Quantstamp Security Assessment Certificate

March 20th 2022 — Quantstamp Verified

Illuvium (Part 4 - Staking V2)

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

Executive Summary

Type	Blockchain based game (staking mechanism)
Auditors	Jan Gorzny, Blockchain Researcher Cristiano Silva, Research Engineer
Timeline	2021-11-29 through 2022-03-11
EVM	London
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review.
Specification	docs folder
Documentation Quality	High
Test Quality	High

Source Code

Repository	Commit
staking-contracts-v2	a4ac29e

Total Issues	**19** (15 Resolved)
High Risk Issues	**2** (1 Resolved)
Medium Risk Issues	**2** (2 Resolved)
Low Risk Issues	**5** (3 Resolved)
Informational Risk Issues	**7** (6 Resolved)
Undetermined Risk Issues	**3** (3 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

Quantstamp has reviewed the V2 staking mechanism for Illuvium. Several issues have been found of various severity levels, and all have been addressed. Although the project has high test coverage, much of the code involves math which is not particularly well justified. Code documentation exists, but is descriptive rather than explanatory. In other cases, issues are likely by design and more-or-less unavoidable (such as the privileged roles). Finally, we note there are a couple of best practices which could be followed, and the code would benefit from additional helper functions to reduce code repetition. Finally, the code has evolved over the evolution of the audit, but new code reintroduced old problems; we strongly recommend keeping this report in mind as the code continues to evolve.

ID	Description	Severity	Status
QSP-1	Missing Validation of the Oracle's Output	⇈ High	Acknowledged
QSP-2	Missing function call	⇈ High	Fixed
QSP-3	Unlimited Approvals	^ Medium	Fixed
QSP-4	Unchecked Return Values	^ Medium	Fixed
QSP-5	Ownership can be Renounced	⌄ Low	Fixed
QSP-6	Incorrect `paramIndex`	⌄ Low	Fixed
QSP-7	Gas Usage / `for` Loop Concerns	⌄ Low	Acknowledged
QSP-8	Privileged Roles and Ownership	⌄ Low	Acknowledged
QSP-9	Unsafe Cast	⌄ Low	Fixed
QSP-10	Clone-and-Own	○ Informational	Acknowledged
QSP-11	Review the Visibility of State Variables	○ Informational	Fixed
QSP-12	Event `LogMigrateUser` Could Present More Information	○ Informational	Fixed
QSP-13	Poorly Named Function: `CorePool.migrateUser(...)`	○ Informational	Fixed
QSP-14	State Variables can Change when the Contract is Paused	○ Informational	Fixed
QSP-15	Suboptimal Code	○ Informational	Fixed
QSP-16	Incorrect and Unclear `Keccak256`	○ Informational	Fixed
QSP-17	External Functions of `ILVPool.sol` can be Called by Anyone	? Undetermined	Fixed
QSP-18	Unrestricted `endTime`	? Undetermined	Fixed
QSP-19	Indistinguishable Return Value	? Undetermined	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 v0.6.6

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 Missing Validation of the Oracle's Output

Severity: *High Risk*

Status: Acknowledged

File(s) affected: `Vault.sol`

Description: The function `_swapETHForILV(...)` swaps ETH for ILV tokens. However, the function does not validate the Oracle's output. What if the price returned is zero? What if it is absurdly large? Moreover, SushiSwap works as an AMM (Automated Market Maker), which can be manipulated by big investors.

Recommendation: Consider putting some logic in the code to guard against the Oracle returning absurd values.

Update: During a call with the Illuvium team, they said that they understood the risks and do not anticipate problems as the problematic function is called by `onlyOwner` (i.e., that they would not call it if it would likely result in bad values).

QSP-2 Missing function call

Severity: *High Risk*

Status: Fixed

File(s) affected: `CorePool.sol`

Description: The comment on the following line is not implemented; only one value (`subVaultRewards`) is updated.

```
// subYieldRewards and subVaultRewards needs to be updated on every `_processRewards` call
user.subVaultRewards = uint256(user.totalWeight).weightToReward(vaultRewardsPerWeight);
```

Recommendation: Add the missing call `user.subYieldRewards = userTotalWeight.weightToReward(yieldRewardsPerWeight);` to the function. This is likely a result of copying and pasting similar functions. Consider refactoring the code to enable greater re-use and prevent errors like this in the future. Or, if this is desired behaviour, change the comment.

QSP-3 Unlimited Approvals

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `Vault.sol`

Description: In `sendILVRewards`, there are approvals with `type(uint256).max` which is dangerous if something goes wrong.

Recommendation: Approve only the amount needed, if possible, or make sure this is known to users, if it is not possible.

Update: This issue has been resolved by only approving the `Vault` contract ILV `balance` for spending.

QSP-4 Unchecked Return Values

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `FlashPool.sol`, `PoolFactory.sol`

Description: The function `stake` calls `transferFrom` but ignores the return value; function `unstake` calls `transfer` and also ignores the return value.

Recommendation: Add a `require` statement to check the return values, and revert if the wrong value is returned.

Update: The two previously listed `transfer` calls have been replaced with their `safe` variants, there is a new instance of this problem. In `PoolFactory.sol`, on lines 295 and 297, `mint` is called of an `ERC20Mintable` token and the return value is not checked; the specific contracts used do not (or will not) return boolean values.

QSP-5 Ownership can be Renounced

Severity: *Low Risk*

Status: Fixed

File(s) affected: `Vault.sol`, `PoolFactory.sol`

Description: All files that inherit from the `Ownable` contract could be left with no owner if the latter renounces his ownership. As one potential consequence, that could prevents update to the deployed contracts, requiring re-deploys and potential downtime.

Recommendation: Unless ownership renouncing is indeed an expected behaviour, override the `renounceOwnership` so that it always reverts.

Update: Resolved by overriding the `renounceOwnership()` function from Open Zeppelin's Ownable contract, removing its functionality.

QSP-6 Incorrect `paramIndex`

Severity: *Low Risk*

Status: Fixed

File(s) affected: `CorePool.sol`, `VaultRecipient.sol`

Description: In `CorePool.sol`, lines 274 and 275 `paramIndex` for `_initTime` and `_weight` are off-by-one. Similarly, line 40 in `VaultRecipient.sol` has an incorrect `paramIndex`.

Recommendation: - `CorePool.sol`: The `paramIndex`value should be `6` and `7`, respectively.

- `VaultRecipient.sol`: The `paramIndex` should be `0`.

Update: The off-by-ones have been corrected.

QSP-7 Gas Usage / `for` Loop Concerns

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `CorePool.sol`, `ILVPool.sol`

Description: Gas usage is a main concern for smart contract developers and users, since high gas costs may prevent users from wanting to use the smart contract. Even worse, some gas usage issues may prevent the contract from providing services entirely. For example, if a `for` loop requires too much gas to exit, then it may prevent the contract from functioning correctly entirely. It is best to break such loops into individual functions as possible.
Any user can make an arbitrary number of stakes. However, this can lead for very large arrays of stakes, and such arrays need to be traversed, such as in function `CorePool.BalanceOf(...)`and `CorePool.getV1StakePosition(...)`(and other functions of the same file). The size of the pools must also be considered, since some functions have to loop over the pools, such as in function `ILVPool.claimYieldRewardsMultiple(...)`.

Recommendation: If possible, consider limiting the maximum number of stakes per user, and the maximum number of participants per pool. Alternatively, allow a user to deal with only some (specified) stakes to avoid out-of-gas issues.

Update: The team has acknowledged the issue: "The issue identified [effects] view functions which are supposed to be used through `eth_calls`, i.e. gas shouldn't be an issue."

QSP-8 Privileged Roles and Ownership

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `Vault.sol`, `PoolFactory.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract.
In this project, only the owner can set pools, convert ETH to ILP in `Vault.sol` and only the owner can perform upgrades in `PoolFactory.sol`.

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: This is issue is acknowledged. Although the code has increased documentation, no new external documentation is provided and no roles have changed.

QSP-9 Unsafe Cast

Severity: *Low Risk*

Status: Fixed

File(s) affected: `FlashPool.sol`, `CorePool.sol`

Description: The function `unstake` in `FlashPool.sol` converts a `uint256` to a `unit128` without checking that the cast is safe. Similar issues - with as small a bit width as 64 - exist elsewhere in the contract. These issues are found in `CorePool.sol` as well.

Recommendation: Check that the value will not change during the cast, i.e., use a safe cast.

Update: The casts have been replaced with calls to a safe casting function.

QSP-10 Clone-and-Own

Severity: *Informational*

Status: Acknowledged

File(s) affected: `IUniswapV2Router01.sol`, `IUniswapV2Router02.sol`

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.

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

Update: The team has acknowledged this issue: "Hardhat framework is used for most libraries, but we decided to keep UniswapV2/Sushi interfaces in the code base."

QSP-11 Review the Visibility of State Variables

Severity: *Informational*

Status: Fixed

File(s) affected: `All`

Description: The proper declaration of the visibility of state variables is a good programming practice and it also reduces the final contract size. Public variables increase the contract size:for each public variable, the solidity compiler automatically generates the `getter` and `setter`functions.
For instance, we do not see a reason for declaring the following state variables as `public` in `Vault.sol`:

- `Pools public pools;`
- `IUniswapV2Router02 public sushiRouter` and
- `IERC20 public ilv`.

Recommendation: Review the visibility of all state variables, and keep `public` only those variables that are really required to be so.

Update: The team has resolved this issue by converting some public state variables to private or internal, which reduced the contract size significantly. Only state variables that aren't important to the client or to external contracts have been converted.

QSP-12 Event `LogMigrateUser` Could Present More Information

Severity: *Informational*

Status: Fixed

File(s) affected: `CorePool.sol`

Description: The function `migrateUser(...)` contains the following: `emit LogMigrateUser(msg.sender, _to)`. Perhaps adding more information to this log would help in future investigations.

Recommendation: Consider adding the rewards in the log message, in order to clarify the computation.

Update: This has been resolved by adding more data to the `LogMigrateUser` event, which is now called `LogMoveFundsFromWallet`.

QSP-13 Poorly Named Function: `CorePool.migrateUser(...)`

Severity: *Informational*

Status: Fixed

File(s) affected: `CorePool.sol`

Description: The function `migrateUser(...)` is related to migrations of wallets of the same user, and not migrating users from V1 to V2. Thus, we encourage the team to provide a better name for this function.

Recommendation: Rename the function `CorePool.migrateUser(...)`to differentiate this functionality from migrating users from V1 to V2. Perhaps something like `migrateWallet(...)` or `migrateFundsFromWallet(...)` are more appropriate

Update: This issue has been resolved by renaming the `migrateUser` function and its related event to `moveFundsFromWallet` and `LogMoveFundsFromWallet` respectively, which makes the function more clear and descriptive.

QSP-14 State Variables can Change when the Contract is Paused

Severity: *Informational*

Status: Fixed

File(s) affected: `CorePool.sol`

Description: The functions `migrateUser(...)`, `sync()`, `receiveVaultRewards(...)`, `setWeight(...)`, and `pendingRewards(...)`can be called when the contract is paused. This can be dangerous since they change state variables.

Recommendation: The Illuvium team should analyse if this is the intended behaviour, or if it is missing the `_requireNotPaused()` modifier on these functions.

Update: The functions `migrateUser(...)`, `sync()`, and `receiveVaultRewards(...)`now require the contract to be not paused. The remaining functions are assumed to be desirable as callable functions on the paused contracts.

QSP-15 Suboptimal Code

Severity: *Informational*

Status: Fixed

Description: The contract size can be optimized by turning `modifiers` into `functions`, as shown in the example below:

```
modifier checkStuff() {}
function doSomething() checkStuff {}
```

The `function doSomething()` can rewritten without modifiers as:

```
function checkStuff() private {}
function doSomething() { checkStuff(); }
```

Recommendation: Maybe the tips in this reference below can help Illuvium to reduce the contract size.

Update: This issue has been resolved by removing the `updatePool` modifier and calling the internal `_sync()` function directly instead, in order to reduce contract size.

QSP-16 Incorrect and Unclear `Keccak256`

Severity: *Informational*

Status: Fixed

File(s) affected: `ILVPool.sol`, `CorePool.sol`

Description: In `ILVPool.sol`, the hash comment `bytes4(keccak256("_migrateYieldWeights(bytes32[],uint256,uint256")))` in function `_migrateYieldWeights` does not have the closing bracket for `_migrateYieldWeights` inside the quote. In `CorePool.sol`, the hash `0x867a0347` is not explained when it should be.

Recommendation: Change the incorrect comment and add the missing comment.

Update: This issue has been fixed.

QSP-17 External Functions of `ILVPool.sol` can be Called by Anyone

Severity: *Undetermined*

Status: Fixed

File(s) affected: `V2Migrator.sol`

Description: The functions from this file can be called by anyone. However, the code comment states that “External `MigrateLockedStakes` call, [is] used in Sushi LP pool”.

Recommendation: The Illuvium team must certify that such freedom does not impose any threat to the application. Consider imposing some restriction on who can call these functions. Since the external functions must be called by the Sushi LP Pool, try to ensure this condition.

Update: This issue has been resolved by adding documentation in `migrateLockedStakes()`. Existing modifiers are intentional.

QSP-18 Unrestricted `endTime`

Severity: *Undetermined*

Status: Fixed

File(s) affected: `FlashPool.sol`, `PoolFactory.sol`

Description: In `setEndTime`, there is no sanity checks for the new value. It could be before the current time, or interact with existing pools in unexpected ways.

Recommendation: Sanity check the value or add documentation to say why it is not a problem.

Update: The team has added the required check in the `setEndTime` and `initialize` functions.

QSP-19 Indistinguishable Return Value

Severity: *Undetermined*

Status: Fixed

File(s) affected: `CorePool.sol`

Description: In `getV1StakePosition`, the value zero is returned if the stake id provided is not found. However, this value is also returned if the stake is the first entry.

Recommendation: Return a different value when the stake is not found or revert in this case.

Update: This issue has been resolved by reverting instead of returning 0 in the `getV1StakePosition` function (unless zero is the desired index).

Automated Analyses

Slither

Slither was unable to analyse the contracts.

Adherence to Best Practices

- `CorePool.sol`: Line 304 has a typo `store stored`. **Update:** Fixed.
- `V2Migrator.sol`: The hex value `0x710276c7` should be explained in a comment. **Update:** Fixed.
- `Vault.sol`: Magic constant `1e7` in function `_getAmountToSend`. **Update:** Fixed (and value changed).
- `Vault.sol`: There is a comment in `estimatePairPoolReserve` where a value is not computed because it's value will "always be zero". If that's really the case - add this check as a require. The comments in general here are not matching the computations - for example, the last computation involves division, which is not described in the comments. **Update:** The comment has been removed.
- `PoolFactory.sol`: There are magic constants in `updateILVPerSecond`.
- `PoolFactory.sol`: There is a commented out import, which could be removed. **Update:** The import has been removed.
- Consider converting `_requireNotPaused` function into a modifier. **Update:** This is avoided to reduce contract size.
- There is inconsistent handling of errors. In some contracts (e.g., `CorePool`) the `ErrorHandler` library is used to handle checks for inputs. In other contracts (e.g., `Vault.sol`) regular `require`statements are used to check for inputs. **Update:** There is now more widespread use of the `ErrorHandler` library.
- `ILVpool.sol`: The naming conventions in arguments for the constructor are inconsistent with regards to where the `_` goes.

Test Results

Test Suite Results

```
CorePools
    Upgrades
       ✓ should upgrade ilv pool (549ms)
       ✓ should upgrade lp pool (507ms)
       ✓ should upgrade factory (266ms)
       ✓ should revert upgrading ilv pool from invalid admin (161ms)
       ✓ should revert upgrading lp pool from invalid admin (66ms)
    #setEndTime
       ✓ should correctly update endTime (44ms)
       ✓ should revert if invalid endTime
       ✓ should revert if invalid caller
    #getPoolData
      ILV Pool
         ✓ should get correct pool data
         ✓ should revert if pool does not exist
      Sushi LP Pool
         ✓ should get correct pool data
         ✓ should revert if pool does not exist
    #blacklistUser
      ILV Pool
         ✓ should blacklist a list of users (39ms)
         ✓ should revert if caller is not factory controller (39ms)
      Sushi LP Pool
         ✓ should blacklist a list of users
         ✓ should revert if caller is not factory controller
    #stake
      ILV Pool
         ✓ should stake and lock (125ms)
         ✓ should get correct stakesLength (104ms)
         ✓ should revert when staking longer than 2 years (45ms)
         ✓ should revert when _lockDuration = 0
         ✓ should revert when _value = 0
      Sushi LP Pool
         ✓ should stake and lock (108ms)
         ✓ should get correct stakesLength (99ms)
         ✓ should revert when staking longer than 2 years (41ms)
         ✓ should revert when _lockDuration = 0
         ✓ should revert when _value = 0
    #pendingYield
      ILV Pool
         ✓ should not accumulate rewards before init time (84ms)
         ✓ should accumulate ILV correctly (101ms)
         ✓ should accumulate ILV correctly for multiple stakers (166ms)
         ✓ should calculate pending rewards correctly after bigger stakes (215ms)
         ✓ should not accumulate yield after endTime (187ms)
      Sushi LP Pool
         ✓ should not accumulate rewards before init time (79ms)
         ✓ should accumulate ILV correctly (91ms)
         ✓ should accumulate ILV correctly for multiple stakers (160ms)
         ✓ should calculate pending rewards correctly after bigger stakes (212ms)
         ✓ should not accumulate yield after endTime (198ms)
    #claimYieldRewards
      ILV Pool
         ✓ should create ILV stake correctly (131ms)
```

```
      ✓ should mint ILV correctly (231ms)
      ✓ should mint sILV correctly (122ms)
    Sushi LP Pool
      ✓ should create ILV stake correctly (141ms)
      ✓ should mint ILV correctly (270ms)
      ✓ should mint sILV correctly (116ms)
  #claimYieldRewardsMultiple
    ✓ should correctly claim multiple pools as ILV (294ms)
    ✓ should correctly claim multiple pools as sILV (285ms)
    ✓ should correctly claim multiple pools as ILV and sILV (348ms)
    ✓ should revert if claiming invalid pool (369ms)
    ✓ should revert if claiming from invalid address (150ms)
  #unstakeLocked
    ILV Pool
      ✓ should unstake locked tokens (152ms)
      ✓ should unstake locked tokens partially (140ms)
      ✓ should revert when _stakeId is invalid (217ms)
      ✓ should revert when _value is 0 (77ms)
      ✓ should revert when _value is higher than stake (88ms)
      ✓ should revert when tokens are still locked (85ms)
    Sushi LP Pool
      ✓ should unstake locked tokens (140ms)
      ✓ should unstake locked tokens partially (124ms)
      ✓ should revert when _stakeId is invalid (87ms)
      ✓ should revert when _value is 0 (79ms)
      ✓ should revert when _value is higher than stake (84ms)
      ✓ should revert when tokens are still locked (73ms)
  #sync
    ILV Pool
      ✓ should sync pool state (106ms)
      ✓ should sync pool state with totalStaked = 0 (53ms)
      ✓ should stop sync after endTime (146ms)
      ✓ should update ilv per second after secondsPerUpdate (114ms)
    Sushi LP Pool
      ✓ should sync pool state (100ms)
      ✓ should sync pool state with totalStaked = 0 (39ms)
      ✓ should stop sync after endTime (135ms)
      ✓ should update ilv per second after secondsPerUpdate (111ms)
  #setWeight
    ILV Pool
      ✓ should change pool weight from 0 to x
      ✓ should change pool weight from x to 0
      ✓ should block unauthorized addresses to change weight through pool
      ✓ should block unauthorized addresses to change weight through factory
      ✓ should revert minting yield from non pool caller
    Sushi LP Pool
      ✓ should change pool weight from 0 to x
      ✓ should change pool weight from x to 0 (42ms)
      ✓ should block unauthorized addresses to change weight through pool (49ms)
      ✓ should block unauthorized addresses to change weight through factory
      ✓ should revert minting yield from non pool caller
  #moveFundsFromWallet
    ILV Pool
      ✓ should migrate a user stake (219ms)
      ✓ should revert if _to = address(0) (190ms)
      ✓ should revert if newUser totalWeight != 0 (215ms)
      ✓ should revert if newUser stakes.length != 0 (420ms)
      ✓ should revert if newUser subYieldRewards != 0 (237ms)
      ✓ should revert if newUser v1IdsLength > 0 (256ms)
      ✓ should revert if user has weight in v1 (111ms)
    Sushi LP Pool
      ✓ should migrate a user stake (291ms)
      ✓ should revert if _to = address(0) (306ms)
      ✓ should revert if newUser totalWeight != 0 (225ms)
      ✓ should revert if newUser stakes.length != 0 (406ms)
      ✓ should revert if newUser subYieldRewards != 0 (273ms)
      ✓ should revert if newUser v1IdsLength > 0 (254ms)
      ✓ should revert if user has weight in v1 (110ms)
  #unstakeLockedMultiple
    ILV Pool
      ✓ should unstake multiple locked tokens (353ms)
      ✓ should revert if unstake parameters length is 0 (236ms)
      ✓ should revert if unstaking value is higher than stake value (299ms)
      ✓ should unstake multiple locked tokens partially (325ms)
      ✓ should unstake multiple locked yield after unlock (412ms)
      ✓ should revert unstaking multiple locked yield before unlock (306ms)
    Sushi LP Pool
      ✓ should unstake multiple locked tokens (300ms)
      ✓ should revert if unstake parameters length is 0 (213ms)
      ✓ should revert if unstaking value is higher than stake value (292ms)
      ✓ should unstake multiple locked tokens partially (304ms)
      ✓ should unstake multiple locked yield after unlock (387ms)
      ✓ should revert unstaking multiple locked yield before unlock (329ms)
  #pause
    ILV Pool
      ✓ should pause the pools
      ✓ should pause and unpause the pools
      ✓ shouldn't allow to stake when paused (40ms)
      ✓ should only allow the factory controller to pause/unpause
    Sushi LP Pool
      ✓ should pause the pools
      ✓ should pause and unpause the pools
      ✓ shouldn't allow to stake when paused
      ✓ should only allow the factory controller to pause/unpause
  Migration tests
    ILV Pool
      ✓ should accumulate ILV correctly - with v1 stake ids (210ms)
      ✓ should accumulate ILV correctly - with v1 stake ids and decreasing v1 weight (487ms)
      ✓ should accumulate ILV correctly - with v1 stake ids and increasing v1 weight (567ms)
      #migrateLockedStake
        ✓ should migrate locked stakes - alice (67ms)
        ✓ should revert if _stakeId doesn't exist (56ms)
        ✓ should migrate locked stakes - carol (60ms)
        ✓ should revert if migrating already migrated stake (111ms)
        ✓ should revert if migrating yield (61ms)
        ✓ should revert if migrating unlocked stake
        ✓ should revert if lockedFrom > v1StakeMaxPeriod
    Sushi LP Pool
      ✓ should accumulate ILV correctly - with v1 stake ids (205ms)
      ✓ should accumulate ILV correctly - with v1 stake ids and decreasing v1 weight (492ms)
      ✓ should accumulate ILV correctly - with v1 stake ids and increasing v1 weight (493ms)
      #migrateLockedStake
        ✓ should migrate locked stakes - alice (59ms)
        ✓ should revert if _stakeId doesn't exist (55ms)
        ✓ should migrate locked stakes - carol (67ms)
        ✓ should revert if migrating already migrated stake (122ms)
        ✓ should revert if migrating yield (59ms)
        ✓ should revert if migrating unlocked stake
        ✓ should revert if lockedFrom > v1StakeMaxPeriod
    Mint yield
      ✓ should mint v1 yield (71ms)
      ✓ should revert if stake !isYield (63ms)
      ✓ should revert if lockedUntil > _now256
      ✓ should revert if yield is already minted (92ms)
      ✓ should mint multiple v1 yield stake (78ms)
      ✓ should revert minting multiple yield stakes if already minted (101ms)
      ✓ should revert if passing !isYield _stakeId (50ms)
      ✓ should revert on mintYieldMultiple if yield is locked (39ms)
  Merkle tree tests
    ILV Pool
      ✓ returns the expected merkle root
      ✓ should validate a merkle proof correctly and mint sILV (75ms)
      ✓ should validate a merkle proof correctly and claim vested ILV (65ms)
      ✓ should validate a merkle proof correctly without stakeIds (42ms)
      ✓ should fail claiming twice - with sILV (101ms)
      ✓ should fail claiming twice - with ILV (106ms)
      ✓ should fail claiming twice - with ILV and sILV (100ms)
      ✓ should fail claiming twice without stakeIds array (76ms)
      ✓ should fail with empty proof (51ms)
      ✓ should fail with invalid index - alice (45ms)
      ✓ should fail with invalid index - bob
      ✓ should fail with invalid msg.sender
      ✓ should set hasMigratedYield correctly (114ms)
      ✓ should not migrate more yieldWeight than allowed
      ✓ should not migrate more pendingV1Rewards than allowed
```

```
FlashPool
  Upgrades
     ✓ should upgrade flash pool (138ms)
     ✓ should revert deploying a pool if factory == address(0)
     ✓ should revert initializing a factory if _ilv == address(0)
     ✓ should revert initializing a factory if silv_ == address(0)
     ✓ should revert initializing a factory if _ilvPerSecond == 0 (55ms)
     ✓ should revert initializing a factory if _secondsPerUpdate == 0
     ✓ should revert initializing a factory if _initTime == 0
     ✓ should revert initializing a factory if _endTime == 0
  #getPoolData
     ✓ should get correct pool data
  #stake
     ✓ should stake (42ms)
     ✓ should revert on _value 0
     ✓ should process rewards on stake (100ms)
  #pendingYield
     ✓ should not accumulate rewards before init time (57ms)
     ✓ should accumulate ILV correctly (73ms)
     ✓ should accumulate ILV correctly for multiple stakers (113ms)
     ✓ should calculate pending rewards correctly after bigger stakes (171ms)
     ✓ should not accumulate yield after endTime (117ms)
  #claimYieldRewards
     ✓ should create ILV stake correctly (119ms)
     ✓ should return if pendingYield = 0 (60ms)
     ✓ should mint sILV correctly (91ms)
  #claimYieldRewardsMultiple
     ✓ should correctly claim multiple pools as ILV (261ms)
     ✓ should correctly claim multiple pools as sILV (228ms)
     ✓ should correctly claim multiple pools as ILV and sILV (238ms)
     ✓ should revert if claiming invalid pool (156ms)
     ✓ should revert if claiming from invalid address (176ms)
  #unstake
     ✓ should unstake (61ms)
     ✓ should revert unstaking 0 (63ms)
     ✓ should revert unstaking more than allowed (59ms)
     ✓ should process rewards on unstake (124ms)
  #setWeight
     ✓ should change pool weight
     ✓ should revert on invalid setWeight caller
  #sync
     ✓ should sync pool state (82ms)
     ✓ should sync pool state with totalStaked = 0
     ✓ should stop sync after endTime (147ms)
     ✓ should update ilv per second after secondsPerUpdate (101ms)
     ✓ should setEndTime
     ✓ should rever trying to setEndTime from unauthorized account
  #moveFundsFromWallet
     ✓ should migrate an user stake (154ms)
     ✓ should revert if _to = address(0) (134ms)
     ✓ should revert if newUser totalWeight = 0 (180ms)
     ✓ should revert if newUser pendingYield = 0 (174ms)
  #pause
     ✓ should pause the pools
     ✓ should pause and unpause the pools
     ✓ shouldn't allow to stake when paused
     ✓ should only allow the factory controller to pause/unpause

Vault
  #claimAllRewards
     ✓ should claim yield and vault rewards in one transaction (532ms)
  #setCorePools
     ✓ should set core pools correctly
     ✓ should revert if ilvPool = address(0)
     ✓ should revert if pairPool = address(0)
     ✓ should revert if lockedPoolV1 = address(0)
     ✓ should revert if lockedPoolV2 = address(0)
     ✓ should receive ether
     ✓ should revert deploying vault with _sushiRouter = address(0)
     ✓ should revert deploying vault with _ilv = address(0)
     ✓ should revert setting vault to address(0)
  #swapETHForILV
     ✓ should swap contract eth balance to ILV (66ms)
     ✓ should revert if _ilvOut = 0
     ✓ should revert if _ethIn = 0
     ✓ should revert if deadline = 0
     ✓ should revert if not enough eth
  #sendILVRewards
     ✓ should distribute ilv revenue (447ms)
     ✓ should buy and distribute ilv revenue in the same transaction (467ms)
     ✓ should send ilv rewards twice (587ms)
  #claimVaultRewards
     ✓ should claim vault rewards (640ms)
     ✓ should return if 0 rev dis (285ms)
     ✓ should claim vault rewards in multiple pools (407ms)
     ✓ should revert if calling receiveVaultRewards from non-vault address (56ms)


 221 passing (3m)
```

Code Coverage

File	Statements	Branches	Functions	Lines
contracts/	99.41% 338/340	93.48% 43/46	96.55% 56/58	99.41% 339/341
contracts/base/	99.63% 267/268	97.37% 37/38	100% 35/35	100% 272/272
contracts/interfaces/	100% 0/0	100% 0/0	100% 0/0	100% 0/0
contracts/libraries/	100% 22/22	100% 8/8	100% 12/12	100% 26/26

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

```
e267e28c8c501e86e2aa5efe19d8d7513b0008362f28c7eeba56ceeaadf43551 ./contracts/PoolFactory.sol
19d45aaf8625eab845126498d71704c4dc4d73f286f71d1b72a0c4949944e740 ./contracts/ILVPool.sol
97c7825c06c2820bdc2009891e28e188bbafdfb6dd549fc0a3b0b80bfa006024 ./contracts/FlashPool.sol
0f7101a36440c5d0646fdbdcb65e45634d62611d7bf0fd573d39d8bf0dc59d25 ./contracts/SushiLPPool.sol
14192c7da52fc04e859ba62b0a8297100221863257726 18ecc626b0073e776c5 ./contracts/Vault.sol
97386c864d79bfd5ce92748d0cddff932b0e35693fd710f99046325349ec04b4 ./contracts/libraries/SafeCast.sol
ef2e278dae6a6599136f8397aced5b950176090c9c3c2a695cdd174d797573a1 ./contracts/libraries/ErrorHandler.sol
f975fc9bbfc50a7f10c44e9de2900e36a1b2a60ebfdee4b596ac6a1435430171 ./contracts/libraries/Stake.sol
a7dbeec9e7c157b8cadc8d3b6ec69196421d142e14113096eb15a645c4290960 ./contracts/mocks/ILVPoolUpgrade.sol
2109db397682fd52cad6a3503076ccbc1837f6048c6e7c90dcf42f45b48dd95c ./contracts/mocks/PoolFactoryMock.sol
f2cfdb3d569d5bff86af2ac0e93e1d7be66e6e1177906de388119e149b117a11 ./contracts/mocks/PoolFactoryUpgrade.sol
f9dea61b56367a80e5fc4793eb8e13c03b3353c282b23da37c2e9c4749e4d01e ./contracts/mocks/SushiLPPoolMock.sol
5829b0fa225ca8fdf90586b5d9e1a2e9266388cc28eae31ba1e7cd65d050da8d ./contracts/mocks/ERC20Mock.sol
81265342a4db2caf417301084ba629151e0abc4e7eb112132b4122e3c4a50514 ./contracts/mocks/SushiLPPoolUpgrade.sol
da0523c7729f1e304667a667bc42828595f93020b2e5af3de3bf08acd5f4b610 ./contracts/mocks/FlashPoolMock.sol
23c5a8c1b32e936ec830c80248c8398eb00e068594020f72f181e2834cf3cd10 ./contracts/mocks/FlashPoolUpgrade.sol
6ba438bee4db21918d81a2ea3f19450e02c6ee4a645d0f604220b3de6c816289 ./contracts/mocks/WETHMock.sol
aca91a92cf63abc35d5735a3f0ed4c247c59dbe77e3856d0d3cef9bd1194baa0 ./contracts/mocks/ILVPoolMock.sol
9660108ca50c3a5943f1e934e9965256ef8bdfe9d968e05a60ed2a6c3de6b3e2 ./contracts/mocks/CorePoolV1Mock.sol
7169731637f6af3e37bfb894907ecb9ce8b8663b7a6946d0e7db0ab0e6752fab ./contracts/base/CorePool.sol
2eff6f0bc19dc27c4275839a0d81f0fe81bc8b23fc8a03c684fd595992a778c1 ./contracts/base/Timestamp.sol
ed6866618555f88914863ee9293e26475ee7f16e8672de37a0bee03b58ab374a ./contracts/base/VaultRecipient.sol
19048f05745ec39970c7bc6032ef92f34ae7610fd0b5d709b80b1865df90ad41 ./contracts/base/V2Migrator.sol
2394f4b7cc31033ea2a0ef16989728207b9136a7a8e57720a5b5b3588c48585d ./contracts/base/FactoryControlled.sol
df837ec28997e2ab1cccaa2c8daa1fac63b8212533af949116c1acaf30308af9 ./contracts/interfaces/IFactory.sol
3ba44482efea286c5a851f49443f46fb2b20f7ab21ab554d84de11d845e9e07a ./contracts/interfaces/IILVPool.sol
2d0ebbc6c2e9495d9d423561aa0cb0a7413c9f5b200d9c8dba3d3bfffac0cccd ./contracts/interfaces/ICorePoolV1.sol
5f68ae7b101b46c3ef82f146af1a74605bc08a3a4fcab50daf72158e2b64c0b5 ./contracts/interfaces/IUniswapV2Router02.sol
7fbb7dd35e3ce41b90c30e00a0e7c2cf2e8509aaac421402037070d9db2664cf ./contracts/interfaces/IERC20Mintable.sol
0ad993ddf916b2f0022d4197f45c2d27b5305588edcc515eebcdb8261525445e ./contracts/interfaces/IUniswapV2Router01.sol
37ee078b2eb48d6ad46c7fc83702460062c12d2f988cbe2e0c7bd6565e5a5a30 ./contracts/interfaces/ICorePool.sol
```

Tests

```
7b1f811661b471f59c8c8aba4e6f702036cfc392fe4f12771ddaa23ac98b9fb4 ./test/Vault.behavior.ts
7cac2d7112f573439b12a847d90c5b9b0a67e6821d820425a32606148bf0d58b ./test/CorePool.spec.ts
6acdc0aa6917efd701ac617c3590061161ef397e1931fa8b8c7f4ef1409b466e ./test/FlashPool.spec.ts
e11e0e6200cd029d8768e4bcec434db3c0a2379e684ec27da649efd73d09424b ./test/Vault.spec.ts
52ca2b7b2a0523a337be8c96696b5f31d191139b49f49bc5bc9665828c06adf9 ./test/CorePool.behavior.ts
914a984852e28c824a357e7b2821340b8f26a4de1b75f945b817bace557759fe ./test/utils/index.ts
2cfb937a163e8b6684dcce4ed71fd5d496e80b00a0825286845d78da2d1409a1 ./test/utils/merkle-tree.ts
2db9ba7568fb5f415e94f39f25f66d59ac74aa41d2e06de9569a2686f1a85c15 ./test/utils/yield-tree.ts
```

Changelog

- 2021-12-10 - Initial report [591c081]
- 2021-12-30 - Revised report [5c41df8]
- 2022-03-11 - Revised report [f146cb8]
- 2022-03-17 - Revised report [a4ac29e]

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.