**Executive Summary**

**Type**: Yield Farming  
**Auditors**:  
Sebastian Banescu, Senior Research Engineer  
Ed Zulkoski, Senior Security Engineer  
Poming Lee, Research Engineer  

**Timeline**: 2021-05-16 through 2021-06-16  
**EVM**: Berlin  
**Languages**: Solidity  
**Specification**: Tokenomics, Launchpad, and Reward Details, README.md  

**Documentation Quality**: Medium  
**Test Quality**: Medium  
**Source Code**  
<table>
<thead>
<tr>
<th>Repository</th>
<th>Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Illuvium-contracts</td>
<td>68297e2 (audit)</td>
</tr>
<tr>
<td>Illuvium-contracts</td>
<td>98697c5 (reaudit)</td>
</tr>
<tr>
<td>Illuvium-contracts</td>
<td>94807fc (review)</td>
</tr>
</tbody>
</table>

Total Issues: 16  
- 12 Resolved  
- 1 High Risk  
- 2 Medium Risk  
- 6 Low Risk  
- 7 Informational  
- 0 Undetermined  

- **High Risk**: The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client’s reputation or serious financial implications for client and users.  
- **Medium Risk**: The issue puts a subset of users’ sensitive information at risk, would be detrimental for the client’s reputation if exploited, or is reasonably likely to lead to moderate financial impact.  
- **Low Risk**: The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client’s business circumstances.  
- **Informational**: The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.  
- **Undetermined**: The impact of the issue is uncertain.  

- **Unresolved**: Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.  
- **Acknowledged**: The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).  
- **Resolved**: Adjusted program implementation, requirements or constraints to eliminate the risk.  
- **Mitigated**: Implemented actions to minimize the impact or likelihood of the risk.
Summary of Findings

After the first audit: Quantstamp has performed a security audit of the Illuvium yield farming contracts (note that the other contracts in the repositories were not in scope). Several findings indicated below have been identified ranging from High to Undetermined severity levels. Additionally, we have identified issues in the specification, code comments and deviations from best practices. Moreover, we have encountered several failing tests when executing the existing test suite. The errors we encountered are included in this report. We recommend fixing all issues before deploying the code in production.

After the reaudit: We have performed a reaudit, which involved checking the fixes performed by the Illuvium team to address the issues found during the first audit. This report has been updated based on commit hash 98697c3.

Contracts that were in the scope of this audit:

- IlluviumCorePool.sol
- IlluviumFlashPool.sol
- IlluviumLockedPool.sol
- IlluviumPoolBase.sol
- IlluviumPoolFactory.sol
- IlluviumVault.sol
- TokenLocking.sol
- LockedPool.sol

After the 2nd reaudit/review: Quantstamp has reviewed commit 94807fc where the following two smart contracts were added:

- FlashPoolV2.sol
- FlashPoolBase.sol

All concerns raised by the auditors on commit 76843ad have been addressed by the current implementation.

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Severity</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>QSP-1</td>
<td>Uniswap Call Susceptible To Price Manipulation Attacks</td>
<td>High</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-2</td>
<td>IlluviumFlashPool. Does Not Check If Lock Period Has Passed</td>
<td>Low</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-3</td>
<td>Unclear ILV Token Bookkeeping For ILV/ETH Pair Pool</td>
<td>Low</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-4</td>
<td>Potentially Uncounted Rewards</td>
<td>Medium</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-5</td>
<td>Potentially Lost Rewards</td>
<td>Medium</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-6</td>
<td>Total Balances Set Larger Than Intended</td>
<td>Low</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-7</td>
<td>Violation Of Check-Effects-Interactions Pattern</td>
<td>Low</td>
<td>Mitigated</td>
</tr>
<tr>
<td>QSP-8</td>
<td>Missing Or Insufficient Input Validation</td>
<td>Low</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-9</td>
<td>Missing invariant checks</td>
<td>Low</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-10</td>
<td>swapEthForIlv Reverts On Zero ETH Balance</td>
<td></td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-11</td>
<td>Inconsistent Initialization Steps</td>
<td></td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-12</td>
<td>blocksPerUpdate Is Defined In Blocks But Is Expected To Be 2 Weeks</td>
<td></td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-13</td>
<td>Privileged Roles and Ownership</td>
<td></td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-14</td>
<td>Clone-and-Own</td>
<td></td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-15</td>
<td>Unused Functions</td>
<td></td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-16</td>
<td>Misaligned Code And Comments</td>
<td></td>
<td>Fixed</td>
</tr>
</tbody>
</table>
Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrage token minting

Methodology

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.

2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.

3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.8.0

Steps taken to run the tools:

1. Installed the Slither tool: pip install slither-analyzer
2. Run Slither from the project directory: slither .

Findings

QSP-1 Uniswap Call Susceptible To Price Manipulation Attacks

Severity: High Risk

Status: Fixed

File(s) affected: IlluviumVault.sol

Description: The function swapETHForILV uses Uniswap to exchange ETH for ILV tokens. The function is declared public with no access control. If the contract holds a large amount of ETH, attackers can manipulate (likely using flash loans) the Uniswap ETH/ILV price such that the IlluviumVault will receive an unfavorable amount of ILV. Further, computing ilvOut in the function leaves it susceptible to sandwich attacks.

Recommendation: Restrict the function such that only a privileged user can invoke it. Rather than relying on Uniswap to compute the ilvOut value in the function, pre-compute an expected...
QSP-2 IlluviumFlashPool Does Not Check If Lock Period Has Passed

Severity: Low Risk

Status: Acknowledged

File(s) affected: IlluviumFlashPool.sol, IlluviumVault.sol

Description: In IlluviumCorePool.sol on L30, the comment states that poolTokenReserve is the “Total value of ILV tokens available in the pool”. However, while functions such as IlluviumCorePool.receiveVaultRewards only increase poolTokenReserve when poolToken == ilv, this is not the case for functions such as IlluviumCorePool._stake. For example, if the poolToken ILV/ETH Pair, the poolTokenReserve is still increased on L205, even though pair tokens are staked (not ILV). This makes the computation in IlluviumVault.sendIlvRewards of ilvPairsRewards unclear:

```solidity
uint256 ilvInPairPool = (pairPoolReserve.mul(ilv.balanceOf(address(ilvEthPair))).div(ilvEthPair.totalSupply())).add(
    user.subVaultRewards.amount); // popular
```

In particular, the expression `(pairPoolReserve.mul(ilv.balanceOf(address(ilvEthPair))).div(ilvEthPair.totalSupply()))` seems to suggest that poolTokenReserve should store the amount of ILV/ETH Pair tokens in the ILV/ETH Pair pool, not the ILV balance itself. With the current setup, it appears that the `ilvInPairPool` computation above will double-count some tokens, since poolTokenReserve is increased for both ILV and ILV/ETH Pair deposits. This will inflate the weight associated with the ILV/ETH Pair pool.

Recommendation: Clarify the intended semantics of poolTokenReserve for the pair pool.

Update: Based on the following quote from dev team we have decided to change the severity of this issue from Medium to Low:

“poolTokenReserve for LP pool gets updated correctly and doesn’t contain any unpaired ILV. Documentation was improved to better reflect the use of poolTokenReserve; LP pool ILV reserve estimation was extracted into a separate function estimatePairPoolReserve to be more clear. See PR #33”

QSP-3 Unclear ILV Token Bookkeeping For ILV/ETH Pair Pool

Severity: Low Risk

Status: Fixed

File(s) affected: IlluviumCorePool.sol, IlluviumLockedPool.sol

Description: In IlluviumCorePool.sol on L30, the comment states that poolTokenReserve is the “Total value of ILV tokens available in the pool”. However, while functions such as IlluviumCorePool.receiveVaultRewards only increase poolTokenReserve when poolToken == ilv, this is not the case for functions such as IlluviumCorePool._stake. For example, if the poolToken ILV/ETH Pair, the poolTokenReserve is still increased on L205, even though pair tokens are staked (not ILV). This makes the computation in IlluviumVault.sendIlvRewards of ilvPairsRewards unclear:

```solidity
uint256 ilvInPairPool = (pairPoolReserve.mul(ilv.balanceOf(address(ilvEthPair))).div(ilvEthPair.totalSupply())).add(
    user.subVaultRewards.amount); // popular
```

In particular, the expression `(pairPoolReserve.mul(ilv.balanceOf(address(ilvEthPair))).div(ilvEthPair.totalSupply()))` seems to suggest that poolTokenReserve should store the amount of ILV/ETH Pair tokens in the ILV/ETH Pair pool, not the ILV balance itself. With the current setup, it appears that the `ilvInPairPool` computation above will double-count some tokens, since poolTokenReserve is increased for both ILV and ILV/ETH Pair deposits. This will inflate the weight associated with the ILV/ETH Pair pool.

Recommendation: Clarify the intended semantics of poolTokenReserve for the pair pool.

Update: Based on the following quote from dev team we have decided to change the severity of this issue from Medium to Low:

“poolTokenReserve for LP pool gets updated correctly and doesn’t contain any unpaired ILV. Documentation was improved to better reflect the use of poolTokenReserve; LP pool ILV reserve estimation was extracted into a separate function estimatePairPoolReserve to be more clear. See PR #33”

QSP-4 Potentially Uncounted Rewards

Severity: Medium Risk

Status: Fixed

File(s) affected: IlluviumPoolBase.sol

Description: The IlluviumPoolBase._updateStakeLock function does not flush rewards before changing the value of user.totalWeight. This may lead to incorrect reward amounts subsequently.

Recommendation: The IlluviumPoolBase._updateStakeLock function should call _processRewards before changing the weight and update user.subYieldRewards after changing the weight.

Update: Quote from dev team:

“_updateStakeLock synchronizes contract state now and processes rewards before updating stake lock. See PR #34 and PR #50”

QSP-5 Potentially Lost Rewards

Severity: Medium Risk

Status: Fixed

File(s) affected: IlluviumLockedPool.sol, IlluviumCorePool.sol

Description: The _processVaultRewards function inside IlluviumCorePool and in IlluviumLockedPool will not give users the full amount of the reward they are entitled to, when pendingVaultClaim > user.totalVaultReserve. Moreover, the function will also stop the users from requesting for the missing amount afterward. Hence the users will lose rewards.

Exploit Scenario: When the function _processVaultRewards is internally invoked, pending claims are transferred to the _staker using _safeIlvTransfer. However, if the ILV balance of the contract is too low, the statement on L28: `IERC20(ilv).safeTransfer(_to, _amount > ilvBalance ? ilvBalance : _amount);` will only transfer a portion of ILV tokens that should be rewarded to the _staker. However, the user.subVaultRewards will be updated as if the total reward were received (e.g., on L262).

Recommendation: Consider either reverting if the ILV balance is too low, or update the pending balance of the user to reflect the shortage.

Update: Quote from dev team:

“_processVaultRewards reverts now if pool balance is too low. See PR #38”

QSP-6 Total Balances Set Larger Than Intended

Severity: Low Risk

Status: Fixed
Update:

Recommendation:

Note that this is not an exhaustive list. User inputs should always be validated.

Description:

File(s) affected: TokenLocking.sol

QSP-7 Violation Of Check-Effects-Interactions Pattern

Severity: Low Risk

Status: Mitigated

File(s) affected: IlluviumLockedPool.sol

Description: The _stake() and _unstake() functions do not follow the Check-Effects-Interactions pattern, because the call to _processVaultRewards() function, makes a call to the the ILV token contract.

The same issue is also encountered in other functions such as receiveVaultRewards(). However, this is not an exhaustive list.

Recommendation: Always follow the Check-Effects-Interactions pattern to avoid reentrancy. This can be done by moving the call to _processVaultRewards() at the end of the aforementioned functions.

Update: Quote from dev team:

“For the best traceability of external interactions, extracted then into separate reused functions. Protected the functions which operate on a pool tokens with reentrancy guard. See PR #37 and PR #39.”

QSP-8 Missing Or Insufficient Input Validation

Severity: Low Risk

Status: Fixed

File(s) affected: TokenLocking.sol, IlluviumLockedPool.sol, IlluviumCorePool.sol, IlluviumPoolBase.sol

Description: The following instances of missing or insufficient input validation have been encountered:

1. The _pool parameter of the of the TokenLocking.setPool() function is not checked to conform to the IILockedPool interface and could be any address.
2. The _rewardsAmount parameter of the IlluviumLockedPool.receiveVaultRewards() function is not checked to be greater than zero. The same applies to the function with the same name in other contracts.
3. The _from and _to parameters of IlluviumLockedPool.changeLockedHolder() are not checked to be different. This could lead to deleting a holder.
4. The _vault parameter of IlluviumCorePool.setVault() is not checked to be different from address(0).
5. The _weight parameter of IlluviumPoolBase.setWeight() is not checked to be greater than zero.

Note that this is not an exhaustive list. User inputs should always be validated.

Recommendation: The items in the following list correspond to the items in the description:

1. Use EIP-165 to check if the address provided through the _pool input parameter respects the IILockedPool interface.
2. Add a require statement to check that _rewardsAmount > 0.
3. Add a require statement to check that _from != _to.
4. Add a require statement to check that _vault != address(0).
5. Add a require statement to check that _weight > 0.

Update: Quote from dev team:
QSP-9 Missing invariant checks
Severity: Low Risk
Status: Fixed
File(s) affected: IlluviumPoolBase.sol
Description: Assumptions about intermediate values during function processing should be explicitly checked, especially if these values depend on outputs returned by external contract calls. For example, we assume that the value of stakeWeight on line 14 inside of IlluviumPoolBase._stake() should be greater than zero. Otherwise, it doesn’t make sense to create a deposit with stakeWeight == 0.
Note that this is one example of what we assume to be an implicit assumption, however, all implicit assumptions should be checked in a similar way.
Recommendation: Add an assert statement that checks if stakeWeight > 0.
Update: Quote from dev team:
"Missing invariant check added. See PR #45"

QSP-10 swapEthForIlv Reverts On Zero ETH Balance
Severity: Informational
Status: Fixed
File(s) affected: IlluviumVault.sol
Description: If the ETH balance of IlluviumVault is zero (possibly due to a previous call to either swapEthForIlv or sendIlvRewards), the function will revert due to the check balance > 0 on line 159. However, since swapEthForIlv is public, a legitimate call to sendIlvRewards could be griefed by any user if they front-run with a call to swapEthForIlv.
Recommendation: Restrict access to swapEthForIlv as suggested above, or change swapEthForIlv to return immediately upon zero balance rather than reverting.
Update: Quote from dev team:
"Resolved in fix for QSP-1. Additionally altered sendIlvRewards not to swap ETH/ILV if ETH balance is zero. See PR #40"

QSP-11 Inconsistent Initialization Steps
Severity: Informational
Status: Fixed
File(s) affected: TokenLocking.sol
Description: According to the inline documentation, step 2 should invoke setPool, and step 3 sets balances (through potentially multiple calls to setBalances). However, the function setBalances requires on line 23 that address(pool) == address(0), so step 3 cannot occur after step 2.
The steps on lines 17-20 appear correct, but lines 126 and 149 do not align with this summary.
Recommendation: Revise the initialization logic.
Update: Quote from dev team:
"Fixed comments for setPool and setBalances functions. See PR #41"

QSP-12 blocksPerUpdate Is Defined In Blocks But Is Expected To Be 2 Weeks
Severity: Informational
Status: Acknowledged
File(s) affected: IlluviumPoolFactory.sol
Description: Several comments suggest that blocksPerUpdate should equal 2 weeks, but is defined in blocks which have variable mining times. It is not clear why timestamp is not used for this variable instead, particularly since block timestamp manipulation will have minimal effect for such a large timespan.
Recommendation: Use block.timestamp for updates instead of block.number. Note that this would also affect related functions such as IlluviumPoolBase._sync.
Update: Quote from dev team:
"Documentation was improved to explicitly state the blocks are used instead of timestamps. The rationale behind using blocks is to make all mined blocks equal in rewards independently of how much time passes for each block to be mined. See PR #54"

QSP-13 Privileged Roles and Ownership
Severity: Informational
Status: Acknowledged
File(s) affected: TokenLocking.sol, IlluviumPoolFactory.sol
Description: Smart contracts will often have owner variables to designate the person with special privileges to make modifications to the smart contract. The following instances of this issue have been identified:
1. The owner of the TokenLocking contract can perform the following privileged actions:
   - Set the pool address for ILV staking (only once).
   - Set the balances of tokens owned by any address, e.g. pre-seed investors, seed investors, team members, etc. This can be done multiple times.
2. The owner of IlluviumPoolFactory can create/register unlimited pools at will.
Recommendation: This centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the owner.

Update: Quote from dev team:

“For TokenLocking this is part of the initialization process, once it is complete, the owner has no privileged access anymore. For IlluviumPoolFactory an ability to register new pools and set their weights is part of the design. Explicitly added that into thesoldoc. See PR #42”

QSP-14 Clone-and-Own

Severity: Informational
Status: Acknowledged
File(s) affected: utils/*

Description: The clone-and-own approach involves copying and adjusting open source code at one's own discretion. From the development perspective, it is initially beneficial as it reduces the amount of effort. However, from the security perspective, it involves some risks as the code may not follow the best practices, may contain a security vulnerability, or may include intentionally or unintentionally modified upstream libraries.

All files in the utils/ sub-directory are cloned from open source repositories such as openzeppelin.

Recommendation: Rather than the clone-and-own approach, a good industry practice is to use the Truffle framework for managing library dependencies. This eliminates the clone-and-own risks yet allows for following best practices, such as, using libraries.

Update: Quote from dev team:

“There are some solidity files (not only libraries) copied from OpenZeppelin. We intentionally copied these files into the source control system to track any intentional/unintentional modifications which may happen there.”

QSP-15 Unused Functions

Severity: Informational
Status: Fixed
File(s) affected: IlluviumLockedPool.sol

Description: The now256() and blockNumber() functions declared on L235 and L224 in IlluviumLockedPool.sol are never used.

Recommendation: Remove unused functions.

Update: Quote from dev team:

“Removed unused functions, removed unused now256() and blockNumber() functions, removed unused LockedPoolMock contract. See PR #46”

QSP-16 Misaligned Code And Comments

Severity: Informational
Status: Fixed
File(s) affected: IlluviumCorePool.sol

Description: In the IlluviumCorePool._stakeAsPool() function on L155 code comment says that the _useSILV should be false, however it’s not in the case on L165 where the 2nd parameter passed to the _processRewards function, which represents the value of _useSILV is hardcoded to true.

Recommendation: Clarify if the code or the comment needs to be adjusted.

Update: Quote from dev team:

“Both code and comments look correct: when the request to process LP pool rewards without sILV (_useSILV = false) is made by staker, stakeAsPool() gets executed internally. Otherwise (if a request is made to process ILV pool rewards, or _useSILV = true), stakeAsPool() doesn’t get executed. Function comment slightly altered to be clearer. See PR #47”

Automated Analyses

Slither

Slither has output 390 results, the majority of which have been filtered out because they were false positives. The remaining issues have been included in this report.

Adherence to Specification

The code seems to adhere to the existing specification with one exception:

1. [Mitigated] The TokenLocking.md files indicates that:
   - Linear unlocking begins: March 30, 2022, 3PM GMT
   - Linear unlocking ends: March 30, 2023, 3PM GMT

   However, these dates are not hard-coded in the smart contract. Instead, the contract is left generic and any cliff and duration can be provided when the contract is deployed. Therefore, we recommend that users check the values of the public cliff and duration state variables of the TokenLocking contract after it has been deployed. In order to verify if the dates have been set correctly.

Update from dev team: Yes, this is part of the deployment scripts, which are also provided and not to be modified.

Additionally, due to gaps in the documentation we have the following open questions:

1. [Fixed] In IlluviumLockedPool.sol, function _processVaultRewards: please confirm if sending all the rewards to the msg.sender immediately instead of having any sort of time lock, is intended by design.
What are the intended values of endBlock? Will it be a short duration or is it larger than the 1 year staking period?

If a user stakes their tokens immediately before the endBlock, are they effectively locking their tokens for free, since _sync will be disabled immediately after?

Should _stake be disabled after the pool is disabled?

**Code Documentation**

1. Good inline documentation.

2. **[Fixed]** It is not fully clear why the local variable TokenLocking.setBalances.totalAmount was created, but it is presumably useful for determining how many tokens the administrators should deposit into the contract. However, it should be noted that if TokenLocking.setBalances is called multiple times, this amount could be misleading as existing balances may exist or be overwritten.

3. **[Fixed]** The comment in IlluviumVault.sol on L105: "Creates (displays) IlluviumVault linked to IlluviumVaultPool..." does not appear correct, as no pool is set in the constructor.

4. **[Fixed]** The comment on L30 of IlluviumCorePool.sol: "// @dev Total value of ILV tokens available in the pool" does not appear correct. The token may not be ILV, but could be ILV/ETH pair tokens instead.

5. **[Fixed]** In IlluviumPoolBase.sol on L583, inline comments should mention that the constant 266 relates to the bonus weight for looking for a full year.

6. **[Fixed]** In TokenLocking.sol L18: "setBeneficiaries" should be changed into "setBalances".

7. **[Fixed]** The off-chain procedure regarding how a holder is able to obtain a signature from the TokenLocking owner or how/where to send such a signature such that the migration is initiated by the owner is not clear. This should be clearly documented.

8. **[Mitigated]** Some code comments indicate concrete values which are not enforced in the code. For example, the comment on L200 in TokenLocking.sol states "will be set during deployment by migration script." The comment on L30 of IlluviumCorePool.sol indicates: "check if blocks/update (2 weeks) have passed since last update". However, the value of blockPerUpdate can be set to any value in the constructor().

**Adherence to Best Practices**

1. **[Fixed]** Since the TokenLocking.release function does not allow the user to specify where the tokens will be unlocked to. The event TokensReleased event has 3 parameters: by, to and amount and is emitted only once on L292 with the first 2 parameters having the same value. It is unclear why both these parameters are needed if they are never different.

2. **[Fixed]** Nested ternary expressions without any code alignment should be avoided. For example L436 contains such an expression without any parentheses which makes it hard to audit and maintain.

3. **[Fixed]** TokenLocking.setBalances should check that each holder and amount is non-zero.

4. **[Fixed]** Magic numbers should be avoided in code and replaced with named constants which provide a semantic meaning and don’t just indicate the constant’s value. For example:

   The value 1e12 appears twice in the IlluviumLockedPool contract and it is unclear why this value is used and what it represents.

   The value 266 appears on L67 in IlluviumCorePool and on L583 in IlluviumPoolBase and it is unclear what it represents.

   The value 1e6 appears multiple times in IlluviumPoolBase and it is unclear what it represents.

5. **[Acknowledged]** There are minor inconsistencies between the IlluviumCorePool and IlluviumLockedPool, such as the core pool using vaultRewardsPerToken as opposed to vaultRewardsPerToken, as it is used SafeMath is not needed.

**Test Results**

**Test Suite Results**

After reaudit: Several failing tests have been encountered when running the existing test suite. We provide the output of the test suite, including the error details below.

**After reaudit:** The dev team has indicated that the failing tests are due to a known issue in Truffle. Running the test files individually helps reduce the probability of failing tests. However, failing tests might still be encountered at seemingly random points.
when locked tokens are distributed
− when locked tokens are distributed without a pool attached
− when locked tokens are distributed with a pool attached
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalidated
− when locked tokens are distributed when tokens are invalid
when no one stakes
linear unlocking routine(s)
holder 3 releases linearly:
t = t2
old address of the migrated holder is no longer registered as a beneficiary
✓ new address of the migrated holder gets appended to lockedHolders
0/365 of the tokens are released
✓ release reverts (372ms)
t = t2 + 1 second
✓ holder 1 ILV balance is 0/50.741k
✓ release reverts (65ms)
✓ userRecord.ilvReleased is zero
✓ userRecord.ilvBalance is 93.363k
✓ userRecord.hasStaked is false
✓ userRecord.ilvReleased is zero
✓ userRecord.ilvBalance is zero
✓ userRecord.ilvReleased is 0/365 of the initial stake (0/927.593k)
✓ userRecord.ilvBalance is 365/365 of the initial stake (927.593k/927.593k)
t = t2 + 1 block
0.000011574074074074073/365 of the tokens are released
✓ userRecord.ilvReleased is 0/365 of the initial stake (0/513.432k)
✓ userRecord.ilvBalance is 365/365 of the initial stake (513.432k/513.432k)
t = t2 + 1 day
0.00017361111111111112/365 of the tokens are released
✓ userRecord.hasStaked is false
✓ userRecord.ilvReleased is 0/365 of the initial stake (0/513.432k)
✓ userRecord.ilvBalance is 364.99998842592595/365 of the initial stake (513.432k/513.432k)
t = t2 + 1 week
1/365 of the tokens are released
1/365 of the tokens are released
✓ holder 2 ILV balance is 0.244212407562/513.432k (40ms)
✓ userRecord.hasStaked is false
✓ userRecord.ilvBalance is 364.99998842592595/365 of the initial stake (513.432k/513.432k)
t = t2 + 1 month
7/365 of the tokens are released
7/365 of the tokens are released
✓ holder 1 ILV balance is 0.024135100984/50.741k
✓ userRecord.hasStaked is false
✓ userRecord.ilvBalance is 364.99998842592595/365 of the initial stake (50.741k/50.741k)

hold 2 release interval

l = -15
release events (15ms)

l = -12
release events (15ms)

l = -8
release events (15ms)

l = -4
release events (15ms)

l = -1
release events (15ms)

l = 0
release events (15ms)

l = 1
release events (15ms)

l = 2
release events (15ms)

l = 3
release events (15ms)

l = 4
release events (15ms)

l = 5
release events (15ms)

l = 6
release events (15ms)

l = 7
release events (15ms)

l = 8
release events (15ms)

l = 9
release events (15ms)

l = 10
release events (15ms)

l = 11
release events (15ms)

l = 12
release events (15ms)

l = 13
release events (15ms)

l = 14
release events (15ms)

l = 15
release events (15ms)

hold 2 release interval

l = 16
release events (15ms)

l = 17
release events (15ms)

l = 18
release events (15ms)

l = 19
release events (15ms)

l = 20
release events (15ms)

hold 2 release interval

l = 21
release events (15ms)

l = 22
release events (15ms)

l = 23
release events (15ms)

l = 24
release events (15ms)

l = 25
release events (15ms)

l = 26
release events (15ms)

l = 27
release events (15ms)

l = 28
release events (15ms)

l = 29
release events (15ms)

l = 30
release events (15ms)

l = 31
release events (15ms)

l = 32
release events (15ms)

hold 3 release interval

l = 33
release events (15ms)

l = 34
release events (15ms)

l = 35
release events (15ms)

l = 36
release events (15ms)

l = 37
release events (15ms)

l = 38
release events (15ms)

l = 39
release events (15ms)

l = 40
release events (15ms)

l = 41
release events (15ms)

l = 42
release events (15ms)

hold 3 release interval

l = 43
release events (15ms)

l = 44
release events (15ms)

l = 45
release events (15ms)

l = 46
release events (15ms)

l = 47
release events (15ms)

l = 48
release events (15ms)

l = 49
release events (15ms)

l = 50
release events (15ms)

l = 51
release events (15ms)

l = 52
release events (15ms)
...
Code Coverage

After audit: Due to the failing tests, the coverage values could not be accurately computed and have resulted in low values as indicated in the table below.

After reaudit: Coverage values have been increased. However, the branch coverage is not sufficiently high at 66%. We recommend increasing this value as close to 100% as possible.

<table>
<thead>
<tr>
<th>File</th>
<th>% Stmts</th>
<th>% Branch</th>
<th>% Funcs</th>
<th>% Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>IlluviumAware.sol</td>
<td>100</td>
<td>50</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>IlluviumCorePool.sol</td>
<td>100</td>
<td>76.92</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>IlluviumFlashPool.sol</td>
<td>88.89</td>
<td>66.67</td>
<td>100</td>
<td>88.89</td>
</tr>
<tr>
<td>IlluviumLockedPool.sol</td>
<td>98</td>
<td>61.54</td>
<td>95.33</td>
<td>96.08</td>
</tr>
<tr>
<td>IlluviumPoolBase.sol</td>
<td>96.35</td>
<td>70</td>
<td>87.5</td>
<td>96.32</td>
</tr>
<tr>
<td>IlluviumPoolFactory.sol</td>
<td>76.19</td>
<td>54.17</td>
<td>70</td>
<td>76.19</td>
</tr>
<tr>
<td>IlluviumVault.sol</td>
<td>98.18</td>
<td>61.11</td>
<td>100</td>
<td>98.21</td>
</tr>
<tr>
<td>ReentrancyGuard.sol</td>
<td>100</td>
<td>50</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>TokenLocking.sol</td>
<td>91.89</td>
<td>69.64</td>
<td>85.71</td>
<td>92.11</td>
</tr>
<tr>
<td>All files</td>
<td>94.44</td>
<td>66.25</td>
<td>89.89</td>
<td>94.25</td>
</tr>
</tbody>
</table>

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

eb6172a9a2b7f018d6cb06b6725627720bed5daef8ff0b6a61ed7a49f72852c //FlashPoolV2.sol
eef65f18b3f5e5d6408f12b6977915d849a1c3649f363ab188ee8b8c0d2f68 //FlashPoolBase.sol
410a81b185a0b376476f9d90f67e158defe1e69af6e36be6a496535a19b3d7 //contracts/Migrations.sol
e2476f84281f6e34c97be71d70c133f022e38821351ba29b622ae1 //contracts/interfaces/IERC20.sol
35181c8847b0a8e3163bbe1a9b0280d59a3a675fca8191ff3b55a5a1b4 //contracts/interfaces/ICorePool.sol
effd2b820645f88867692d2b540c74c7e166f3a31fcb2a7e63a7e738 //contracts/interfaces/ILockedPool.sol
2be8e8ff81c4a65432cc6b9fd049a686868201c7de6722feb4e56f7548d6b4b5d2bf //contracts/interfaces/IPool.sol
7ae014468647a7b859ad2d16b3a2b22a784e2a187e3b01c15c1f12f82c9e1b92 //contracts/interfaces/IVaultReceiver.sol
Tests
Changelog

- 2021-05-28 - Initial report based on commit hash 68297e2
- 2021-06-16 - Updated report based on commit hash 98697c5
- 2021-08-14 - Updated report based on commit hash 948807c
About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp's team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp’s collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp, Inc. (Quantstamp). Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on the websites and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.