initialize`.

Recommendation:

Consider initializing the value of `_farmContract` in function `initialize()`.

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.

Icons explanation

: Issue resolved

: Issue not resolved / Acknowledged. The team will be fixing the issues in the own timeframe.

: Issue partially resolved. Not all instances of an issue was resolved.