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

Methods

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>
</tbody>
</table>

Total Issues
16 (12 Resolved)

High Risk Issues
1 (1 Resolved)

Medium Risk Issues
2 (2 Resolved)

Low Risk Issues
6 (5 Resolved)

Informational Risk Issues
7 (4 Resolved)

Undetermined Risk Issues
0 (0 Resolved)

- **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**

- **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 `98697c5`.

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

<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>Informational</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-11</td>
<td>Inconsistent Initialization Steps</td>
<td>Informational</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>Informational</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-13</td>
<td>Privileged Roles and Ownership</td>
<td>Informational</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-14</td>
<td>Clone-and-Own</td>
<td>Informational</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-15</td>
<td>Unused Functions</td>
<td>Informational</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-16</td>
<td>Misaligned Code And Comments</td>
<td>Informational</td>
<td>Fixed</td>
</tr>
</tbody>
</table>
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
- Arbitrarily 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 is 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 ilvInPairPool unclear:

```
uint256 ilvOut = swapEthForIlv(msg.sender, ilvInPairPool);
```

In particular, the expression `pairPoolReserve.mul(1lv.balanceOf(address(1lvEthPair))).div(1lvEthPair.totalSupply())` seems to suggest that pairPoolReserve 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 `swapEthForIlv` computation above will double-count some tokens, since `pairPoolReserve` 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:

"Flash pools don’t lock tokens by design. Documentation was improved to address the confusion. See PR #35".

QSP-3 Unclear ILV Token Bookkeeping For ILV/ETH Pair Pool
Severity: Low Risk
Status: Fixed
File(s) affected: IlluviumCorePool.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 is 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 `ilvInPairPool` unclear:

```
uint256 ilvOut = swapEthForIlv(msg.sender, ilvInPairPool);
```

With the current setup, it appears that the `swapEthForIlv` computation above will double-count some tokens, since `pairPoolReserve` 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 #36".

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 #4 and PR #57".

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 > poolTokenReserve. Moreover, the function will also stop the users from requesting for the missing amount afterward. Hence the users will lose rewards.

Exploit Scenarios: 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 L285: `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 reward 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
File(s) affected: TokenLocking.sol

Description: The TokenLocking.setBalances() function may be called multiple times in order to "allow setting balance to zero in case of accidental addition of the holder". However, due to missing checks/assertions there exists a possibility for human error which could lead to unlocking a total balance larger than intended.

Exploit Scenario: For the sake of simplicity let's assume that the total amount of locked ILV tokens should be 100. The TokenLocking contract owner performs the following actions:

1. Sets the balances of 2 holders by calling setBalances:
   - Holder1's balance is set to 70 ILV, using address 0x111
   - Note that there are no checks in the smart contract to verify that:
     - The total amount, which is equal to 110 ILV is greater than the intended total, which is 100 ILV.
   - The same address appears twice in the holders array.

2. Notices that the address and amount used for Holder2 was wrong and sets it again by calling setBalances:
   - Holder2's balance is set to 40 ILV, using address 0x111
   - Note that at this point there exist 2 holders 0x111 that has 40 ILV and 0x222 that has 30 ILV, which is again incorrect. This is possible because the setBalances() function does not keep track of holders whose balances were set in previous calls to setBalances().

Recommendation: The following countermeasures should be implemented to mitigate this issue:

1. Keep track of holders whose balances were set in previous calls to setBalances() by storing them in a list that can be iterated.
2. Whenever setBalances() is called check that the sum of all balances set (including the balances set in previous calls to this function AND which were not modified by the current call) is equal to the expected total amount, that is 3.8 million ILV (18 decimals).
3. Check that there are no duplicate addresses in the holders input argument.

Note that as part of the fix it shouldn't be necessary to assume that the value of holders is always the same for each call, because if the list is too long then this function might revert with an out-of-gas error.

Update: Quote from dev team:

*"Added duplicate holders check; added total expected balance check; added previously set holders cleanup. setBalances to be used to set/update balances in a single transaction (up to 100 balances setup fit into 4.5mil gas)." See [PR #37](https://github.com/Illuvium/contracts/pull/37)

QSP-7 Violation Of Check-Effects-Interactions Pattern

Severity: Low Risk

Status: Mitigated

File(s) affected: IlluviumLockedPool.sol

Description: The _staking and _unstaking 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 issues 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 them into separate reused functions. Protected the functions which operate on a pool tokens with reentrancy guard. See transferPoolToken in IlluviumPoolBase, see [PR #39](https://github.com/Illuvium/contracts/pull/39) and [PR #40](https://github.com/Illuvium/contracts/pull/40)*

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 ILockedPool 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](https://eips.ethereum.org/eips/eip-165) to check if the address provided through the _pool input parameter respects the ILockedPool 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:
*Introduced validations for ILV and sILV tokens, pool factory. Contracts are to be deployed with a well-tested script, which enforces correctness of the addresses set. changeLockedHolder() is called only by TokenLocking which validates the inputs. setWeight should allow zero input by design to disable the pool (added.sol). See PR #45*  

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 420 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 #51"

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 an ILV address `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.number` for updates instead of `blocksPerUpdate`. 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 the soldoc. 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 lines 235 and 224 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 line 155 code comment says that the _useSILV should be false, however it’s not in the case on line 165 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 #49”

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.
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 (deploys) IlluviumVault linked to IlluviumYieldPool..." 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 2e6 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` indicates: "check if blocks/update (2 weeks) have passed since last update". However, the value of `blockPartupdate` can be set to any value in the `constructor()`.

Update: Will be set during deployment by migration script.

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 L262 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: `now256 = now256 < cliff ? cliff : now256 > duration ? cliff + duration : now256;`. Also the comment on L435 is vague as it refers to "safe bounds". It should be explicitly indicated what those bounds are. We recommend using nested if-then-else statements and adding more precise comments.

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 26e6 appears on L167 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 `vaultRewardsPerWeight` as opposed to `vaultRewardsPerToken`.

   Update from dev team: "This is intended by design since core pools allow staking for different time intervals as opposed to the locked pool. That’s where the "weight" comes into play: it reflects the difference in the period tokens are locked for."

6. [Fixed] SafeMath is used interchangeably with normal arithmetic symbols throughout, however since it Solidity 0.8 is used SafeMath is not needed.

7. Event parameters with type `address` should be indexed. The following deviations from this best practice were identified:

   - [Fixed] L19 in `TokenLocking.sol` where `poolAddr` parameter of the `PoolUpdated` event is not indexed.

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.
Contract: TokenLocking and its flows (excluding staking)

- Contract: IlluviumPoolFactory
- Contract: IlluviumLockedPool
- Contract: IlluviumFlashPool

when token locking TokenLocking is deployed without a pool attached

- unix timestamp –> Date conversion
- ✓ should mint exact amount of ILV during yield farming (135702ms)
- ✓ should revert on unauthorized pool weight change (6245ms)
- ✓ should correctly change a given pool weight (2795ms)
- ✓ should revert on invalid ilv per block update (2420ms)
- ✓ should correctly update ilv per block (3962ms)
- ✓ should calculate pool reserve correctly (1937ms)
- ✓ should unstake locked tokens correctly (1094ms)
- ✓ should stake locked tokens correctly (97ms)
- ✓ should process and lock ilv yield rewards correctly (3644ms)
- ✓ should mint sILV correctly (1614ms)
- ✓ should always set total weight (2732ms)
- ✓ should unstake correctly (2056ms)
- ✓ should create a flash pool correctly (1539ms)
- ✓ should not accumulate yield after yield farming ends (2560ms)
- ✓ should calculate pending rewards correctly after bigger stakes (3893ms)
- ✓ should calculate pending rewards correctly for multiple users (4631ms)
- ✓ should calculate pending rewards correctly (1890ms)
- ✓ should process and lock ilv yield rewards correctly (1295ms)
- ✓ should correctly set users locking weight (4203ms)
- ✓ should correctly set last yield distribution (3070ms)
- ✓ should not accumulate rewards before init block (200k)

- ✓ unlocking formula gives 6341958396752917 out of 200k in 1 sec
- ✓ unlocking formula gives 3170979198376458 out of 100k in 1 sec

- Wed Mar 30 2022 18:00:00 GMT+0300 (Eastern European Summer Time) converts to 1648652400

- ✓ TokensMigrated event is emitted when balances are first set to [40, 70, 0]

- ✓ lockedHolders array is [a1, a2] (89ms)
- ✓ lockedHolders array is [a1, a3] (78ms)
- ✓ lockedHolders array is [a2, a3] (67ms)
- ✓ lockedHolders array is [a2, a4] (56ms)

- ✓ operation fits into a single block
- ✓ account 1 balance gets set to 60 and not 70
- ✓ account 2 balance gets set to 60 and not 70

- ✓ lockedHolders array is [a2, a3] (89ms)
- ✓ lockedHolders array is [a2, a4] (56ms)

- ✓ lockedHolders array is [a3, a4] (56ms)
- ✓ lockedHolders array is [a4, a5] (45ms)

- ✓ lockedHolders array is [a3, a4, a5] (44ms)
- ✓ lockedHolders array is [a1, a2, a3] (34ms)

- ✓ lockedHolders array is [a1, a2, a3, a4] (23ms)
- ✓ lockedHolders array is [a1, a2, a3, a4, a5] (12ms)

- ✓ lockedHolders array is [a1, a2, a3, a4, a5, a6] (11ms)
- ✓ lockedHolders array is [a1, a2, a3, a4, a5, a6, a7] (10ms)

- ✓ lockedHolders array is [a1, a2, a3, a4, a5, a6, a7, a8] (9ms)
- ✓ lockedHolders array is [a1, a2, a3, a4, a5, a6, a7, a8, a9] (8ms)

- ✓ lockedHolders array is [a1, a2, a3, a4, a5, a6, a7, a8, a9, a10] (7ms)
- ✓ lockedHolders array is [a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11] (6ms)

- ✓ lockedHolders array is [a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12] (5ms)
- ✓ lockedHolders array is [a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12, a13] (4ms)

- ✓ lockedHolders array is [a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12, a13, a14] (3ms)
- ✓ lockedHolders array is [a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12, a13, a14, a15] (2ms)

- ✓ lockedHolders array is [a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12, a13, a14, a15, a16] (1ms)
- ✓ lockedHolders array is [a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12, a13, a14, a15, a16, a17] (0ms)
when no one stakes

holder 3 releases linearly:

holder 2 releases linearly:

holder 1 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

✓ holder 3 ILV balance is 0/927.593k (358ms)

✓ release reverts (372ms)

0/365 of the tokens are released

✓ holder 2 ILV balance is 0.01628082717/513.432k (42ms)

✓ userRecord.ilvReleased is 0/365 of the initial stake (0/513.432k)

✓ 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

✓ holder 2 ILV balance is 0.244212407562/513.432k (40ms)

✓ userRecord.ilvReleased is 0.000011574074074074073/365 of the initial stake (0.01628082717/513.432k)

✓ userRecord.ilvBalance is 364.99998842592595/365 of the initial stake (513.432k/513.432k)

t = t2 + 1 day

0.00017361111111111112/365 of the tokens are released

✓ holder 2 ILV balance is 1.406k/513.432k (44ms)

✓ userRecord.ilvReleased is 0.00017361111111111112/365 of the initial stake (0.244212407562/513.432k)

✓ userRecord.ilvBalance is 364.9998263888889/365 of the initial stake (513.431k/513.432k)

t = t2 + 1 week

1/365 of the tokens are released

✓ holder 2 ILV balance is 9.846k/513.432k (235ms)

✓ userRecord.hasStaked is false

✓ userRecord.ilvReleased is 1/365 of the initial stake (1.406k/513.432k)

✓ userRecord.ilvBalance is 364/365 of the initial stake (512.025k/513.432k)

t = t2 + 1 month

7/365 of the tokens are released

✓ holder 2 ILV balance is 9.846k/513.432k (235ms)

✓ userRecord.hasStaked is false

✓ userRecord.ilvReleased is 1/365 of the initial stake (1.406k/513.432k)

✓ userRecord.ilvBalance is 364/365 of the initial stake (512.025k/513.432k)

t = t2 + 1 month

✓ holder 1 ILV balance is 139.018181671232/50.741k (135ms)

✓ userRecord.hasStaked is false

✓ userRecord.ilvReleased is 1/365 of the initial stake (139.018181671232/50.741k)

✓ userRecord.ilvBalance is 364/365 of the initial stake (50.741k/50.741k)

30/365 of the tokens are released

✓ userRecord.ilvBalance is 30/365 of the initial stake (42.199k/513.432k)

✓ holder 1 ILV balance is 49.768k/50.741k

✓ userRecord.hasStaked is false

✓ userRecord.ilvReleased is 30/365 of the initial stake (4.17k/50.741k)

✓ userRecord.ilvBalance is 335/365 of the initial stake (50.741k/50.741k)

358/365 of the tokens are released

✓ userRecord.ilvBalance is 30/365 of the initial stake (42.199k/513.432k)

✓ holder 1 ILV balance is 49.768k/50.741k

✓ userRecord.hasStaked is false

✓ userRecord.ilvReleased is 358/365 of the initial stake (503.585k/513.432k)

✓ userRecord.ilvBalance is 7/365 of the initial stake (9.846k/513.432k)

364/365 of the tokens are released

✓ holder 1 ILV balance is 49.768k/50.741k

✓ userRecord.hasStaked is false

✓ userRecord.ilvReleased is 364/365 of the initial stake (50.602k/50.741k)

✓ userRecord.ilvBalance is 7/365 of the initial stake (973.12727169863/50.741k)

364/365 of the tokens are released

✓ holder 1 ILV balance is 49.768k/50.741k

✓ userRecord.hasStaked is false

✓ userRecord.ilvReleased is 364/365 of the initial stake (50.602k/50.741k)

✓ userRecord.ilvBalance is 7/365 of the initial stake (973.12727169863/50.741k)

all the tokens are released

✓ holder 2 ILV balance is 513.432k/513.432k (167ms)

✓ userRecord.hasStaked is false

✓ userRecord.ilvReleased is 364.99998842592595/365 of the initial stake (513.432k/513.432k)

all the tokens are released

✓ holder 1 ILV balance is 50.741k/50.741k

✓ userRecord.hasStaked is false

✓ userRecord.ilvReleased is 364.99998842592595/365 of the initial stake (50.741k/50.741k)

✓ userRecord.ilvBalance is zero (0/513.432k)

✓ userRecord.hasStaked is false

✓ userRecord.ilvBalance is zero (0/50.741k)
Simulation starting
Staking '0.01' into ILV and ILV/ETH LP pools to init them
finalizing TokenLocking setup (locked balances, locked pool setup)
prepared 32 accounts with '320k' total ILV. deploying yield farming infrastructure

'10.00516042' ILV left in the vault/pools

All tokens released. '30' reward sent. '320.019k' ILV released.

Simulation complete.

Day 1134:

Day 879:

Day 327:

Day 3:

'0.0017054' ILV left in the vault/pools

Simulation complete.
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>96.3%</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 changed condition or potential vulnerability that was not within the scope of the review.

Contracts
4105ab1853403746235df9d9f76f150e1e694a4e63fe0a68e9653a19a431d7 ./contracts/Migrations.sol
e247e60428165e34c9b7bdec17f0eb97bcb13eb224ed38821351b29bcb22ae1 ./contracts/interfaces/IERC20.sol
35181c81febe88e186ba19b2880df5a3a67f565ca8192f3bf354eb31a43b4 ./contracts/interfaces/ICorePool.sol
effd2042c465887625b9979357de774e7ccb11f3c3cf73 ./contracts/interfaces/ILockedPool.sol
2b08f818c141543285cf0b199a68e82b10d71d6e72d2b9e45624464db45d2b8a ./contracts/interfaces/IERC1155.sol
c10e7757a7c79ea6e3b976e664646e8e0efd4d41c32830526787e7c0bad1d7ae ./contracts/interfaces/IERC721.sol
7c5b4c6c8eb29a78f7d2a983cee4e82bb66f9b451c3f8a9e1e86277f70ad54a5 ./contracts/interfaces/INFTClaimManager.sol
eef05bdf6074a0bf2e28d4b5940dc1c689c9bcb486c0e82666161e19bfc ./contracts/interfaces/IWETH.sol
284a9eb7102b1258d4eacc4576f522a5c3a3577b9f8131e6a6c8e1d9b49ec ./contracts/interfaces/IERC20Mintable.sol
9917a791e8eb355a15c5bc11f1653db0cc97262ab154f0315130c9573ae06e1c ./contracts/interfaces/IPool.sol
7ae0f144686847a7953a2d163b32a22a470e1a6948a4b8845b05d587d ./contracts/interfaces/IVaultReceiver.sol
aa54f88eb1a56a4ac247e52a5d0b87a2d778eb6eb85140bbda8c03791 ./contracts/interfaces/IDisperse.sol
d51869f89e19f487558f18937674d7df30eb62d3870ad2c25d8b019618c3bcb8 ./contracts/locking_erc20/__/NonLockingListener.sol
Changelog

- 2021-05-28 - Initial report based on commit hash 68297a2
- 2021-06-16 - Updated report based on commit hash 98697c5
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.