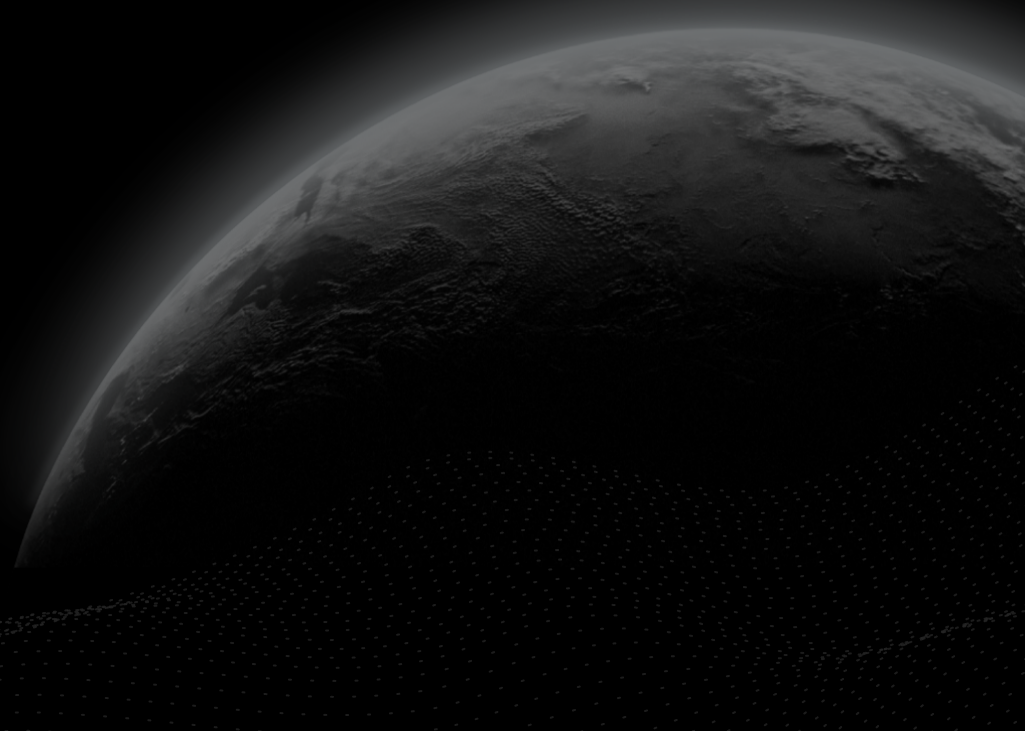
CERTIK

Security Assessment

Perion

CertiK Verified on Feb 6th, 2023



CertiK Verified on Feb 6th, 2023

Perion

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Others	Other	Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 02/06/2023	N/A

CODEBASE

https://github.com/Periondao/perion-liquidity-mining/

...View All

COMMITS

- 5cd0cb6ddbd22d1054c34dceb5c11025864632a7
- 76f03d2c2d566c4f4f930176aa4b723806baa8fc

...View All

Vulnerability Summary

7	5	1	0	1	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined	Unresolved

0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
1	Major	1 Mitigated	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
0	Medium		Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
0	Minor		Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
6	Informational	5 Resolved, 1 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | PERION

CODEBASE | PERION

Repository

https://github.com/Periondao/perion-liquidity-mining/

Commit

- 5cd0cb6ddbd22d1054c34dceb5c11025864632a7
- 76f03d2c2d566c4f4f930176aa4b723806baa8fc

AUDIT SCOPE | PERION

11 files audited ● 1 file with Resolved findings ● 10 files without findings

ID	Repo	Commit	File	SHA256 Checksum
● TPP	Periondao/perion-liquidity-mining	76f03d2	TimeLockPool.sol	18afa872d6d6d32e1f3fec137e7053cf3d5b9ceea34be255a858a6c6631dbf52
● ABS	Periondao/perion-liquidity-mining	76f03d2	base/AbstractRewards.sol	e270601f91d6125f40e9561270b476381d91628655006fef5d0e70ac9b89807b
● BAE	Periondao/perion-liquidity-mining	76f03d2	base/BasePool.sol	756b87b1ac1858e910e371ae6f078c07b1f4d2c925eee119f0bc080b10ddbcea
● BOR	Periondao/perion-liquidity-mining	76f03d2	base/BoringBatchable.sol	d49f31dbbabb52cd64bbd9464d3ae53d194459bd9b145c116721627e0bec1e4c
● IPU	Periondao/perion-liquidity-mining	76f03d2	Imports.sol	e7ca8219d5b41b79ec5ee4d4d2535f567ee94b76c31d35c7d1999d62f2ae5974
● TLT	Periondao/perion-liquidity-mining	76f03d2	TimeLockNonTransferablePool.sol	2529dfabdfd533d8522f0bda0c1f9b3ba987df4ed2302ba7b335d960e5c9a2f7
● UPP	Periondao/perion-liquidity-mining	76f03d2	TransparentUpgradeableProxy.sol	55a1e36d47a2f81882bb37dce50c39c6c26002c5f51f81f660f258701c043c2b
● VPU	Periondao/perion-liquidity-mining	76f03d2	View.sol	46480967f8a6d25c0be1e8c3265d42613802426a67cde655c584e0b3cb7897e2
● IAP	Periondao/perion-liquidity-mining	76f03d2	interfaces/IAbstractRewards.sol	e0394c1ee77ded691dabe466d7f215f96d250078dd7dfc27065453abbedd9d9c
● IPP	Periondao/perion-liquidity-mining	76f03d2	interfaces/IBasePool.sol	4d2377ee376d612be01f7f068f5f7f3c66b3d1724933d473bc8a820ba273cb0b
● ITP	Periondao/perion-liquidity-mining	76f03d2	interfaces/ITimeLockPool.sol	16f0f4ab85fde5c437b1ea83fd6e06f08bf183c148d5d6bc01dfa1d827002e3d

APPROACH & METHODS | PERION

This report has been prepared for Perion to discover issues and vulnerabilities in the source code of the Perion project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis 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:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- 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.

REVIEW NOTES | PERION

Overview

The Perion team has created a **staking functionality** where users can deposit and lock ERC20 tokens in order to earn a yield.

The contract (sPERC-LP) has been deployed to address 0xc014286360Ef45aB15A6D3f6Bb1E54a03352aC8f.

The contract (SPERC) has been deployed to address 0xf64F48A4E27bBC299273532B26c83662ef776b7e.

External Dependencies

In the Perion stacking contract, the project relies on a few external contracts and addresses to fulfill the needs of its business logic.

Addresses

The following addresses interact at some point with specified contracts, making them an external dependency.

- The address `escrowPool` with the interface `ITimeLockPool`.
- The address `rewardToken` with the interface `IERC20`.
- The address `depositToken` with the interface `depositToken`.

During the review, no hardcoded address values were found in the codebase. All following values are initialized either at deploy time or by specific functions in smart contracts.

Contracts

The project uses OpenZeppelin libraries and contracts for contract format and functionality as well as for functions such as security and verification.

The following contracts are referenced in various contracts:

- `SafeERC20Upgradeable`, `ERC20VotesUpgradeable.sol`, `SafeCastUpgradeable.sol`, `AccessControlEnumerableUpgradeable.sol`, `Initializable.sol`, `DraftIERC20Permit.sol`
- `TransparentUpgradeableProxy.sol`, `MathUpgradeable.sol`.

FINDINGS | PERION

7	0	1	0	0	6
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Perion. Through this audit, we have uncovered 7 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
TUP-01	**Centralized Control Of Contract Upgrade**	**Centralization / Privilege**	**Major**	**Mitigated**
BBP-01	Payable Function Using `delegatecall` Inside A Loop	Volatile Code	Informational	Resolved
CON-01	Potential Front-Run Attack Can To Unprotected Initializer	Logical Issue	Informational	Resolved
GLOBAL-01	Incompatibility With Non-Standard ERC20 Tokens	Logical Issue	Informational	Acknowledged
TLN-01	Lack Of Storage Gap In Upgradeable Contract	Logical Issue	Informational	Resolved
TLP-01	Potential Issues On Rewards Distribution	Volatile Code	Informational	Resolved
TPP-01	Potential Failure To Extend Lock Period	Logical Issue	Informational	Resolved

TUP-01 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

Category	Severity	Location	Status
Centralization / Privilege	**Major**	**contracts/TransparentUpgradeableProxy.sol (5cd0cb): 6**	**Mitigated**

Description

`TransparentUpgradeableProxy` is an upgradeable contract, the owner can upgrade the contract without the community's commitment. If an attacker compromises the account, he can change the implementation of the contract and drain tokens from the contract.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND

- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[**Perion Team**, 02/03/2023]: The Perion Team has implemented a multi-signature wallet prior to this audit to protect against a single point of failure resulting from a compromised private key. These measures include the use of multi-signature wallets. The corresponding wallet `admin_` is a GnosisSafe deployed at the address 0x12D73beE50F0b9E06B35Fdef93E563C965796482 | ETH.

The multisig wallet addresses are:

- eth:0x98b729F0212AC8a4647c70dCFbf982439372Eaf9
- eth:0x0E57EefdD1eDb66e4984c3E3026A48189F4B064C
- eth:0xb9F686e51038A6F05Ae7D8d18Ae8F1D4c7f6dcF0

BBP-01 | PAYABLE FUNCTION USING `delegatecall` INSIDE A LOOP

Category	Severity	Location	Status
Volatile Code	Informational	contracts/base/BoringBatchable.sol (5cd0cb): 37	Resolved

Description

`delegatecall()` is used inside a loop in a payable function. If the called function uses `msg.value`, the incoming payment may be processed multiple times unexpectedly.

```
35  function batch(bytes[] calldata calls, bool revertOnFail) external payable {
36    for (uint256 i = 0; i < calls.length; i++) {
37              (bool success, bytes memory result) =
address(this).delegatecall(calls[i]);
38              if (!success && revertOnFail) {
39                  revert(_getRevertMsg(result));
40              }
41          }
42      }
```

This usage can be dangerous as the implementation contract can change in the future. Especially in the case of *double spending* issues.

Recommendation

if this is not necessary we advise removing the `payable` attribute, and also not using logic that relies on the `msg.value` in the future upgrade.

Alleviation

[**Perion Team**, 01/25/2023]: The team heeded the advice and resolved the finding by removing the `payable` of the function `batch()` in the commit cad907a36b889f9b385f8f65cae469167507fdb4.

CON-01 POTENTIAL FRONT-RUN ATTACK CAN TO UNPROTECTED INITIALIZER

Category	Severity	Location	Status
Logical Issue	Informational	contracts/TimeLockNonTransferablePool.sol (5cd0cb): 7; contracts/base/BoringBatchable.sol (5cd0cb): 35	Resolved

Description

The `TimeLockNonTransferablePool` contract does not protect its initializer function `initialize()`, which makes it vulnerable and permits an attacker to take control of the logic contract and perform privileged operations that could either destroy the proxy.

```
6  contract TimeLockNonTransferablePool is TimeLockPool {
```

- `TimeLockNonTransferablePool` is an upgradeable contract that does not protect its initializer.

```
7      function initialize()
```

- `initialize` is an unprotected initializer function.

If the team does not deploy the transparent proxy and the implementation in the same transaction, it leaves an opportunity for an attacker to carry out a "front-run" attack. This would allow the attacker to call the `initialize()` function on the implementation before the team and claim ownership of the contract, potentially enabling them to perform privileged operations and steal ownership. A potential scenario has been described below.

The auditors are requesting information regarding the process of deploying contracts and asking if that will be executed within the same transaction?

Scenario

1. The developer deploys the implementation contract.
2. The attacker "frontruns" the developer and calls the `initialize()` function on the implementation.
3. The developer then deploys the upgradeable contract that points to the implementation.

Recommendation

We advise calling `_disableInitializers` in the constructor or giving the constructor the `initializer` modifier to prevent the initializer from being called on the logic contract.

Reference: https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#initializing_the_implementation_contract

Alleviation

[**Perion Team**, 01/25/2023]: The team heeded the advice and resolved the finding by adding the function `_disableInitializers()` inside the constructor of the contract in the commit 3639b5f096f2999492878fc4d68821e6b1b1768f.

GLOBAL-01 INCOMPATIBILITY WITH NON-STANDARD ERC20 TOKENS

Category	Severity	Location	Status
Logical Issue	Informational		Acknowledged

Description

The current design is not compatible with **non-standard ERC20** tokens, such as deflationary tokens or tokens that adhere to the ERC777 standard. Implementing non-ERC20 compliant versions of the `depositToken` and `rewardToken` functions may result in unexpected behavior and potential security issues.

```
31  IERC20 public depositToken;
32  IERC20 public rewardToken;
```

Recommendation

We advise our client to ensure that the tokens set during initialization adhere to the ERC20 standard as implemented in the OpenZeppelin library to avoid any unexpected behavior or security issues.

Alleviation

[**Perion Team**, 01/25/2023]: The team acknowledged the finding and decided to not change the codebase, as the team will only use ERC20 tokens compliant with the project.

TLN-01 | LACK OF STORAGE GAP IN UPGRADEABLE CONTRACT

Category	Severity	Location	Status
Logical Issue	Informational	contracts/TimeLockNonTransferablePool.sol (5cd0cb): 7	Resolved

Description

For upgradeable contracts, there must be a storage gap to "allow developers to freely add new state variables in the future without compromising the storage compatibility with existing deployments". Otherwise, it may be very difficult to write new implementation code. Without a storage gap, the variable in **child contract** might be overwritten by the upgraded base contract if new variables are added to the base contract. Lack of Storage can lead to unexpected behavior during the upgrading of the base contract `TimeLockPool`.

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity 0.8.7;
3
4  import "./TimeLockPool.sol";
5
6  contract TimeLockNonTransferablePool is TimeLockPool {
7      function initialize(
8          string memory _name,
9          string memory _symbol,
10         address _depositToken,
11         address _rewardToken,
12         address _escrowPool,
13         uint256 _escrowPortion,
14         uint256 _escrowDuration,
15         uint256 _maxBonus,
16         uint256 _maxLockDuration,
17         uint _endDate
18     ) public initializer {
19         __TimeLockPool_init(_name, _symbol, _depositToken, _rewardToken,
_escrowPool, _escrowPortion, _escrowDuration, _maxBonus, _maxLockDuration,
_endDate);
20     }
21
22     // disable transfers
23     function _transfer(address _from, address _to, uint256 _amount) internal
override {
24         revert("NON_TRANSFERABLE");
25     }
26 }
```

The auditors wish to inform the team that while there is currently no vulnerability, it may appear in the future if a developer introduces variable parameters inside the `TimeLockNonTransferablePool` contract.

Recommendation

We recommend adding the following code to prevent storage collisions inside the `TimeLockNonTransferablePool` contract.

```
uint256[50] private __gap;
```

Alleviation

[**Perion Team**, 01/27/2023]: The team heeded the advice and resolved the finding by adding a *storage gap* into the contract `TimeLockNonTransferablePool` in the commit 20ce00782871b549bc1ddad9748986463079a729.

[**CertiK**, 01/27/2023]: The `__gap[50]` has been removed from the `BasePool` contract. In this case, when the team wants to upgrade and add new variables they need to add the new variables into `TimeLockNonTransferablePool` contract and update the gap (`__gap[x]`) storage accordingly. Otherwise, this can lead to *storage collision* and unexpected behavior.

TLP-01 | POTENTIAL ISSUES ON REWARDS DISTRIBUTION

Category	Severity	Location	Status
Volatile Code	Informational	contracts/TimeLockPool.sol (5cd0cb): 217	Resolved

Description

According to the design of the `TimeLockPool.sol`, users can deposit tokens and gain "share tokens", which determines how many reward tokens can be received. The reward is distributed by the project that calls `distributeRewards()`, and holding more share tokens means gaining more reward tokens in this round. This design could cause some more profitable strategies to deposit tokens into the pool.

For example, whales may gain the most reward for a short locking time. A whale may maintain a small number of tokens deposited in the pool and call `increaseLock()` to increase the deposit dramatically to gain more shares every time when rewards are going to distribute. This can be achieved by front running. As the `increaseLock()` function does not update the end time of the locking, the whale can leave the pool in a short locking period, thus gaining most of the reward in this round.

Recommendation

We would like to learn more about how the reward will be distributed and ensure the above situation will not cause actual issues to the project.

Alleviation

[**Perion Team**, 1/29/2023]: This is not an issue as the increaseLock function only credits the depositor with the remaining time on their deposit (giving them zero advantage).

```
// Multiplier should be according the remaining time  from the deposit until its
end.
uint256 remainingDuration = uint256(userDeposit.end - block.timestamp);

uint256 mintAmount = (_increaseAmount * getMultiplier(remainingDuration)) / ONE;
```

A whale can wait to increaseLock before the reward distribution but that would provide no advantage and would actually result in them receiving less than if they had just deposited the full amount at the beginning.

We are also disclosing our reward distribution schedule so it will not be a surprise to anyone.

[**Perion Team**, 2/2/2023]: Because the minimum lock time is one month and rewards are distributed at the same time every day. This should be the expected behavior.

TPP-01 | POTENTIAL FAILURE TO EXTEND LOCK PERIOD

Category	Severity	Location	Status
Logical Issue	Informational	TimeLockPool.sol: 175~182	Resolved

Description

In the function `extendLock()`, users can extend their lock period on their deposits. However, in the following scenarios, there's possibility that users may fail to extend their deposit locks.

Scenario

Considering the following scenarios, the related parameters are simplified without losing generality.

Case 1, with only `extendLock()` considered:

1. An user makes his initial deposit by calling the `deposit()` at time 0, with `_amount = 100`, `_duration = 10`, where the maximum duration `maxLockDuration` is assumed as 10 as well, and `endDate` here is 20.
2. Representative tokens are minted based on the deposited `_amount` and the `multiplier`. Further, for simplicity, `maxBonus = 10`,

   $multiplier = 1 + \frac{maxBonus \times duration}{maxLockDuration} = 1 + \frac{10 \times 10}{10} = 11$

   $mintAmount = amount \times multiplier = 10 \times 11 = 110$

   Hence, the user currently has a number of shares equal to 110.
3. Before the lock period expires, which is at time 10, the user could call `extendLock()` to extend the lock period.
4. Assume at time 2, the user calls the function `extendLock()` with `_increaseDuration = 1` to extend the lock to time 11, which is assumed equal to or greater than the `MIN_LOCK_DURATION`.
5. Therefore, the `duration` on L175 is

   $duration = \min(maxLockDuration, depositEnd - currentTime + increaseDuration) = \min(10, 10 - 2 + 1) = 9$

   $mintAmount = amount \times multiplier = 10 \times (1 + \frac{maxBonus \times duration}{maxLockDuration}) = 10 \times (1 + \frac{10 \times 9}{10}) = 10 \times (1 + 9) = 100$
6. Since the new `mintAmount = 100 < userDeposit.shareAmount`, the user can't extend the lock once they had maxed the duration in the past.

Case 2, both `extendLock()` and `increaseLock()` are considered:

1. An user makes his initial deposit by calling the `deposit()` at time 0, with `_amount = 100`, `_duration = 10`, where the maximum duration `maxLockDuration` is assumed as 10 as well, and `endDate` here is 20.

2. Representative tokens are minted based on the deposited `_amount` and the `multiplier`. Further, for simplicity, `maxBonus = 10`,

   $$multiplier = 1 + \frac{maxBonus \times duration}{maxLockDuration} = 1 + \frac{10 \times 10}{10} = 11$$

   $$mintAmount = amount \times multiplier = 10 \times 11 = 110$$

   Hence, the user currently has a number of shares equal to 110.

3. Before the lock period expires, which is at time 5, the user could call `increaseLock()` to increase the locked amount.

4. Assume at time 2, the user calls the function `increaseLock()` with `_increaseAmount = 10`.

5. Therefore, the `remainingDuration = 8` on L227 is

   $$mintAmount = amount \times multiplier = 10 \times (1 + \frac{maxBonus \times remainingDuration}{maxLockDuration}) = 10 \times (1 + \frac{10 \times 9}{10}) = 10 \times (1 + 8) = 90$$

6. `userDeposit.shareAmount = 110 + 90 = 200`.

7. Assume at time 5, the user calls the function `extendLock()` with `_increaseDuration = 1` to extend the lock to time 11, which is assumed equal to or greater than the `MIN_LOCK_DURATION`.

8. Therefore, the `duration` on L175 is

   $$duration = \min(maxLockDuration, depositEnd - currentTime + increaseDuration) = \min(10, 10 - 5 + 1) = 6$$

   $$mintAmount = amount \times multiplier = 20 \times (1 + \frac{maxBonus \times duration}{maxLockDuration}) = 20 \times (1 + \frac{10 \times 6}{10}) = 20 \times (1 + 6) = 140$$

9. Since the new `mintAmount = 140 < userDeposit.shareAmount`, which is 200, the user can't extend the lock even if the increased lock amount at time 2 doesn't have the max duration.

Recommendation

Recommend reconsidering the logic of deposit lock extension implementation to ensure that it's intended.

Alleviation

[**Perion Team**, 02/06/2023]: This is the intended behavior by the project, so the team acknowledged the finding and decided to not change the codebase.

OPTIMIZATIONS | PERION

ID	Title	Category	Severity	Status
CON-02	User-Defined Getters	Gas Optimization	Optimization	Acknowledged
TLP-02	Unused State Variable	Gas Optimization	Optimization	Resolved

CON-02 | USER-DEFINED GETTERS

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/TimeLockPool.sol (5cd0cb): 235~237; contracts/base/AbstractRewards.sol (5cd0cb): 54~56	Acknowledged

Description

The linked functions are equivalent to the compiler-generated getter functions for the respective variables.

```
    function getDepositsOf(address _account) public view returns(Deposit[] memory) {
        return depositsOf[_account];
    }
...
  function withdrawnRewardsOf(address _account) public view override returns
(uint256) {
    return withdrawnRewards[_account];
  }
```

It is a better practice to instead declare the variable as `public` as compiler-generated getter functions are less prone to error and more maintainable than manually written ones.

Recommendation

We advise that the linked variables are instead declared as `public` as compiler-generated getter functions are less prone to error and much more maintainable than manually written ones.

Alleviation

[**Perion Team**, 01/20/2023]: The team acknowledged this finding and decided not to change the codebase at the current stage.

TLP-02 | UNUSED STATE VARIABLE

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/TimeLockPool.sol (5cd0cb): 25	Resolved

Description

The `unit` state variable is never used in the codebase.

```
25      uint256 public unit;
```

Recommendation

We advise removing the unused variables.

Alleviation

[**Perion Team**, 01/25/2023] :

The team heeded the advice and resolved the finding by removing the unused code in the commit hash 6887a9e999f9781ffd32cd7daf51ca03d1929ec7.

APPENDIX | PERION

Finding Categories

Categories	Description
Centralization / Privilege	Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.
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.
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.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

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 | CERTIK

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 you ("Customer" or 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, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

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.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE

FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

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.