Security Assessment

ETHST

May 29th, 2021

Table of Contents

Summary

This report has been prepared for ETHST smart contracts, 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 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.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	ETHST
Description	ETHST stands for Ethereum Standard Hashrate Token. It's the first global perpetual Ethereum standard hashrate protocol, a hashrate token that anchors the mining power of Ethereum.
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/ethst20210317/ETHST
Commits	1.a067e0c1ea25ce40a2ea6b9152c43a0f4dccd782 2.0662f02ddd560c2876d60e79e99d1ef8a59369f0

Audit Summary

Delivery Date	May 29, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

Total Issues	17
Critical	0
Major	0
Medium	0
Minor	4
Informational	13
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
ETT	ETToken.sol	db62463e67553813bbdb38367adae56d2f8334455efa4fa856f778a2d6d4363b
ETF	ETTokenFactory.sol	ae98fe2a7c3dd47a414361b0eb28fe143aace5a2f8a923b9d2498aecd591c6eb
ETE	ExchangeToken.sol	6230a1ee6d9fb12907fdfecc42993000113dc65e467f3a46b614f7afa0675882
IRE	InviteReward.sol	2cec39635f6868a9850adff8f827d5bc91a101cede7b949088d9617b4047a657
LME	LpMining.sol	af3c03ad45c742478289d226ba411cebf7567c7475cce4484ec57b6a6e2ab76a
NME	NodeMining.sol	bffdd4f5cf1bc14604e7e6b334cc2e154d22c901547521b2adb6799f74079304
PME	PledgeMining.sol	b44daba776ce33fb6c0bec8fe1140d7d03059465776858cbd06b9311f590f8b6
RET	Recommend.sol	58e0512c0e4a49c2f2a59821f75232bcb7a5ab74103616543029a3e919d3b5e1
TME	TeamMining.sol	ee34355f8f32b3a601f5bed2380f13d44db79f2eb5a06c323cfad8210c166dc8

System Overview

ETHST is the first global perpetual Ethereum standard hash rate token, a hash rate token that anchors the mining power of Ethereum.

Users who hold ETHST passes can obtain two tokens, ETH and ET, enjoying double benefits for mining with 2 tokens, among them, ET is the governance token of the first global perpetual Ethereum-based standard hash rate token ETHST. The total amount of issuance is constant at 100 million and will never be issued again.

There are five specific distribution plans of ET tokens, such as Node Reward, Liquidity Mining, Developer Rewards, ETHST Mining, and Invitation Incentives.

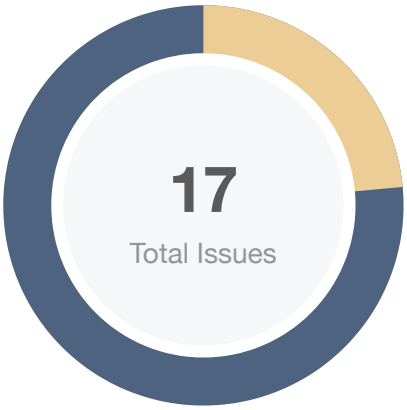
In general, the code implements most of the functions described in the white paper.

Review Summary

There is an inconsistent point between the audit code and the white paper as the following.

- Team's reward ET will be unlocked in 12 equal batches within one year. The fact is that the developer will get this reward by calling the function `withdrawal` in the contract `TeamMining.sol`. It will unlock 1% in the remaining ET to the team every day.

Findings



Critical	**0** (0.00%)
Major	**0** (0.00%)
Medium	**0** (0.00%)
Minor	**4** (23.53%)
Informational	**13** (76.47%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
ETE-01	Redundant Judgment Condition	Control Flow	Minor	Resolved
ETE-02	Typos In The Contract	Coding Style	Informational	Resolved
ETF-01	Inconsistent Of ET Token Release Rate	Logical Issue	Minor	Acknowledged
ETF-02	Missing Zero Address Validation	Logical Issue	Informational	Resolved
ETH-01	Unlocked Version Of Solidity	Optimizaition	Informational	Acknowledged
ETT-01	Divide Before Multiply	Mathematical Operations	Informational	Acknowledged
LME-01	Redundant Judgment Condition	Control Flow	Minor	Resolved
LME-02	Typos In The Contract	Coding Style	Informational	Resolved
NME-01	Incorrect Warning Message	Logical Issue	Informational	Resolved
NME-02	Costly Operations Inside A Loop	Gas Optimization	Informational	Resolved
NME-03	Unused Variables	Gas Optimization	Informational	Partially Resolved
NME-04	Incorrect Warning Message	Logical Issue	Informational	Resolved
RET-01	Variable Could Be Declared Constant	Gas Optimization	Informational	Resolved
RET-02	Boolean Equality	Coding Style	Informational	Resolved
RET-03	Missing Zero Address Validation	Logical Issue	Informational	Resolved

ID	Title	Category	Severity	Status
TME-01	Inconsistent Of Team Reward	Inconsistency	Minor	Resolved
TME-02	Missing Zero Address Validation	Logical Issue	Informational	Resolved

ETE-01 | Redundant Judgment Condition

Category	Severity	Location	Status
Control Flow	Minor	ExchangeToken.sol: 257	Resolved

Description

There some judgments that are redundant. Such as the following codes could improve.

- Variable `amount` in the function `buy`, contract `ExchangeToken.sol`.
- Variable `weight` in functions `addPool`, contract `LpMining.sol`.
- Variable `weight` in function `updatePool`, contract `LpMining.sol`.

Example:

```
function buy(uint256 amount, uint256 id)
    external
    virtual
    override
    nonReentrant
{
        ......
    require(amount >= 0, "The payment amount is too small");
        ......
}
```

The type of `amount` is `uint256`, so it must be satisfied with the `require` statement.

Recommendation

Consider modifying the `require` statement.

```
function buy(uint256 amount, uint256 id)
    external
    virtual
    override
    nonReentrant
{
        ......
    require(amount > 0, "The payment amount is too small");
```

```
        ......
    }
```

Alleviation

The team heeded the advice and resolved this issue in commit `0662f02ddd560c2876d60e79e99d1ef8a59369f0`.

ETE-02 | Typos In The Contract

Category	Severity	Location	Status
Coding Style	Informational	ExchangeToken.sol	Resolved

Description

There are several typos of function name in the code:

`_updateAllPoolRewardPreShare`, `_updateSingePoolReward`, `currenSingePoolETBlockRewardShare`, `_updateSingePoolRewardPreShare` in the contract `LpMining.sol`;

`updateCrrentPrice` in the contract `ExchangeToken.sol`.

Recommendation

We recommend correcting all typos in the contract.

Alleviation

The team heeded the advice and resolved this issue in commit `0662f02ddd560c2876d60e79e99d1ef8a59369f0`.

ETF-01 | Inconsistent Of ET Token Release Rate

Category	Severity	Location	Status
Logical Issue	Minor	ETTokenFactory.sol: 153, 163	Acknowledged

Description

In the white paper, this protocol will release 0.1% of the total remaining amount of ET tokens daily. In the code logic, this protocol will release 1% of the total remaining amount of ET tokens daily. And in the code comment, this protocol will release 10% of the total remaining amount of ET tokens daily.

Please change all of these points to follow the intention of the design.

Alleviation

[ETHST Team]: In the future, we will deploy a new token, replacing the current ET token. The new token will follow the description on the white paper.

ETF-02 | Missing Zero Address Validation

Category	Severity	Location	Status
Logical Issue	Informational	ETTokenFactory.sol: 43, 58	Resolved

Description

Functions `updateConfig` and `constructor` in contract `ETTokenFactory.sol`, function `updateConfig` in contract `Recommend.sol`.

All of them are missing address zero checks.

Recommendation

Consider adding zero address check, for example:

```
function updateConfig(address _token)
    external
    requireImpl
{
    require(_token != address(0), "ERR_ZERO_ADDR");
    token = _token;
}
```

Alleviation

The team heeded the advice and resolved this issue in commit `0662f02ddd560c2876d60e79e99d1ef8a59369f0`.

ETH-01 | Unlocked Version Of Solidity

Category	Severity	Location	Status
Optimizaition	Informational		Acknowledged

Description

We do not recommend using unlocked versions of solidity for deployment.

Recommendation

Deploy with any of the following Solidity versions:

- `0.5.16 - 0.5.17`
- `0.6.11 - 0.6.12`
- `0.7.5 - 0.7.6`

Use a simple pragma version that allows any of these versions. Consider using the latest version of Solidity for testing.

Alleviation

No Alleviation.

ETT-01 | Divide Before Multiply

Category	Severity	Location	Status
Mathematical Operations	Informational	ETToken.sol: 175~178	Acknowledged

Description

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

Recommendation

Consider ordering multiplication before division.

Alleviation

[ETHST Team]: The precision loss is too tiny to care.

LME-01 | Redundant Judgment Condition

Category	Severity	Location	Status
Control Flow	Minor	LpMining.sol: 456, 483	Resolved

Description

There some judgments that are redundant. Such as the following codes could improve.

- Variable `amount` in the function `buy`, contract `ExchangeToken.sol`.
- Variable `weight` in functions `addPool`, contract `LpMining.sol`.
- Variable `weight` in function `updatePool`, contract `LpMining.sol`.

Example:

```
function buy(uint256 amount, uint256 id)
    external
    virtual
    override
    nonReentrant
{
        ......
    require(amount >= 0, "The payment amount is too small");
        ......
}
```

The type of `amount` is `uint256`, so it must be satisfied with the `require` statement.

Recommendation

Consider modifying the `require` statement.

```
function buy(uint256 amount, uint256 id)
    external
    virtual
    override
    nonReentrant
{
        ......
    require(amount > 0, "The payment amount is too small");
```

```
        ......
    }
```

Alleviation

The team heeded the advice and resolved this issue in commit `0662f02ddd560c2876d60e79e99d1ef8a59369f0`.

LME-02 | Typos In The Contract

Category	Severity	Location	Status
Coding Style	Informational	LpMining.sol	Resolved

Description

There are several typos of function name in the code:

`_updateAllPoolRewardPreShare`, `_updateSingePoolReward`, `currenSingePoolETBlockRewardShare`, `_updateSingePoolRewardPreShare` in the contract `LpMining.sol`;

`updateCrrentPrice` in the contract `ExchangeToken.sol`.

Recommendation

We recommend correcting all typos in the contract.

Alleviation

The team heeded the advice and resolved this issue in commit `0662f02ddd560c2876d60e79e99d1ef8a59369f0`.

NME-01 | Incorrect Warning Message

Category	Severity	Location	Status
Logical Issue	Informational	NodeMining.sol: 153	Resolved

Description

The following warning message is inappropriate:

```
    function settlement(address[] calldata users)
        external
        virtual
        override
        requireImpl
        nonReentrant
    {
        require(users.length <= 21, "Settlement users length <= 21");
    ........
```

The warning message has opposite meanings.

Recommendation

Consider modifying like below:

```
        require(users.length <= 21, "Settlement users length > 21");
```

Alleviation

The team heeded the advice and resolved this issue in commit `0662f02ddd560c2876d60e79e99d1ef8a59369f0`.

NME-02 | Costly Operations Inside A Loop

Category	Severity	Location	Status
Gas Optimization	Informational	NodeMining.sol: 171	Resolved

Description

Costly operations inside a loop might waste gas, so optimizations are justified, refer to:
https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop

Recommendation

Use a local variable to hold the loop computation result. Example like below:

```
uint256 sendAmount = ETReward.div(2);
uint256 etReward = ETReward;
for (uint256 i = 0; i < users.length; i++) {
     ......
     etReward = etReward.sub(rewardAmount);
     emit Settlement(uAddr, rewardAmount);
}
ETReward = etReward;
```

Alleviation

The team heeded the advice and resolved this issue in commit `0662f02ddd560c2876d60e79e99d1ef8a59369f0`.

NME-03 | Unused Variables

Category	Severity	Location	Status
Gas Optimization	Informational	NodeMining.sol: 37, 39	Partially Resolved

Description

Variables `ETRewardPerShare` and `Contribution_Total` are not used.

Recommendation

Consider removing them.

Alleviation

The team heeded the advice and resolved this issue in commit `0662f02ddd560c2876d60e79e99d1ef8a59369f0`.

NME-04 | Incorrect Warning Message

Category	Severity	Location	Status
Logical Issue	Informational	NodeMining.sol: 153	Resolved

Description

The following warning message is inappropriate:

```
    function settlement(address[] calldata users)
        external
        virtual
        override
        requireImpl
        nonReentrant
    {
        require(users.length <= 21, "Settlement users length <= 21");
    ........
```

The warning message has opposite meanings.

Recommendation

Consider modifying like below:

```
        require(users.length <= 21, "Settlement users length > 21");
```

Alleviation

The team heeded the advice and resolved this issue in commit `0662f02ddd560c2876d60e79e99d1ef8a59369f0`.

RET-01 | Variable Could Be Declared Constant

Category	Severity	Location	Status
Gas Optimization	Informational	Recommend.sol: 54	Resolved

Description

Variable `_recommendDepthLimit` is not modified within the contract and thus could be declared `constant`.

Recommendation

We recommend declaring `_recommendDepthLimit` as `constant` and renaming as `RECOMMEND_DEPTH_LIMIT` to comfort the UPPER_CASE_WITH_UNDERSCORE format.

Alleviation

The team heeded the advice and resolved this issue in commit `0662f02ddd560c2876d60e79e99d1ef8a59369f0`.

RET-02 | Boolean Equality

Category	Severity	Location	Status
Coding Style	Informational	Recommend.sol: 136, 161	Resolved

Description

Detects the comparison to boolean constants.

Recommendation

Consider modifying like below:

```
require(
          !_recommerBindMapping[_owner],    "Can not bind repeatedly"
       );
```

Alleviation

The team heeded the advice and resolved this issue in commit `0662f02ddd560c2876d60e79e99d1ef8a59369f0`.

RET-03 | Missing Zero Address Validation

Category	Severity	Location	Status
Logical Issue	Informational	Recommend.sol: 61	Resolved

Description

Functions `updateConfig` and `constructor` in contract `ETTokenFactory.sol`, function `updateConfig` in contract `Recommend.sol`.

All of them are missing address zero checks.

Recommendation

Consider adding zero address check, for example:

```
function updateConfig(address _token)
    external
    requireImpl
{
    require(_token != address(0), "ERR_ZERO_ADDR");
    token = _token;
}
```

Alleviation

The team heeded the advice and resolved this issue in commit `0662f02ddd560c2876d60e79e99d1ef8a59369f0`.

TME-01 | Inconsistent Of Team Reward

Category	Severity	Location	Status
Inconsistency	Minor	TeamMining.sol: 41~47	Resolved

Description

In the white paper, the team‘s reward of ET will be unlocked in 12 equal batches within one year, but it is inconsistent with implementation in function `withdraw`.

Alleviation

[ETHST Team]: The team rewards will be collected regularly.

TME-02 | Missing Zero Address Validation

Category	Severity	Location	Status
Logical Issue	Informational	TeamMining.sol: 24~26, 28~30	Resolved

Description

Functions `updateConfig` and `constructor` in contract `ETTokenFactory.sol`, function `updateConfig` in contract `Recommend.sol`.

All of them are missing address zero checks.

Recommendation

Consider adding zero address check, for example:

```
function updateConfig(address _token)
    external
    requireImpl
{
    require(_token != address(0), "ERR_ZERO_ADDR");
    token = _token;
}
```

Alleviation

The team heeded the advice and resolved this issue in commit `0662f02ddd560c2876d60e79e99d1ef8a59369f0`.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings 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 findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings 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.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather 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.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is 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 assessment. This report does 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.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.