PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Swerve
(Liquidity Bootstrapping Pool)

13 March 2023

paladinsec.co   info@paladinsec.co

Table of Contents

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains the right to re-use any and all knowledge and expertise gained during the audit process, including, but not limited to, vulnerabilities, bugs, or new attack vectors. Paladin is therefore allowed and expected to use this knowledge in subsequent audits and to inform any third party, who may or may not be our past or current clients, whose projects have similar vulnerabilities. Paladin is furthermore allowed to claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for Swerve's Liquidity Bootstrapping Pool contract on the Avalanche network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	Swerve
URL	TBC
Platform	Avalanche
Language	Solidity
Preliminary Contracts	https://github.com/swervesys/lbp-contracts/blob/main/contracts/SwerveLBP.sol

1.2 Contracts Assessed

Name	Contract	Live Code Match
`SwerveLBP`		

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
🔴 High	2	2	-	-
🟠 Medium	3	2	-	1
🟡 Low	6	6	-	-
🟣 Informational	3	2	-	1
Total	**14**	**12**	**-**	**2**

Classification of Issues

Severity	Description
🔴 **High**	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
🟠 **Medium**	Bugs or issues with that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
🟡 **Low**	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
🟣 **Informational**	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 SwerveLBP

ID	Severity	Summary	Status
01	HIGH	Adding liquidity to TraderJoe will result in stuck funds	✓ RESOLVED
02	HIGH	Fees can be circumvented by partial withdrawals	✓ RESOLVED
03	MEDIUM	`_distributeTokens` is not explicitly called with the received amount	ACKNOWLEDGED
04	MEDIUM	Both exit functions are vulnerable to frontrunning	✓ RESOLVED
05	MEDIUM	Users can circumvent the fee on `fundToken`	✓ RESOLVED
06	LOW	`_recipientAddresses` distribution is flawed	✓ RESOLVED
07	LOW	Leftover tokens can get stuck during createLBP	✓ RESOLVED
08	LOW	LBP creators are limited in their privileges	✓ RESOLVED
09	LOW	Lack of validation	✓ RESOLVED
10	LOW	`getPools` will eventually malfunction due to gas issues	✓ RESOLVED
11	LOW	`_feeRecipientsBPS` and `_recipientAddresses` are private	✓ RESOLVED
12	INFO	The contract does not work with tokens that have a fee on transfer	✓ RESOLVED
13	INFO	Typographical issues	✓ RESOLVED
14	INFO	Gas optimizations	ACKNOWLEDGED

2 Findings

2.1 SwerveLBP

`SwerveLBP` is a utility extension for Balancer's LBP feature. A liquidity bootstrapping pool on Balancer is a smart contract that allows users to pool their funds together to trade tokens on Balancer. It allows liquidity providers to earn fees on trades and helps projects bootstrap liquidity for their tokens.

Users can simply create their own LBP via the function `createPool` which allows them to pass arguments such as:

- `startWeights`
- `endWeights`
- `startTime`
- `endTime`
- `swapFeePercentage`

After a pool has been created, the caller becomes the authorized address to interact with the pool via `SwerveLBP`, while `SwerveLBP` becomes the owner of the LBP itself. This results in some configurational limitations, such as the inability to reconfigure the weights.

After the LBP has been created via `LBPFactory`, the authorized address can then transfer its authorization privilege and can enable swaps for the corresponding pools. Moreover, `SwerveLBP` will get the corresponding minted `LBPTokens` which represents the provided liquidity.

Whenever the authorized address decides to withdraw the liquidity, there are two options:

1. `exitPool`, which allows the authorized address to withdraw liquidity to itself.

2. `exitToTraderJoe`, which allows the authorized address to withdraw liquidity to itself and / or create a liquidity pair on TraderJoe.

Both methods can be used to either withdraw the whole share or just a portion of it. During both `exit` functions, the contract applies a fee to the `fundToken` if the withdrawal amount is larger than the initial amount. This fee is then transferred to the owner of `SwerveLBP`.

2.1.1 Privileged Functions

- `setSwapEnabled (onlyPoolOwner)`
- `transferPoolOwnership (onlyPoolOwner)`
- `exitPool (onlyPoolOwner)`
- `exitToTraderJoe (onlyPoolOwner)`
- `transferOwnership`
- `renounceOwnership`

2.1.2 Issues & Recommendations

Issue #01	Adding liquidity to TraderJoe will result in stuck funds
Severity	● HIGH SEVERITY
Description	Within `exitToTraderJoe`, the caller can choose to add liquidity to TraderJoe. There will often already be liquidity due to other users adding it from the liquidity bootstrapping pool. Since `addLiquidity` will almost always be called with amounts that are not in a 50/50 ratio, this could result in a leftover amount which will be stuck in the `SwerveLBP` contract. This issue is amplified due to the high volatility in LBP's which means that ratios can be down to 99/1.
Recommendation	There are several ways to deal with this issue: - Refund excess to the pool owner - Validate that the LBP ended in 50/50 weights and that the refunds are negligible - Swap the remainder of the tokens for an optimal liquidity add All solutions have their advantages and disadvantages and we are happy to enter into a discussion about this.
Resolution	✓ RESOLVED The contract now executes the before-after check as desired and transfers the leftover balance back to the caller.

Issue #02	Fees can be circumvented by partial withdrawals
Severity	● HIGH SEVERITY

Description

Both withdrawal functions only take a fee if the withdrawal amount is larger than `fundTokenInputAmount`.

`exitPool` executes this check within the `_distributeTokens` function:

```
if (fundTokenFromPool > poolData.fundTokenInputAmount) {

    uint256 totalPlatformAccessFeeAmount =
((fundTokenFromPool - poolData.fundTokenInputAmount) *
platformAccessFeeBPS) / _TEN_THOUSAND_BPS; // Fund amount
after substracting the fee remainingFundBalance =
fundTokenFromPool - totalPlatformAccessFeeAmount;

    if (isStandardFee) {
        _distributePlatformAccessFee(pool, fundToken,
totalPlatformAccessFeeAmount);
    }

    else { _distributeSafeFee(pool, fundToken,
totalPlatformAccessFeeAmount);
    }
}
```

`exitToTraderJoe` executes this check within the `splitTokens` function:

```
if (amountFund > fundTokenInputAmount) {

    totalPlatformAccessFeeAmount = ((amountFund -
fundTokenInputAmount) * platformAccessFeeBPS) /
_TEN_THOUSAND_BPS;
    remainingFund = amountFund -
totalPlatformAccessFeeAmount;

}
```

	However, the issue within these checks is that `fundTokenInputAmount` is never decreased, which means that users can circumvent these checks by simply withdrawing partial shares lower than `fundTokenInputAmount`.
Recommendation	Consider decreasing `fundTokenInputAmount` by the withdrawal amount.
Resolution	✓ RESOLVED The variable was now decreased in storage and logic was implemented which also decreases the variable in `exitToTraderJoe`.

Issue #03	**`_distributeTokens` is not explicitly called with the received amount**
Severity	● MEDIUM SEVERITY
Description	`exitPool` calls `_distributeTokens` with the following parameter: `balances[fundTokenIndex] - balancesAfterExit[fundTokenIndex]`. Since the underlying Balancer protocol uses a lot of arithmetic operations within the exit path, this calculation might eventually differ from the actual received amount. However, this might not be ideal since this approach is vulnerable to rounding issues. If the contract receives less tokens than expected due to rounding, this will result in a revert of `_distributeTokens`. We do not really expect this to happen in practice but it might make sense to test the contract balances instead using a before-after pattern.
Recommendation	Consider testing the contract balances instead using a before-after pattern.
Resolution	● ACKNOWLEDGED The client indicated that they do not expect any potential rounding errors.

Issue #04	Both `exit` functions are vulnerable to frontrunning
Severity	● MEDIUM SEVERITY
Description	Both `exit` functions provide zero as `minAmountsOut`. If the owner expects certain output amounts, this could drastically change with a big sale of any market participants, resulting in a loss for the owner.
Recommendation	Consider providing the minimum desired output as input parameter.
Resolution	✓ RESOLVED Corresponding parameters were added to the functions.

Issue #05	Users can circumvent the fee on `fundToken`
Severity	● MEDIUM SEVERITY
Description	The contract allows the provision of `fundToken` and `mainToken` and a parameter called `isCorrectOrder`. Users can simply provide the `mainToken` as `fundToken` and the `fundToken` as `mainToken` with `isCorrectOrder = true`, which will result in a fee deducted from `mainToken` instead of `fundToken`.
Recommendation	This appears to be an authorization issue as anyone can create a pool. We are happy to resolve it in whichever way the Swerve team sees fit.
Resolution	✓ RESOLVED The developer team has added an `enumerableSet` with `allowedFundTokens` where the owner can add `fundTokens`. The `createLBP` function then checks that the `fundToken` is indeed allowed.

Issue #06	`_recipientAddresses` **distribution is flawed**
Severity	LOW SEVERITY
Description	`_distributePlatformAccessFee` distributes the fee on all addresses within the `_recipientAdresses` set with their assigned share (`_feeRecipientsBPS`). However, there is no way to add additional addresses to the array nor any shares to `_feeRecipientsBPS`. Moreover, even if such functionality is implemented, the function would not work because the first address, the `owner()`, already has `_TEN_THOUSAND_BPS` assigned which results in a revert if there are any additional addresses with shares added because the contract would not have enough tokens to distribute. Additionally, when ownership ever gets transferred, this recipient would become permanently outdated to the old owner.
Recommendation	Consider implementing functions for adding/setting recipients and their assigned shares. Also consider implement a safeguard that the sum of these shares is equal to `_TEN_THOUSAND_BPS`. Idiomatically, the rounding remainder should also be distributed but that might not be worth it if the tokens have large numbers of precision.
Resolution	✓ RESOLVED Logic was added to support multiple fee recipients.

Issue #07	Leftover tokens can get stuck during `createLBP`
Severity	● LOW SEVERITY
Description	`createLBP` transfers the desired amount from `msg.sender` to the contract. Afterwards, it approves the amounts to the vault and executes `vault.joinPool` which then will transfer these tokens in. However, within `joinPool`, the vault and the LBP itself execute a bunch of logic including arithmetic operations where the desired amount is first rounded down and then rounded up. This results in the vault withdrawing less tokens than desired, which will leave a leftover amount in the `SwerveLBP` contract that can never be withdrawn.
Recommendation	Consider caching the balance before the transfer from `msg.sender` to the contract and then simply transferring back any leftover amount which exceeds the balance before to the `msg.sender`. We encourage providing test coverage on this issue as well.
Resolution	✓ RESOLVED The balances are now cached correctly and the leftover amount is transferred back.

Issue #08	LBP creators are limited in their privileges
Severity	● LOW SEVERITY
Description	Contrary to a direct LBP creation via the `LBPFactory`, a pool creator does not have the full control over the created LBP. For example important functions such as `updateWeightsGradually`, `setSwapFeePercentage`, `pause` and `unpause` can never be called.
Recommendation	Consider if this is intentional, otherwise it might make sense to add these important functions to the `SwerveLBP` contract.
Resolution	✓ RESOLVED These functions have been added.

Issue #09	Lack of validation
Severity	● LOW SEVERITY

Description	The contract lacks important validations which might break functionality. In order to keep the report size reasonable we have conslidated these issues.
	
	Line 119
	`platformAccessFeeBPS = _platformAccessFeeBPS;`
	
	`platformAccessFeeBPS` should never exceed 10_000.
	
	Line 264
	`function transferPoolOwnership`
	
	This function lacks a non-zero check.
	
	Many of the lengths and variables within `poolConfig` are not validated, e.g. the weights. This might be fine as the weights are checked in the LBP settings code.

Recommendation	Consider carefully validating these parameters.
Resolution	✓ RESOLVED

Issue #10	`getPools` will eventually malfunction due to gas issues
Severity	● LOW SEVERITY
Description	`getPools` is a function that returns all pool addresses. However, if there are many pools, this function will always run out of gas due to its gas cost increasing linearly with the number of pools.
Recommendation	There is already an `index` function so we can resolve this issue on the note that this eventual malfunctioning is acceptable. However, since anyone can create pools, someone can DoS this function, so this should be taken into consideration. It might be better to either add pagination (eg. a `from` and `to` index) or just straight up delete the function.
Resolution	✓ RESOLVED The function has been removed.

Issue #11	`_feeRecipientsBPS` and `_recipientAddresses` are `private`
Severity	● LOW SEVERITY
Description	Important variables that third-parties might want to inspect should be marked as `public` so that these variables be easily inspected through the explorer, web3 and derivative contracts.
Recommendation	Consider adding getter functions for the recipient addresses and marking the other variable as public.
Resolution	✓ RESOLVED

Issue #12	The contract does not work with tokens that have a fee on transfer
Severity	INFORMATIONAL
Description	The contract contains multiple sections which will not work with tokens that have a fee on transfer. For example, `createLBP` will create an LBP with the same amount that has been initiated for the transfer before. If a token has a fee on transfer, the contract will not have enough tokens to fulfill `joinPool`. `_distributeTokens` is called with the parameter `balances[fundTokenIndex] - balancesAfterExit[fundTokenIndex]`, which also does not work with tokens that have a fee on transfer. As Balancer does not support such tokens, this issue has been marked as informational: https://docs.balancer.fi/security/token-compatibility
Recommendation	No action needs to be taken. This issue can be resolved on the note that Swerve does not plan to ever support such tokens.
Resolution	✓ RESOLVED

Issue #13	Typographical issues
Severity	● INFORMATIONAL

Description	<u>Line 10</u>
	`import { IJoeFactory } from './interfaces/IJoeFactory.sol';`
	
	This import is unused, and the state variable and the setter within the constructor can be removed.
	
	<u>Line 58</u>
	`uint256 private constant MAX_INT = 2**256 - 1;`
	
	This variable is unused. Consider removing it.
	
	<u>Line 223</u>
	`require(_pools.add(pool), 'exists already');`
	
	This appears to be an assertion thus the assert keyword can be used.
	
	<u>Line 429</u>
	`uint256 _deadline = deadline;`
	
	Since this variable is only used once, caching it does not save gas but instead reduces the readability of the code.
	
	We also recommend the future developers to be careful with extending this contract as `_transferTokenToPoolOwner` has been optimized to always transfer the tokens to `msg.sender` — this could be a potential issue if a function callable by the contract owner was ever added for example.

Recommendation	Consider fixing the typographical errors.
Resolution	✓ RESOLVED

Issue #14	Gas optimizations
Severity	● INFORMATIONAL
Description	Line 169 `function createLBP(PoolConfig memory poolConfig) external returns (address) {` `poolConfig` can be provided as `calldata` to save gas. Various other gas optimizations can be made in the `exit` functions by caching variables in `memory` but since they would make the code more verbose we understand that this trade-off was not made.
Recommendation	Consider implementing the gas optimizations mentioned above.
Resolution	● ACKNOWLEDGED

PALADIN
BLOCKCHAIN SECURITY