Quantstamp Security Assessment Certificate

June 30th 2021 — Quantstamp Verified

Illuvium Yield Farming Rewards

This smart contract audit was prepared by Quantstamp, the leader in blockchain security.

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	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Tokenomics, Launchpad, and Reward Details README.md
Documentation Quality	Medium
Test Quality	Medium

Source Code

Repository	Commit
illuvium-contracts	68297e2 (audit)
illuvium-contracts	98697c5 (reaudit)
illuvium-contracts	94807fc (review)

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	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 `98697c5`.

Contracts that were in the scope of this audit:

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

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

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
- Arbitrary 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.

Exploit Scenario: The following steps describe how this issue could be exploited:

1. The attacker causes an imbalance in the Uniswap pool by either increasing the amount of ETH or decreasing the amount of ILV in the pool (possibly utilizing a flash loan).
2. The attacker invokes `swapEthForIlv`. The `IlluviumVault` will receive a lower than market-value amount of ILV due to the imbalance in step 1.
3. The attacker buys back the ETH from step 1 at a favorable price (paying back the flash loan if needed).

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

amount (accounting for a small amount of slippage) and pass it into the function. This additional parameter may be avoided by using an independent price oracle for the ILV token if such an oracle is available.

Update: Added `ilvOut` and `deadline` as input parameters to `swapEthForIlv`, made function access restricted. See PR #34

QSP-2 `IlluviumFlashPool` Does Not Check If Lock Period Has Passed

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `IlluviumFlashPool.sol`

Description: The `IlluviumPoolBase._unstake` function (inherited by `IlluviumFlashPool`) does not check if `block.timestamp > lockedUntil` of a deposit. The `IlluviumCorePool` adds a check in its overridden function `_unstake`, however, `IlluviumFlashPool` does not override the `_unstake` function.

Recommendation: Check that `lockedUntil` is in the past for a deposit that is unstaked.

Update: Based on the following quote from dev team we have decided to change the severity of this issue from High 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 ilvInPairPool =
    (pairPoolReserve.mul(ilv.balanceOf(address(ilvEthPair))).div(ilvEthPair.totalSupply())).add(
        pairPoolIlvBalance
    );
```

In particular, the expression `(pairPoolReserve.mul(ilv.balanceOf(address(ilvEthPair))).div(ilvEthPair.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 `ilvInPairPool` 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 #44 and PR #55"

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 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 L281: `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`
    . Holder2's balance is set to 40 ILV, using address `0x111`
    . **Note** that there are no checks in the smart contract to verify that:
        · The `totalAmount`, 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 30 ILV, using address `0x222`
    . **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

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 `transferPoolToken*` in `IlluviumPoolBase`, see PR #39 and PR #43"

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 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 soldoc). 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 L420 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 L159. 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 L164 that `address(pool) == address(0)`, so step 3 cannot occur after step 2.
The steps on L17-20 appear correct, but L126 and L149 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 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 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 (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 locking 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 `IlluviumPoolFactory.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()`.
   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 L242 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 - cliff > 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 `2e6` 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]** L94 in `TokenLocking.sol` where `poolAddr` parameter of the `PoolUpdated` event is not `indexed`.

Test Results

Test Suite Results

After audit: 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.

```
Using network 'test'.


Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.

[initial migration] local network - skipping the migration script
[ILV ERC20 deployment] local network - skipping the migration script
[Token Locking deployment] local network - skipping the migration script
[pools deployment] local network - skipping the migration script


  Contract: IlluviumCorePool
     ✓ should correctly create a core pool (828ms)
     ✓ should stake correctly (755ms)
     ✓ should unstake correctly (2624ms)
     ✓ should revert on invalid unstake (3159ms)
```

```
    ✓ should not accumulate rewards before init block (1622ms)
    ✓ should correctly set last yield distribution (3070ms)
    ✓ should update stake lock correctly (1515ms)
    ✓ should revert on invalid update stake lock (2048ms)
    ✓ should correctly set users locking weight (4203ms)
    ✓ should mint sILV correctly (1407ms)
    ✓ should mint ILV correctly (3253ms)
    ✓ should process and lock ilv yield rewards correctly (1295ms)
    ✓ should calculate pending rewards correctly (1890ms)
    ✓ should calculate pending rewards correctly for multiple users (4631ms)
    ✓ should calculate pending rewards correctly after bigger stakes (3893ms)
    ✓ should not accumulate yield after yield farming ends (2560ms)

Contract: IlluviumFlashPool
    ✓ should create a flash pool correctly (1539ms)
    ✓ should unstake correctly (2056ms)
    ✓ should always set total weight (2732ms)
    ✓ should not accumulate rewards after pool is disabled (1755ms)
    ✓ should mint sILV correctly (1614ms)
    ✓ should process and lock ilv yield rewards correctly (3644ms)

Contract: IlluviumLockedPool
    ✓ should stake locked tokens correctly (97ms)
    ✓ should unstake locked tokens correctly (1094ms)
    ✓ should allow only token locking calls (96ms)
    ✓ should calculate pool reserve correctly (1937ms)

Contract: IlluviumPoolFactory
    ✓ should create core pools correctly (2642ms)
    ✓ should create flash pools correctly (11639ms)
    ✓ should correctly update ilv per block (3962ms)
    ✓ should revert on invalid ilv per block update (2420ms)
    ✓ should correctly change a given pool weight (2795ms)
    ✓ should revert on unauthorized pool weight change (6245ms)
    ✓ should mint exact amount of ILV during yield farming (135702ms)

Contract: TokenLocking and its flows (excluding staking)
  unix timestamp <-> Date conversion
     ✓ Wed Mar 30 2022 18:00:00 GMT+0300 (Eastern European Summer Time) converts to 1648652400
  when token locking TokenLocking is deployed without a pool attached
    ✓ unlocking formula gives 3170979198376458 out of 100k in 1 sec
    ✓ unlocking formula gives 6341958396752917 out of 200k in 1 sec
    ✓ unlocking formula gives 9512937595129375 out of 300k in 1 sec
  token locking state variables are initialized correctly
     ✓ owner is a0
     ✓ duration is t3 - t2
     ✓ cliff is t2
     ✓ cliff is 1648652400
     ✓ ILV token address (ilv) is set correctly
     ✓ locking pool is not set (staking is not supported)
  setting the balances
    when pool is not set
      when expected total is correct
        when there are no duplicate holders
          when t < t2
            when t = t1
               ✓ succeeds (252ms)
            when t = t2 - 1
               ✓ succeeds (99ms)
          when t ⩾ t2
            when t = t2
               ✓ reverts (104ms)
            when t = t2 + 1
               ✓ reverts (87ms)
            when t = t3
               ✓ reverts (70ms)
        when there are duplicate holders
          when t < t2
            when t = t1
               ✓ reverts (109ms)
            when t = t2 - 1
               ✓ reverts (598ms)
          when t ⩾ t2
            when t = t2
               ✓ reverts (712ms)
            when t = t2 + 1
               ✓ reverts (674ms)
            when t = t3
               ✓ reverts (352ms)
      when expected total is not correct
        when t < t2
          when t = t1
             ✓ reverts (333ms)
          when t = t2 - 1
             ✓ reverts (102ms)
        when t ⩾ t2
          when t = t2
             ✓ reverts (196ms)
          when t = t2 + 1
             ✓ reverts (717ms)
          when t = t3
             ✓ reverts (930ms)
    when pool is set
      when t < t2
        when t = t1
           ✓ reverts (382ms)
        when t = t2 - 1
           ✓ reverts (94ms)
      when t ⩾ t2
        when t = t2
           ✓ reverts (150ms)
        when t = t2 + 1
           ✓ reverts (131ms)
        when t = t3
           ✓ reverts (71ms)
  resetting the balances
    when balances are first set to [40, 70, 0]
       ✓ lockedHolders array is [a1, a2] (78ms)
      when balances are then set to [0, 60, 30]
         ✓ lockedHolders array is [a2, a3] (89ms)
         ✓ account 1 balance gets erased (46ms)
         ✓ account 2 balance gets set to 60 and not 70
         ✓ account 3 balance gets set to 30 (67ms)
    when 100 balances are set for the first time
4495075 gas is used
       ✓ operation fits into a single block
      when 150 balances are set next
4556668 gas is used
         ✓ operation fits into a single block
  when locked tokens are allocated
     ✓ staking fails for holder 1 (no locking pool) (85ms)
     ✓ holder 1 ILV balance is zero
     ✓ staking fails for holder 2 (no locking pool) (55ms)
     ✓ holder 2 ILV balance is zero
     ✓ staking fails for holder 3 (no locking pool) (52ms)
     ✓ holder 3 ILV balance is zero (57ms)
    holder 1 is registered as a beneficiary:
       ✓ userRecord.ilvBalance is 50.741k
       ✓ userRecord.ilvReleased is zero
       ✓ userRecord.hasStaked is false
    holder 2 is registered as a beneficiary:
       ✓ userRecord.ilvBalance is 513.432k
       ✓ userRecord.ilvReleased is zero
       ✓ userRecord.hasStaked is false
    holder 3 is registered as a beneficiary:
       ✓ userRecord.ilvBalance is 927.593k
       ✓ userRecord.ilvReleased is zero
       ✓ userRecord.hasStaked is false
    migrateWithSig (locked tokens migration)
      when a1 is overridden with an account with known key - migration
         ✓ reverts if the signature is invalid (68ms)
         ✓ reverts if the nonce is bad  (72ms)
         ✓ reverts if the signature has expired (184ms)
         ✓ reverts if the signature is wrong (414ms)
         ✓ reverts when performed by low privileged user pair (153ms)
       old (overridden) address of the migrated holder is registered as a beneficiary
          ✓ userRecord.ilvBalance is 502.734k
          ✓ userRecord.ilvReleased is zero
          ✓ userRecord.hasStaked is false
       new address of the migrated holder is not registered as a beneficiary
          ✓ userRecord.ilvBalance is zero
          ✓ userRecord.ilvReleased is zero
          ✓ userRecord.hasStaked is false
       succeeds when performed by an admin on user request
          ✓ TokensMigrated event is emitted
          ✓ new address of the migrated holder gets appended to lockedHolders (440ms)
         old address of the migrated holder is no longer registered as a beneficiary
            ✓ userRecord.ilvBalance is zero
            ✓ userRecord.ilvReleased is zero
            ✓ userRecord.hasStaked is false
         new address of the migrated holder is registered as a beneficiary
            ✓ userRecord.ilvBalance is 502.734k
            ✓ userRecord.ilvReleased is zero
            ✓ userRecord.hasStaked is false
       succeeds when performed by a user on admin request
          ✓ TokensMigrated event is emitted
```

```
        ✓ new address of the migrated holder gets appended to lockedHolders
      old address of the migrated holder is no longer registered as a beneficiary
         ✓ userRecord.ilvBalance is zero
         ✓ userRecord.ilvReleased is zero
         ✓ userRecord.hasStaked is false
      new address of the migrated holder is registered as a beneficiary
         ✓ userRecord.ilvBalance is 93.363k
         ✓ userRecord.ilvReleased is zero
         ✓ userRecord.hasStaked is false
when no one stakes
  linear unlocking routine(s)
    holder 1 releases linearly:
      t = t2
         ✓ release reverts (65ms)
         ✓ holder 1 ILV balance is 0/50.741k
        0/365 of the tokens are released
           ✓ userRecord.ilvBalance is 365/365 of the initial stake (50.741k/50.741k)
           ✓ userRecord.ilvReleased is 0/365 of the initial stake (0/50.741k)
           ✓ userRecord.hasStaked is false
        t = t2 + 1 second
           ✓ holder 1 ILV balance is 0.001609006732/50.741k (96ms)
          0.000011574074074074073/365 of the tokens are released
             ✓ userRecord.ilvBalance is 364.99998842592595/365 of the initial stake (50.741k/50.741k)
             ✓ userRecord.ilvReleased is 0.000011574074074074073/365 of the initial stake (0.001609006732/50.741k)
             ✓ userRecord.hasStaked is false
          t = t2 + 1 block
             ✓ holder 1 ILV balance is 0.024135100984/50.741k
            0.000173611111111111112/365 of the tokens are released
               ✓ userRecord.ilvBalance is 364.9998263888889/365 of the initial stake (50.741k/50.741k)
               ✓ userRecord.ilvReleased is 0.000173611111111111112/365 of the initial stake (0.024135100984/50.741k)
               ✓ userRecord.hasStaked is false
            t = t2 + 1 day
               ✓ holder 1 ILV balance is 139.018181671232/50.741k (135ms)
              1/365 of the tokens are released
                 ✓ userRecord.ilvBalance is 364/365 of the initial stake (50.602k/50.741k)
                 ✓ userRecord.ilvReleased is 1/365 of the initial stake (139.018181671232/50.741k)
                 ✓ userRecord.hasStaked is false
              t = t2 + 1 week
                 ✓ holder 1 ILV balance is 973.12727169863/50.741k
                7/365 of the tokens are released
                   ✓ userRecord.ilvBalance is 358/365 of the initial stake (49.768k/50.741k)
                   ✓ userRecord.ilvReleased is 7/365 of the initial stake (973.12727169863/50.741k)
                   ✓ userRecord.hasStaked is false
                t = t2 + 1 month
                   ✓ holder 1 ILV balance is 4.17k/50.741k
                  30/365 of the tokens are released
                     ✓ userRecord.ilvBalance is 335/365 of the initial stake (46.571k/50.741k)
                     ✓ userRecord.ilvReleased is 30/365 of the initial stake (4.17k/50.741k)
                     ✓ userRecord.hasStaked is false
                  t = t3 - 1 month
                     ✓ holder 1 ILV balance is 46.571k/50.741k (258ms)
                    335/365 of the tokens are released
                       ✓ userRecord.ilvBalance is 30/365 of the initial stake (4.17k/50.741k)
                       ✓ userRecord.ilvReleased is 335/365 of the initial stake (46.571k/50.741k)
                       ✓ userRecord.hasStaked is false
                    t = t3 - 1 week
                       ✓ holder 1 ILV balance is 49.768k/50.741k
                      358/365 of the tokens are released
                         ✓ userRecord.ilvBalance is 7/365 of the initial stake (973.12727169863/50.741k)
                         ✓ userRecord.ilvReleased is 358/365 of the initial stake (49.768k/50.741k)
                         ✓ userRecord.hasStaked is false
                      t = t3 - 1 day
                         ✓ holder 1 ILV balance is 50.602k/50.741k
                        364/365 of the tokens are released
                           ✓ userRecord.ilvBalance is 1/365 of the initial stake (139.018181671232/50.741k)
                           ✓ userRecord.ilvReleased is 364/365 of the initial stake (50.602k/50.741k)
                           ✓ userRecord.hasStaked is false
                        t = t3 - 1 block
                           ✓ holder 1 ILV balance is 50.741k/50.741k (404ms)
                          364.9998263888889/365 of the tokens are released
                             ✓ userRecord.ilvBalance is 0.000173611111110858474/365 of the initial stake (0.024135100984/50.741k)
                             ✓ userRecord.ilvReleased is 364.9998263888889/365 of the initial stake (50.741k/50.741k)
                             ✓ userRecord.hasStaked is false
                          t = t3 - 1 second
                             ✓ holder 1 ILV balance is 50.741k/50.741k (119ms)
                            364.99998842592595/365 of the tokens are released
                               ✓ userRecord.ilvBalance is 0.000011574074051168282/365 of the initial stake (0.001609006732/50.741k)
                               ✓ userRecord.ilvReleased is 364.99998842592595/365 of the initial stake (50.741k/50.741k)
                               ✓ userRecord.hasStaked is false
                            t = t3
                               ✓ holder 1 ILV balance is 50.741k/50.741k
                              all the tokens are released
                                 ✓ userRecord.ilvBalance is zero (0/50.741k)
                                 ✓ userRecord.ilvReleased is equal to initial stake (50.741k/50.741k)
                                 ✓ userRecord.hasStaked is false
holder 2 releases linearly:
  t = t2
     ✓ release reverts (178ms)
     ✓ holder 2 ILV balance is 0/513.432k (64ms)
    0/365 of the tokens are released
       ✓ userRecord.ilvBalance is 365/365 of the initial stake (513.432k/513.432k)
       ✓ userRecord.ilvReleased is 0/365 of the initial stake (0/513.432k)
       ✓ userRecord.hasStaked is false
    t = t2 + 1 second
       ✓ holder 2 ILV balance is 0.01628082717/513.432k (42ms)
      0.000011574074074074073/365 of the tokens are released
         ✓ userRecord.ilvBalance is 364.99998842592595/365 of the initial stake (513.432k/513.432k)
         ✓ userRecord.ilvReleased is 0.000011574074074074073/365 of the initial stake (0.01628082717/513.432k)
         ✓ userRecord.hasStaked is false
      t = t2 + 1 block
         ✓ holder 2 ILV balance is 0.244212407562/513.432k (40ms)
        0.000173611111111111112/365 of the tokens are released
           ✓ userRecord.ilvBalance is 364.9998263888889/365 of the initial stake (513.431k/513.432k)
           ✓ userRecord.ilvReleased is 0.000173611111111111112/365 of the initial stake (0.244212407562/513.432k)
           ✓ userRecord.hasStaked is false
        t = t2 + 1 day
           ✓ holder 2 ILV balance is 1.406k/513.432k (44ms)
          1/365 of the tokens are released
             ✓ userRecord.ilvBalance is 364/365 of the initial stake (512.025k/513.432k)
             ✓ userRecord.ilvReleased is 1/365 of the initial stake (1.406k/513.432k)
             ✓ userRecord.hasStaked is false
          t = t2 + 1 week
             ✓ holder 2 ILV balance is 9.846k/513.432k (235ms)
            7/365 of the tokens are released
               ✓ userRecord.ilvBalance is 358/365 of the initial stake (503.585k/513.432k)
               ✓ userRecord.ilvReleased is 7/365 of the initial stake (9.846k/513.432k)
               ✓ userRecord.hasStaked is false
            t = t2 + 1 month
               ✓ holder 2 ILV balance is 42.199k/513.432k (78ms)
              30/365 of the tokens are released
                 ✓ userRecord.ilvBalance is 335/365 of the initial stake (471.232k/513.432k)
                 ✓ userRecord.ilvReleased is 30/365 of the initial stake (42.199k/513.432k)
                 ✓ userRecord.hasStaked is false
              t = t3 - 1 month
                 ✓ holder 2 ILV balance is 471.232k/513.432k (152ms)
                335/365 of the tokens are released
                   ✓ userRecord.ilvBalance is 30/365 of the initial stake (42.199k/513.432k)
                   ✓ userRecord.ilvReleased is 335/365 of the initial stake (471.232k/513.432k)
                   ✓ userRecord.hasStaked is false
                t = t3 - 1 week
                   ✓ holder 2 ILV balance is 503.585k/513.432k (347ms)
                  358/365 of the tokens are released
                     ✓ userRecord.ilvBalance is 7/365 of the initial stake (9.846k/513.432k)
                     ✓ userRecord.ilvReleased is 358/365 of the initial stake (503.585k/513.432k)
                     ✓ userRecord.hasStaked is false
                  t = t3 - 1 day
                     ✓ holder 2 ILV balance is 512.025k/513.432k (510ms)
                    364/365 of the tokens are released
                       ✓ userRecord.ilvBalance is 1/365 of the initial stake (1.406k/513.432k)
                       ✓ userRecord.ilvReleased is 364/365 of the initial stake (512.025k/513.432k)
                       ✓ userRecord.hasStaked is false
                    t = t3 - 1 block
                       ✓ holder 2 ILV balance is 513.431k/513.432k (457ms)
                      364.9998263888889/365 of the tokens are released
                         ✓ userRecord.ilvBalance is 0.000173611111110858474/365 of the initial stake (0.244212407562/513.432k)
                         ✓ userRecord.ilvReleased is 364.9998263888889/365 of the initial stake (513.431k/513.432k)
                         ✓ userRecord.hasStaked is false
                      t = t3 - 1 second
                         ✓ holder 2 ILV balance is 513.432k/513.432k
                        364.99998842592595/365 of the tokens are released
                           ✓ userRecord.ilvBalance is 0.000011574074051168282/365 of the initial stake (0.01628082717/513.432k)
                           ✓ userRecord.ilvReleased is 364.99998842592595/365 of the initial stake (513.432k/513.432k)
                           ✓ userRecord.hasStaked is false
                        t = t3
                           ✓ holder 2 ILV balance is 513.432k/513.432k (167ms)
                          all the tokens are released
                             ✓ userRecord.ilvBalance is zero (0/513.432k)
                             ✓ userRecord.ilvReleased is equal to initial stake (513.432k/513.432k)
                             ✓ userRecord.hasStaked is false
holder 3 releases linearly:
  t = t2
     ✓ release reverts (372ms)
     ✓ holder 3 ILV balance is 0/927.593k (358ms)
    0/365 of the tokens are released
       ✓ userRecord.ilvBalance is 365/365 of the initial stake (927.593k/927.593k)
       ✓ userRecord.ilvReleased is 0/365 of the initial stake (0/927.593k)
```

```
      ✓ userRecord.hasStaked is false
    t = t2 + 1 second
      ✓ holder 3 ILV balance is 0.029413801208/927.593k
      0.000011574074074074073/365 of the tokens are released
        ✓ userRecord.ilvBalance is 364.99998842592595/365 of the initial stake (927.593k/927.593k)
        ✓ userRecord.ilvReleased is 0.000011574074074074073/365 of the initial stake (0.029413801208/927.593k)
        ✓ userRecord.hasStaked is false
      t = t2 + 1 block
        ✓ holder 3 ILV balance is 0.441207018132/927.593k
        0.000173611111111111112/365 of the tokens are released
          ✓ userRecord.ilvBalance is 364.9998263888889/365 of the initial stake (927.593k/927.593k)
          ✓ userRecord.ilvReleased is 0.000173611111111111112/365 of the initial stake (0.441207018132/927.593k)
          ✓ userRecord.hasStaked is false
        t = t2 + 1 day
          ✓ holder 3 ILV balance is 2.541k/927.593k (186ms)
          1/365 of the tokens are released
            ✓ userRecord.ilvBalance is 364/365 of the initial stake (925.052k/927.593k)
            ✓ userRecord.ilvReleased is 1/365 of the initial stake (2.541k/927.593k)
            ✓ userRecord.hasStaked is false
          t = t2 + 1 week
            ✓ holder 3 ILV balance is 17.789k/927.593k (300ms)
            7/365 of the tokens are released
              ✓ userRecord.ilvBalance is 358/365 of the initial stake (909.804k/927.593k)
              ✓ userRecord.ilvReleased is 7/365 of the initial stake (17.789k/927.593k)
              ✓ userRecord.hasStaked is false
            t = t2 + 1 month
              ✓ holder 3 ILV balance is 76.24k/927.593k (109ms)
              30/365 of the tokens are released
                ✓ userRecord.ilvBalance is 335/365 of the initial stake (851.353k/927.593k)
                ✓ userRecord.ilvReleased is 30/365 of the initial stake (76.24k/927.593k)
                ✓ userRecord.hasStaked is false
              t = t3 - 1 month
                ✓ holder 3 ILV balance is 851.353k/927.593k
                335/365 of the tokens are released
                  ✓ userRecord.ilvBalance is 30/365 of the initial stake (76.24k/927.593k)
                  ✓ userRecord.ilvReleased is 335/365 of the initial stake (851.353k/927.593k)
                  ✓ userRecord.hasStaked is false
                t = t3 - 1 week
                  ✓ holder 3 ILV balance is 909.804k/927.593k
                  358/365 of the tokens are released
                    ✓ userRecord.ilvBalance is 7/365 of the initial stake (17.789k/927.593k)
                    ✓ userRecord.ilvReleased is 358/365 of the initial stake (909.804k/927.593k)
                    ✓ userRecord.hasStaked is false
                  t = t3 - 1 day
                    ✓ holder 3 ILV balance is 925.052k/927.593k (391ms)
                    364/365 of the tokens are released
                      ✓ userRecord.ilvBalance is 1/365 of the initial stake (2.541k/927.593k)
                      ✓ userRecord.ilvReleased is 364/365 of the initial stake (925.052k/927.593k)
                      ✓ userRecord.hasStaked is false
                    t = t3 - 1 block
                      ✓ holder 3 ILV balance is 927.593k/927.593k (167ms)
                      364.9998263888889/365 of the tokens are released
                        ✓ userRecord.ilvBalance is 0.000173611111110858474/365 of the initial stake (0.441207018132/927.593k)
                        ✓ userRecord.ilvReleased is 364.9998263888889/365 of the initial stake (927.593k/927.593k)
                        ✓ userRecord.hasStaked is false
                      t = t3 - 1 second
                        ✓ holder 3 ILV balance is 927.593k/927.593k
                        364.99998842592595/365 of the tokens are released
                          ✓ userRecord.ilvBalance is 0.000011574074051168282/365 of the initial stake (0.029413801208/927.593k)
                          ✓ userRecord.ilvReleased is 364.99998842592595/365 of the initial stake (927.593k/927.593k)
                          ✓ userRecord.hasStaked is false
                        t = t3
                          ✓ holder 3 ILV balance is 927.593k/927.593k (302ms)
                          all the tokens are released
                            ✓ userRecord.ilvBalance is zero (0/927.593k)
                            ✓ userRecord.ilvReleased is equal to initial stake (927.593k/927.593k)
                            ✓ userRecord.hasStaked is false

  Contract: TokenLocking Sim
prepared 32 accounts with '320k' total ILV. deploying yield farming infrastructure
finalizing TokenLocking setup (locked balances, locked pool setup)
Staking '0.01' into ILV and ILV/ETH LP pools to init them
Simulation starting
Day 350:
        staked:   '*             *                '
        released: '                               '
Day 776:
        staked:   '*             *                '
        released: '       *   ** **  *** ***    * *'
Day 1099:
        staked:   '*             *                '
        released: '* * ** * **** ** ********    * *'
Simulation complete.
All tokens released. '30' reward sent. '320.019k' ILV released.
        Staked:   '*             *                '
        Balances: '!*************!*****************'
'10.00516042' ILV left in the vault/pools
     ✓ evolve from t1 to t4 = t3 + 1 year (low complexity) (47644ms)
prepared 32 accounts with '320k' total ILV. deploying yield farming infrastructure
finalizing TokenLocking setup (locked balances, locked pool setup)
Staking '0.01' into ILV and ILV/ETH LP pools to init them
Simulation starting
Day 3:
        staked:   '*       *     ***              '
        released: '                               '
Day 327:
        staked:   '*     * *     ***              '
        released: '                               '
Day 879:
        staked:   '*     * *     ***              '
        released: '         * *       *  **   * * *'
Day 1134:
        staked:   '*     * *     ***              '
        released: ' * **    * * **  * *  **   * * *'
Simulation complete.
All tokens released. '20' reward sent. '320.019k' ILV released.
        Staked:   '*     * *     ***              '
        Balances: '!*****!*!*****!!!***************'
'0.0017054' ILV left in the vault/pools
     ✓ evolve from t1 to t4 = t3 + 1 year [ @skip-on-coverage ] (58222ms)

  Contract: TokenLocking and its flows (including staking)
    when yield farming infrastructure is deployed
      deployment looks correct (rough integrity check)
        ✓ uniswapV2Router02.WETH is expected WETH9
        ✓ uniswapV2Router02.factory is expected UniswapV2Factory (513ms)
        ✓ uniswapV2Factory.getPair(ILV, WETH) is expected pair ILV/ETH
        ✓ pair.factory is expected UniswapV2Factory
        ✓ pair.token0 is ILV
        ✓ pair.token1 is WETH
        ✓ pair.totalSupply is √XY (53ms)
        ✓ zero address balance is √XY - 1000
        ✓ H0 pair balance is √XY - 1000 (436ms)
        ✓ ILV_Pool.ilv is ILV
        ✓ ILV_Pool.sIlv is sILV (38ms)
        ✓ ILV_Pool.factory is expected PoolFactory (41ms)
        ✓ ILV_Pool.poolToken is ILV (183ms)
        ✓ ILV_Pool.vault is expected Vault
        ✓ ILV_Pool ILV balance is zero (59ms)
        ✓ LP_Pool.ilv is ILV (369ms)
        ✓ LP_Pool.sIlv is sILV (77ms)
        ✓ LP_Pool.factory is expected PoolFactory (75ms)
        ✓ LP_Pool.poolToken is ILV/ETH Pair
        ✓ LP_Pool.vault is expected Vault (562ms)
        ✓ LP_Pool ILV balance is zero (145ms)
        ✓ LockedPool.ilv is ILV
        ✓ LockedPool.tokenLocking is expected TokenLocking (685ms)
        ✓ LockedPool.vault is expected Vault (72ms)
        ✓ LockedPool ILV balance is zero (329ms)
        ✓ TokenLocking.cliff is t2 (79ms)
        ✓ TokenLocking.duration is 1 year (179ms)
        ✓ TokenLocking.ilv is ILV
        ✓ TokenLocking.pool is not set (290ms)
        ✓ TokenLocking ILV balance is zero (58ms)
        ✓ Vault.ilv is ILV (389ms)
        ✓ Vault.uniswap is expected UniswapV2Router02
        ✓ Vault ILV balance is zero (67ms)
      Vault.pools is set
        ✓ pools.ilvPool is expected ILV CorePool
        ✓ pools.pairPool is expected LP CorePool
        ✓ pools.lockedPool is expected LockedPool
    when TokenLocking setup is finalized (locked tokens issued, vault attached)
      setup looks correct
        ✓ TokenLocking.pool is expected LockedPool
        ✓ TokenLocking ILV balance is a sum of the locked balances
        ✓ TokenLocking.userRecord.hasStaked is false for all holders (933ms)
      when holders 1 and 2 stake
        staking can be observed
          ✓ LockedPool.poolTokenReserve gets increased by total amount
          for holder 1
            ✓ TokenLocking.TokensStaked event gets emitted correctly
            ✓ LockedPool.Staked event gets emitted correctly (169ms)
            ✓ TokenLocking.userRecord.hasStaked is updated to true (533ms)
            ✓ ILV.Transferred event doesn't get emitted (121ms)
          LockedPool.User record gets created properly
            ✓ user.tokenAmount is equal to staked
```

```
              ✓ user.subVaultRewards is zero
            LockedPool._processVaultRewards is not executed
              ✓ LockedPool.VaultRewardsClaimed event doesn't get emitted
          for holder 2
             ✓ TokenLocking.TokensStaked event gets emitted correctly
             ✓ LockedPool.Staked event gets emitted correctly
             ✓ TokenLocking.userRecord.hasStaked is updated to true
             ✓ ILV.Transferred event doesn't get emitted
            LockedPool.User record gets created properly
              ✓ user.tokenAmount is equal to staked
              ✓ user.subVaultRewards is zero
            LockedPool._processVaultRewards is not executed
              ✓ LockedPool.VaultRewardsClaimed event doesn't get emitted
          total stake amount doesn't get transferred from TokenLocking to LockedPool
             ✓ ILV.Transferred event doesn't get emitted
            LockedPool.User record gets created properly
              ✓ user.tokenAmount is equal to staked
              ✓ user.subVaultRewards is zero
            LockedPool._processVaultRewards is not executed
              ✓ LockedPool.VaultRewardsClaimed event doesn't get emitted
          total stake amount doesn't get transferred from TokenLocking to LockedPool
             ✓ TokenLocking balance remains (296ms)
             ✓ LockedPool balance remains zero (89ms)
        when vault rewards are distributed
          pools receive rewards
             ✓ ILV_Pool 0.0000999 (1821ms)
             ✓ LP_Pool 0.0009996 (152ms)
             ✓ LockedPool 2.9989002 (252ms)
             ✓ Vault 3e-7 (520ms)

Contract: IlluviumVault
   ✓ should correctly swap eth for ilv (804ms)
   ✓ should revert on swap without eth (109ms)
   ✓ should revert on zero ILV swap (1029ms)
   ✓ should revert on expired swap (635ms)
   ✓ should correctly set core pools (1469ms)
   ✓ should correctly send vault rewards to core pools (7309ms)
   ✓ should properly distribute vault rewards to staker (12145ms)
   ✓ should properly distribute vault rewards to multiple stakers (10497ms)
   ✓ should correctly set poolTokenReserve for pools (13482ms)
   ✓ should properly distribute vault rewards using stake() (6102ms)


331 passing (51m)
```

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.

File	% Stmts	% Branch	% Funcs	% Lines
IlluviumAware.sol	100%	50%	100%	100%
IlluviumCorePool.sol	100%	76.92%	100%	100%
IlluviumFlashPool.sol	88.89%	66.67%	100%	88.89%
IlluviumLockedPool.sol	98%	61.54%	93.33%	96.08%
IlluviumPoolBase.sol	96.35%	70%	87.5%	96.32%
IlluviumPoolFactory.sol	76.19%	54.17%	70%	76.19%
IlluviumVault.sol	98.18%	61.11%	100%	98.21%
ReentrancyGuard.sol	100%	50%	100%	100%
TokenLocking.sol	91.89%	69.64%	85.71%	92.11%
All files	**94.44%**	**66.25%**	**89.89%**	**94.25%**

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

```
ebb172aa0a2b701fdacb09bb725627720bed3def08ff060e561ed7a43972852c ./FlashPoolV2.sol
1c653f8b38e35fe50640ff22ba779215d8831c6346c3634b1ab88e88b7cd2fc8 ./FlashPoolBase.sol
4105ab18554b037462f5d9dd9e76103dfe1e694afa63fe060ae9653a19a3b1d7 ./contracts/Migrations.sol
e247e60428165e34c9b7bdecf7f0e9b7bdc133b224ed38821351ba29bc622ae1 ./contracts/interfaces/IERC20.sol
33181c04f1be0e88e8163bae190c280fd59a3a675fca8191fbb3fb554e03a1b4 ./contracts/interfaces/ICorePool.sol
effd2024364368480656697 2be54bd2380c74cfeb1d6fa3b1f1cb2ae763ea738 ./contracts/interfaces/ILockedPool.sol
2be0ff818c14a134320c5fb0199e860201ac7d672d20e945684864bdb4b5d2ba ./contracts/interfaces/IERC1155.sol
c1e0f757a7c7eb8ea63b9976de66468c8e0ffd441c32830528778c7c0a1d77ae ./contracts/interfaces/IERC721.sol
7cb5b4cfb9c4eb09a7bdf2a9a836ac4e82b66f9b45123f8e1e86027f70d5d45a ./contracts/interfaces/INFTClaimManager.sol
ef05bdf6074a0bf2e28d4b5940dcfc609e9cbf3c8c406f0208e82666116d19bc ./contracts/interfaces/IWETH.sol
284a49b71012bc58d64eca4576f522a01c31a5757b0f81331e6aecb1ed1b49ec ./contracts/interfaces/IERC20Mintable.sol
99f7a79ff8e8a0635ae15c50cf11f630bbc9f826ab15480313130c9575ae06c1 ./contracts/interfaces/IPool.sol
7ae0144686847a705dad2d16b3a2b2a20fae1184e2aa1dbb48ea954a0d56057d ./contracts/interfaces/IVaultReceiver.sol
```

aa54f80eb1a568a64cc24742ce56d56b07a2d27860eb56514bbcd6a8ccc03791 ./contracts/interfaces/IDisperse.sol

d51869f89e3f94f07558f18917647d5f30e62ad03001d2c2d58d019618c3bc04 ./contracts/locking_erc20/__NopLockingListener.sol

f05c1bbf51d18f2e9fccd26c344a5291c5fe5d655a8b29932372b92d5cdea354 ./contracts/locking_erc20/__LockingERC20.sol

5f8e954aa9773d07bb443c4cb1c7c596f4f3cd51af09c2ada4f575abe6c71d67 ./contracts/locking_erc20/LockingListener.sol

05593b2d41b9325f70bc430b93c7ef948598d8a8b011e9600e35570f51af31f6 ./contracts/locking_erc20/LockingERC20.sol

b8e1f5b09cd26de89741351b44fa6502b3901ba09b82bdf8275e1f2e56ca553c ./contracts/utils/ERC20.sol

a20e8915751d5dc88469c76d9f1a8583d0fca63ace73d58dbc0f23dcd783341c ./contracts/utils/Ownable.sol

4593bae74a40321d74e1ff2fdd19ce503fa7da14dcf05ffeaf4c59f25b46ef89 ./contracts/utils/SafeERC20.sol

1d90a734c956271ac90b7c46ac29aad359cad8f5e361498bc3092ccd920517e4 ./contracts/utils/SafeMath.sol

c626c394d1c3f1ebb4c3fb278d54d53dc28259d3848b965bb3d7b40d87d1afe6 ./contracts/utils/Address.sol

8f1fdee146be83108fb7cf8d0f54c88d72998da3198fe725bece06bef65703a2 ./contracts/utils/AddressUtils.sol

9e06ea99bbad80a7c86c3b5ed2d3c67b1fde4445f693146072a78e3a121e9c41 ./contracts/utils/AccessControl.sol

b263443181dba6c3e9e0a411b0bc75a8734eb04b68c40da7042d300ad613c708 ./contracts/pools/IlluviumCorePool.sol

9bb1a9ff913604b356f2599dde7a1995297c387f690adea6469229696fe6bbef ./contracts/pools/IlluviumVault.sol

63577bfc42ecb7e0fd93ca57880fee94d0b7f7b9de4a66bdc7064c3725edcc67 ./contracts/pools/IlluviumPoolBase.sol

0b9f9a14f3095313b405ceb4530cb57222bf1793d00290d48f443ffaef388e3d ./contracts/pools/IlluviumPoolFactory.sol

91b3696813c093d6be3eeba607e7d06c5fffd8b32b6921a40f86bbd6656d94c3 ./contracts/pools/IlluviumLockedPool.sol

facb218ff7fcfcff633361e6f9f09144136911594d950e6786561d3d9fe6694c ./contracts/pools/IlluviumFlashPool.sol

ccf6fc0d61caa9928b5c9c5198146b7708e2371426a16b6ad461f5c583f10d75 ./contracts/pools/TokenLocking.sol

82e2af4fb2dde5653f665a06775b937fe84eb2be0e4bab3bc9e8deb2fd265d26 ./contracts/token/ERC721Receiver.sol

65c2398dd19a042f445611bd5c5b008d720c035ed33b71b61725dce3e83aade3 ./contracts/token/EscrowedIlluviumERC20.sol

e0b24aaa9d62977fd7a1572b2ece1bba7a27921a55cc022624a67f5f59b90f33 ./contracts/token/IlluviumERC20.sol

d653a70303ca168c629a9bb437988f8021ba47170ca1a2c924706dd644ac9cbc ./contracts/token/ERC20Receiver.sol

2e70b68a4d93e6169ee6432701b6029d0ffb86265c4ff6d520bdca0530365ef0 ./contracts/mocks/PoolFactoryMock.sol

d39b33cbd013483b16f4f09b4499e51fd019da71ee5aa4521d02b908845b0746 ./contracts/mocks/__NopReceiver.sol

60448cc58d667f12175c39186755dbca0ad5ffa3a2baff26e02a3dcd18ef04a6 ./contracts/mocks/__IlluviumERC20.sol

44e3e206260ba5247d86cf3108563c1860efca3e90ecc13bb5c4c1665dc9952b ./contracts/mocks/ERC20Mock.sol

42a2483f3ef3340a389354c282ac4a62109c071306100fb2f0328d335dedab5e ./contracts/mocks/FlashPoolMock.sol

60098e2b5f6e1eb7ca7ed4b0eed83dc2a303e43d4fd623f40466129edc7f7b52 ./contracts/mocks/TokenLockingMock.sol

549608bcc2d8c736bac84cc49a9feb70b0ed2f213670adb3eea0680ab16f7402 ./contracts/mocks/CorePoolMock.sol

8a04c15c30356270bc860907e0ae52168408c4f88598879503b73035ec02e1fc ./contracts/mocks/LockedPoolMock.sol

Tests

49345091fae96d988b96cd527f0c88c3dead607e5a17f03aa01960ef30dc553a ./test/ilv_vault_yield.js

0f12d61518183174dbdf254bd8a14459a24ab09f56fce268a3d2b43c52a098de ./test/locking_erc20/locking_erc20_locking.js

60ab2fa1de46bbb0fddbcc9f019d11892c0f78f9976be13f2692c9097a29fb9c ./test/locking_erc20/locking_erc20_zeppelin.js

aca568ab7cb6f80af3e53334a1f6c25ca4d9c9a5e39d1b82ec63986030501883 ./test/locking_erc20/locking_erc20_evolution.js

ee6e7a8620fa47e9e88fbffdfa50e46b4a4c777f463ce79c53c66b619803c59e ./test/locking_erc20/locking_erc20_transfers.js

c628b9104510578f627c199b7c341da030d49a95f7b068cd99af6806d03b8ec8 ./test/locking_erc20/locking_erc20_unit.js

c54533124099d73395a666abd09aa306accf3d7780b5194cb0dd2a62284b343d ./test/locking_erc20/include/locking_erc20_dates.js

33ab58c45936c8fbeafd0d15057aec1452ddb96d4486ce100a8f7385ad122074 ./test/locking_erc20/include/locking_erc20_features_roles.js

0c747c1977d49c3d1a8a723bcf3ea78dd682fb78c2d8db27375365f83547d2ef ./test/locking_erc20/include/locking_erc20_constants.js

5b1da9a4397c98add0161ca442f114a078f12cd242cdc1bece0522fa1596425a ./test/locking_erc20/include/locking_erc20_functions.js

180b9f52e217fd2edfa0dec74314277e34ffb1296d384eca70e863c772274098 ./test/ilv_erc20/ilv_erc20_unit.js

32ae7b0f520252aed0bb2215356895354f22a3ff894db30ce4946a32a04b645e ./test/ilv_erc20/ilv_erc20_deployment.js

34dbb51a541952793eb75d4186819440c9e39d12e0dcc60fc8adae0b9667d5b2 ./test/ilv_erc20/ilv_erc20_non_func.js

c2130892aaaf22e3479508ad1d595f2233be6d06bace62075f6f208b0706e5b1 ./test/ilv_erc20/ilv_erc20_func_req.js

10a1ee6c6dca3d8e949db18f6dd5997f5f39f16517395bab4f9b28d5a07128e4 ./test/ilv_erc20/ilv_erc20_qs_audit.js

d47e2a21b85e4247350d0903f40b1c493063b184b7fb35b2d94373da269bdca8 ./test/ilv_erc20/ilv_erc20_acl.js

e2e966e44dc6eaee3972dfcab13b8bc4403bc3a42c3a5800f3ad8bdbda99785d ./test/ilv_erc20/ilv_erc20_mint_burn.js

26437fe38bf579c783f872d917c102870eb4913fa4dd5c42d37466986de71d54 ./test/ilv_erc20/ilv_erc20_dao.js

1db83f0afa0b949ec902fb82836dc6e5f5cab1e822724dbed0a1975ffb3fe363 ./test/include/rpc_api.js

e3e68db852dd3a2a0144be85fd51a7fcfc0f0b79e005a561deae137b149d5561 ./test/include/locking_settings.js

13ad7f0f3fa857b6d3a3b392035f06dc17521eb6603d468a6d829a447979d4a4 ./test/include/ERC20.behavior.ext.js

a3ef3d2f46be99e75d7b7e42f8432b5867f35de19f7f611a069c9b54c072a7d6 ./test/include/Comp.behavior.js

fc834c8eb817650b20018c2be35b0f77463e11c7bcdbaeb6c44598a60246afde ./test/include/yield_functions.js

f0e3c229964d8e555613db5f21629fa94c04e2a4f66e87d0740028bccd88b9ed ./test/include/ilv_erc20_features_roles.js

241f8279b15aaf1008056a48ab28cffe0c528aea799ad7edb40a5656dcf5a5cc ./test/include/EIP712.js

d6307283af76481c95c265a51501692fd38bcc841d348b15e80349d7461a2ffc ./test/include/ilv_erc20_constants.js

91c40bc532a4f95bb5750ef5345d4f232c7b8da8b5815657324d21578c628ffe ./test/include/ERC20.behavior.js

29befca9e49ab9b83dc9c0a75c4c020b3eb45a6c21f6ff6351a29526c9288c22 ./test/yield_farming/TokenLocking.test.js

```
ceb7514147f8754a68e0c41046c4c20b04852ddb6a61f2c62d87ff74d1d08f4f ./test/yield_farming/CorePool.test.js
7195b8808ff9736495f490378eac1e3073acdc43e468fbbd582cda2c47841987 ./test/yield_farming/Vault.test.js
52ce056d7f42f8f125d65c30d541a4759a545300fc8f780c92bc844fb2e27561 ./test/yield_farming/TokenLocking-sim.test.js
620b421ab160957bb4d92d9227a805a4c0308dfbaf00574e7e5721e4743c1f1e ./test/yield_farming/LockedPool.test.js
725486cb7279138ad10d91e36786a6af8733a8dd2cac67bde77d0e5c57ce03dd ./test/yield_farming/TokenLocking-ns.test.js
d829720443202511652c624d2430b3aa1592213e201114da044a13e045b2cd4d ./test/yield_farming/PoolFactory.test.js
6393455230517881 0b72cf6783428f9b4f42ed2f92a40c11be87be14de363139 ./test/yield_farming/FlashPool.test.js
bc0eb47961ed8634ae59d5fac4dfc156af07c351434284a9afdfb5faca6022d5 ./test/yield_farming/utils/index.js
de3b144981392003fb616f3a5e995853fb60938fdad8c136a46abdd14fab5f00 ./test/yield_farming/include/yield_farming_functions.js
```

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 `94807fc`

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.