PUBLIC

Code Assessment

of the SafeguardPool Smart Contracts

June 27, 2023

Produced for

Swaap

by

CHAINSECURITY

Contents

1 Executive Summary

Dear Swaap Finance team,

Thank you for trusting us to help Swaap Labs with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of SafeguardPool according to Scope to support you in forming an opinion on their security risks.

Swaap Labs implements a Safeguard Pool, utilizing the Balancer V2 infrastructure. It is an AMM pool with restrictions on certain swap transactions, in accordance with predefined parameters, known as safeguards. To perform a swap, a valid quote from a privileged signer must be provided. This quote encapsulates the swap price and associated penalties.

The most critical subjects covered in our audit are asset solvency, functional correctness, and precision of arithmetic operations. Security regarding all the aforementioned subjects is good.

The general subjects covered are integration with external systems, signature handling and sanity checks. Security regarding signature handling and sanity checks is high. The pool is integrated with the Balancer V2 infrastructure, which is an out-of-scope system. The issue Reentrancy via Vault was fixed, however other not yet discovered issues may remain since the Balancer V2 infrastructure is not covered by this audit. Thus, security regarding external systems integration is improvable.

In summary, we find that the codebase provides a good level of security regarding the most critical subjects, assuming that the Balancer V2 infrastructure does not contain any severe issues.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	1
• Code Corrected	1
Medium-Severity Findings	3
• Code Corrected	3
Low-Severity Findings	3
• Code Corrected	2
• Risk Accepted	1

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the `./pkg/safeguard-pool/contracts` folder of the SafeguardPool repository based on the documentation files.

The scope consists of the five solidity smart contracts:

1. ChainlinkUtils.sol
2. SafeguardFactory.sol
3. SafeguardMath.sol
4. SafeguardPool.sol
5. SignatureSafeguard.sol

The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	08 May 2023	6537ca745fba4aaf8b971b89e1f7043ce7b1b0a3	Initial Version
2	07 June 2023	1fb3afd6ee78a1c0f7686fb18b2bcb59b8d2255b	Version with fixes
3	12 June 2023	227eb2fc6fbcf4477bb4329f8bdb7b83f619fdf0	Fix rounding direction
4	21 June 2023	b6e118f19dcc179a11f46fdf51d749ae5c206ca4	Fix Vault reentrancy

For the solidity smart contracts, the compiler version `0.7.1` was chosen.

2.1.1 Excluded from scope

Any other files not explicitly mentioned in the scope section. In particular tests, scripts, external dependencies, and configuration files are not part of the audit scope.

2.2 Assumptions

This assessment was performed under certain assumptions:

- The system will be used with ERC20 tokens that do not break any internal accounting.
- It is expected, that swaps with valid quotes can revert due to the changes of the smart contract state caused by other swaps, joins or exits.
- Assessed system will be deployed properly and initialized with the correct parameters.

2.3 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Swaap Labs offers a SafeguardPool - an AMM pool that restricts certain swaps, based on a set of criteria that are called safeguards. The system is based on Balancer V2 architecture. The default Vault Balancer V2 holds and manages accounting for all tokens in each pool. And the SafeguardPool contract provides logic to perform the operations (swaps / joins / exits) on those assets.

SafeguardPool features a quote-directed safeguards-protected swap, a quote must be submitted as part of the `userData` which: (1) is signed by a privileged signer off-chain. (2) encodes the swap price and the penalty parameters. (3) is bound to each user. As a result, the quote sets the price and slippage of a future swap. In addition, a swap with a certain quote will only succeed if all the safeguards are passed. An imbalanced join (providing liquidity) / exit (withdrawing liquidity) works as a composition of swap and balanced join/exit. Swap in such joins/exits is restricted by the same criteria as the normal swaps.

Some safeguards are working by comparing the latest on-chain state of the pool with the historical state of the pool. In detail, 2 values are recorded periodically (every 0.5-1.5 days, depending on configuration) to capture the performance of the pool liquidity tokens (PT): `hodlBalancePerPT0` and `hodlBalancePerPT1`. This HODL PT token perfomances are utilized by safeguards to prevent certain swaps.

Specifically, the following swap safeguards are present:

- Swap Signature Safeguard: checks that the privileged `signer` role holder signed the quote parameters, the signature is not expired and the `quoteIndex` is not used.
- Fair Pricing Safeguard: checks that the quote price with slippage is within the max deviation compared to the chainlink price feed.
- Performance Safeguard: checks if the performance of one PT is within the max allowed deviations, compared to the HODL data.
- Target Balances Safeguard: checks if the total number of output tokens per PT is within the max deviation from the HODL data.

2.3.1 *SafeguardPool*

SafeguardPool implements `onSwap()`, `_onInitializePool()`, `_onJoinPool()`, `_onExitPool()` to be called during swap, join and exit.

2.3.1.1 *Swap*

For each swap, users must submit the following abi encoded data as their `userData`:

- `signature` - 65 bytes `r,s,v` ECDSA signature from `signer` role holder.
- `quoteIndex` - used as an unordered nonce for the signature.
- `deadline` - signature expiration time
- `swapData` - abi encoded pricing params.

The `signature` should be an ECDSA signature from `signer` of following EIP712 hashed struct:

- kind - GIVEN_IN or GIVEN_OUT
- isTokenInToken0
- sender
- recipient

- keccak256(swapData)
- quoteIndex
- deadline - signature-specific deadline. Swap-specific deadline is handled by Vault.swap function.

Thus, each signature can only be used by fixed sender and only be received by fixed recipient. Each quote is one use only and valid only until a certain deadline. The quote dictates swap price and maximum swap amount.

The swapData is an encoded set of pricing parameters:

- address expectedOrigin
- uint256 originBasedSlippage
- bytes32 priceBasedParams: consists of quoteAmountInPerOut, and maxSwapAmount.
- bytes32 quoteBalances: this is packed quoteBalanceIn and quoteBalanceOut
- bytes32 balanceBasedParams: packed balanceChangeTolerance and balanceBasedSlippage
- bytes32 timeBasedParams: packed startTime and timeBasedSlippage

In the onSwap function, the exact out token amount per in token amount is computed based on this data and the balanceTokenIn and balanceTokenOut provided by the Vault contract. The quoteAmountInPerOut is an initial swap price for any given swap. The final price of swap is computed as quoteAmountInPerOut * penalty. The default penalty is 100%, however, different special penalties can increase it:

- Balance based penalty: if on-chain balanceTokenIn or balanceTokenOut is smaller than the ones in quoteBalances, balanceBasedSlippage * max deviation penalty increase is applied. If max deviation exceeds balanceChangeTolerance, swap is reverted.
- Time based penalty: timeBasedSlippage * (block.timestamp - startTime) is the penalty increase.
- Origin based penalty: originBasedSlippage penalty increase is applied if expectedOrigin != tx.origin.

If specified amountIn (or amountOut) exceeds maxSwapAmount the swap reverts.

2.3.1.2 Init, Join and Exit

_onInitializePool() and _onJoinPool() can be permissionless if allow list is not enabled, otherwise, users need to provide a valid signature from the privileged signer which signs the sender and a deadline. In addition, the remaining time before deadline should be larger than _MAX_REMAINING_SIGNATURE_VALIDITY (5 minutes). Upon initialization, a user deposits any amount of tokens and a constant _INITIAL_BPT shares will be minted. There are two types of joins:

- _joinAllTokensInForExactBPTOut(): will deposit the tokens in a balanced way w.r.t. pool's balances for the exact shares wanted.
- _joinExactTokensInForBPTOut(): is equivalent to a swap followed by a balanced join, where the swap must meet the aforementioned restrictions, including Swap Signature Safeguard.

Similarly, there are two types of exits:

- _exitExactBPTInForTokensOut(): will withdraw the tokens in a balanced way w.r.t. pool's balances for the exact shares redeemed.
- _exitBPTInForExactTokensOut(): is equivalent to a swap followed by a balanced exit, where the swap must meet the aforementioned restrictions, including Swap Signature Safeguard.

2.3.1.3 Parameter setters

The following permissioned parameter setters are restricted to the authenticated address:

- setManagementFees: claims the previous management fees and set the new yearly rate.
- setFlexibleOracleStates: update the flexible oracle states.
- setMustAllowlistLPs: join is permissionless if the flag is unset, otherwise join requires a privileged role's signature.
- setSigner: updates the privileged signer who will sign the quotes and allowed join actions.
- setPerfUpdateInterval: sets the performance update interval within 0.5 to 1.5 days.
- setMaxPerfDev: sets the max performance deviation within 5%.
- setMaxTargetDev: sets the max target deviation within 15%.
- setMaxPriceDev: sets the max swap price deviation w.r.t. the current oracle price within 3%.

SafeguardPool also inherits the BasePool permissioned functionalities which are also restricted to the authenticated address:

- `pause()`: pauses the pool.
- `unpause()`: unpauses the pool.
- `enableRecoveryMode()`: enables the recovery mode where only the balanced exit is executed without scaling and complex math.
- `disableRecoveryMode()`: disables the recovery mode.

2.3.1.4 Permissionless management functions

The following management functions are permissionless and can be called by everyone.

- `evaluateStablesPegStates`: evaluates the flexible stablecoin oracle price deviations and peg or unpeg them from constant ONE accordingly. If the deviation is smaller than `0.2`%, anyone can come to peg the price to constant ONE. And if the deviation is larger than `0.5`%, anyone can come to unpeg the price from constant ONE. Therefore the flexible stablecoin oracle only works when there is a depegging.
- `claimManagementFees`: claims the management fees (computed based on the time elapsed) by minting new shares of pool to the fee collector.
- `updatePerformance`: updates the performance of one unit of pool share when an update interval has elapsed.

2.3.2 SafeguardPoolFactory

This contract extends the default Balancer V2 `BasePoolFactory`.

The `create` function of this contract deploys a new SafeguardPool. The access control management of the newly deployed `SafeguardPool` is delegated to the Vault`s Authorizer. Note that the deployment of new pool is permissionless, only the legitimate pools should be used.

2.3.3 Trust Model

This trust model only covers the contracts in Scope. The access control of the Balancer V2 contracts is out of scope.

In SafeguardPool, the signer is a privileged role that signs quotes for all the swaps and joins if allow list is enabled. It is assumed to be fully trusted, always honest and never to behave against the interest of the users. As such, it is assumed:

- Signer is always able to produce the quotes (Liveness)

- Signer won't produce quotes that do not benefit the pool.
- Signer won't censor the swappers and the liquidity providers (if allow list is enabled).
- It can give different quotes depending on different users. Some may get good prices and penalties while others may not.

Each `SafeguardPool` can be initialized with `AssetsManagers`. These default Balancer V2 asset managers, which can manipulate pools funds, are assumed to be fully trusted. By default, `SafeguardPoolFactory` does not allow asset managers.

Any pool deployed is fully trusted. All the parameters that pool deployed specifies (e.g. oracles) are assumed to be well-thought and correct.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Security: Related to vulnerabilities that could be exploited by malicious actors
- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	1

- Performance Updates Can Be Sandwiched Risk Accepted

5.1 Performance Updates Can Be Sandwiched

Correctness Low Version 1 Risk Accepted

CS-SLSGP-008

The performance safeguard validates that the performance based on a unit of pool token does not deviate too much from the old performance after a swap. Updating the performance is permissionless when a `perfUpdateInterval` (within 0.5 to 1.5 days) has elapsed. If the allowed performance deviation is x%, one can bundle a performance update within two swaps to achieve around 2x% deviation, which breaks the assumption that performance can at most change x% within one `perfUpdateInterval`.

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0

High-Severity Findings	1

- Reentrancy via Vault Code Corrected

Medium-Severity Findings	3

- Incorrect Rounding Directions Code Corrected
- Incorrect Target Deviation Computation Code Corrected
- Missing Sanity Checks at Pool Initialization Code Corrected

Low-Severity Findings	2

- Balance Based Penalty Can Be Manipulated Code Corrected
- Price Feed Data Validity Checks Code Corrected

6.1 Reentrancy via Vault

Design High Version 1 Code Corrected

CS-SLSGP-012

The Balancer V2 Vault has a known vulnerability to read-only-reentrancy:

> https://forum.balancer.fi/t/reentrancy-vulnerability-scope-expanded/4345.

The balances during onJoin/onExit are updated after new shares are minted/burned. And before balance update, the Vault performs a call to external address with the remaining ETH.

The following scenario is possible:

- A large LP awaits the time when the `updatePerformance()` can be called.
- LP exits in a balanced way(no updatePerformance triggered yet) and triggers the reentrancy from the Vault.
- In the reentrant call the pool.updatePerformance() is executed. The reentrancy guard on Pool won't be triggered, because it is the Vault that makes the reentrant call.
- The performance snapshots wrong values. PT will be already burned, but `Vault.getPoolTokens()` will return not yet updated balances. Thus the performance will be too high.

This reentrancy is due to the way Vault contract deals with the ETH that is sent along with swap/join/exit call using `_handleRemawiningEth` function.

As a result, wrong performances will be saved for a given performance update period. This will cause DoS in case of exit (performances are too high), or disable the performance based checks for the whole period.

This applies to `getPoolPerformance` function as well.

Code corrected:

The reentrancy issue has been fixed in the Vault contract, where the update of the balances is now done before the token transfers.

6.2 Incorrect Rounding Directions

Security Medium Version 1 Code Corrected

CS-SLSGP-002

Most computations in SafeguardPool are based on 18 decimals for higher precision. However, rounding errors are not properly handled in some cases, where it may round towards the advantage of users instead of the pool.

In `calcJoinSwapAmounts()`, the `swapAmountIn` is computed using `divDown`. Then the `rOpt` is computed from this value:

```
uint256 swapAmountIn = num.divDown(denom);
uint256 swapAmountOut = swapAmountIn.divDown(quoteAmountInPerOut);

function calcJoinSwapROpt(uint256 excessTokenBalance, uint256 excessTokenAmountIn,
    uint256 swapAmountIn ) internal pure returns (uint256) {
    uint256 num   = excessTokenAmountIn.sub(swapAmountIn);
    uint256 denom = excessTokenBalance.add(swapAmountIn);
    return num.divDown(denom);
}
```

However, the num will be computed as `excessTokenAmountIn.sub(swapAmountIn)`, thus it will be effectively rounded up. This might result in minting more shares than intended.

A similar case exits in `calcExitSwapAmounts()` though it is unclear which `rOpt` is larger.

In addition, in `_exitBPTInForExactTokensOut()` the `bptAmountOut` is rounded down. This lowers the amount of shares the user needs to burn. As a result, the pool tokens can lose value with time due to exit conditions that do not favor remaining pool token holders.

```
uint256 bptAmountOut = totalSupply().mulDown(rOpt);
```

In another note, `_getOnChainAmountInPerOut()` and `calcBalanceDeviation` round down the computations. This may make the `fairPricingSafeguard` and balance based checks slightly weaker. However, in other places, it is unclear if the computation should round up or down (e.g. computation of currentPerformance in `_updatePerformance()`).

Code corrected:

The `calcJoinSwapROpt()` now subs 1 wei from numerator and adds 1 wei to denominator. This effectively lowers the number of tokens minted during the deposit by a small amount, that always guarantees that the balances per PT values won't decrease during balanced join.

The `calcExitSwapROpt()` now adds 1 wei to numerator and subs 1 wei from denominator. This effectively increases the number of tokens burned during the withdrawal by a small amount, that always guarantees that the balances per PT values won't decrease during balanced exit.

The `_exitBPTInForExactTokensOut()` has been fixed to use mulUp instead of mulDown to compute the amount of pool tokens burned upon a withdrawal.

The rounding in `calcBalanceDeviation` can effectively be accounted by the quote generating front-end.

6.3 Incorrect Target Deviation Computation

Correctness **Medium** **Version 1** **Code Corrected**

CS-SLSGP-003

The balance safeguard validates that the HODL balance of the output token after a swap does not deviate too much from it before the swap (target deviation). This is computed in the wrong way in `_getPerfAndTargetDev()`, where the numerator should be `newBalanceOutPerPT` instead of `newBalanceOut`. The target deviation should be in %, however, this wrongly computed value represents the amount of pool tokens.

Code corrected:

The target deviation now is correctly computed as `newBalancePerPTOut.divDown(hodlBalancePerPTOut)`.

6.4 Missing Sanity Checks at Pool Initialization

Security **Medium** **Version 1** **Code Corrected**

CS-SLSGP-004

There is no sanity check on the user's input token amounts `amountsIn` as well as the initial HODL balance at the pool initialization.

In case a user initializes the pool with 0 `amountsIn`, the pool becomes useless irreversibly:

- Anyone can mint any amount of pool tokens by depositing 0 liquidity.
- No swap is possible as there is no liquidity.

A user can also disable swaps by initializing the pool with a small `amountsIn`, where the HODL balance rounds down to 0. Assuming there is a pool of two tokens with 18 decimals, due to the following behavior of the `_onInitializePool`:

- User initializes with `amountsIn = [1 wei, 1 wei]`.
- After scaleUp, `amountsIn = [1 wei, 1 wei]` because the tokens already have 18 decimals.
- the HODL balance is computed as `1 * 10^18 / (100 * 10^18)`, which rounds down to 0.

If both hold balances are 0, the `_updatePerformace` and `_getPerfAndTargetDev` will revert due to the division by 0.

Code corrected:

A check was added in the `_onInitializePool()` function, that requires both `amountsIn[0]` and `amountsIn[1]` to be at least `_MIN_INITIAL_BALANCE = 1e8`. This way, issues due to division by zero will be avoided.

6.5 Balance Based Penalty Can Be Manipulated

Design Low Version 1 Code Corrected

CS-SLSGP-001

In case the current pool balance is less than the pool balance at the quote time, a penalty will be enforced on the quote price during a swap. However, the balance of the pool can be easily manipulated by Join or Exit.

In case there is a balance based penalty, a user can bypass it by Just In Time (JIT) liquidity provision:

- Join the pool to push the balance back to quote time.
- Swap without balance based penalty.
- Exit after the swap.

By having a valid quote and doing join-swap-exit bundle, users can bring the state of the pool balances in a state, where other "pending" quotes are blocked by the balance based penalty. Thus, using join-swap-exit bundle user can:

- Bypass paying the balance based penalty fees
- Avoid the `maxDeviation` check.

However, in join-swap-exit the user will only get fraction of the `maxSwapAmount` total swap value, due to the need to provide out token as an asset during join.

A swap can also be front-run by a liquidity provider's exit, which aggravates the balance based penalty. This way an exit-swap-join, (swap is sandwiched by malicious LP) can:

- Revert the swap
- Enforce the higher balance penalties on the swap.

This can be seen as a DoS attack, however it requires significant gas with no clear benefit for the attacker.

Code corrected:

Swaap Labs responded:

> The new balance based penalty also takes into consideration the balance change per PT as well as the balance change: penalty = max(balanceChange, balanceChangePerPT) * slippage

Since joins and exits do not change the balances per PT, this check will not be bypassable by join-swap-exit bundle. Thus, swaps with quoted balances that differ too much from the onchain conditions will not be executable.

6.6 Price Feed Data Validity Checks

Security Low Version 1 Code Corrected

CS-SLSGP-005

SafeguardPool uses chainlink oracle to retrieve the price feed for tokens. However the checks in `ChainlinkUtils.getLatestPrice` are missing or not strong enough:

- `_ORACLE_TIMEOUT` is a constant of 1.5 days which could be too large. The heartbeat of most datafeeds is much smaller: `https://docs.chain.link/data-feeds/price-feeds/addresses#Ethereum%20Mainnet`. Any round that is older than the Heartbeat cannot be considered fresh. This might happen due to potential ChainLink failures.

- ChainLink `getLatestRound` returns `roundId` and `answeredInRound`. However, they are not inspected. In ChainLink OCR pricefeeds the `roundId` and `answeredInRound` are always equal. However, older versions of pricefeeds require validation, that the round data was not computed in an old round(answeredInRound should not be less than roundId): https://docs.chain.link/data-feeds/historical-data#getrounddata-return-values. Please be aware of this and check for each deployed pool what pricefeed version is used.

Code corrected:

Swaap Labs responded:

> Each oracle in a pool has its own maximum timeout (=< 1.5 days) which is immutable and defined at deployment time. The roundId and answeredInRound are checked .

6.7 Events Indexed Params

Informational Version 1 Code Corrected

CS-SLSGP-013

The `quoteIndex` in `ISignatureSafeguard` is not indexed. It functions as a random-order nonce for quote signatures. Querying on-chain information about which quote is exhausted is easier if this field is indexed.

```
event SwapSignatureValidated(bytes32 digest, uint256 quoteIndex);
event AllowlistJoinSignatureValidated(bytes32 digest);
```

Similarly, `digest` params in both events can be indexed.

Code corrected:

`quoteIndex` as well as `digest` of both events has been marked as indexed in the updated code.

6.8 Outdated Dependency of Balancer Pool Factory

Informational Version 1 Code Corrected

CS-SLSGP-011

One of SafeguardPool's dependency Balancer's `BasePoolFactory` has been updated in March where `create2()` is used instead of `create()`. The full merge request can be found here: https://github.com/balancer/balancer-v2-monorepo/pull/2362

Code corrected:

Balancer dependency is updated. `CREATE2` opcode with an extra `salt` parameter is now used to deploy the pools.

6.9 Performance Safeguard Sensitivity

Informational Version 1 Code Corrected

CS-SLSGP-007

The HODL balances are set on initializing pool, and during the updates they are multiplied by performance. This effectively fixes during the initialization the proportion of assets that are used for performance safeguard. If the price of assets changes significantly over time, the difference between balance0/balance1 and hodlBalance0/hodlBalance1 can cause significant sensitivity to price changes. In addition, this imbalance can be caused intentionally during the initialization.

For example:

1. Pool initialized with 1 Eth and 100k USD as assets. The hodlBalanceETH = 1, hodlBalanceUSD = 100k. Assume that BPT is always 1. At this time 1 ETH == 1000 USD. TLV = 101000 USD == holdTVL
2. Over time, with help of swap the balance of pool becomes: 50 ETH and 1000 USD, with 2000 USD as ETH price. TLV = 101000 USD. old hodlTVL = 1100 Since TVL does not change, the holdBalanes will not change as well.
3. Without any balance changes, if price of ETH becomes 1900 USD == 5% drop: TLV = 96k USD. hodlTVL = 101900. newTVL/hodlTVL = 0.942 > 5% drop

Thus, due to the initial proportion of hold balances the hodl performance of the pool was affected more than the asset price. Also, note that the balances of tokens itself did not change between 2 and 3. Just the change of the oracle price can be enough to make swaps fail due to the performance safeguard.

Code corrected:

Swaap Labs have updated the code that the performance safeguard will be bypassed if a swap is rebalancing the current pool towards the hodl balance ratio.

```
if (newBalancePerPTOut < hodlBalancePerPTOut || newBalancePerPTIn > hodlBalancePerPTIn)
{
   _srequire(
         _getPerfFromBalancesPerPT(newBalancePerPTIn,newBalancePerPTOut,
            hodlBalancePerPTIn,hodlBalancePerPTOut,onChainAmountInPerOut
         ) >= _getMaxPerfDev(packedPoolParams),
         SwaapV2Errors.LOW_PERFORMANCE
   );
}
```

Swaap Labs stated:

> The idea is to allow the rebalancing of assets even if we do not have good performance in order not to find the pool stuck with undesired asset ratios.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 Imbalanced Join Order

Informational Version 1 Acknowledged

CS-SLSGP-006

User can call `_joinExactTokensInForBPTOut()` to join the pool in an imbalanced way. There are two approaches to achieve the same imbalanced join:

1. excess tokens are swapped for limited tokens first, then a balanced join is executed.
2. a balanced join is executed first, then do a swap to achieve the same result.

SafeguardPool takes the first approach. However, as the pool balance at swap time is smaller in approach 1 compared to approach 2, it could induce higher balance based penalty and consequently prevent a transition that actually benefits the system.

Acknowledged:

Swaap Labs responded:

> We chose to keep this approach as it is easier to produce a quote for this kind of operation & it’s more gas efficient and easier to check the post trade safeguards. In addition a user can separately swap and then join the pool even if we change the approach.

7.2 Invalidation of Quotes

Informational Version 1

CS-SLSGP-009

The signed quotes remain valid until they are either executed or reach their deadline. No functionality allows a specific quote to be invalidated. However, changing the signer will invalidate all previously signed quotes. In case the signer role holder is changed from Alice to Bob and then back to Alice, all the un-expired quotes Alice signed before will become valid again. These facts must be considered throughout the contract's lifespan.

7.3 Management Fees and Swap Safeguards Relation

Informational Version 1

CS-SLSGP-010

The `_claimManagementFees` is called before any swap or join, but not during the swaps. This can affect the safeguard that rely on per PT values. E.g. if `_claimManagementFees` is called after a long period, the hodl balances per pt will drop, due to newly minted PT shares. Then, the safeguards can fail until next snapshot of the hodl balances. Due to the low rate of management yearly fees (5%), this should not be a problem.

8 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

8.1 Consistency of Input Arguments Scale

Note Version 1

Most of the computations work with values of 18 decimals. Input amounts for tokens that have less than 18 decimals will be first scaled up by a scaling factor to reach 18 decimals. In SafeguardPool, some of the input argument amounts are expected to be already scaled up, while the others (mostly coming from `Vault`) are not.

Examples of such differences:

- In `_onInitializePool()`, `amountsIn` in `userData` needs to be not upscaled.
- In `_joinExactTokensInForBPTOut()`, `joinAmounts` in `userData` needs to be already scaled up.
- In `_exitBPTInForExactTokensOut()`, `exitAmounts` in `userData` needs to be already scaled up.
- In `onSwap()`, `SwapRequest.amount` needs to be not upscaled, however `quote.maxSwapAmount` needs to be upscaled.

Scaling the value off-chain is gas-efficient, but requires the correct input data generation. If directly submitting a transaction to the contract, users should be aware of which parameters should be scaled up and which should not.