Quantstamp Security Assessment Certificate

November 20th 2022 — Quantstamp Verified

Key Finance

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

Executive Summary

Type	Defi Aggregator
Auditors	Faycal Lalidji, Senior Security Engineer Fatemeh Heidari, Security Auditor Nikita Belenkov, Security Auditor
Timeline	2022-10-03 through 2022-10-21
EVM	Arrow Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Online Documentation
Documentation Quality	Medium
Test Quality	Medium

Source Code

Repository	Commit
cryptohiveteam/key-contract	bb3664c initial audit

Total Issues	**18** (16 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	**5** (5 Resolved)
Low Risk Issues	**7** (7 Resolved)
Informational Risk Issues	**6** (4 Resolved)
Undetermined Risk Issues	0 (0 Resolved)

ALL ISSUES ADDRESSED

Documentation Issues Addressed

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).
Fixed	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

Initial Audit
Overall the code is complex and requires more documentation. We have found multiple issues ranging from medium to informational severity that must be fixed before deployment. We highly recommend implementing sufficient testing to verify all the project specifications possibly.
Fix Review
All highlighted issues have been either fixed or acknowledged.

ID	Description	Severity	Status
QSP-1	Assignment to Out of Bounds Array Index Leads to Transaction Failure	Medium	Fixed
QSP-2	Lockers Addresses Cannot Be Added Back After Being Removed	Medium	Fixed
QSP-3	Possible Attack by Manipulating Computation Precision	Medium	Fixed
QSP-4	Withdrawing Rewards on-Behalf of Contracts Represent a Risk	Medium	Mitigated
QSP-5	Any Address Can Extend Reward Distribution Period	Medium	Fixed
QSP-6	Underflow Leads to Transaction Failure	Low	Fixed
QSP-7	`KlapVeDepositor.transferfrom()` Is Missing a `notPaused` Modifier	Low	Fixed
QSP-8	Gas Usage / Loop Concerns	Low	Mitigated
QSP-9	Reduction of Distributed Funds	Low	Fixed
QSP-10	ERC20 Compliance	Low	Fixed
QSP-11	`Depositor.deleverageOnce()` Borrow Rate Input Is Not Fully Validated	Low	Fixed
QSP-12	Unseen Balance Contract Has to Be Validated	Low	Fixed
QSP-13	Missing Input Validation	Informational	Fixed
QSP-14	Zero-Division Leads to Transaction Failure	Informational	Fixed
QSP-15	Block Timestamp Manipulation	Informational	Acknowledged
QSP-16	Using Custom Errors in Revert Is More Gas Efficient	Informational	Fixed
QSP-17	Allowance Double-Spend Exploit	Informational	Fixed
QSP-18	Functions Missing Access Restriction	Informational	Acknowledged

Quantstamp Audit Breakdown

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

DISCLAIMER:
If the final commit hash provided by the client contains features that are not within the scope of the audit or an associated fix review, those features are excluded from consideration in this report.

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 v0.8.3

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Findings

QSP-1 Assignment to Out of Bounds Array Index Leads to Transaction Failure

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `Rewards.sol`

Description: In `_getAirdropTokensOfIncentivizedERC20()` function in `Rewards` contract, if there is an `i` for which `getRewardToken(token, i)` returns zero address, and `i` is higher than `tokenCount`, the transaction fails(due to trying to assign to a non-existing index in `airdropTokens`).

Recommendation: Revise implementation to prevent accessing the non-existing index.

Update: Fixed in commit "1a83468".

QSP-2 Lockers Addresses Cannot Be Added Back After Being Removed

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `KeyMinter.sol`

Description: `KeyMinter.removeLocker()` removes locker address but does not reset the receivers `emissionWeights` meaning that the addresses won't be usable in the future since the `emissionWeights` value for an address is required to be zero to be able to add it again.

Recommendation: We recommend removing the `emissionWeights` related fields for that specific address or changing the requirements.

Update: Fixed in commit "79f7af7".

QSP-3 Possible Attack by Manipulating Computation Precision

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `Depositor.sol`

Description: In `Depositor. deposit()`, the value of the initial shares is directly proportional to the deposit amount. This functionality permits the user to set the initial share value as desired and may introduce a lower precision when calculating shares of subsequent users.
In a more elaborate scenario, if the interest rate is high enough or if it can be manipulated on the KLAP liquidity pool, an attacker might make a small initial deposit and manipulate the yield through KLAP or any other means making the precision loss when computing even higher.

Exploit Scenario: As an example: (please note that this is without taking into account the minimum deposit amount set since it is an unknown variable)

- An attacker deposits a small amount.
- the attacker manipulates the yield value to make the share-to-amount ratio equal to 1/100.
- If a user deposits 50, Bob's share will be equal to 50*1/100, equal to zero.

Recommendation: Set a hard-coded deposit minimum value.

Update: Fixed in commit "1c746e4bc734548d018db03978bd995e13613807".

QSP-4 Withdrawing Rewards on-Behalf of Contracts Represent a Risk

Severity: *Medium Risk*

Status: Mitigated

File(s) affected: `Rewards.sol`, `KeyRewards.sol`

Description: `KeyRewards.getAllReward()`, `KeyRewards.getReward()`, `Rewards.getReward()` and `Rewards.getAirdropReward()` allow withdrawing on behalf of users. The implemented reward withdrawal can be problematic in case of third-party integration with the protocol.
For example, suppose contract developers are unaware that funds can be withdrawn on behalf of the developed contract. In that case, they might not implement the necessary logic to handle the tokens sent to them.

Recommendation: Either document this behavior clearly in the developer documentation or do not withdraw contract addresses on behalf.

Update: The issue was mitigated by adding the following comment to the related functions in commit "3958b0b": "Anyone can claim the amount that vesting ended on behalf of the receiver. Therefore, contracts need to implement withdrawal if funds are sent to their addresses."

QSP-5 Any Address Can Extend Reward Distribution Period

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `KlapStakingRewards.sol`

Description: `KlapStakingRewards.updatePeriodFinishForKeyOnce()` allows anyone to extend the reward distribution for any remaining reward. Even if a requirement exists that validates if the period has ended, the function does not update the reward state. Rewards that should have been distributed will be extended following the new period.

Recommendation: The reward per unit asset state should be updated before setting the new period extension.

Update: Fixed in commit "01ab277".

QSP-6 Underflow Leads to Transaction Failure

Severity: *Low Risk*

Status: Fixed

File(s) affected: `Depositor.sol`

Description: In `_withdraw()` function, when `borrowBal` == 0 the amount that is getting withdrawn from pool is calculated by deducting `assetBal` from `amount`. Add a control to make sure `amount` is higher than `assetBal` to prevent underflow.

Update: Fixed in "405e984".

QSP-7 `KlapVeDepositor.transferfrom()` Is Missing a `notPaused` Modifier

Severity: *Low Risk*

Status: Fixed

File(s) affected: `KlapVeDepositor.sol`

Description: The `transfer()` function has a modifier that checks if the protocol is not paused, while `transferFrom()` does not, which is inconsistent with the general contract logic.

Exploit Scenario: Add the `notPaused` modifier to `transferFrom()` function.

Update: Fixed in commit "c6d5efec".

QSP-8 Gas Usage / Loop Concerns

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `KeyMinter.sol`, `KeyRewards.sol`

Related Issue(s): SWC-126, SWC-134

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 loop requires too much gas to finish processing, then it may prevent the contract from functioning correctly entirely.

- `KeyMinter.addLocker()` iterates over `tokenLockerIds`. New addresses are checked against the array before being excluded or included. If the array length grows, then gas issues might arise.
- Similarly, `KeyMinter.removeLocker()`might iterates over all `tokenLockerIds` array before successfully removing an element.
- In `KeyToken._distribute()`, it seems non-feasible to iterate over the whole `emissionWeights` array. Depending on the array, gas issues might arise.

Recommendation: If possible, we recommend breaking loops into individual functions and/or adding function arguments that allow users to continue loop processing in a separate transaction. Another possible alternative is to use Openzeppelin list implementation.

Update: Partially fixed in commit "0462438".
"For the 3rd bullet, since the admin registers receivers and corresponding `emissionWeights`, we as an admin will limit the number of it, and the gas fee used for the 'applyEmission' function calls so that `mintAndDistribute`call succeeds."

QSP-9 Reduction of Distributed Funds

Severity: *Low Risk*

Status: Fixed

File(s) affected: `KeyMinter.sol`

Description: The condition stated below, part of `KeyMinter._distribute()` function, reduces the receiver amount for the ongoing calculated period if the remaining amount is less than expected.

```
if (remainAmount < dividedAmount) dividedAmount = remainAmount;
```

Recommendation: If such an error exists, we recommend either reverting the transaction with a valid error message or emitting an event to be able to log and reimburse the funds later on.

Update: Fixed in commit "63eae5a".

QSP-10 ERC20 Compliance

Severity: *Low Risk*

Status: Fixed

File(s) affected: `DepositToken`

Description: Following the ERC20 Token standard, `DepositToken._transfer()` must emit a `Transfer` event even if the value is equal to zero, which is not the case in this contract.

Recommendation: The `Transfer` event can be placed outside of the condition.

Update: Fixed in "b7e0931".

QSP-11 `Depositor.deleverageOnce()` Borrow Rate Input Is Not Fully Validated

Severity: *Low Risk*

Status: Fixed

File(s) affected: `Depositor.sol`

Description: In `Depositor.deleverageOnce()`, the input borrow rate should also be validated against the KLAP protocol liquidation threshold dynamically with a certain margin to avoid liquidation.
The implemented function is withdrawing, meaning that the MAX TVL is not applied, but the limit is the liquidation threshold; please bear in mind that `borrowRateMax` might not always reflect the correct value if KLAP pool is updated.

Recommendation: Add the required check.

Update: Fixed in commits 2674d78 and 2b044d8.

QSP-12 Unseen Balance Contract Has to Be Validated

Severity: *Low Risk*

Status: Fixed

File(s) affected: `KlapStakingRewards.sol`

Description: `KlapStakingRewards.getReward()` checks the unseen balance and re-distributes it as a reward. It should be strictly checked that the staking token address is not equal to the

reward tokens when initializing the contracts.

Update: Fixed in commit "5496719".

QSP-13 Missing Input Validation

Severity: Informational

Status: Fixed

File(s) affected: `Depositor.sol`, `Configurator.sol`

Description: - In `setBorrowParameters()` and `setBorrowMax()` functions there is no control that `_borrowRate` and `borrowRateMax` are less than 100 or higher than zero.

- `Configurator.setupPreBoost()` does not validate `endsAt` to be greater than `block.timestamp`.

Recommendation: Add the required checks.

Update: Fixed in commit "a61b78f".

QSP-14 Zero-Division Leads to Transaction Failure

Severity: Informational

Status: Fixed

File(s) affected: `Depositor.sol`

Description: In `Depositor` contract if in `_deleverage` function:

```
uint256 adjBorrowRateX100 = borrowRateMax[asset] > 0 ? borrowRateMax[asset] * 100 - 1 : borrowRateMax[asset]; // Use borrowRateMax as close value with liquidation threshold
```

if `borrowRateMax[asset]` is zero, the transaction will fail because of zero division.

Recommendation: `borrowRateMax` mapping has to be set correctly for all listed assets.

Update: Fixed in commit "f6a9a2a"`

QSP-15 Block Timestamp Manipulation

Severity: Informational

Status: Acknowledged

File(s) affected: `KeyRewards.sol`, `KlapFeeDistributor.sol`, `KlapStakingRewards.sol`, `KlapVeDepositor.sol`, `Rewards.sol`, `KeyMinter.sol`, `TokenLocker.sol`

Related Issue(s): SWC-116

Description: Projects may rely on block timestamps for various purposes. However, it's important to realize that miners individually set the timestamp of a block, and attackers may be able to manipulate timestamps for their own purposes. If a smart contract relies on a timestamp, it must take this into account.

Recommendation: Consider at the protocol design level and write tests showing timestamp manipulation does not pose a problem.

Update: The key Finance team acknowledged the issue as follows: "Every contract mentioned doesn't depend on `block.timestamp`'s accuracy. The only function that can make differ by account is `KeyRewards.updateRewardPerUnitAssets()`. However, this is constrained to be called only an hour after the last call. A small error in the `block.timestamp` is negligible".

QSP-16 Using Custom Errors in Revert Is More Gas Efficient

Severity: Informational

Status: Fixed

Description: In solidity 0.8, a custom error type has been introduced that is much more gas efficient and allows to return of values along with the error message.
It is recommended to use a custom error inside `revert()` calls over a string error.
Reverts are used in the following lines:

1. contracts/klap/KlapVeDepositor.sol
    1. L228: `revert('Invalid Token Type')`
    2. L321: `revert('Invalid Token Type')`
    3. L368: `revert('Invalid Token Type')`
    4. L376: `revert('Invalid Token Type')`
    5. L441: `revert("Cannot renounce ownership")`
2. contracts/klap/Depositor.sol
    1. L182: `revert("Cannot renounce ownership")`
    2. L482: `revert("Wrong Token Type")`
3. contracts/klap/DepositorToken.sol
    1. L160: `revert("Cannot renounce ownership")`
4. contracts/klap/KlapFeeDistributor.sol
    1. L257: `revert("Cannot renounce ownership")`
5. contracts/klap/KlapVeVoter.sol
    1. L394: `revert("Cannot renounce ownership")`
6. contracts/klap/KlapStakingRewards.sol

1. L239: `revert("Cannot renounce ownership")`

7. contracts/klap/RewardAbstract.sol
   1. L55: `revert("Cannot renounce ownership")`
8. contracts/KeyMinter.sol
   1. L274: `revert("Cannot renounce ownership")`
9. contracts/KeyToken.sol
   1. L126: `revert("Cannot renounce ownership")`
10. contracts/TokenLocker.sol
    1. L378: `revert("Cannot renounce ownership")`

Recommendation: Define a custom error instead of passing a string to `revert()`. For example:

```
error invalidTokenType(IKlapVotingEscrow.TokenType tokenType);

revert invalidTokenType(tokenType);
```

Update: Fixed in commit "8abb4ea".

QSP-17 Allowance Double-Spend Exploit

Severity: Informational

Status: Fixed

File(s) affected: `KeyToken.sol`, `DepositToken.sol`, `KlapVeDepositor.sol`

Description: As it presently is constructed, the contract is vulnerable to the allowance double-spend exploit, as with other ERC20 tokens.

Exploit Scenario: 1. Alice allows Bob to transfer `N` amount of Alice's tokens (`N>0`) by calling the `approve()` method on `Token` smart contract (passing Bob's address and `N` as method arguments)

1. After some time, Alice decides to change from `N` to `M` (`M>0`) the number of Alice's tokens Bob is allowed to transfer, so she calls the `approve()` method again, this time passing Bob's address and `M` as method arguments
2. Bob notices Alice's second transaction before it was mined and quickly sends another transaction that calls the `transferFrom()` method to transfer `N` Alice's tokens somewhere
3. If Bob's transaction will be executed before Alice's transaction, then Bob will successfully transfer `N` Alice's tokens and will gain an ability to transfer another `M` tokens
4. Before Alice notices any irregularities, Bob calls `transferFrom()` method again, this time to transfer `M` Alice's tokens.

Recommendation: The exploit (as described above) is mitigated through use of functions that increase/decrease the allowance relative to its current value, such as `increaseAllowance()` and `decreaseAllowance()`. Furthermore, we recommend that developers of applications dependent on `approve()` / `transferFrom()` should keep in mind that they have to set allowance to 0 first and verify if it was used before setting the new value.

Update: Fixed in commit "e0fe170".

QSP-18 Functions Missing Access Restriction

Severity: Informational

Status: Acknowledged

File(s) affected: `KeyRewards.sol`, `Rewards.sol`

Description: Few functions that modify the contract state are left without an access restriction modifier, even if it seems trivial, and without risk, we recommend allowing such actions only through admin or special managing addresses to avoid any possible manipulation later on through a possible exploit.

Recommendation: Add admin modifier to `KeyRewards.recordPastLastKeyRPUOnce()`, `KeyRewards.recordPastKeyRPU()`, `Rewards.recordPastKlapRPUOnce()`.

Update: The Key Finance team acknowledged this issue as follows: "We intended to make the three functions able to be called by anyone since anyone can get stuck while calling the functions using corresponding values like `KeyRewards.keyRewardPerUnitAs()` set, `KeyRewards.lastKeyRewardPerUnitAsset()`, `Rewards.klapRewardPerUnitAsset()`".

Automated Analyses

Slither

Slither did not raise any significant findings.

Code Documentation

• `Depositor.getTotalOriginalBalance()` the function definition might be misleading the total amount available when deleveraging is less than what the function returns. Since there is a minimum liquidation threshold that has to be respected and is not taken into account. The comments must be changed.

• Reward.sol:L749 is missing description for the require statement.

• Depositor.sol:L547 I missing description for the require statement.

Adherence to Best Practices

- `KeyToken.setMinter()` does not revert in case the new minter address is already set. We recommend adding the necessary check.
- `KeyMinter.totalEmission()` naming is counterintuitive and really represents the emission in one week.
- Seems like all the computation done in `DepositToken.balanceOf()` can be executed in a single function in `Depositor` contract and avoid multiple external static calls, a similar issue is applicable to `DepositToken._transfer()`.
- `DepositToken.deleverageOnce()` does not check if the asset is listed as opposed to the other functions.
- `minLeverage` is not specified or set when `Depositor.addAsset()` is called.

Test Results

Test Suite Results

```
Generating typings for: 43 artifacts in dir: types for target: ethers-v5
Successfully generated 83 typings!
Compiled 43 Solidity files successfully

Network Info
============
> HardhatEVM: v2.9.3
> network:    hardhat

No need to generate any newer typings.


  KeyKlapVoter:unit
     ✓ function availableVotes(address _user) (816ms)
     ✓ voteForPools #1
case 1: vote ( user -> 0: 100, user2 -> 0: 100, 0: 300 ) (1020ms)
     ✓ voteForPools #2
case 2: reverts when trying to vote with more than own voting weight (316ms)
     ✓ voteForPools #3
case 3: cannot vote when pool's lastRewardTime = 0 (58ms)
     ✓ voteForPools #4
case 4: vote 100 pools (10302ms)
     ✓ voteForPools #5
case 5: vote with 0


  KeyMinter
     ✓ init #1 (59ms)
     ✓ init #2: fails if called again (75ms)
     ✓ default function calls (205ms)
     ✓ totalEmissionForCurrentPeriod #1
     ✓ totalEmissionForCurrentPeriod #2 (60ms)
     ✓ updateMiningParameters #1 (45ms)
     ✓ addLocker #1: new ones (225ms)
     ✓ addLocker #2: already one in the lockers (220ms)
     ✓ addLocker #3: fails when called with zero-address locker (59ms)
     ✓ addLocker #4: fails when called with zero-address receiver (42ms)
     ✓ addLocker #5: fails when called with zero-address locker (77ms)
     ✓ addLocker #6: fails when called with receiver already set (107ms)
     ✓ removeLocker #1 (207ms)
     ✓ removeLocker #2 (214ms)
     ✓ mintAndDistribute #1 (2009ms)
     ✓ mintAndDistribute #2: when emergency set for a receiver (1297ms)
     ✓ mintAndDistribute #3: fails when called with 0 periodsToProceed (216ms)
     ✓ mintAndDistribute #3: fails when startEpochTime is not set (144ms)
     ✓ getCurrentPeriodNo #0: (120ms)
    ownership
       ✓ renounceOwnership #0: fails

  new_integration/Depositor.deposit.test.ts: Integration Test for Depositor.deposit. Done in more elaborate way
    deposit
       ✓ deposit #0: successful deposit (54861ms)
       ✓ deposit #1: deposit 0 DAI, which results in fail (406ms)
       ✓ deposit #2: deposit 10**(-18) DAI, which results in fail (because it's less than minLeverage) (384ms)
       ✓ deposit #3: deposit 1 DAI (10**18) (3438ms)
beforeState.totalShare 1000000000000000000
beforeState.totalOriginalBalance 999999997618789244
       ✓ deposit #4: deposit 1 DAI (10**18) when some amount less than it is deposited already (8594ms)
       ✓ deposit #5: deposit 10**9(+10**(-18)) DAI (10**27 + 1) when some amount less than it is deposited already (6176ms)
       ✓ deposit #6: deposit 0 DAI when some amount more than it is deposited already, then it fails (2462ms)
       ✓ deposit #7: deposit 1 DAI (10**decimal) when some amount more than it is deposited already (5550ms)
       ✓ deposit #8: deposit 10**9+10**(-18) DAI (10**(decimal+9)+1) when some amount more than it is deposited already (6134ms)
       ✓ deposit #9: deposit 1 DAI (10**decimal) in pre-boost phase (5855ms)
       ✓ deposit #10: deposit 1 DAI (10**decimal) in pre-boost phase when the amount less than 10**decimal is already deposited (5399ms)
       ✓ deposit #11: deposit 10**9+10**(-18) DAI (10**(decimal+9)+1) in pre-boost phase when the amount less than 10**(decimal+9)+1 is already deposited (2407ms)
       ✓ deposit #12: deposit 1 DAI (10**decimal) in pre-boost phase when the amount more than 10**decimal is already deposited (5634ms)
       ✓ deposit #13: successful deposit (21313ms)
       ✓ deposit #14: deposit 0 USDC, which results in fail (342ms)
       ✓ deposit #15: deposit 10**(-6) USDC, which results in fail (because it's less than minLeverage) (352ms)
       ✓ deposit #16: deposit 1 USDC (10**6) (3028ms)
       ✓ deposit #4: deposit 1 USDC (10**6) when some amount less than it is deposited already (7148ms)
       ✓ deposit #5: deposit 10**9(+10**(-6)) USDC (10**15 + 1) when some amount less than it is deposited already (5115ms)
       ✓ deposit #6: deposit 0 USDC when some amount more than it is deposited already, then it fails (2253ms)
       ✓ deposit #7: deposit 1 USDC (10**decimal) when some amount more than it is deposited already (5618ms)
       ✓ deposit #8: deposit 10**9+10**(-6) USDC (10**(decimal+9)+1) when some amount more than it is deposited already (5924ms)
       ✓ deposit #9: deposit 1 USDC (10**decimal) in pre-boost phase (5529ms)
       ✓ deposit #10: deposit 1 USDC (10**decimal) in pre-boost phase when the amount less than 10**decimal is already deposited (4926ms)
       ✓ deposit #11: deposit 10**9+10**(-6) USDC (10**(decimal+9)+1) in pre-boost phase when the amount less than 10**(decimal+9)+1 is already deposited (2824ms)
       ✓ deposit #12: deposit 1 USDC (10**decimal) in pre-boost phase when the amount more than 10**decimal is already deposited (5579ms)
       ✓ deposit #26: N cases for deposit (33786ms)
       ✓ deposit #27: deposit 1 DAI (10**18) (2058ms)
       ✓ deposit #28: deposit 1 DAI (10**18) when some amount less than it is deposited already (3430ms)
       ✓ deposit #29: fails when paused (409ms)
       ✓ deposit #30: deposit 1 DAI (10**18) when some amount less than it is deposited already by the same account (4862ms)
       ✓ deposit #31: deposit 1 DAI (10**18) when borrowDepth = 1 & borrowRate = 0 (2660ms)
       ✓ deposit #32: deposit 1 DAI (10**18) when borrowDepth = 1 & borrowRate = 0 (371ms)

  new_integration/Depositor.leverage.test.ts: Integration Test for Depositor._leverage, Depositor._deleverage, and other related functions. Done in more elaborate way
    initialize & setAddresses
       ✓ initialize #1: fails when called with zero-address (1914ms)
       ✓ setAddresses #0:
       ✓ setAddresses #1: fails with 0 address (70ms)
       ✓ setAddressesForLaunchPhase #0:
       ✓ setAddressesForLaunchPhase #1: fails with 0 address (38ms)
    admin & ownership
       ✓ setAdmin #1: fails
       ✓ onlyAdmin #1: call fails
       ✓ renounceOwnership #0: fails
    leverage-related functions
       ✓ pause #0: (47ms)
       ✓ pause #1: fails when called with 0 address
       ✓ pause #2: calling pause again doesn't change paused valued as true  (69ms)
    leverage
       ✓ leverage #0: successful leverage (1786ms)
       ✓ leverage #1: successful leverage (2072ms)
       ✓ leverage #2: successful leverage (2225ms)
       ✓ leverage #3: successful leverage (1838ms)
       ✓ leverage #4: successful leverage (2034ms)
       ✓ leverage #5: successful leverage (3606ms)
       ✓ leverage #6: successful leverage (3650ms)
       ✓ resume #0: succeeds, variated from leverage #1 (5519ms)
       ✓ resume #1: fails when not deleveraged before calling resume, variated from leverage #1 (1933ms)
       ✓ setBorrowParams #1: setBorrowParams is callable only after deleveraged and before leverage, variated from leverage #1 (7314ms)
    deleverage
       ✓ deleverage #0: successful deleverage (4969ms)
       ✓ deleverage #1: successful deleverage (755ms)
       ✓ deleverage #2: successful deleverage (913ms)
       ✓ deleverage #3: successful deleverage (4687ms)
       ✓ deleverage #4: successful deleverage (5112ms)
       ✓ deleverage #5: successful deleverage (3256ms)
       ✓ deleverage #6: deleverage fails (11558ms)
       ✓ deleverage #7: successful deleverage (5357ms)
       ✓ deleverage #8: fails when asset is not inIncluded (6702ms)
       ✓ deleverage #9: fails when asset is not paused (1899ms)
       ✓ deleverage #10: fails when cannot repay debt (2897ms)
       ✓ deleverage #11: when Depositor's remaining asset balance is larger than last debt (3403ms)
       ✓ deleverage #13: borrowRate = 0, borrowDepth = 1 when Depositor's remaining asset balance is larger than last debt (2457ms)
       ✓ deleverage #12: borrowRate = 0, borrowDepth = 1 (2411ms)
       ✓ pauseAndDeleverage #0: (4262ms)
       ✓ pauseAndDeleverage #1: fails when asset is not inIncluded (237ms)
```

```
      ✓ deleverageOnce #0: deleverageOnce call succeeds (3075ms)
      ✓ deleverageOnce #1: deleverageOnce call succeeds when using borrowRate which is less than liquidation threshold when there has been no rebalancing for a day (4870ms)
      ✓ deleverageOnce #2: deleverageOnce call fails when borrowDepth = 1 & borrowRate = 0 (variated from #0) (910ms)
      ✓ deleverageOnce #3: deleverageOnce call fails when using maxLTV when there has been no rebalancing for a day and borrowDepth = 1 & borrowRate = 0 (variated from #1) (926ms)

new_integration/Depositor.withdraw.test.ts: Integration Test for Depositor.withdraw. Done in more elaborate way
  withdraw
      ✓ withdraw #0: successful withdrawal (6438ms)
      ✓ withdraw #1: withdraw 0 DAI, which results in fail (2889ms)
      ✓ withdraw #2: withdraw 1 DAI when less than it is deposited by the withdrawer, which results in success (7278ms)
      ✓ withdraw #3: withdraw almost 1 DAI when equal amount is deposited by the withdrawer, which results in success (7661ms)
      ✓ withdraw #4: withdraw 1 DAI when more than it is deposited by the withdrawer, which results in success (7764ms)
      ✓ withdraw #5: withdraw almost 1 DAI when just 10**(-18) DAI more is deposited by the withdrawer, which results in success (7092ms)
      ✓ withdraw #6: withdraw almost 1 DAI when just 333 * 10**(-18) DAI more is deposited by the withdrawer, which results in success (7921ms)
      ✓ withdraw #7: withdraw almost 1 DAI when just 10**(-15) DAI more is deposited by the withdrawer, which results in success (7478ms)
      ✓ withdraw #8: withdraw almost 1 DAI when same amount is deposited by withdrawer and just 333 * 10**(-18) DAI more is deposited by another user, which results in success (14579ms)
      ✓ withdraw #9: withdraw almost 1 DAI when same amount is deposited by withdrawer and just 1000 * 10**(-18) DAI more is deposited by another user, which results in success (14232ms)
      ✓ withdraw #10: withdraw 10**9(+10**(-18)) DAI when more than is deposited by the withdrawer, which results in success (8514ms)
      ✓ withdraw #11: withdraw all by MaxUint256 DAI when more than is deposited by the withdrawer, which results in success (8100ms)
      ✓ withdraw #12: withdraw 10 DAI when more than is deposited by the withdrawer and less than it is deposited by another user, which results in success (11060ms)
      ✓ withdraw #13: withdraw 10 DAI when more than is deposited by the withdrawer and more than it is deposited by another user, which results in success (10489ms)
      ✓ withdraw #14: withdraw 1 DAI when more than is deposited by the withdrawer and less than it is deposited by another user, which results in success (7614ms)
      ✓ withdraw #15: withdraw 1 DAI when more than is deposited by the withdrawer and less than it is deposited by another user, which results in success (8184ms)
      ✓ withdraw #16: withdraw 1 DAI when more than is deposited by the withdrawer and less than it is deposited by another user, which results in success (7294ms)
      ✓ withdraw #17: withdraw 1 DAI when less than it is deposited by the withdrawer, which results in success (3689ms)
      ✓ withdraw #18: withdraw almost 1 DAI when same amount is deposited by withdrawer and just 1000 * 10**(-18) DAI more is deposited by another user, which results in success (13765ms)
      ✓ withdraw #19: withdraw 10 DAI when more than is deposited by the withdrawer and more than it is deposited by another user, which results in success (5465ms)
      ✓ withdraw #20: withdraw 1 DAI when less than it is deposited by the withdrawer, which results in success (13999ms)
      ✓ withdraw #21: withdrawal fails when paused (2390ms)
      ✓ withdraw #22: withdraw 1 DAI when more than it is deposited by the withdrawer, which results in success (7144ms)
      ✓ withdraw #23: withdraw 1 DAI when less than it is deposited by the withdrawer, which results in success (3753ms)
      ✓ withdraw #24: withdraw 1 DAI when more than it is deposited by the withdrawer, which results in success (4085ms)

new_integration/DepositToken.test.ts: Integration Test for DepositToken. Done in more elaborate way
    ✓ initialize #1: fails when called with zero-address (1202ms)
  transfer
      ✓ transfer #1 (6176ms)
      ✓ transfer #2 (6991ms)
      ✓ transfer #3 (21747ms)
      ✓ transfer #4 (4612ms)
      ✓ transfer #5: fails when send to zero-address (2318ms)
      ✓ transfer #6: nothing happens when send 0 amount (2662ms)
      ✓ transfer #7: succeeds in preboost phase (6817ms)
      ✓ transfer #8: fails when paused (4930ms)
  transferFrom
      ✓ transferFrom #1 (6989ms)
      ✓ transferFrom #2: fails when no approval (2112ms)
      ✓ transferFrom #3: fails when sending from zero-address (2072ms)
      ✓ transferFrom #4: when approved maxUint (2966ms)
  etc
      ✓ balanceOf #1: 0 when not deposited
      ✓ totalSupply #0: (2242ms)
      ✓ approve #1: (2205ms)
      ✓ mint #0: fails when not from depositor (93ms)
      ✓ burn #0: fails when not from depositor (2485ms)
      ✓ positionAmount #0: (5368ms)
  ownership
      ✓ renounceOwnership #0: fails

new_integration/KeyKlapVoter
  initialize & setAddresses
      ✓ initialize #1: fails when called with zero-address (458ms)
      ✓ setAddresses #0:
      ✓ setAddresses #1: fails with 0 address (67ms)
  basic function calls
      ✓ view function calls (111ms)
      ✓ function calls (94ms)
      ✓ function calls params (103876ms)
  voteForPools
      ✓ voteForPools #0: basic success call (2067ms)
  submitVotes
      ✓ submitVotes #0: basic success call (12570ms)
  submitVotesWithTokenID
      ✓ submitVotesWithTokenID #0: basic success call (3541ms)

new_integration/KeyRewards.test.ts: Integration test for KeyRewards. Done in more elaborate way
  updateRewardPerUnitAssets
      ✓ updateRewardPerUnitAssets #0: initial (7043ms)
      ✓ updateRewardPerUnitAssets #1: call after 2 weeks since minting KEY (13648ms)
  recordPastKeyRPU
      ✓ recordPastKeyRPU #0: (3921ms)
      ✓ recordPastKeyRPU #1: fails when there was no key calculation before
      ✓ recordPastKeyRPU #2: update 2 weeks (3388ms)
  recordPastLastKeyRPUOnce
      ✓ recordPastLastKeyRPUOnce #0 (3213ms)

new_integration/KeyKlapVoter
  KeyToken: basic function calls
      ✓  (179ms)

new_integration/KlapFeeDistributor
  initialize & setAddresses
      ✓ initialize #1: fails when called with zero-address (194ms)
  basic function calls
      ✓ function calls (659ms)

new_integration/KlapStakingRewards
  initialize & setAddresses
      ✓ initialize #1: fails when called with zero-address (243ms)
  basic function calls
      ✓ view function calls (40ms)
      ✓ function calls (635ms)
  getReward
      ✓ getReward #0 (239ms)
  updatePeriodFinishForKeyOnce
      ✓ updatePeriodFinishForKeyOnce #0 (2745ms)

new_integration/KlapVeDepositor
    ✓ onERC721Received #1: succeeds for first 2 ERC20-typed tokens when not in preBoost phase (960ms)
    ✓ onERC721Received #2: onERC721Received(address,address,uint256,bytes) short lock duration (524ms)
    ✓ onERC721Received #3: onERC721Received(address,address,uint256,bytes) small to large number of KLAP (907ms)
    ✓ onERC721Received #4: onERC721Received(address,address,uint256,bytes) short to long lock duration (42984ms)
    ✓ onERC721Received #5: onERC721Received(address,address,uint256,bytes) depositor checkup (11254ms)
    ✓ onERC721Received #6: onERC721Received(address,address,uint256,bytes) not from veDepositor (269ms)
    ✓ depositTokens basic case #1: (2394ms)
    ✓ depositTokens basic case #1-1: only whitelisted account can deposit token when in PreBoost phase (2356ms)
    ✓ depositToken basic case 2 (844ms)
    ✓ depositToken basic case 3 (105ms)
    ✓ depositToken basic case 4 (71ms)
    ✓ depositToken basic case 5 (456ms)
    ✓ VE lock amount test (11831ms)
    ✓ claimFromVeDistributor #0: (1775ms)
    ✓ mergeMulti #1: fails when in preBoost phase (260ms)
  initialize & setAddresses
      ✓ initialize #1: fails when called with zero-address (726ms)
      ✓ setAddresses (87ms)
      ✓ setAddressesLaunchPahse (55ms)
  basic function calls
      ✓ function calls (193ms)
      ✓ setStakerRatio #0:
      ✓ approve #0: (113ms)
      ✓ transfer #0: (81ms)
      ✓ burn #0: (58ms)

new_integration/Rewards.new.test.ts: Integration test for Rewards. Done in more elaborate way
  Klap contract verification: ChefIncentivesController.claim, MultiFeeDistribution.exit
      ✓ ChefIncentivesController.claim & MultiFeeDistribution.exit #1 (2039ms)
      ✓ ChefIncentivesController.claim & MultiFeeDistribution.exit #2 (1166ms)
  updateRPU
      ✓ updateRPU #1:  (280ms)
      ✓ updateRPU #1-2:  (5837ms)
      ✓ updateRPU #2: updateRPU success in normal case (3051ms)
      ✓ updateRPU #2-2: updateRPU success in normal case (18127ms)
      ✓ updateRPU #3: (2519ms)
      ✓ updateRPU #4: (3246ms)
      ✓ updateRPU #4-2: (5634ms)
      ✓ updateRPU #5: (2719ms)
      ✓ updateRPU #6: (3284ms)
      ✓ updateRPU #7: (4074ms)
      ✓ updateRPU #8: (7898ms)
  getReward
      ✓ getReward #1: (7344ms)
      ✓ getReward #2: (7547ms)
      ✓ pendingRewards #1: case corresponding to 'getReward #1' (5898ms)
      ✓ pendingRewards #2: case corresponding to 'getReward #2' (7906ms)
      ✓ getReward #3: (5189ms)
      ✓ getReward #4: (5972ms)
```

```
      ✓ pendingRewards #3: case corresponding to 'getReward #4' (5957ms)
      ✓ pendingRewardsInDetail #1: case corresponding to 'getReward #1' (6565ms)
      ✓ pendingRewardsInDetail #2: case corresponding to 'getReward #4' (6101ms)
      ✓ claimLockerRewards #1: claimLockerRewards after getReward #1 (6059ms)
      ✓ claimLockerRewards #2: claimLockerRewards after getReward #4 (9601ms)

new_integration/Rewards
   ✓ updateRPU
   ✓ recordRewards
   ✓ basic case #1 (125375ms)
   ✓ multiple deposit amount (3593ms)
   ✓ deposit n times (18616ms)
   ✓ multiple users (23891ms)
   ✓ Claim Locker Rewards (1790ms)
   ✓ Claim Locker Rewards Preboost (138ms)
   ✓ Claim Locker Rewards Require check (1862ms)

new_integration/TokenLocker
  initialize & setAddresses
     ✓ initialize #1: fails when called with zero-address (193ms)
  basic function calls
     ✓ view function calls (183ms)
     ✓ function calls (320ms)

new_unit/Rewards.new.test.ts: Unit test for Rewards. Done in more elaborate way
  getAirdropTokens
     ✓ getAirdropTokens #1 (991ms)

new_unit/Rewards.new.test.ts: Unit test for Rewards. Done in more elaborate way
  calculateReward
     ✓ calculateReward #1: initial call (RPU=0) (613ms)
     ✓ calculateReward #2: initial call (RPU>0) (43ms)
     ✓ calculateReward #3 (42ms)
     ✓ calculateReward #4 (44ms)
     ✓ calculateReward #5 (46ms)
     ✓ calculateReward #6 (44ms)
     ✓ calculateReward #7 (45ms)
     ✓ calculateReward #8 (45ms)
     ✓ calculateReward #9 (39ms)
     ✓ calculateReward #10 (46ms)
     ✓ calculateReward #10 (47ms)
     ✓ calculateReward #11 (43ms)
     ✓ calculateReward #12 (42ms)
     ✓ calculateReward #13 (44ms)
     ✓ calculateReward #14 (43ms)
     ✓ calculateReward #15 (43ms)
     ✓ calculateReward #16 (43ms)
     ✓ calculateReward #17 (38ms)
     ✓ calculateReward #18 (39ms)
     ✓ calculateReward #19 (38ms)
     ✓ calculateReward #20 (48ms)
     ✓ calculateReward #21 (94ms)
     ✓ calculateReward #22 (75ms)
     ✓ calculateReward #23 (67ms)
     ✓ calculateReward #24 (60ms)
     ✓ calculateReward #25 (58ms)
     ✓ calculateReward #26 (187ms)
     ✓ calculateReward #27 (58ms)
     ✓ calculateReward #28 (73ms)
     ✓ calculateReward #29 (74ms)
     ✓ calculateReward #30 (82ms)
     ✓ calculateReward #31 (58ms)
     ✓ calculateReward #32 (94ms)
     ✓ calculateReward #32 (94ms)
     ✓ calculateReward #32 (91ms)
     ✓ calculateReward #33 (78ms)
     ✓ calculateReward #34 (123ms)
     ✓ calculateReward #35 (72ms)
     ✓ calculateReward #35 (71ms)
     ✓ calculateReward #36 (72ms)
     ✓ calculateReward #36 (75ms)
     ✓ calculateReward #37 (44ms)
     ✓ calculateReward #38 (44ms)
     ✓ calculateReward #39 (43ms)
     ✓ calculateReward #40
  recordKlapRewards
     ✓ recordKlapRewards #1: globally initial call on first week (74ms)
     ✓ recordKlapRewards #2: globally initial call on 6th week (128ms)
     ✓ recordKlapRewards #3: initial call on first week when RPU>0 (80ms)
     ✓ recordKlapRewards #4: initial call on 6th week (107ms)
     ✓ recordKlapRewards #5: initial call on 6th week (121ms)
     ✓ recordKlapRewards #5: initial call on 6th week (159ms)
     ✓ recordKlapRewards #6: initial call on 6th week (227ms)
  recordPastKlapRPUOnce
     ✓ recordPastKlapRPUOnce #0 (45ms)
     ✓ recordPastKlapRPUOnce #1: called twice (96ms)

TokenLocker
   ✓ #constructor (119ms)
  lock
     ✓ #lock #0: (518ms)
     ✓ #lock after lock with same locking period (364ms)
     ✓ #lock after lock with different locking period (358ms)
     ✓ #lock for 0 week
     ✓ #lock for 18 weeks
     ✓ #lock zero amount
  #extendLock
     ✓ extendLock after a lock (241ms)
     ✓ extendLock after a lock with 0 week (67ms)
     ✓ extendLock after a lock with 18 new weeks (72ms)
     ✓ extendLock after a lock with new weeks less than weeks (68ms)
     ✓ extendLock after a lock with 0 amount (67ms)
  exitStream
     ✓ withdrawAndInitExitStream keyToken.address#1, claimableExitStreamBalance, userBalance, totalWeight, activeUserLocks (1268ms)
     ✓ streamableBalance #1 (997ms)


296 passing (22m)
```

Code Coverage

Quantstamp usually recommends developers increase the branch coverage to 90% and above before a project goes live to avoid hidden functional bugs that might not be easy to spot during the development phase. For branch code coverage, the current targeted files by the audit achieve a low score that must be enhanced before deployment.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	98.11	79.58	98.04	98.87	
Configurator.sol	85.71	62.5	100	100	
KeyMinter.sol	100	80	100	100	
KeyToken.sol	94.87	64.29	90.91	93.02	45,46,47
PauseableUpgradeable.sol	100	75	100	100	
TokenLocker.sol	99.06	90.38	100	100	
contracts/klap/	89.79	70.37	94.09	89.81	
DepositToken.sol	100	100	100	100	
Depositor.sol	95.82	70.28	95.92	95.58	… 602,603,713
KeyKlapVoter.sol	100	87.93	100	100	
KeyRewards.sol	89.5	66.1	92.86	91.35	… 467,470,471
KlapFeeDistributor.sol	56.34	50	81.82	56.16	… 252,254,259
KlapStakingRewards.sol	82.02	82.5	92.86	82.22	… 216,217,218
KlapVeDepositor.sol	87.41	65.56	100	86.49	… 430,431,432
RewardAbstract.sol	100	100	80	90.91	57
Rewards.sol	88.67	64.93	89.74	89.04	… 801,802,803
All files	**91.19**	**71.92**	**94.94**	**91.36**	

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

```
02f0db15e04a47099c508ae3a2df90f7f01e97358545f4b520f6e7ee0297ea26 ./contracts/Configurator.sol
ea0a96f632c45ba001af1a286d62d55d97063af721353aa1ef8f3fcf959429a6 ./contracts/TokenLocker.sol
d54a056d27e3b97994a3c803fb41ae1bcd534575e957cd5088439ee0a40dd75b ./contracts/PauseableUpgradeable.sol
a6d134b307cef1304f981c08c3ea636acd1378786cecfd728a2e254dc735d427 ./contracts/KeyToken.sol
a42eb3d6dc5ddfdd915430d9c1e73d18d7046f2ee6c24bb941c4595dde3c43a0 ./contracts/KeyMinter.sol
34d8f1c38fc7f1892a0f165e237d1835214d6d73025e74bdd2160dd6ef34e09a ./contracts/klap/KeyRewards.sol
e3aad6e1a530402cf473189f01800df44b0967ea2c6f7750461c5daca46a793c ./contracts/klap/RewardAbstract.sol
353aafb840c79250edd32aae8f5747c8e8a4c0c2e6eefd37c8506be61aed3f4d ./contracts/klap/KeyKlapVoter.sol
381a2303971d3f2d04bfbd833b72a8d575a21e8e5dbeb89842e4b8ea6cfe71f8 ./contracts/klap/Depositor.sol
17c56636bed5fa81da0dc1c7afa2af64d3ac54cd212997def6b2df59f94db35e ./contracts/klap/KlapVeDepositor.sol
6eb38eb6cf9d2bebcc05da07e3daf863c6264e54f61cf7ed38805879f6120de5 ./contracts/klap/KlapFeeDistributor.sol
d5e006f298aa95ca4f6d1e1d25e7016ee0c9d4267c35ca86228923c5c850a60d ./contracts/klap/DepositToken.sol
5fd5581ba46690bd3b5571b2b82e77bf1a6e921f38566ca9b7b92d00caa00fef ./contracts/klap/KlapStakingRewards.sol
24c235e099513c7482af87eaea633f0a2f2a0061df50131ef50c12bf67bbe3c5 ./contracts/klap/Rewards.sol
```

Tests

```
4cc5d9f80bd8d3fad9321b4728af1f9d0c608af9e659190b59fee001878d68ba ./test/TokenLocker.test.ts
b2fdefc5f5fc79e3b24c88b3cb27dee1a8d98298cdddf5a7f627dbad0c263901 ./test/utils.ts
867c98d519e96a1d5cb1c28122f53a8fbd959fa16225d105ab592340b5c99c58 ./test/KeyKlapVoter.ts
7e462dc46f1f9f55602eda297aa3d896c6691c2787af9c0c2be5483aba817da5 ./test/KeyMinter.test.ts
a3de11f9fffbb53806b5bd0d8fe9b9f1716ffd70d4989fc4144ca77a982b1571 ./test/new_integration/TokenLocker.new.test.ts
7336f5dacbd57d2880618d5dfe4d7a0483868feeb79e3eed6b325b21d3ceb8ac ./test/new_integration/KeyToken.test.ts
df103dd90cd08f4a9f7fded24829acb6e6721e04e9f09a236bb79d8f3fdb7a91 ./test/new_integration/KeyKlapVoter.new.test.ts
81889d740d3d87703e6211ce708ac1c46a0931e0e1358c9f0bb0edeaaf766086 ./test/new_integration/KlapStakingRewards.new.test.ts
5ffc84931616dfdd27cefb308b25069b58078e4d65fc1743de4093e403b8a6fe ./test/new_integration/KeyRewards.test.ts
50393e2c8aede44b8d4d26da46dc54ea4ee4d0185c776fadbdeceafe2c5293ad ./test/new_integration/Depositor.withdraw.test.ts
dd35362f3ae8fde6bcdf8fade914b8a991350058b36afe14a9159476b5c3ec1b ./test/new_integration/Depositor.deposit.test.ts
cd5d1f2bcdb078625bda0e1d0ac1bdd7deb7e562accfb811d7b60666a9c6b6a1 ./test/new_integration/KlapVeDepositor.test.ts
edfa230928e2a9e567d78907e619a61a46c43141a07882e7f166a6fd739c3772 ./test/new_integration/Depositor.leverage.test.ts
94aba2737b3afdb5122293c7f3809ad2a91799557db4a334136d1de84c33a248 ./test/new_integration/KlapFeeDistributor.new.test.ts
ce4a22959e8cdb07329af8851d3b7a0b29434fb21a0de950be7eb89e001df906 ./test/new_integration/Rewards.test.ts
48a55fbd0a70770d386797f5c57784445f5d46fc504181c2ff2fa9ad2d2b9451 ./test/new_integration/Rewards.new.test.ts
c26bd064b3a3d1ef083e94ef49e5ef9f8947844006144a83d2590cae9bd9e11f ./test/new_integration/DepositToken.test.ts
395d54c924b7a38df46dfa4cc8dfcdf21dc88f3a9f94ae2f832b50234f8cb4c0 ./test/new_unit/Rewards.airdrop.test.ts
a920c9a81d4414c1805a8e636778985edf524fa18e9e589d1c4566aaf0138cd1 ./test/new_unit/Rewards.new.test.ts
8a2bf2d37274428d9a1406816a6466162e9d4cb193340f13829b840eeb3d65b3 ./test/harness/klapex.ts
d1586aa3f3e2f36caaccf29c2bae882cda488e5eb722fa6f64ba4de82e24225c ./test/harness/Klap.ts
26a870df6992a0444174e8aaa2683fda5dfdb981db0d6707a9ef616f8f8d3295 ./test/harness/RealKlap.ts
2a8569c02fa068ef129742a03b66f0bdea85b60908108cd84c65ff32b7bbfbed ./test/harness/helpers.ts
```

Changelog

- 2022-10-21 - Initial report
- 2022-11-13 - Fix review (e93f349)
- 2022-11-20 - Report update

About Quantstamp

Quantstamp is a global leader in blockchain security backed by Pantera, Softbank, and Commonwealth among other preeminent investors. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its white glove security and risk assessment services.

The team consists of web3 thought leaders hailing from top organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Many of the auditors hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 250 audits and secured over $200 billion in digital asset risk from hackers. In addition to providing an array of security services, Quantstamp facilitates the adoption of blockchain technology through strategic investments within the ecosystem and acting as a trusted advisor to help projects scale.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Aave, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT