HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Parallax
Date: March 03, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Parallax
Approved By	Marcin Ugarenko | Lead Solidity SC Auditor at Hacken OU
Type	ERC721 token; Vault; Yield Strategy
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://parallaxfinance.org/
Changelog	11.01.2023 - Initial Review 06.02.2023 - Second Review 03.03.2023 - Third Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Parallax (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is smart contracts in the repository:

Initial review scope

Repository	https://bitbucket.ideasoft.io/projects/PAR/repos/solidity
Commit	ce3b18c0ceea7ae75c3170c3452b08d1adf6936f
Functional Requirements	https://bitbucket.ideasoft.io/projects/PAR/repos/solidity/browse/docs/docs.pdf
Technical Requirements	https://bitbucket.ideasoft.io/projects/PAR/repos/solidity/browse/docs/docs.pdf
Contracts	File: ./contracts/extensions/TokensRescuer.sol SHA3: 246bfb15ae892c4e362f401eed46488e27f3fbdc4f9839752c6d2fafc4f8a42e File: ./contracts/interfaces/IParallax.sol SHA3: 840c1f76fade58ea77d7a17bf9eaef15731c67c1fb75795175f36fb6cb03de4c File: ./contracts/interfaces/IParallaxStrategy.sol SHA3: 4101d0c193b79e7b523d6a8b43db43b6eb65a1144af5fdaae5d7997aabbdec6b File: ./contracts/interfaces/ITokensRescuer.sol SHA3: 3f6271838e43b2ddc72edccc65fb01c674bfcc86ac0638b0e5ce500868f1185b File: ./contracts/Parallax.sol SHA3: 62c12caa414ca3144b3e9e402834d1321841dca56fadcd4adb424ea143071e8f File: ./contracts/strategies/curve-sorbettiere/CurveSorbettiereStrategy.sol SHA3: 5fc73b0a783bcf127635b88ee5507028c71d686924f2765d35ebd6c10bc77dd3 File: ./contracts/strategies/curve-sorbettiere/interfaces/ICurve.sol SHA3: e05f60ddaea8c751400f222419e2bb198b4dc00121e805b576169f25fa914060 File: ./contracts/strategies/curve-sorbettiere/interfaces/ISorbettiere.sol SHA3: ceaa566d10d2c17d2708e3e3ef1ac9e0563ac94a8c535bc638f76e0d7bcc2e99

Second review scope

Repository	https://bitbucket.ideasoft.io/projects/PAR/repos/solidity
Commit	f87b8cae7dbd9970a24276c31885c15f8f7c1bb0

Functional Requirements	https://bitbucket.ideasoft.io/projects/PAR/repos/solidity/docs/Curve APY calculation.pdf https://bitbucket.ideasoft.io/projects/PAR/repos/solidity/docs/Documentation.pdf
Technical Requirements	https://bitbucket.ideasoft.io/projects/PAR/repos/solidity/docs/Curve APY calculation.pdf https://bitbucket.ideasoft.io/projects/PAR/repos/solidity/docs/Documentation.pdf
Contracts	File: ./contracts/ERC721/ERC721UpgradeableParallax.sol SHA3: f8ef74003134a72c7d0d9ae0586fad2e2c0a27e54947940d685853aa7933c93c File: ./contracts/extensions/CheckerZeroAddr.sol SHA3: cba220ed74dad247a08fca1c42bf59e224fb870df8d2563dde9b714203647cda File: ./contracts/extensions/Timelock.sol SHA3: 1ff7e0e42b9bf073f4163c87c33d0c8c15c63b8cdb4a25283079e00843629521 File: ./contracts/extensions/TokensRescuer.sol SHA3: 64c9f6a22a113fa24dc756a1b161905875d25ad95eb662a1809aaeb6a45e51b6 File: ./contracts/interfaces/IERC721UpgradeableParallax.sol SHA3: cf5a1e58d8995ec3365b18b0297b88fb9c4d2675aca60df46eee17cc6ce72e66 File: ./contracts/interfaces/IFees.sol SHA3: d578ca8dd568ce2762143f547e9dde41055bdea0efdef48f076f15324fbea673 File: ./contracts/interfaces/IParallax.sol SHA3: 55c454e22c05101fdae0bf5d448b077136b84b05d7bc7bfc3414b1db5b37c68b File: ./contracts/interfaces/IParallaxStrategy.sol SHA3: f6490988308bf7332743f7dface7b886880aaa2df084fbcdfde8db9629acd8d0 File: ./contracts/interfaces/ITokensRescuer.sol SHA3: a65a0522cb31bd0a3d7d6f84ed990c790a923893936874f9ecb3e3978bd3382c File: ./contracts/Parallax.sol SHA3: 8df0a2562cb4d812fcdd2a0e68dffd72f04dc10b268fb8f8781f8851684ab79a File: ./contracts/strategies/curve-sorbettiere/CurveSorbettiereStrategy.sol SHA3: 182cd00a689e5a96e8e38eeb5b541763527874c86652f33fc64456beb8168e22 File: ./contracts/strategies/curve-sorbettiere/interfaces/ICurve.sol SHA3: d0010f1c18c1c03ce828b6b77d58c4bd199d5fb8dcf4ff5815bf3fd17a4633c2 File: ./contracts/strategies/curve-sorbettiere/interfaces/ISorbettiere.sol SHA3: b573b94f9c91e5888d898102f3a8f073c0160760fb143ef83d7acb2ca476b4fb

Third review scope

Repository	https://bitbucket.ideasoft.io/projects/PAR/repos/solidity
Commit	8bd4562ece9e838956a0295a1ba10f76b8b5da4a
Functional Requirements	https://bitbucket.ideasoft.io/projects/PAR/repos/solidity/docs/Curve APY calculation.pdf https://bitbucket.ideasoft.io/projects/PAR/repos/solidity/docs/Documentation.pdf

Technical Requirements	https://bitbucket.ideasoft.io/projects/PAR/repos/solidity/docs/Curve APY calculation.pdf https://bitbucket.ideasoft.io/projects/PAR/repos/solidity/docs/Documentation.pdf
Contracts	File: ./contracts/ERC721/ERC721UpgradeableParallax.sol SHA3: f8ef74003134a72c7d0d9ae0586fad2e2c0a27e54947940d685853aa7933c93c File: ./contracts/extensions/CheckerZeroAddr.sol SHA3: aab073312777d6da58c1e72e94df568dbbd05c70b11c0c2d5b2e096e66c25947 File: ./contracts/extensions/Timelock.sol SHA3: 5a11d7bd1f22bc331242f62299a77114b9ebbfc9c8d3d6b53d8962f3cf9de18c File: ./contracts/extensions/TokensRescuer.sol SHA3: 64c9f6a22a113fa24dc756a1b161905875d25ad95eb662a1809aaeb6a45e51b6 File: ./contracts/interfaces/IERC721UpgradeableParallax.sol SHA3: cf5a1e58d8995ec3365b18b0297b88fb9c4d2675aca60df46eee17cc6ce72e66 File: ./contracts/interfaces/IFees.sol SHA3: d578ca8dd568ce2762143f547e9dde41055bdea0efdef48f076f15324fbea673 File: ./contracts/interfaces/IParallax.sol SHA3: 2a20504f74a3521272688528ae271228b2d356166da4cc9ea4d68eb147d5f9e4 File: ./contracts/interfaces/IParallaxStrategy.sol SHA3: 6228cdd9b0c903bbffedeb1f4ae2f13a616a3977e164fb986f0490a56b74a6f5 File: ./contracts/interfaces/ITokensRescuer.sol SHA3: a65a0522cb31bd0a3d7d6f84ed990c790a923893936874f9ecb3e3978bd3382c File: ./contracts/Parallax.sol SHA3: ec9cfe750dbdf77309a50e8b5209b94781125128ed494d1045cae3a2f4c9aebb File: ./contracts/strategies/curve-sorbettiere/CurveSorbettiereStrategy.sol SHA3: 4a80149cad715cf50aad7cd8ab9a4aa6f0b55de2ca08d906efe923ac4c1a1ada File: ./contracts/strategies/curve-sorbettiere/interfaces/ICurve.sol SHA3: f225a8b5861d7d60905bcb64ceaf9a510158b89d5f10d3f00c7b7806fd1d2186 File: ./contracts/strategies/curve-sorbettiere/interfaces/ISorbettiere.sol SHA3: fca77ebe07476bf371d274808e11ff6bced247712178c834733b8a5e6ca8799b

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation by external or internal actors.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor Gas optimization. These issues won't have a significant impact on code execution but affect code quality

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are detailed.
- Technical description is precise.
- There is a diagram explaining the flow.
- NatSpec covers most of the code.

Code quality

The total Code Quality score is **10** out of **10**.

- The development environment is configured.
- The code is structured and function/contract interactions are clear.

Test coverage

Code coverage of the project is **100%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Negative cases coverage is present.
- Interactions by several users are tested thoroughly.

Security score

As a result of the audit, the code contains no issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **10**.

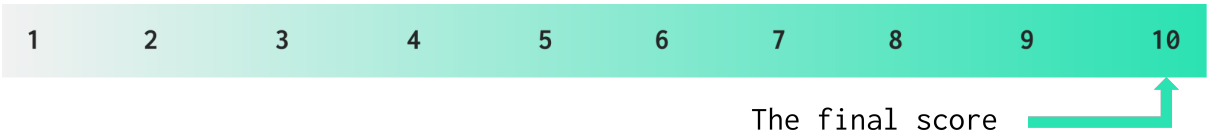


Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
11 January 2023	20	4	4	1
6 February 2023	1	2	1	0
3 March 2023	0	0	0	0

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Not Relevant
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Not Relevant
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed

Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Not Relevant
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155 EIP-712	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of Unused Variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP Standards Violation	EIP	EIP standards should not be violated.	Passed
Assets Integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed
User Balances Manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Passed

Token Supply Manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Style Guide Violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Passed
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, which may be changed in the future.	Passed

System Overview

Parallax is a platform that allows different compounding strategies; at the moment, in the scope of the audit, there is only one, CurveSorbetteriesStrategy, that allows users to deposit into the USDC-USDT-MIM LP on curve and auto-compound the rewards.

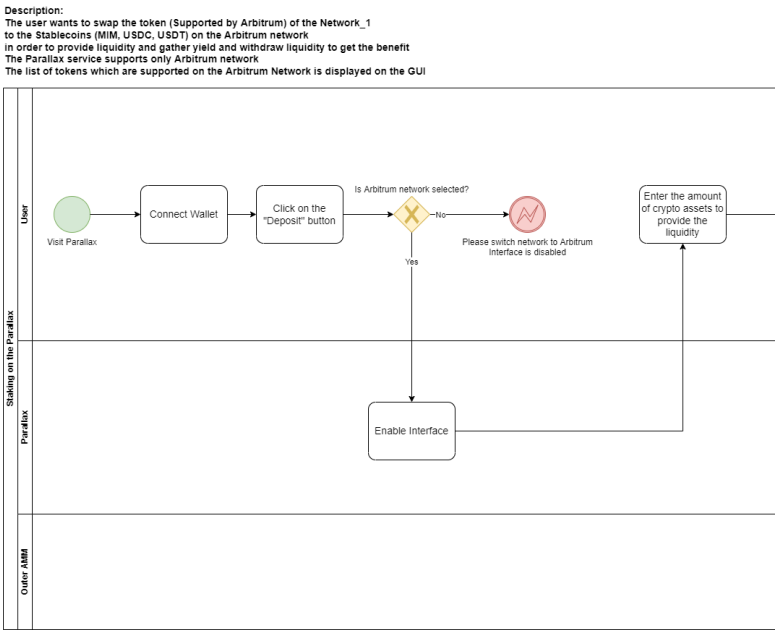
When a position is created, the user receives an ERC721 that is burned on complete withdrawal and used to transfer the position to another user.

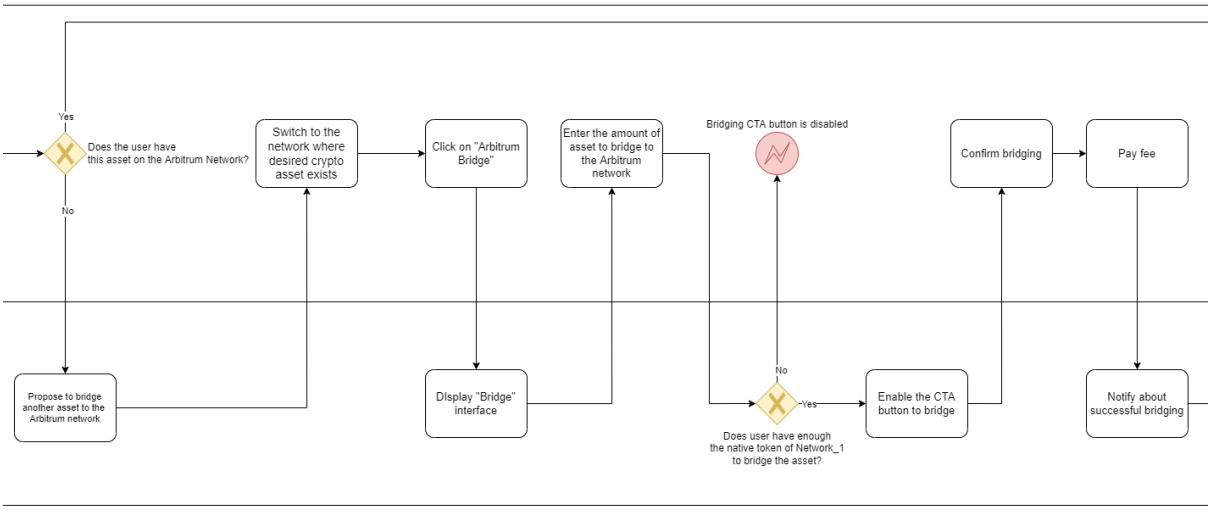
The files in the scope:

- **TokenRescuer.sol** - Contract to rescue tokens and native coins wrongfully sent to parallax and Strategy contracts.
- **IParallax.sol** - Interface for the parallax contract.
- **IParallaxStrategy.sol** - Interface for the parallax strategy.
- **ITokenRescuer.sol** - Interface for the TokenRescuer.
- **Parallax.sol** - Main contract that allows the owner to add new strategies and users to deposit, withdraw positions and transfer NFTs.
- **CurveSorbetterieStrategy.sol** - First strategy that uses curve as base layer, compounding the rewards of a LP on curve.
- **ICurve.sol** - Interface to interact with curve.
- **ISorbetterie.sol** - Interface for the first strategy.
- **Timelock.sol** - The contract to handle timelocks in case of state changes in the contract to give users time to react.
- **CheckerZeroAddr.sol** - The contract responsible for zero address checks.
- **ERC721UpgradeableParallax.sol** - The system NFT that is used for confirmation of positions.
- **IERC721UpgradeableParallax.sol** - The interface for ERC721UpgradeableParallax.sol.
- **IFees.sol** - The contract to hold the withdrawal fees.

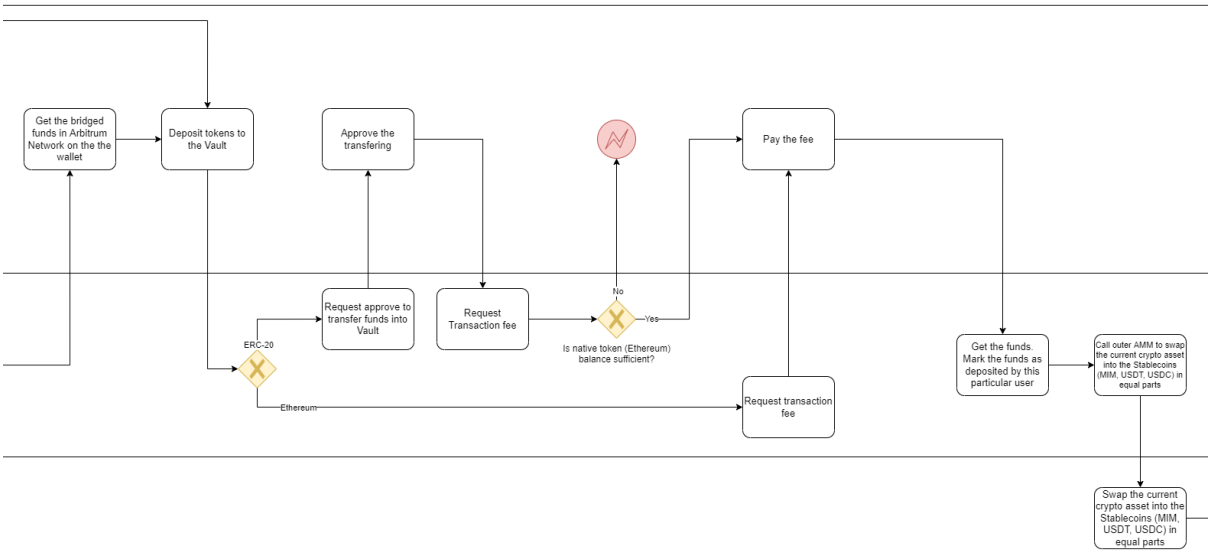
Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

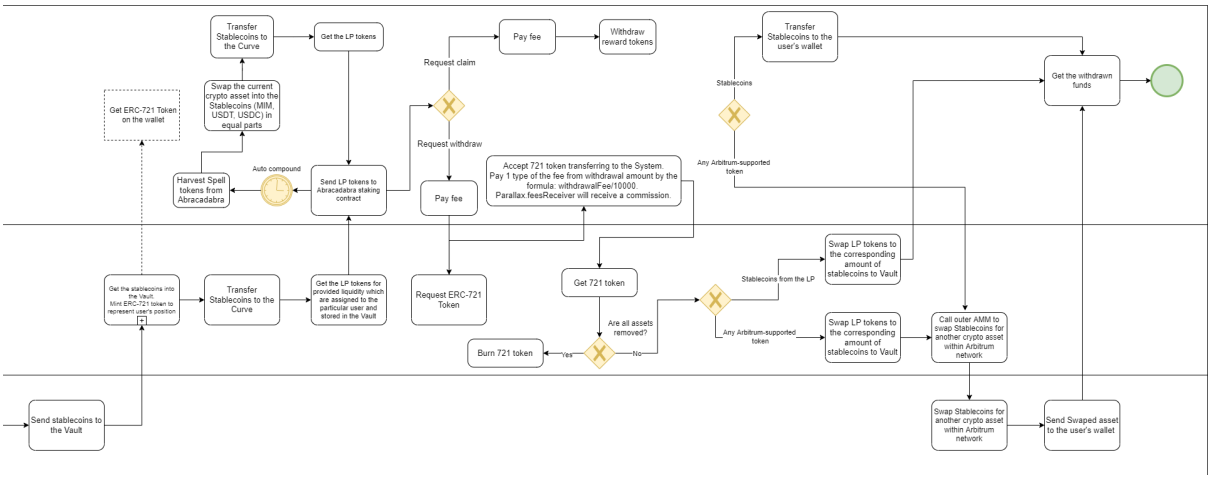
Flow of the project:





Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io





Privileged roles

Roles defined in the system:

- Owner: The owner of the parallax contract can:
  - Rescue tokens sent wrongfully to the contract parallax and strategy contracts (both native and non native).
  - Whitelist and unwhitelist a token that can be deposited.
  - Add a strategy and modify its parameters.
- User: The user in the parallax system can:
  - Open positions and do deposits, withdrawals, and token transfers.

Risks

- The project relies on external factors:
  - USDT-USDC-MIM peg for the first strategy.
  - Sushiswap.
  - USDT, USDC, MIM liquidity on Sushiswap with themselves and other ERC20/ETH.
- There are swaps to deposit and withdraw with whitelisted ERC20 and ETH, which creates a possibility for a race condition where they could be subject to a sandwich attack or lose a part of the deposit due to slippage.
- The upgradeable nature of the contracts puts the user funds at risk in case of logic upgrade.

Recommendations

- The system relies on the secureness of the Owner's private keys, which can impact the execution flow and secureness of the funds. We recommend this account to be at least ⅗ multi-sig.

Findings

■■■■ Critical

C01. Invalid Hardcoded Value

In CurveSorbettiereStrategy.sol contracts' *_harvest()* function, there is a hardcoded route (SPELL, MIM) to be used on Sushiswap on Arbitrum.

The routes/pairs may not always have desired liquidity and this may result in all harvest being lost.

Path:
./contracts/strategies/curve-sorbettiere/CurveSorbettiereStrategy.sol : _harvest()

Recommendation: The route (SPELL, WETH, MIM) should be used, or there should be an admin function that controls the route values in case of low liquidity.

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0)

■■■ High

H01. Front-Running Attack

In the CurveSorbettiereStrategy.sol contract, the parameters for the *compound()* -> *_harvest()* function: *swapMimAmountOutMin* is driven by off-chain data and can be manipulated.

This may result in easy front-running attacks on SPELL - MIM swaps (it can even be done in one transaction without the need to watch the mempool). An attacker can initiate the attack by calling the *compound()* function directly from the Parallax.sol contract.

Path:
./contracts/strategies/curve-sorbettiere/CurveSorbettiereStrategy.sol : compound(), _harvest()

Recommendation: Take the swapping price from an on-chain oracle and calculate swapMimAmountOutMin according to that value. Or sign the data to prevent manipulation.

Status: Fixed
(Revised commit: 8bd4562ece9e838956a0295a1ba10f76b8b5da4a)

H02. Denial of Service

In the CurveSorbettiereStrategy.sol contract, the call *IParallax(PARALLAX).feesReceiver()* is used many times without zero address checks.

Since the safeTransfer() function checks for zero addresses, the transaction will fail if *feesReceiver()* returns a zero address and there will be denial of service.

Path:
./contracts/strategies/curve-sorbettiere/CurveSorbettiereStrategy.sol : withdrawLPs(), withdrawTokens(), withdrawAndSwapForNativeToken(), withdrawAndSwapForERC20Token()

Recommendation: Either there should be zero address checks in CurveSorbettiereStrategy.sol contracts calls or in Parallax.sol, there should be a requirement so that feesReceiver cannot be zero address.

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0)

H03. Invalid Calculations

In the CurveSorbettiereStrategy.sol the *withdrawTokens()* function has underflow if withdrawalFee is larger than 3333.

This will cause Denial of Service in the *withdrawTokens()* function.

Path:
./contracts/strategies/curve-sorbettiere/CurveSorbettiereStrategy.sol : withdrawTokens()

Recommendation: There should be a check in every strategy that extends IParallaxStrategy.sol so that the withdrawalFee or any other data cannot break its logic.

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0)

H04. Checks-Effects-Interactions Pattern Violation

In the Parallax.sol contract, during the functions execution, some state variables are updated after the external calls, which is against best practices.

This may lead to reentrancies, race conditions, and denial of service vulnerabilities during the implementation of new functionality.

- In *_claim()* function, *rewardToken.safeTransfer()* call is made before doing state changes on position.former.
- In *_transferPosition()* function, the *_claim()* function, which has external calls made inside, is called before doing state changes on many variables.
- In *safeTransferFrom(address from, address to, uint256 tokenId)*, and *safeTransferFrom(address from, address to, uint256 tokenId, bytes memory data)* *_transferPosition()* function is called after external call.

Path:
./contracts/Parallax.sol : _claim(), _transferPosition(), safeTransferFrom(address from, address to, uint256 tokenId), safeTransferFrom(address from, address to, uint256 tokenId, bytes memory data)

Recommendation: Common best practices should be followed, functions should be implemented according to the Check-Effect-Interaction pattern. If not possible, the nonReentrant modifier can be used.

- In *_claim()* function, position.former variable can be updated before the *rewardToken.safeTransfer()* call.
- In *_transferPosition()* function, the external call depends on the state variables, so the Checks-Effects-Interactions pattern cannot be followed. That is why a nonReentrant modifier should be used.
- In *safeTransferFrom(address from, address to, uint256 tokenId)*, and *safeTransferFrom(address from, address to, uint256 tokenId, bytes memory data)* *_transferPosition()* function should be called before external call, or nonReentrant modifier should be used.

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0)

Medium

M01. Admin Privilege Actions

In the Parallax.sol contract withdrawal functionality, the owner can set withdrawal fees as much as 100% at any point, change the required timelocks, and change the reward token at any time.

This may result in users losing all their assets in case of maximum fee, indefinite lock of their funds in case of constant timelock manipulation, and change in their expected reward token. Since these changes can be made after users deposit tokens.

Path:
./contracts/Parallax.sol : setFees(), setTimelock(), setRewardToken()

Recommendation: Consider using timelocks for state changes in these functions so that users can respond to the changes. Additionally, consider adding capped limits.

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0)

M02. Best Practice Violation - Unchecked Transfer

In TokenRescuer.sol contracts' *rescueERC20Token()* function, *IERC20Upgradeable(token).transfer()* call is made.

The transfer call can cause unwanted results if the receiver is an address that is not compatible with ERC20 tokens.

Path:
./contracts/extensions/TokensRescuer.sol : rescueERC20Token()

Recommendation: *safeTransfer()* should be used instead of *transfer()*

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0)

M03. Unscalable Functionality - Bad Struct Naming

The Deposit0..3 and Withdraw0..3 functions are non-declarative, in contrast to the *IParallaxStrategy* struct names, which are declarative.

Path:
./contracts/interfaces/IParallax.sol

Recommendation: Struct names should be more declarative.

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0)

M04. Contradiction - Documentation Mismatch

In documentation it is stated that the view function *tokenURI()* returns a URI for a token by its ID and can be called by anyone.

This function will only return an empty string "" as _baseUri is not overridden in Parallax and there is no option to set baseUri.

This behavior contradicts the documentation.

Path:
./contracts/Parallax.sol : tokenURI()

Recommendation: The _baseUri should be overridden so that there is a URI returned for every *tokenURI()* call, or documentation should be updated.

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0)

M06. Unchecked Input Value

In the *CurveSorbettiereStrategy.sol* contract, many functions use the params.path parameter, which is a user input. This parameter is not checked whether it is the right path or not.

The first and last item in the path should match the expected tokens. The lack of check may lead to unexpected behavior and the loss of user funds.

Path:
./contracts\strategies\curve-sorbettiere\CurveSorbettiereStrategy.sol : swapNativeTokenAndDeposit, swapERC20TokenAndDeposit, withdrawAndSwapForNativeToken, withdrawAndSwapForERC20Token

Recommendation: Add a check for the first and last items in path to match the expected tokens.

Status: Fixed
(Revised commit: 8bd4562ece9e838956a0295a1ba10f76b8b5da4a)

Low

L01. Floating Pragma

In every Solidity file in the scope, the expression of *pragma solidity ^0.8.15* is used while specifying the pragma version.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively.

Paths:
./contracts/extensions/TokensRescuer.sol
./contracts/interfaces/IParallax.sol
./contracts/interfaces/IParallaxStrategy.sol
./contracts/interfaces/ITokensRescuer.sol
./contracts/Parallax.sol
./contracts/strategies/curve-sorbettiere/CurveSorbettiereStrategy.sol
./contracts/strategies/curve-sorbettiere/interfaces/ICurve.sol
./contracts/strategies/curve-sorbettiere/interfaces/ISorbettiere.sol

Recommendation: Lock the pragma version and consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0)

L02. Missing Require Check

In Parallax.sol contracts' *getUsers()* and *getUsersByStrategy()* functions, the cursor input parameter is not checked.

If the cursor parameter is given as 0, there will be underflow at *result[i - 1]* expression.

Path:
./contracts/Parallax.sol : getUsers(), getUsersByStrategy()

Recommendation: Add a requirement check so that the cursor is at least given as 1.

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0) (There is an error message typo in *_cursorIsLessThanOne(uint256 cursor)* function.)

L03. Missing NatSpec

There are code explanations in the documentation; however, NatSpec in code is missing.

It is best practice to use NatSpec in the code.

Paths:
./contracts/extensions/TokensRescuer.sol
./contracts/interfaces/IParallax.sol

./contracts/interfaces/IParallaxStrategy.sol
./contracts/interfaces/ITokensRescuer.sol
./contracts/Parallax.sol
./contracts/strategies/curve-sorbettiere/CurveSorbettiereStrategy.sol
./contracts/strategies/curve-sorbettiere/interfaces/ICurve.sol
./contracts/strategies/curve-sorbettiere/interfaces/ISorbettiere.sol

Recommendation: NatSpec should be added to the code.

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0)

L04. Style Guide Violation: Order of Functions

The provided projects should follow the official guidelines. Functions should be grouped according to their visibility and ordered:

1. Constructor
2. Receive function (if exists)
3. Fallback function (if exists)
4. External
5. Public
6. Internal
7. Private

Paths:
./contracts/Parallax.sol
./contracts/strategies/curve-sorbettiere/CurveSorbettiereStrategy.sol

Recommendation: Follow the official Solidity guidelines.

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0)

L05. Functions that Can Be Declared Internal

The initialize function should be called only once and it is already called in the *__CurveSorbettiereStrategy_init* and *__Parallax_init*.

Paths:
./contracts/Parallax.sol : __Parallax_init_unchained;
./contracts/strategies/curve-sorbettiere/CurveSorbettiereStrategy.sol : __CurveSorbettiereStrategy_init_unchained;

Recommendation: Change the visibility of the initialize functions.

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0)

L06. Inconvenient Naming

In Parallax.sol contracts' Strategy struct, the lastUpdate variable name can be misleading as it looks like a timestamp.

Path:
./contracts/Parallax.sol

Recommendation: Consider adding Block to the name.

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0)

L07. Redundant Code

In Parallax.sol contracts' *__Parallax_init()* function calling *__Context_init_unchained()* and *__TokensRescuer_init_unchained()* are redundant.

In CurveSorbettiereStrategy.sol contracts' *__CurveSorbettiereStrategy_init()* function calling *__Context_init_unchained()* and *__TokensRescuer_init_unchained()* are redundant.

Paths:
./contracts/Parallax.sol : __Parallax_init()
./contracts/strategies/curve-sorbettiere/CurveSorbettiereStrategy.sol : __CurveSorbettiereStrategy_init()

Recommendation: Consider removing redundant code.

Status: Fixed
(Revised commit: 8bd4562ece9e838956a0295a1ba10f76b8b5da4a)

L08. Parameter Name Contradiction

In Parallax.sol contracts' *onlyContract()* modifier, parameter named '*address strategy*'; contradicts with modifier name as it can be not only strategy but any contract.

Path:
./contracts/Parallax.sol : onlyContract()

Recommendation: Give more general naming to 'strategy'

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0)

L09. Wrong Modifier Usage

In CurveSorbettiereStrategy.sol contracts' *__CurveSorbettiereStrategy_init_unchained()* function, the initializer modifier is used incorrectly.

In Parallax.sol contracts' *__Parallax_init_unchained()* function, the initializer modifier is used incorrectly.

In TokenRescuer.sol contracts' *__TokensRescuer_init_unchained()* function, the initializer modifier is used incorrectly.

Openzeppellin has just added mitigation for this mistake.

Paths:
./contracts/Parallax.sol : __Parallax_init_unchained()
./contracts/strategies/curve-sorbettiere/CurveSorbettiereStrategy.sol : __CurveSorbettiereStrategy_init_unchained()

./contracts/extensions/TokensRescuer.sol : __TokensRescuer_init_unchained()

Recommendation: onlyInitializing modifier should be used.

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0)

L10. Constructor Usage

In Parallax.sol and CurveSorbettiereStrategy.sol contracts, recommended *_disableInitializers()* call should be added to constructors.

Paths:
./contracts/Parallax.sol : constructor()
./contracts/strategies/curve-sorbettiere/CurveSorbettiereStrategy.sol : constructor()

Recommendation: Consider adding constructor with *_disableInitializers()* call.

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0)

L11. Missing Zero Address Validation

Address parameters are used without checking against the possibility of being 0x0.

This can lead to unwanted external calls to 0x0.

Paths:
./contracts/Parallax.sol : __Parallax_init_unchained(), rescueNativeToken(), rescueERC20Token(), addToken()
./contracts/extensions/TokensRescuer.sol : rescueNativeToken(), rescueERC20Token()

Recommendation: Implement zero address validations.

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0)

L12. Functions that Can Be Declared External

There are some functions that can be declared external.

External functions consume less Gas.

Paths:
./contracts/Parallax.sol : rescueNativeToken(), rescueERC20Token()
./contracts/extensions/TokensRescuer.sol : rescueNativeToken(), rescueERC20Token()

Recommendation: Use the *external* attribute for functions that are never called from the contract.

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0)

L13. Readability

In the Parallax.sol contracts' *_claim()* function the:

if (value == 0) {} else if (

can be converted to guard:

if(value == 0) return;

+ if statement for readability.

Path:
./contracts/Parallax.sol : _claim()

Recommendation: This conversion can be made.

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0)

L14. Typos in Variable Names

There are some typos in naming, such as fromUserPosistion, toUserPosistion, witdrawalAmount, usdcAmoutOutMin.

Paths:
./contracts/Parallax.sol : fromUserPosistion, toUserPosistion
./contracts/extensions/TokensRescuer.sol : witdrawalAmount, usdcAmoutOutMin

Recommendation: Spellings should be fixed.

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0)

L15. Gas Optimization

In the Parallax.sol contracts' *_getStakedBySharesAmount()* there is a double read from storage on the variable *strategies[strategyId].totalStaked*.

Path:
./contracts/Parallax.sol : _getStakedBySharesAmount(),

Recommendation: Use a memory variable to read from storage only once.

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0)

L16. Zero Value Check

In the Parallax.sol contract, there is no check for strategies[strategyId].totalShares == 0 in *_getEarnedBySharesAmount()* function and this may result in division by 0.

Path:
./contracts/Parallax.sol : _getEarnedBySharesAmount(),

Recommendation: Consider adding a zero-value check.

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0)

L17. Redundant Code Block

uint256[50] private __gap at the end of the Parallax and CurveSorbettiereStrategy are redundant; there is no need to reserve storage slots in top-level contracts.

Paths:
./contracts/Parallax.sol;
./contracts/strategies/curve-sorbetterie/CurveSorbetterieStrategy.sol

Recommendation: Remove the redundant code block.

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0)

L18. Redundant Code Block

Override specifier is not needed when only overriding interface declaration starting from Solidity 0.8.8.

Paths:
./contracts/Parallax.sol
./contracts/strategies/curve-sorbetterie/CurveSorbetterieStrategy.sol

Recommendation: Consider removing redundant override specifiers to clarify what really needs to be overridden.

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0)

L19. Zero Valued Transactions

Best practice is to check and transfer funds when the amount is > 0. In the *CurveSorbettiereStrategy*, in case the *fee == 0*, there are redundant transfer operations.

Path:
./contracts/strategies/curve-sorbetterie/CurveSorbetterieStrategy.sol

Recommendation: Implement conditional checks for the zero-valued transaction.

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0)

L20. Dead Code

The function *_toDynamicArray(uint256[3] memory input)* is declared private but never used.

Path:
./contracts/strategies/curve-sorbetterie/CurveSorbetterieStrategy.sol
: _toDynamicArray(uint256[3] memory input)

Recommendation: Remove the dead code.

Status: Fixed
(Revised commit: f87b8cae7dbd9970a24276c31885c15f8f7c1bb0)

Findings Of The Customer Team

These Findings were **found by the Parallax development team** during the internal testing process.

■■■■ Critical

SC01. Data Inconsistency

In the Parallax.sol contracts *_withdraw()* function, when a position is closed, the positionCount parameter is decreased.

However, if the closed position is not the last position in the array, the next deposit with a zero positionId replenished the last position and a new position is not created.

Path:
./contracts/Parallax.sol : _withdraw()

Fix: A different parameter named positionsIndex is being incremented with every new position and used to create new positions. The positionsCount parameter is not used to create new positions anymore.

Found in: ce3b18c0ceea7ae75c3170c3452b08d1adf6936f

Status: Fixed
(Revised commit: 8bd4562ece9e838956a0295a1ba10f76b8b5da4a)

■■■ High

SH01. Contradiction

In the _transferPosition() function of the Parallax.sol contract, the _claim call is made for the *msg.sender* instead of *from* which is the user which the position is being transferred from.

This can lead to funds losses of the *“from”* user.

Path:
./contracts/Parallax.sol : _transferPosition()

Fix: The *msg.sender* is now converted to *from* parameter.

Found in: ce3b18c0ceea7ae75c3170c3452b08d1adf6936f

Status: Fixed
(Revised commit: 8bd4562ece9e838956a0295a1ba10f76b8b5da4a)

■■ Medium

SM01. Data Inconsistency

In the *_transferPosition()* function of the Parallax.sol contract, the user migrations were not checked like a new position or last position.

This was creating inconsistencies with the data flow when a user deposits directly or gets transferred a position from someone else.

Path: ./contracts/Parallax.sol : _transferPosition()

Fix: Two new functions are introduced which are named _addNewUserIfNeeded() and _deleteUserIfNeeded(). These functions implement the addition of a new user if the user is new and the removal of a user if the position is the last position that the user withdraws/transfers. These functions are called every time a deposit or withdrawal is made. These are also called on position transfers which solves the data inconsistency issue.

Found in: ce3b18c0ceea7ae75c3170c3452b08d1adf6936f

Status: Fixed
(Revised commit: 8bd4562ece9e838956a0295a1ba10f76b8b5da4a)

SM02. Logic Error

In the Parallax.sol contracts' *getUsers()* and *getUsersByStrategy()* functions the expressions result[i-1] = users[i]; and result[i-1] = strategy.users[i]; were errors since i is starting from the cursor value but the results arrays should be addressing indexing starting from 0 to howMany parameter.

Path: ./contracts/Parallax.sol : getUsers(), getUsersByStrategy()

Fix: There is now a new variable j which starts from 0 and iterates through the result array by being incremented by 1 at the end of the for loop. The new expressions are the following:

```
result[j] = users[i];
++j;
```

and

```
result[j] = strategy.users[i];
++j;
```

Found in: ce3b18c0ceea7ae75c3170c3452b08d1adf6936f

Status: Fixed
(Revised commit: 8bd4562ece9e838956a0295a1ba10f76b8b5da4a)

SM03. Logic Error

In the CurveSorbettiereStrategy.sol contract, the *_swapETHForTokens()* and *_swapTokensForETH()* functions have checks to see if the path.length is equal to zero.

If the path.length is zero, the function *return 0* as its value; this is incorrect.

If the user mistakenly provides a path of zero length, the return value from the function call used in deposits and withdrawals will be zero, resulting in the user's funds being locked in the contract until token rescuing functionality is used.

Path:
./contracts\strategies\curve-sorbettiere\CurveSorbettiereStrategy.sol : _swapETHForTokens, _swapTokensForETH

Fix: The functions were modified so that in case of path.length == 0, the functions would not return 0 but revert through the swapping protocol. The manual check was removed.

Found in: ce3b18c0ceea7ae75c3170c3452b08d1adf6936f

Status: Fixed
(Revised commit: 8bd4562ece9e838956a0295a1ba10f76b8b5da4a)

SM04. Undocumented Behavior

When using the method *withdrawTokens()* the withdrawal proportion could have been different and this led to an incorrect calculation of the fees and rewards.

This was an undocumented behavior of the *withdrawTokens()* function; it was expected to work in the same way as the *depositTokens()* function that takes as an input the tokens in the same proportion.

Path:
./contracts\strategies\curve-sorbettiere\CurveSorbettiereStrategy.sol : withdrawTokens()

Fix: The function calculates the fees and transfers a percentage of the LP instead of transferring one of the tokens.

Also, the other functions for the withdrawal of the liquidity have been changed.

Found in: ce3b18c0ceea7ae75c3170c3452b08d1adf6936f

Status: Fixed
(Revised commit: 8bd4562ece9e838956a0295a1ba10f76b8b5da4a)

Low

SL01. Logic Error

In the Parallax.sol contracts' *getClaimableRewards()* function if the rewards token is set as zero address, the accumulated rewards are shown incorrectly as if there is a non zero token address.

This would happen if the accrual of rewards is stopped by setting the token address to zero address instead of setting rewardsPerBlock to 0.

Setting rewardToken to address(0) should not be considered as a proper way of stopping the rewards occurrence. As this also creates a DoS issue within claim() and _transferPossition() functions. Only valid way should be to set rewardPerBlock = 0.

Path: ./contracts/Parallax.sol : getClaimableRewards()

Fix: There is now a check so that if the rewardsPerBlock is larger than 0, the rewardToken cannot be a zero address. Also when setting rewardToken after the first set, it cannot be set as zero address.

Found in: ce3b18c0ceea7ae75c3170c3452b08d1adf6936f

Status: Fixed
(Revised commit: 8bd4562ece9e838956a0295a1ba10f76b8b5da4a)

SL02. Missing Validation

If the *compoundMinAmount* value is set to 0, or a small value compared to the liquidity present in the liquidity pool that is being used, every deposit, withdraw or compound will revert with this error message *UniswapV2Library: INSUFFICIENT_INPUT_AMOUNT.*

Path:
./contracts/Parallax.sol : setCompoundMinAmount()
./contracts/strategies/CurveSorbetterieStrategy.sol : setCompoundMinAmount(), __CurveSorbetterieStrategy_init_unchained()

Found in: 8bd4562ece9e838956a0295a1ba10f76b8b5da4a

Status: Mitigated (It is advised that the *compoundMinAmount* should be set correctly by the Parallax team to prevent the situation of DoS resulting from too small swap amounts, or to implement a validation for the *compoundMinAmount* and *initialCompoundMinAmount*.)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.