Quantstamp

Swaap Finance Safeguard Pool

Executive Summary

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

Type	Exchange Pool
Timeline	2023-06-13 through 2023-06-23
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Source Code	• swaap-labs/swaap-v2-monorepo #85e0ef6
Auditors	• Rabib Islam Auditing Engineer • Danny Aksenov Senior Auditing Engineer • Julio Aguliar Auditing Engineer • Ruben Koch Auditing Engineer

Documentation quality	High
Test quality	Medium
Total Findings	8 Fixed: 3 Acknowledged: 5
High severity findings	0
Medium severity findings	2 Acknowledged: 2
Low severity findings	3 Fixed: 1 Acknowledged: 2
Undetermined severity findings	0
Informational findings	3 Fixed: 2 Acknowledged: 1

Summary of Findings

Update: We find that all issues have been addressed being either fixed or acknowledged. However, regarding SWP-2, a medium-severity issue, we found that it would benefit from a direct resolution that would further minimize risks to the protocol inherent to depegging events. Moreover, we are concerned that the test suite does not yet have full coverage, as code of this complexity would benefit strongly from a robust test suite.

Initial audit: Quantstamp performed an audit on Swaap Finance's Safeguard Pool mechanism based on the code in the linked repository.

Swaap Finance is an exchange protocol that allows users to swap tokens as well as provide liquidity in "pools". Safeguard Pool represents on-chain logic that aims to implement features including:

- using on-chain oracles in order to ensure accurate swap prices;
- implementing "safeguards" and "penalties" in order to protect liquidity providers and traders from undesired effects.

The safeguards and penalties can be roughly summarized as follows:

- quotes must be signed by an authorized signer;
- the "performance" of the pool (relative to a basic holding strategy) must not deviate too quickly;
- the price of a quote must be sufficiently close to an oracle's price (save for the case of stablecoins, where this check may be conditionally foregone);
- traders may be penalized for waiting too late to execute a swap;
- traders may be penalized for if their swap induces more volatility than was expected at the time of a quote's signing;
- traders may be penalized for using a quote that was expected to be executed by someone else (as determined by account address).

Key concerns for the audit were

- determining whether the safeguards and penalty systems were implemented correctly;

- determining whether any loopholes were present that may allow the safeguards to be somehow circumvented;
- assessing whether the safeguards and penalties sufficiently fulfil the stated objectives of the developer.

Ultimately, the most concerning issues centered around the unusually high power given to the quote signer (a single address for a given pool) and the potential for depegging events to be exploited to some degree.

We recommend the client address all the issues discussed in this report.

ID	DESCRIPTION	SEVERITY	STATUS
SWP-1	Privileged Roles and Ownership	● Medium	Acknowledged
SWP-2	Front-Run Potential Around Evaluating Token Price Peg States	● Medium	Acknowledged
SWP-3	Signature Malleability	● Low	Acknowledged
SWP-4	Avoid Potential Integer Overflow Underflow	● Low	Fixed
SWP-5	Missing Chainlink Pricefeed Decimal Check	● Low	Acknowledged
SWP-6	Unlocked Pragma	● Informational	Fixed
SWP-7	Users Can Front-Run Swaps	● Informational	Acknowledged
SWP-8	Notation of Equations in Documentation Differs From Implementation	● Informational	Fixed

Assessment Breakdown

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

Disclaimer

Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code 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

1. Code review that includes the following
   1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential

vulnerabilities.
    3. 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:
    1. 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.
    2. 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, clarity, 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.

Scope

The scope of this audit was limited to the Safeguard Pool feature of Swaap Finance.

Files Included

Repo: https://github.com/swaap-labs/swaap-v2-monorepo(6537ca745fba4aaf8b971b89e1f7043ce7b1b0a3) Files: pkg/safeguard-pool/contracts/*

Files Excluded

Repo: https://github.com/swaap-labs/swaap-v2-monorepo(6537ca745fba4aaf8b971b89e1f7043ce7b1b0a3) Files: pkg/safeguard-pool/contracts/test

Findings

SWP-1 Privileged Roles and Ownership

● Medium Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

> All the signer and pool controller permissions will be extensively explained in the user's documentation upon release. Slippage penalties are per-quote specific and thus cannot be fixed by the pool.

File(s) affected: `SafeguardPool.sol`

Description: The following actions can be executed exclusively by the contract owner:

- `setFlexibleOracleStates()`
- `setMustAllowlistLPs()`
- `setSigner()`
- `setPerfUpdateInterval()`
- `setMaxPerfDev()`
- `setMaxTargetDev()`
- `setMaxPriceDev()`
- `setManagementFees()`

In addition to the above, there is a designated `signer` account that signs the quotes that are allowed to be used with the pool. In determining and signing quotes, this account has much power, including the following:

- the ability to set `quoteBalances`, which is a determining factor in whether the balance-based penalty is triggered;
- the ability to set `expectedOrigin` and `originBasedSlippage`, which are used to determine how much a user who is not `expectedOrigin` is penalized for executing the associated quote;
- the ability to set `startTime` and `timeBasedSlippage`, which are determining factors in how much one is penalized for waiting to execute a quote.

However, in general it must be noted that `signer` is able to determine all aspects of quotes. This means that the quality of quotes (as pertains both to LPs and traders) is contingent on the proper functionality of `signer`. In the case that keys are lost or that `signer` for whatever reason behaving inappropriately, the quotes may be unsuitable for one or more parties involved with a pool.

Furthermore, the `SafeguardPool` inherits from BalancerV2's `BasePool` contract which in turn inherits from the `TemporarilyPausable` contract. This inheritance enables the setting of a time window in which the contract can be paused by the owner and a buffer time window where it can be unpaused again. The maximum values for those values, as defined by the inherited Balancer code, is `270 days` and `90 days`, respectively.

Recommendation: Make sure that the above permissions and powers are well-represented to the users in documentation. Consider further decentralizing the `signer` role, for example, by making slippage penalties rely on contract storage if per-quote specificity is not absolutely required.

SWP-2 Front-Run Potential Around Evaluating Token Price Peg States

● Medium ⓘ Acknowledged

Update

Although we understand that gas is an important point of consideration in protocol design and implementation, we find that assuming the price of any given asset is risky in principle, and we recommend against it.

The client provided the following explanation:

> A lot of conditions must overlap in order to have significant losses if the price oracle peg is on: 1- Quote Signer is malicious or misbehaving 2- balancesSafeguard() deviations are important (set by the pool managers) 3- Price depeg is important and fast (for slow depeg evaluateStablesPegStates() can be used to unpeg the price oracle permissionlessly)

File(s) affected: `SafeguardPool.sol`

Description: Within `SafeguardPool`, if certain conditions are met, tokens can be pegged to the value of one, which is intended to reduce gas costs for swaps with stablecoins. This pegging can be activated or deactivated in a permissionless manner if it is within or outside, respectively, of a certain price range compared to the Chainlink oracle.

There is a guaranteed profit in frontrunning a successful switch of the `isTokenPegged` flag for an asset in either way (assuming a favorable quote is provided). However, the scenario of frontrunning a removal of a pegging could have severe consequences. If the pegged token's real price is de-pegging in the sub-dollar direction, swaps with the pegged token as the input could be very rewarding. More importantly, the performance and balances safeguard are also performed with the pegged asset value, resulting in the reality of the drained TVL of a swap potentially vastly exceeding `1 - maxTargetDev`.

However, given that the `_balancesSafeguard()` function limits the potential losses in the event of a depegging, the risk associated with (temporary) non-reliance on the oracle price may be considered acceptable.

Recommendation: While the change of the pegging flag would be a very rare event, we do believe the protocol is not sufficiently protected. The most secure approach to the issue would be the removal of the pegging feature.

SWP-3 Signature Malleability

● Low ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

> Since the signature is not included in the quote's digest, processes should use the quote's digest as a unique identifier

File(s) affected: `SignatureSafeguard.sol`

Related Issue(s): SWC-117

Description: EIP-2 allows signature malleability for `ecrecover()`. `EOASignaturesValidator._isValidSignature()` accepts malleable signatures, allowing a signature different from the one an `owner` signed to be successfully submitted. This would make the permitting transaction hash different from the one the permit signer is expecting, and may interfere with processes that respond to on-chain data. This function is inherited and used by `SignatureSafeguard`.

Recommendation: Consider modifying the signature check in `EOASignaturesValidator._isValidSignature()` to require unique signatures by requiring signature s-values to be in the lower half of the potential range. The Ethereum Yellow paper (EY) defines s-value's full range as follows:

```
* 0 < s < secp256k1n ÷ 2 + 1 (EY 311)
```

```
 * With secp256k1n =
 115792089237316195423570985008687907852837564279074904382605163141518161494337 (EY 313)
```

OpenZeppelin's ECDSA contract which checks signature uniqueness can be used as a reference.

SWP-4 Avoid Potential Integer Overflow Underflow

● Low ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `fe17a9b68fa8fea582aa60f1d7613e20bce4942e`. The client provided the following explanation:

> Fixed potential overflows

File(s) affected: `SafeguardMath.sol`

Description: The following calculations could benefit from using a SafeMath library to avoid any potential underflow/overflow:

```
function calcAccumulatedManagementFees(..) {
    uint256 expInput = yearlyRate * elapsedTime;
    ...
}
```

```
function calcYearlyRate(..)  {
    uint256 logInput = FixedPoint.ONE - yearlyFees;
    ...
}
```

Recommendation: Use the `SafeMath` library to perform these calculations.

SWP-5 Missing Chainlink Pricefeed Decimal Check

● Low ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

> We chose to keep it as it is because it adds significant gas cost for an event that is unlikely to happen

File(s) affected: `ChainlinkUtils.sol`

Description: Currently, the scaling factors for the two assets of the oracle prices are calculated once based on the return value of `oracle.decimals()` in the constructor of the `SafeguardPool`, however, technically, the decimals could be subject to change. Therefore, it is a good best practice to check that the decimals meet the expectations when fetching prices from a Chainlink oracle.

Recommendation: Make `oracleDecimals` a global variable. In `getLatestPrice()`, check `oracle.decimals() == oracleDecimals`.

SWP-6 Unlocked Pragma

● Informational ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `2b24c53ca92881da0041d72e616d8c3d5903bdd4`. The client provided the following explanation:

> Fixed solidity version to 0.7.6 to the contracts related to SafeguardPool.sol

File(s) affected: `ChainlinkUtils.sol`, `SafeguardFactory.sol`, `SafeguardMath.sol`, `SafeguardPool.sol`, `SignatureSafeguard.sol`

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.7.*`. The caret (`^`) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, we recommend to remove the caret to lock the file onto the latest version of the 0.7.x version, namely 0.7.6.

SWP-7 Users Can Front-Run Swaps

● Informational ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

> The signed quote contains the expected sender and recipient's address which could prevent users from using and front running other traders quote data from the mempool

File(s) affected: `SafeguardPool.sol`

Description: When validating a swap, the `_balancesSafeguard()` function is called in order to ensure that the performance of a pool does not change drastically within `perfUpdateInterval` seconds of the last time that performance was updated. This means that if a trader finds a quote and submits a transaction to the mempool in a specfic time window such that it takes full advantage of the allowable performance deviation within the performance update interval, that same quote cannot be used again within the block.

However, once the transaction is in the mempool, another user can take the exact same quote data from the pending transaction and submit another swap transaction with a greater amount of gas. This will cause the original pending transaction to revert. However, this can be mitigated through the use of the expected origin penalty.

Recommendation: Consider assisting users in using Flashbots in order to submit transactions with a greater degree of privacy.

SWP-8 Notation of Equations in Documentation Differs From Implementation

● Informational ⓘ Fixed

Update

The client provided the following explanation:

> The documentation has been corrected

File(s) affected: `SafeguardMath.sol`, `SafeguardPool.sol`

Description: The documentation explains how `_joinExactTokensInForBPTOut()` works. This function is called when the amount of tokens in is disproportional to the balances of the pool, in which case, the function swaps some of the input tokens before letting the user join the pool.

According to the doc, we substitute function 2 in 1 to get 3 by solving for sas_asa, where jaj_aja is assumed to be the join amount in excess:

1. ja−saba+sa=jb+sbbb−sb\frac{j_a - s_a}{b_a + s_a} = \frac{j_b + s_b}{b_b - s_b}ba+saja−sa = bb−sbjb+sb
2. sb=p∗sas_b = p * s_asb = p ∗ sa
3. sa=ja∗bb−jb∗babb+jb+p∗(ba+ja)s_a = \frac{j_a * b_b - j_b * b_a}{b_b + j_b + p * (b_a + j_a)}sa = bb+jb+p∗(ba+ja)ja∗bb−jb∗ba

Looking at the code, the equation and the naming are different. The implemented equation in `SafeguardMath.calcJoinSwapAmounts()` appears to solve for a variable sInsInsIn:

4. sIn=aE∗bL−aL∗bEbL+aL+(1/p)∗(bE+aE)sIn = \frac{aE * bL - aL * bE}{bL + aL + (1 /p) * (bE + aE)}sIn = bL+aL+(1/p)∗(bE+aE)aE∗bL−aL∗bE

According the comments above the function, we know that:

- aEaEaE = amountIn in excess ⇒ jaj_aja
- aLaLaL = limiting amountIn ⇒ jbj_bjb
- bEbEbE = balance of excess token ⇒ bab_aba

- bLbLbL = balance of limiting token ⇒ bbb_bbb
- sIn=p∗sOutsIn = p * sOutsIn = p ∗ sOut ⇒ sas_asa = sOutsOutsOut, and sbs_bsb = sInsInsIn

We know that sInsInsIn should correspond to sas_asa; however, the form of equation (4) and equation (3) disagree due to the variable ppp in equation (2) representing its inverse in equation (4).

Recommendation: We recommend adjusting the equations in the documentation to accord with the form of the calculations in the implementation.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – 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.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **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).

Code Documentation

1. Most functions are missing proper NatSpec comments. Consider adding it for `public` and `external` functions and for complex `internal` and `private` ones.

Adherence to Best Practices

1. Fixed The visibility of the `SafeguardPool.getOracleParams()` and `getPoolParameters()` functions should be aligned with the visibility defined in the interface and set to `external`, since both are also not called from within the contract.
2. Unresolved Generic variable names such as `foo` and `bar` as in `SafeguardMath.calcJoinSwapAmount()` should be replaced with more meaningful names.
3. Fixed Typo in the variable name `SwaapV2Errors.PERFORMANCE_UPDATE_TOO_S*0*ON`.
4. Fixed In `SafeguardPool._claimManagementFees()`, save gas by placing the call to `_payProtocolFees()` inside the `if(yearlyRate > 0)` condition's block. Consider additionally placing the event emission inside the same block to save more gas.
5. Fixed Consider renaming `_performanceSafeguard` with a more accurate name, as the HODL benchmark comparison is being handled by `_balancesSafeguard()`.

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

- `839...0d5 ./contracts/SafeguardFactory.sol`
- `978...888 ./contracts/ChainlinkUtils.sol`

- `b29...979 ./contracts/SafeguardPool.sol`
- `1f6...6c1 ./contracts/SafeguardMath.sol`
- `969...c1e ./contracts/SignatureSafeguard.sol`

Tests

- `115...e78 ./test/SafeguardPool.test.ts`
- `f8f...db7 ./test/SafeguardMath.test.ts`
- `0d9...5ac ./test/SignatureSafeguard.test.ts`
- `6c1...736 ./test/SafeguardFactory.test.ts`

Toolset

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

Setup

Tool Setup:

- Slither v0.9.3

Steps taken to run the tools:

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

Automated Analysis

Slither

Slither found instances of dead code that were included into the report.

Test Suite Results

Update: We did not find additional tests after a round of fixes.

Initial audit: The test suite does not include tests for all features. For example, tests of the important yearly fee feature are lacking. We highly recommend testing all features extensively in the test suite.

```
  SafeguardFactory
    constructor arguments
      ✓ sets the vault
      ✓ registers tokens in the vault
      ✓ starts with no BPT
      ✓ sets the pool parameters
      ✓ sets mustAllowlistLPs
      ✓ sets the oracles parameters
      ✓ sets the asset managers to zero
      ✓ sets the owner to delegate owner
      ✓ sets the name
      ✓ sets the symbol
      ✓ sets the decimals
    peg state
      ✓ set peg state
      ✓ stable tokens with flexible oracles
    temporarily pausable
      ✓ pools have the correct window end times
      ✓ multiple pools have the same window end times
      ✓ pools created after the pause window end date have no buffer period

  SafeguardMath
    Management fees
```

```
      ✓ Check helper functions
    calcYearlyRate
      ✓ 0 rate
      ✓ non-0 rate
    calcAccumulatedManagementFees
      ✓ 0 fee
      ✓ 1-year fee
BigNumber { value: "218065375461025610" }
      ✓ n-year rate
    calcTimeSlippagePenalty
      ✓ no penalty: currentTimestamp < startTime
      ✓ no penalty: timeBasedSlippage = 0
      ✓ non 0 penalty
    calcOriginBasedSlippage
      ✓ slippage
      ✓ no slippage
    calcBalanceBasedPenalty
      ✓ slippage on balance of tokenIn
      ✓ slippage on balance of tokenOut
      ✓ slippage on balance per PT change
      ✓ reverts on large balance change
      ✓ reverts on large balance per PT change
    calcJoinSwapAmounts
      ✓ calcJoinSwapAmounts
    calcJoinSwapROpt
      ✓ calcJoinSwapROpt
    calcExitSwapAmounts
      ✓ calcExitSwapAmounts
    calcExitSwapROpt
      ✓ calcExitSwapROpt

  SafeguardPool
    Creation
      when the creation succeeds
        ✓ sets the name
        ✓ sets the symbol
        ✓ sets the decimals
        ✓ sets the owner
        ✓ sets the vault correctly
        ✓ uses two token pool specialization
        ✓ registers tokens in the vault
        ✓ starts with 0 SPT
        ✓ sets pool parameters correctly
        ✓ sets mustAllowList correctly
        ✓ sets management fees correctly
        ✓ sets the oracles parameters correctly
    Initialize
      when not initialized
        when not paused
          ✓ transfers the initial balances to the vault
          ✓ mints 100 BPT
          ✓ mints the minimum BPT to the address zero
          ✓ reverts for low initial amountsIn
          ✓ reverts with wrong initial balances length
        when paused
          ✓ reverts
        in recovery mode
          ✓ does not revert
      when it was already initialized
        ✓ initial balances are correct
        ✓ initial hodl balances are correct
        ✓ reverts
    Post-init
      Join/Exit
        generic
          ✓ fails if caller is not the vault
          ✓ fails if no user data
          ✓ fails if wrong user data
        joinAllGivenOut
          when paused
            ✓ reverts
          when in recovery mode
            ✓ valid
          when in normal mode
            ✓ valid
          when in allowlist mode
            ✓ valid
        joinGivenIn
          when paused
            ✓ reverts
          when in recovery mode
            ✓ valid
          when in normal mode
            ✓ reverts: wrong excess token
            ✓ valid
        exitGivenOut
          when paused
            ✓ reverts
          when in recovery mode
            ✓ valid
          when in normal mode
            ✓ reverts: wrong excess token
            ✓ valid
        multiExitGivenIn
          when paused
            ✓ reverts
          when in recovery mode
            ✓ valid
          when in normal mode
            ✓ valid
      Swap
        swapGivenIn
          ✓ onSwap fails on a regular swap if caller is not the vault
          when paused
            ✓ reverts
          when in recovery mode
            ✓ valid
          when in normal mode
            ✓ valid
        swapGivenOut
          ✓ onSwap fails on a regular swap if caller is not the vault
          when paused
            ✓ reverts
          when in recovery mode
            ✓ valid
          when in normal mode
            ✓ valid
        Detailed Swap
          ✓ Swap given in
          ✓ Swap given out
      Protocol Fees
        ✓ Management Fees: 47304000s
        ✓ Management Fees: 94608000s
        ✓ Management Fees: 141912000s
        ✓ Management Fees: 189216000s
        ✓ Management Fees: 236520000s
        ✓ Management Fees: 283824000s
        ✓ Management Fees: 331128000s
        ✓ Management Fees: 378432000s
        ✓ Management Fees: 425736000s
        ✓ Management Fees: 473040000s
      Setters / Getters
        Unauthorized caller
          ✓ Fail to call setFlexibleOracleStates
          ✓ Fail to call setMustAllowlistLPs
          ✓ Fail to call setSigner
          ✓ Fail to call setPerfUpdateInterval
          ✓ Fail to call setMaxPerfDev
          ✓ Fail to call setMaxPriceDev
          ✓ Fail to call setMaxTargetDev
          ✓ Fail to call setManagementFees
        when paused
          ✓ Fail to call setFlexibleOracleStates
          ✓ Fail to call setMustAllowlistLPs
          ✓ Fail to call setSigner
          ✓ Fail to call setPerfUpdateInterval
          ✓ Fail to call setMaxPerfDev
          ✓ Fail to call setMaxPriceDev
          ✓ Fail to call setMaxTargetDev
          ✓ Fail to call setManagementFees
          ✓ Fail to call claimManagementFees
          ✓ Fail to call updatePerformance
          ✓ Fail to call evaluateStablesPegStates
        Correctly set values wihtout affecting others params
          ✓ valid setFlexibleOracleStates
          ✓ valid setMustAllowlistLPs
          ✓ valid setSigner
          ✓ valid setPerfUpdateInterval
          ✓ valid setMaxPerfDev
          ✓ valid setMaxTargetDev
          ✓ valid setMaxPriceDev
          ✓ valid setManagementFees
          ✓ valid claimManagementFees
          update performance
            ✓ cannot updatePerformance when last update is close
            ✓ update performance after 10% of balances
            ✓ update performance after 10% of TVL
        Oracle
          ✓ Revert if oracle price is zero
          setFlexibleOracleStates
            ✓ isFlexibleOracle0=true, isFlexibleOracle1=true
            ✓ isFlexibleOracle0=true, isFlexibleOracle1=false
            ✓ isFlexibleOracle0=false, isFlexibleOracle1=true
            ✓ isFlexibleOracle0=false, isFlexibleOracle1=false
          isPegged
            ✓ checking isPegged values
            ✓ checking isPegged values
            ✓ checking isPegged values
            ✓ checking isPegged values
            ✓ checking isPegged values
            ✓ checking isPegged values
            ✓ checking isPegged values
            ✓ checking isPegged values
            ✓ checking isPegged values

  SafeguardPool
    Post-init
      Swap Safeguards
        validateSwap
          ✓ valid
          ✓ exceeded swap amount in
          ✓ exceeded swap amount out
          ✓ unfair price: index 0
          ✓ unfair price: index 1
          ✓ min balance out is not met: index 0
          ✓ min balance out is not met: index 1
          ✓ low performance
        Signature Safeguards
          ✓ fails on empty signatureSGUserData
          ✓ fails on wrong signatureSGKind
          ✓ fails on wrong inIs0
          ✓ fails on wrong signatureSGSender
          ✓ fails on wrong recipient
          ✓ fails on wrong quoteIndex
          ✓ fails on wrong deadline
          ✓ fails on replay
          ✓ fails on wrong signer
          ✓ fails on expected origin
          ✓ valid
          ✓ correctly sets quoteIndex
        isLPAllowed
          ✓ valid
          ✓ fails on too large deadline
          ✓ fails on expired deadline
          ✓ fails on wrong signer
          ✓ fails on wrong sender
```

Code Coverage

Update: We did not find additional tests after a round of fixes.

Initial audit: Branch coverage for `SafeguardPool.sol` is at 84.21%. We recommend increasing this to 100%, especially given the sensitive nature of the contract.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	96.71	85.25	95.7	95.98	
ChainlinkUtils.sol	100	100	100	100	
SafeguardFactory.sol	100	100	100	100	
SafeguardMath.sol	100	100	100	100	
SafeguardPool.sol	95.39	84.21	94.29	94.7	... 0,1201,1203
SignatureSafeguard.sol	100	100	100	100	
All files	96.71	85.25	95.7	95.98	

Changelog

- 2023-06-23 - Initial report
- 2023-07-04 - Final report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers 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 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

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, 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

Quantstamp