vacuumlabs

WingRiders v2

Audit Report v1

September 4, 2024

Contents

Revision table

Report version	Report name	Date	Report URL
1.0	Main audit	2024-09-04	Full report link

1 Executive summary

THIS REPORT DOES NOT PROVIDE ANY WARRANTY OF QUALITY OR SECURITY OF THE AUDITED CODE and should be understood as a best efforts opinion of Vacuumlabs produced upon reviewing the materials provided to Vacuumlabs. Vacuumlabs can only comment on the issues it discovers and Vacuumlabs does not guarantee discovering all the relevant issues. Vacuumlabs also disclaims all warranties or guarantees in relation to the report to the maximum extent permitted by the applicable law. This report is also subject to the full disclaimer in the appendix of this document, which you should read before reading the report.

Project overview

WingRiders v2 is an extension and a rewrite of WingRiders' AMM decentralized exchange version 1, featuring both constant product and stableswap liquidity pools. The version 1 which runs on the mainnet today was delivered in two milestones with the constant product pools delivered first and the stableswap option later. Rather than explaining the whole functionality, we will focus on explaining the differences introduced in this version.

Most importantly, the version 2 contains entirely new code. It is a rewrite and a refactor from Plutus to Plutarch. It utilizes Plutus v2 features such as reference scripts, inline datums, and more. That optimizes the scripts' performance. Additionally, it includes further optimizations across many functions. The version 2 merges the two codebases for constant product and stableswap pools by having a single factory script that creates both types of pools. The liquidity pool validity tokens and share tokens have the same minting policy. Previously, the two types of pools were deployed as separate instances of the protocol. Constant product requests and liquidity pools still have different validator hashes compared to the stableswap versions. They share the code, though. Other than that, the following features have been added in the WingRiders v2:

1. *Stableswap decimals support*. Previously, the stableswap pools worked only with assets with the same number of decimals. Support for different decimals has been added in the v2 version. As it is not easy to determine the ***correct*** number of decimals on-chain, a stableswap liquidity pool can be created for any combination of decimals for two assets. Naturally, only one such combination is the right one. The decimals are saved in the liquidity pool's datum. It is critical that users or any frontend interface they use check this value to determine the correct pool they should interact with. Interacting with a pool with different decimals could result in funds

lost. Further, the decimals are part of the request datum so that no request can be applied to a pool with wrong decimals. Finally, the share tokens are also different for any decimals chosen. *Practically, two stableswap pools with the same assets but different decimals are totally different.*

2. *Updateable fees*. Previously, the fees such as the agent fee or the protocol fee were fixed. In the current version, they are stored in a pool's datum and can be updated at any time in a direct pool interaction by a party authorized by a special new token just for this. There are two different token types. An *agent fee authority token* whose holder can update the agent fee. *There is no grace period and any requests applied after the change are automatically subject to the new fee.* That may deem some already created requests invalid or it can allow for an agent to take more Ada from the requests than the user expected. However, users are still protected by their chosen minimum compensation thresholds. This is true also in the case of routed swaps explained below, where it can invalidate an entire routed chain in a middle point.

   The other token is called the *fee authority token*. Its holder can freely change the other fees, including the protocol fee, reserve fee and project fee. The last two fees were added in this version. They are separated from the protocol fee and are accounted for separately in the pool datum. Additionally, they list a reserve fee beneficiary and a project fee beneficiary per pool. These values can be also freely changed by the fee auth token holder at any time. The fees are extracted the same way as the protocol fee was extracted before. There are two different extract treasury request types for this purpose. The particular extraction request puts the accumulated fees to the fee beneficiary listed in the datum at the time. The *fee authority token holder fully owns all the accumulated project and reserve fees* until the time they are extracted and sent to a different party. These fees can also be changed at any time and the change will affect all requests applied from the change onwards.

   Both parties are privileged and trusted. They could spam the pool by direct interactions with the pool or withdraw all accumulated fees for themselves (see WR2-401). Additionally, they could collect more fees out of the already existing requests if the minimum compensation thresholds are benevolent.

3. *Routing*. It is possible to create routed swaps. More specifically, it is possible to specify a beneficiary address and a compensation datum. The request compensation will be created at that address with that datum. By directing the compensation again to the request address and specifying custom nested datum, it is possible to route any number of steps this way, including two swaps with different liquidity pools, adding liquidity and then locking the share tokens into a farm. To enable this,

a custom request oil Ada is allowed as it may be needed to supply more Ada to cover the min Ada requirement for bigger inline datums.

The risk of such swaps is entirely up to the users creating them. For example, if a routed swap comprising of three swaps is created, it might get stuck in the middle swap due to a change in the liquidity pool's asset ratio and e.g. too strict minimum compensation thresholds set. That would mean that the owner needs to reclaim that swap and could be left out with tokens he never wanted. It should also be noted that by creating a routed swap, the entire chained intent is submitted publicly. That makes it possible for malicious players to front-run such swaps and take the profit up to the thresholds specified.

4. *Zap-in and zap-out*. Previously, the zap-in functionality was supported only in stableswap pools. In the current version, both zap-in and zap-out functionalities are supported for both constant product and stableswap pools. Additionally, stableswap zap-ins incur all protocol, reserve and project fees for its swap part.

The current version of WingRiders Launchpad does not work with WingRiders v2 straight away. A new launchpad version is needed to support these new pools. Additionally, we did not audit any liquidity migration scripts nor plan.

Audit overview

We started the audit at commit `b0c2f59b6e813bcb7580ec96c5b87a39c179ec56` and it lasted from 22 May 2024 to 4 Sep 2024. The timeframe is inclusive of periods in which we were awaiting the implementation of fixes by the client. We interacted mostly on Slack and gave feedback in GitHub pull requests. The team fixed all issues to our satisfaction, except for 1 minor issue that was acknowledged and 1 minor issue that was resolved only partially. They do not represent security threats to the protocol. The first issue can be mitigated by frequent treasury extractions. The second issue can be mitigated if advanced users that make use of routing requests to their own scripts make sure that their scripts can handle additional tokens.

The scope of the audit was limited to the smart contract files only. We started with audited versions of the code and reviewed its rewrite and new features. We did not review any tests as part of this audit, even though a large set of tests was a part of the codebase. We appreciate the test coverage. We easily created a proof of concept for one tricky issue thanks to it. This all helped smooth the audit process and raise our and their confidence in the code.

We performed a design review along with a deep manual audit of the code and reported findings along with remediation suggestions to the team in a continuous fashion, allowing the time for a proper remediation that we reviewed afterwards. See more about our methodology in Methodology.

The commit `32c371b5d1cacedb264db8acae09dc0413d733c3` represents the final version of the code. The status of any issue in this report reflects its status at that commit. You can see all the files audited and their hashes in Audited files. The smart contract language used is Plutarch and the contracts are intended to run on Cardano. To avoid any doubt, we did not audit Plutarch itself.

Summary of findings

During the audit, we found and reported: 3 critical, 3 major, 1 medium, 3 minor, and 5 informational findings. All findings except for two minor findings were fully resolved. The findings that were not fully resolved:

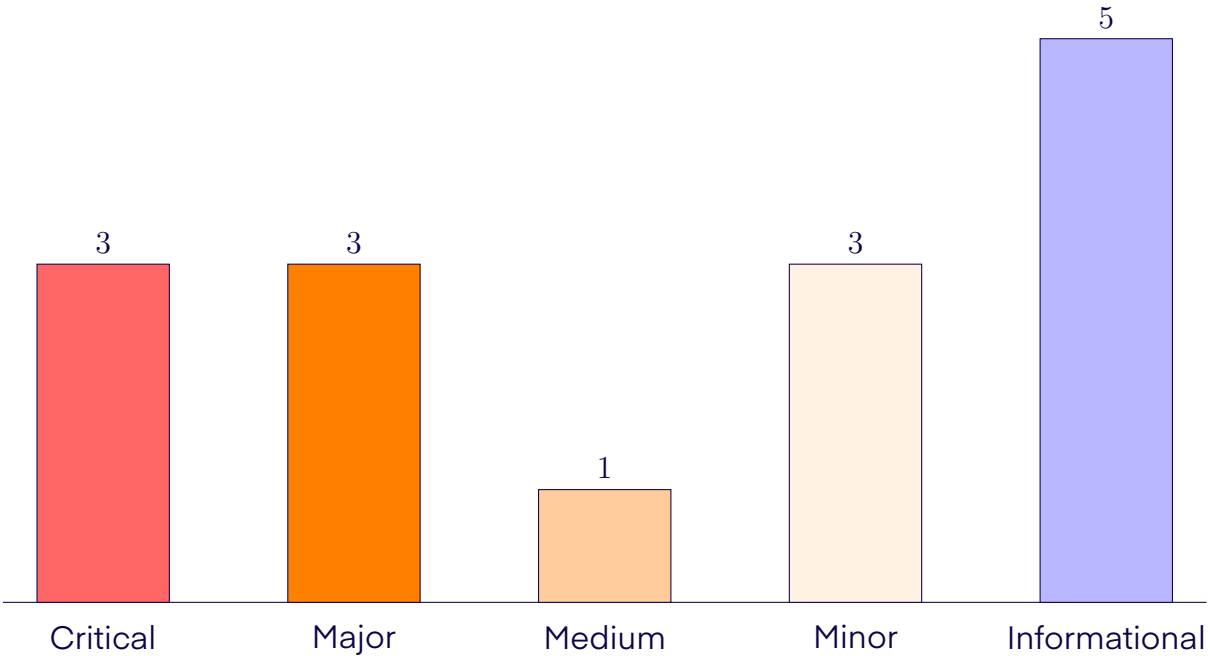
- **WR2-301** – Fee auth token holders can withdraw accumulated project and reserve fees. This issue was acknowledged as intended. The fee auth token holders are supposed to own the project and reserve fees until the point when they are extracted from the pool and e.g. sent to the project address. This property was not adequately documented at the time and seemed counter-intuitive to us, since the accounting is separate for those fees. The documentation was improved but the risk remains. However, since the parties are trusted and if the treasuries are frequently extracted, the risk is small.
- **WR2-303** – Additional tokens may make compensation output unspendable. This issue was partially resolved. The number of additional tokens was limited. However, it is not as strict as it could be – mainly due to decreased performance if it was. There can be a few additional tokens in compensation outputs. That theoretically means that if a script compensation address is used and an agent puts additional tokens in it, the script may not be able to handle it. It is quite an edge case. We advise anyone forwarding their request compensation to their custom made scripts to make sure the scripts can handle additional tokens.

A few issues such as issues WR2-001 and WR2-302 were plain code bugs, e.g. a few fields from big datum structures that were left unchecked. This was likely due to new fields being added while not all validations in the code were updated alongside the change. There was also one major issue of this kind that the WR team found and fixed themselves during the on-going audit.

Another set of vulnerabilities concerned the stableswap invariant. The issue WR2-102 was about an add liquidity formula that needed a big rework because of the fees that were added to the imbalanced liquidity provision and with whose the formula did not count correctly. The issue WR2-101 describes how the stableswap invariant would have been broken in a rare emergency withdrawal scenario. Finally, the issue WR2-103 also describes a discrepancy in the stableswap invariant, given Ada stableswap pools and staking rewards' addition to the pool.

The rest of the issues were either about more complex edge cases in the protocol or numerous code style and documentation suggestions.

2 Severity overview



Findings

ID	TITLE	SEVERITY	STATUS
WR2-001	Liquidity pools can be emptied by the fee auth token holder	CRITICAL	RESOLVED
WR2-002	Requests can be unlocked by agents	CRITICAL	RESOLVED
WR2-003	Malicious fee change can block all liquidity	CRITICAL	RESOLVED
WR2-101	Emergency withdrawal in a stableswap pool breaks the invariant	MAJOR	RESOLVED
WR2-102	Stableswap zap-in formulas are wrong about non-pool fees	MAJOR	RESOLVED
WR2-103	Adding staking rewards to a stableswap pool breaks the invariant	MAJOR	RESOLVED

Continued on next page

ID	TITLE	SEVERITY	STATUS
WR2-201	Anyone can block script fee beneficiary funds	MEDIUM	RESOLVED
WR2-301	Fee auth token holders can withdraw accumulated project and reserve fees	MINOR	ACKNOWLEDGED
WR2-302	`pinitialStableswapPoolCorrect` does not check the value of `agentFeeAda`	MINOR	RESOLVED
WR2-303	Additional tokens may make compensation output unspendable	MINOR	PARTIALLY RESOLVED
WR2-401	Dead code	INFORMATIONAL	RESOLVED
WR2-402	Documentation	INFORMATIONAL	RESOLVED
WR2-403	Naming	INFORMATIONAL	RESOLVED
WR2-404	`feeInBasis` semantics and naming	INFORMATIONAL	RESOLVED
WR2-405	Zap-in `swapA` is not sanitized	INFORMATIONAL	RESOLVED

WR2-001 Liquidity pools can be emptied by the fee auth token holder

Category	Vulnerable commit	Severity	Status
Code Issue	`b0c2f59b6e`	CRITICAL	RESOLVED

Description

A fee authority token serves to identify actors that are able to update the pool fees and the addresses where they can be withdrawn. The updated fees are only applied for new requests. To change the fees, an actor spends the pool directly and updates the fee fields in its datum. The token is the same across all liquidity pools.

When interacting with the pool, however, it is not checked that the datum fields `reserveTreasuryA` and `reserveTreasuryB` are kept unchanged. That means that they can be set to any `Integer` value. That allows an attacker to empty the whole liquidity pool, and, since all pools suffer from this and all rely on the same fee auth token, he can empty all the pools. The attacker can achieve it by setting the `reserveTreasuryA` to the whole quantity of asset A in the liquidity pool and doing the same for the asset B. Additionally, he can lawfully update the `reserveBeneficiary` address in the same transaction. Having set the critical field in the pool datum, the attacker can now make an `ExtractReserveTreasury` request to empty the liquidity pool. Finally, he can repeat this for all the pools.

It is important to note, that only the fee authority token holder can do this. It is also crucial to note, that an agent has to execute the `ExtractReserveTreasury` request. However, as all the protocol and user funds are at stake and the agent is automated, we still consider this a critical issue. Even without emptying the liquidity pools, the attacker could damage the pools by setting the fields to malicious values that would make any other requests fail. To achieve this, the agent is not even necessary.

Recommendation

We suggest checking that the `reserveTreasuryA` and `reserveTreasuryB` pool datum fields are left untouched in the `pvalidateChangeFees` pool validation path.

Resolution

The issue was fixed according to our recommendation in the pull request number 935.

WR2-002 Requests can be unlocked by agents

Category	Vulnerable commit	Severity	Status
Code Issue	`b0c2f59b6e`	CRITICAL	RESOLVED

Description

As it is not possible to choose the order of transaction inputs, the locations of request inputs belonging to the compensation outputs are part of the redeemer. The `porder` function is an optimized function that outputs the ordered request inputs based on the redeemer. The vulnerability lies in the initial call of the function. In short, the function has the following signature:

```
pfix #$ plam \recur all' rest last' locs'
```

Whereas `all'` represents all inputs, `last'` represents the index of the input last parsed, such that the rest represents the inputs after it and `locs'` represents the locations to be found. Based on whether the next index in `locs'` is before or after `last'`, is the `last'` and rest either reset, iterating again from the start of the inputs or it continues iterating just in the rest.

As mentioned, the issue lies in the initial call of this function. When called for the first time, the `rest` is set to all the inputs and the `last'` is set to 999, a constant seemingly large enough such that there is no input index smaller than it. Translated, what this means, is that the last index after which there are still all the inputs, is 999. This can be misused. If the redeemer contains $requestLocations = [1000, 1001, 1002, 0, 1, 2]$, it returns the inputs on indices 0, 1, 2 and again the same inputs on indices 0, 1, 2.

This alone is not a critical problem of itself. However, the other validations validating non-malformance of the request locations part of the redeemer checks only the indices. For example, noDuplicates check correctly validates as the $[1000, 1001, 1002, 0, 1, 2]$ array does not contain any duplicates.

We now show how an agent is able to utilize this to unlock any requests of his choice. To continue with the `requestLocations` mentioned above, he will prepare 3 very small requests. He will effectively reference his 3 requests twice in the `requestLocations` by using the above array. That means that he indeed has to execute his 3 requests twice – e.g. if one of them is a withdraw liquidity request, he needs to supply more share tokens from his wallet in order to execute the request twice. This transaction enables, however, for him

to include 3 other requests of his choice, not execute them and unlock the contents for himself.

Even though the agents are currently trusted, this might not hold in the future. Moreover, even a trusted agent should not have the ability to unlock any user funds contained within requests.

Recommendation

We recommend modifying the `porder` function's initial call arguments. To avoid the magic number as well, you might introduce a `Maybe` type there. Even without it, consider setting the `rest'` to an empty list if the `last'` is high and checking that the number of inputs is indeed lower than the number.

Resolution

The issue was fixed according to our recommendation in the pull request number 944.

WR2-003 Malicious fee change can block all liquidity

Category	Vulnerable commit	Severity	Status
Code Issue	`2db11b740f`	CRITICAL	RESOLVED

Description

There are two parties that can update different fees – the agent fee token holder can update the agent fee and the fee auth token holder can update the protocol, project and reserve treasury fees. By setting the agent fee too high, the pool might be unable to accommodate new requests. By setting the other fees very close to 100%, all the liquidity from the requests could be taken. This is all expected. They are trusted parties, after all. The fee change needs to be monitored.

However, they should not be able to block all the liquidity that is already in the pool. This statement is clear, as there is also the emergency withdrawal use case. Even if there is no agent, users can still withdraw their liquidity using the emergency withdrawal. There should be nothing stopping liquidity providers from withdrawing their liquidity.

The vulnerability lies in the fact that the fee change authorities can, in fact, disable remove liquidity code path by setting either the `agentFeeAda` value or the `feeBasis` value to such a high and large number that they are still able to update the fee to it, but any following emergency withdrawal transaction would exceed the transaction size. Ultimately, this means that they can block all the liquidity across all the pools. We accompany this finding with proof of concept tests showcasing these vulnerabilities.

Recommendation

We recommend mandating small enough maximum numbers for `agentFeeAda` and `feeBasis`.

Resolution

The issue was fixed in the pull request number 959.

WR2-101 Emergency withdrawal in a stable-swap pool breaks the invariant

Category	Vulnerable commit	Severity	Status
Logical Issue	e1ab5f7342	MAJOR	RESOLVED

Description

The parameter D, crucial to the stableswap invariant, is carried in the pool datum. Even though a liquidity withdrawal changes the D parameter, in an emergency withdrawal, the new parameter D is not verified and it is not changed in the datum. It stays the same. As it roughly corresponds to the amount of balanced liquidity, it should be lower than is noted in the datum.

This breaks the main stableswap invariant. Furthermore, it influences the application of some requests. Most notably, it influences the computation of swaps where the invariant is checked for the new pool reserves but with the old pool parameter D. It either makes swap requests fail if they are too low or it requires unbalanced swaps overall upping the liquidity and making the compensation smaller for the party making the swap request – hence increasing D to the level that is carried in the datum.

The imbalance can be fixed, for example by applying a remove liquidity request whose computation does not depend on the old D but allows for a new D to be verified and saved into the datum.

The attack requires an emergency withdrawal which requires the pool to be considered abandoned, meaning no interaction with it for 2 weeks. After that, it can deny the service and cause some users to lose funds on their swaps. The swaps feature a minimum wanted assets protection fields. When directly swapping, a user might be cautious to agree to the computed value. However, the swap might be hidden in a zap-in functionality. It is harder to estimate the fair value of share tokens that a user wants to receive.

Recommendation

We recommend requiring new parameter D to be supplied alongside an emergency withdrawal action, checking that the invariant holds and saving it into the pool datum.

Resolution

The issue was fixed according to our recommendation in the pull request number 946.

WR2-102 Stableswap zap-in formulas are wrong about non-pool fees

Category	Vulnerable commit	Severity	Status
Design Issue	`2db11b740f`	MAJOR	RESOLVED

Description

Previously in `v1`, all imbalanced liquidity provisions incurred only a swap fee that went into the pool. The formulas reflected this. Currently, some fees such as the protocol fee, the project fee and the reserve fee do not count towards the pool liquidity, they are collected and accounted for separately. The formulas were not updated correctly to reflect this fact, though.

Notably, the function `pstableswapAddLiquidity` computes the amount swapped, `swapA`, as if all the liquidity went into the pool, which is not true. Further, it subtracts the (incorrectly computed) non-pool fees from the new pool reserves. That further goes into the `pcheckAddLiquidity` function. The purpose of that function is to check that the correct amount of share tokens was earned. However, the function as well as its documentation, are again not updated correctly. From the already subtracted new reserves, it subtracts all the fees again and uses that to decide the number of fair share tokens.

Put together, the protocol, project and reserve fees in stableswap imbalanced liquidity provisions (zap-ins) are paid approximately slightly more than twice. Once directly, although the formula is incorrect. The second time indirectly, in a smaller number of shares earned.

Recommendation

We recommend revisiting all the stableswap add liquidity formulas with a strong focus on what fees go into the pool vs. what fees don't. We recommend updating documentation with this as well.

Resolution

The issue was fixed in the pull requests number 964 and 973. The formulas governing stableswap liquidity addition were simplified and changed to reflect the different fees.

WR2-103 Adding staking rewards to a stable-swap pool breaks the invariant

Category	Vulnerable commit	Severity	Status
Code Issue	e1ab5f7342	MAJOR	RESOLVED

Description

Similar to the issue WR2-101 about breaking the stableswap invariant, adding staking rewards to a stableswap pool where one of the assets is Ada breaks the invariant as well. It happens because such addition increases the liquidity, thus increases the new parameter `D` which should be verified and saved into the datum. Instead, it is not updated nor verified. As a result, lower parameter `D` is saved in the pool compared to reality. The main stableswap invariant is broken.

The impact is different in this issue. The whole staking rewards can be stolen in a followup swap request since the invariant is checked for the new after-swap asset values and the old pre-swap parameter `D`. The parameter `D` was not updated, hence it did not reflect the Ada addition. That means that it can be freely taken away and the invariant will hold.

Recommendation

We recommend requiring new parameter `D` to be supplied alongside an add staking rewards action, checking that the invariant holds, updating the new pool state with it and, ultimately, saving it into the pool datum.

Resolution

The issue was fixed according to our recommendation in the pull request number 945.

WR2-201 Anyone can block script fee beneficiary funds

Category	Vulnerable commit	Severity	Status
Design Issue	e1ab5f7342	MEDIUM	RESOLVED

Description

If the project or the reserve fee beneficiary is set to a script address, such as a multi-sig or DAO-controlled treasuries, there is no check for the datum of extracted funds. That means that anyone can create an extract project fee treasury request or an extract reserve fee treasury request and supply their own arbitrary compensation datum in the request. It will then become the datum of the extracted funds. An attacker can block those funds by supplying a datum that can not be parsed by the beneficiary script, a datum that is malformed, or by supplying a datum hash that does not have a known datum.

Furthermore, after the change in the pull request number 937, the protocol treasury is affected as well. An unknown datum hash can be supplied, blocking the fees.

Recommendation

We recommend keeping track of the expected output datum in the pool datum, similar to how the fee beneficiary address is handled. The expected datum can then be compared to the actual request's compensation datum. Regarding the protocol treasury, we recommend checking that the compensation datum is a fixed datum.

Resolution

The issue was fixed in the pull request number 938. Then new features were implemented and it was further worked on in the pull request number 949. Only a dummy fixed inline datum is allowed for project and reserve fee UTxOs. Further, even though any datum or a datum hash are allowed for the protocol treasury UTxOs, the datum hash is computed on-chain based on the compensation datum provided in the request datum. Hence, the datum corresponding to the datum hash is known.

WR2-301 Fee auth token holders can withdraw accumulated project and reserve fees

Category	Vulnerable commit	Severity	Status
Design Issue	`b0c2f59b6e`	MINOR	ACKNOWLEDGED

Description

Fee authorization tokens serve to identify parties that can change the fees and the beneficiary addresses where those fees can be extracted to. They are the same parties across all liquidity pools.

It is by definition that the parties should be able to update the fee amounts and addresses. However, they can update it and immediately withdraw all accumulated project and reserve fees across all pools.

The fee auth token holders are trusted and most likely will be controlled by the WingRider's team or DAO. The impact depends a lot on the frequency of emptying the accumulated project and reserve fees. The higher the potential reward, the higher the motivation.

Recommendation

We recommend requiring an extract project or reserve treasury action upon its beneficiary address change.

Resolution

The WingRiders' team clarified that this was intended. Fee auth token holders are supposed to fully own all the project and reserve fees accumulated in all pools until the point they are extracted and sent to different parties. As such, they can decide to change the addresses and withdraw the fees at any point, even the already accumulated fees.

We further recommended the team to clearly state this fact in the comments of the code as we consider this counter-intuitive from the code. The comments were added in the pull request number 947. Furthermore, there is the centralization concern that can be mitigated by frequent treasury extractions which we recommend performing. As this concern remains, the issue is left as acknowledged.

WR2-302 `pinitialStableswapPoolCorrect` does not check the value of `agentFeeAda`

Category	Vulnerable commit	Severity	Status
Code Issue	`2db11b740f`	MINOR	RESOLVED

Description

The value of `agentFeeAda` is not checked when creating new stableswap pools. This can lead to attackers creating new pools with any value for this fee, for example very high numbers or even negative numbers. We investigated the consequences of this and we believe there are 2 possible attacks in Ada stableswap pools:

1. A malicious agent could create a pool with a higher than usual fee. Such agent could then siphon the funds from users. However, the stolen amount would still be within the bounds of the user's slippage allowance.
2. DoS attack – This is more serious in a way, as anyone could front-run the pool creation and create a malformed pool with e.g. impossibly high fees, and thus block users from interacting with the pool. The fee could be set high enough so that requests would fail. The DoS can be repaired by the owner of the agent fee authority token. It would take time, though, and could damage the reputation.

Furthermore, setting the fee to a negative value increases the request's Ada value in Ada stableswap pools on paper, even though there is not that much Ada in reality. The final pool value and the compensation outputs are checked, however, so it is not easy to game this.

Recommendation

We recommend checking the `agentFeeAda` value in the `pinitialStableswapPoolCorrect` function in the same way as it's done in the constant product's function.

Resolution

The issue was fixed according to our recommendation in the pull request number 948.

WR2-303 Additional tokens may make compensation output unspendable

Category	Vulnerable commit	Severity	Status
Design Issue	`2db11b740f`	MINOR	PARTIALLY RESOLVED

Description

It is not checked that the compensation output contains only the tokens it should contain. Hence, an agent may add any number of additional tokens into it. Naturally, it needs to fit into the pool evolve transaction. It is more of a problem now, since the compensation output might be routed into any script and the script might break or exceed the execution limits if there are more tokens than expected.

Specifically, one of the intended use cases describes a direct locking of the compensation shares into a farm. One farm code branch checks that no tokens are taken from the input farm, iterating through all the tokens on the input. This might exceed the limits, requiring the user to terminate the farm and recreate it. Of course, there are more scripts that might break because of this. Other examples include project and reserve treasury scripts, as well as arbitrary user-defined scripts.

Recommendation

We recommend enforcing that no additional tokens are put into the compensation outputs, especially if the outputs are located on script addresses.

Resolution

The issue was partially resolved in the pull request number 957. The number of tokens in any compensation output is limited to 5. That should be strict enough for any reasonable attack on execution limits. However, the limit is generic and not request type specific. That means that there can still be a few unexpected tokens in the compensation output. Such an attack is dependent on agents constructing the transaction. Ultimately, even though unlikely, it might break some scripts that can not handle the unexpected additional tokens.

WR2-401 Dead code

Category	Vulnerable commit	Severity	Status
Code Style	`e1ab5f7342`	INFORMATIONAL	RESOLVED

Description

Across the codebase, there are some functions that are either duplicated or not used at all. Examples include:

- `maxTxValidityRangeSize` in both Constants and Utils.
- `pisTokenPresent` and `pisTokenSpent`.
- `pzipWith''`.
- A few functions in the `src/Plutarch/Mint/Util.hs` file, e.g. `pcheckMintingPolicyRedeemer`.
- The `RequestLocation` type in the pool types' file.
- All functions about the extended pool id in the pool types' file, s.a. `startExtendedPoolId`, `endExtendedPoolId` and their plutarch p-* equivalents.
- `DEX` and `ExchangeConfig` in the `DEX/Base/Types.hs` file. If they are not used on-chain, they do not have to be part of the on-chain codebase.
- The `type PoolValidatorHash`.
- The `type Deadline`, `isSwapFromA`, `isSwapFromB` from the request types' file.

Recommendation

We recommend removing all the mentioned unnecessary code.

Resolution

The issue was fixed according to our recommendation in the pull request number 941. All points except for the `DEX/Base/Types` structures were addressed. Those two types were non-trivial to move out and a comment was added explaining that they are used only in tests.

WR2-402 Documentation

Category	Vulnerable commit	Severity	Status
Documentation	`e1ab5f7342`	INFORMATIONAL	RESOLVED

Description

Across the codebase, there are cases where the documentation is incorrect, outdated, lacking and the function is not self-explanatory or the function has specific nuances that might not be expected and hence the documentation is welcome. Let's go to the examples:

- **Incorrect documentation**:
  - In the `pfoldl2AllowMore` function, you say that "We allow more as, but not more bs". The reality is the opposite. You error out if there is no output for a request input.
  - In the request apply docs, you say that the transaction has to run either `validatePool` or `validateEmergencyWithdrawal`. There are two more redeemers now, hence the documentation is outdated.
  - In the `poolEvolve` docs, you say that the pool datum has to remain the same. There are more fields in the datum now and it does not stay the same.
  - There is a typo `Stablaswap` instead of `Stableswap` in the docs.
- **Lacking documentation**:
  - `pvalueOfWithOilCheck` is equivalent to the `quantityOfWithOilCheck` function from `v1`, but its documentation is lost. Also, there's a change that allows for more Ada now. Please add and update the documentation.
  - `PFlatAssets` section contains no docs anymore, the type is not self-explanatory.
  - Documentation was lost in the `pexpectedNumberOfTokensInThePool` function.
- **Non-intuitive edge cases**:
  - `divideCeil` and `pdivideCeil` utility functions do not work with negative numbers as `mod a b` can be negative, add a comment about it – it was there in the `v1`.

- `p2elemsAt'` util function errors out for two indices that are the same.
- `pdrop'` does not throw if it can not drop so many items.
- `porder` goes into an endless loop when `i == last'`, i.e. when two equal indices are desired after each other.

Recommendation

We recommend correcting incorrect documentation, adding back documentation that is lacking and was useful, further documenting non self-explanatory functions or types and commenting on non-intuitive edge cases of various utility functions.

Resolution

The issue was fixed according to our recommendation in the pull request number 943.

WR2-403 Naming

Category	Vulnerable commit	Severity	Status
Code Style	`e1ab5f7342`	INFORMATIONAL	RESOLVED

Description

Naming of variables and functions is important so that the code is clear, makes sense and no mistakes are accidentally introduced. Hence we recommend improving on a few names across the codebase:

- In the treasury holder validator, a currency symbol is named `tn` and a token name is named `cs`. Here, the types were the same and it could have critical consequences. There was the same inverted mistake in the other place which mitigated it. We still recommend renaming `(tn, cs)` to `(cs, tn)`.
- `defaultMinTimeToConsiderPoolAbandoned` is hardcoded and not default. It can not be changed.
- In the `pparseRequest` function, the `requestOutput` name is used. In other places, `compensationOutput` is used to refer to the same output.
- Short local variable names do not always match the meaning or type, s.a. `ptryUniqueScriptTxInInfo –> o -> i` and `pvalueContainsToken –> vs -> v`.
- `pfix` contstructs use either `self` or `recur` to refer to the same thing.
- Both `stakingRewardsSymbol` and `stakingRewardSymbol` are used; both `shareToken` and `sharesToken` are used. It makes looking for all occurences harder.
- In the reward mint redeemer, the parameter is named `tn` but later it is referred to as pool id. Without documentation, this naming is not self-explanatory.
- `validitySymbol`. It is often supplied when shares currency symbol is required, factory token's currency symbol is required, etc. A more suitable name covering all of those could be used, s.a. `dexSymbol`, `dexCurrencySymbol`.
- `requestValidatorHashSw`. `Sw` referring to stableswap is not commonly found in the codebase, `stableswapRequestValidatorHash` hence looks more self-explanatory.
- In the factory validation, `adaIsSwapped` variable is used. However, nothing is swapped when a new pool is being created.

- In the factory validation, `expectedShares` variable is used. However, there are a lot of different variables referring to amounts of share tokens. Hence we recommend adding more information to it, s.a. `expectedMintedShares`.
- Argument `request` of `pPoolDatum` contains the request's script hash. Consider changing the name to reflect this.

Recommendation

We recommend improving on the naming in the mentioned examples.

Resolution

The issue was fixed according to our recommendation in the pull request number 942.

WR2-404 `feeInBasis` semantics and naming

Category	Vulnerable commit	Severity	Status
Code Style	`e1ab5f7342`	INFORMATIONAL	RESOLVED

Description

The semantics of the important `feeInBasis` constant changed. In the `v1` code, the `swapFeeInBasis` referred to the combined fee amount of the protocol fee and the swap fee that is left in the pool for the liquidity providers, in the number of basis points. The constant is now called just `feeInBasis` and its semantics are different. It refers to just the part that stays in the pool for the liquidity providers. In the code, it is sometimes referred to as the swapFee, `swapFeeInBasis`, or just `feeInBasis`. As there are more fees now and the handling has changed, we think that it deserves a better and consistent naming.

The new semantics is also not reflected in the comment for the `pstableswapNewReserves` function and in the documentation file, `stableswap.md`.

Recommendation

We recommend using a more explanatory and consistent naming for `feeInBasis`. Examples include `swapFeeInBasis` if the swap fee is explained to refer to only the liquidity providers' part, `lpFeeInBasis` or just `poolFeeInBasis`. Furthermore, we recommend fixing the documentation to reflect the new semantics.

Resolution

The issue was fixed according to our recommendation in the pull request number 967 – `swapFeeInBasis` naming is used.

WR2-405 Zap-in swapA is not sanitized

Category	Vulnerable commit	Severity	Status
Code Issue	e1ab5f7342	INFORMATIONAL	RESOLVED

Description

In the zap-in scenario, the liquidity is provided in a single token. A portion of it needs to be first swapped to the other token and just then can the liquidity be added. This is done behind the scenes in the zap-in. However, the on-chain code requires that the amount to be swapped, called `swapA` is provided in the redeemer. The amount is not sanitized; it is not checked to be positive, to be less than the amount of tokens supplied, etc. The number goes through a lot of equations that rely on it being reasonable afterwards. The only thing checked is that the liquidity provided in the end is balanced. It appears that it can not be broken and there is no malicious `swapA` that would not break the final check. However, it certainly is better to not rely on this and sanitize the input to the equations.

Recommendation

We recommend sanitizing the `swapA` parameter taken from the redeemer – checking that it is positive and that it is lower than the amount of assets provided.

Resolution

The issue was fixed according to our recommendation in the pull requests number 958 and 966.

A Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the agreement between VacuumLabs Bohemia s.r.o. (Vacuumlabs) and WingRiders (Client) (the Agreement), or the scope of services, and terms and conditions provided to the Client in connection with the Agreement, and shall be used only subject to and to the extent permitted by such terms and conditions. This report may not be transmitted, disclosed, referred to, modified by, or relied upon by any person for any purposes without Vacuumlabs's prior written consent.

This report is not, nor should be considered, an endorsement, approval or disapproval of any particular project, team, code, technology, asset or anything else. This report is not, nor should be considered, an indication of the economics or value of any technology, product or asset created by any team or project that contracts Vacuumlabs to perform a smart contract assessment. This report does not provide any warranty or guarantee regarding the quality or nature of the technology analysed, nor does it provide any indication of the technology's proprietors, business, business model or legal compliance.

To the fullest extent permitted by law, Vacuumlabs disclaims all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. This report is provided on an as-is, where-is, and as-available basis. Vacuumlabs does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by Client or any third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services, assets and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and Vacuumlabs will not be a party to or in any way be responsible for monitoring any transaction between you and client and/or any third-party providers of products or services.

This report should not be used in any way by anyone to make decisions around investment or involvement with any particular project, services or assets, especially not to make decisions to buy or sell any assets or products. This report provides general information and is not tailored to anyone's specific situation, its content, access, and/or usage thereof, including any associated services or materials, shall not be considered or

relied upon as any form of financial, investment, tax, legal, regulatory, or other advice.

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Vacuumlabs prepared this report as an informational exercise documenting the due diligence involved in the course of development of the Client's smart contract only, and THIS REPORT MAKES NO CLAIMS OR GUARANTEES CONCERNING THE SMART CONTRACT'S OPERATION ON DEPLOYMENT OR POST-DEPLOYMENT. This report provides no opinion or guarantee on the security of the code, smart contracts, project, the related assets or anything else at the time of deployment or post deployment. Smart contracts can be invoked by anyone on the internet and as such carry substantial risk. VACUUMLABS HAS NO DUTY TO MONITOR CLIENT'S OPERATION OF THE PROJECT AND UPDATE THE REPORT ACCORDINGLY.

THE INFORMATION CONTAINED IN THIS REPORT MAY NOT BE COMPLETE NOR INCLUSIVE OF ALL VULNERABILITIES. This report is not comprehensive in scope, it excludes a number of components critical to the correct operation of this system. You agree that your access to and/or use of, including but not limited to, any associated services, products, protocols, platforms, content, assets, and materials will be at your sole risk. On its own, it cannot be considered a sufficient assessment of the correctness of the code or any technology. This report represents an extensive assessing process intending to help Client increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology, however blockchain technology and cryptographic assets present a high level of ongoing risk, including but not limited to unknown risks and flaws.

While Vacuumlabs has conducted an analysis to the best of its ability, it is Vacuumlabs's recommendation to commission several independent audits, a public bug bounty program, as well as continuous security auditing and monitoring and/or other auditing and monitoring in line with the industry best practice. The possibility of human error in the manual review process is highly real, and Vacuumlabs recommends seeking multiple independent opinions on any claims which impact any functioning of the code, project, smart contracts, systems, technology or involvement of any funds or assets. VACUUMLABS'S POSITION IS THAT EACH COMPANY AND INDIVIDUAL ARE RESPONSIBLE FOR THEIR OWN DUE DILIGENCE AND CONTINUOUS SECURITY.

B Audited files

The files and their hashes reflect the final state at commit `32c371b5d1cacedb264db8acae09dc0413d733c3` after all the fixes have been implemented.

SHA256 hash	Filename
d8106...ccbc9	src/DEX/Constants.hs
8b212...7c2f9	src/DEX/Factory.hs
d7c71...30a65	src/DEX/Mint/Validity.hs
c6648...ceee9	src/DEX/Pool.hs
d5b9e...eabc9	src/DEX/Pool/ConstantProduct.hs
06dbe...24189	src/DEX/Pool/Stableswap.hs
1632c...2665b	src/DEX/Pool/Util.hs
ec3a3...37397	src/DEX/Request.hs
4280b...8d966	src/DEX/Staking/RewardMint.hs
f17ce...a1859	src/DEX/Treasury/Holder.hs
c371e...7d989	src/DEX/Types/Base.hs
83eb0...7a50a	src/DEX/Types/Classes.hs
32aa5...31fb4	src/DEX/Types/Factory.hs

Continued on next page

SHA256 hash	Filename
3f569...314be	src/DEX/Types/Pool.hs
ccc38...b3439	src/DEX/Types/Request.hs
516c3...23a87	src/DEX/Types/Treasury.hs
da956...1e84e	src/Other/FixedSupplyPolicy.hs
3fb0e...b84ab	src/Plutarch/Mint/Util.hs
28f99...5e3a9	src/Plutarch/PlutusScript.hs
fabec...010ca	src/Plutarch/Types/Base.hs
c2a17...6f19b	src/Plutarch/Types/Classes.hs
f364d...097c7	src/Plutarch/Util.hs
22951...d679f	src/Plutus/Util.hs

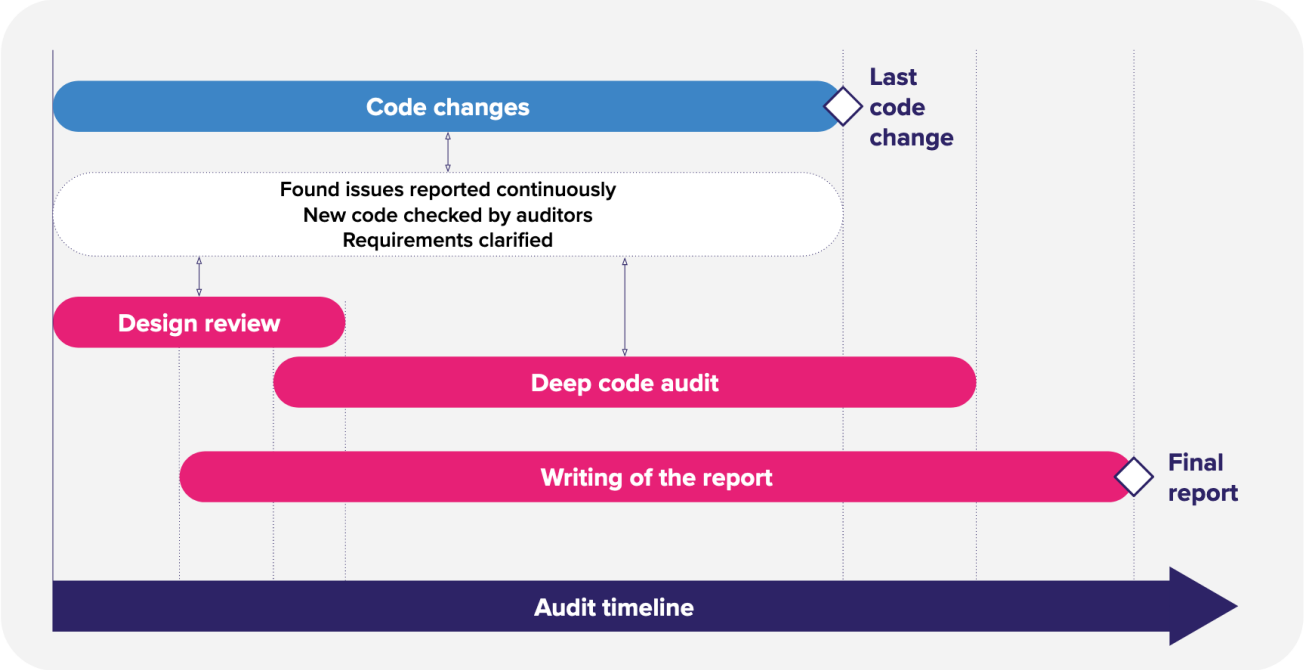
The code uses a few functions from Liqwid-Plutarch-Extra that is part of the `liqwid-libs` repository. We have audited those used in the files and revisions above as well.

Please note that we did not audit the files not included in the above list that may or may not be part of the commit hash. We also did not assess the security of Plutarch itself. The assessment builds on the assumption that Plutarch is secure and delivers what it promises.

C Methodology

Vacuumlabs' agile methodology for performing security audits consists of several key phases:

1. Design reviews form the initial stage of our audits. The goal of the design review is to find larger issues which result in large changes to the code fast.
2. During the deep code audit, we verify the correctness of the given code and scrutinize it for potential vulnerabilities. We also verify the client's fixes for all discovered vulnerabilities. We provide our clients with status reports on a continuous basis providing them a clear up-to-date status of all the issues found so far.
3. We conclude the audit by handing over a final audit report which contains descriptions and resolutions for all the identified vulnerabilities.



Throughout our entire audit process, we report issues as soon as they are found and verified. We communicate with the client for the duration of the whole audit. During our audits, we check several key properties of the code:

- Vulnerabilities in the code
- Adherence of the code to the documented business logic
- Potential issues in the design that are not vulnerabilities
- Code quality

During our manual audits, we focus on several types of attacks, including but not limited to:

1. Double satisfaction
2. Theft of funds
3. Violation of business requirements
4. Token uniqueness attacks
5. Faking timestamps
6. Locking funds indefinitely
7. Denial of service
8. Unauthorized minting
9. Loss of staking rewards

D Issue classification

Severity levels

The following table explains the different severities.

Severity	Impact
CRITICAL	Theft of user funds, permanent freezing of funds, protocol insolvency, etc.
MAJOR	Theft of unclaimed yield, permanent freezing of unclaimed yield, temporary freezing of funds, etc.
MEDIUM	Smart contract unable to operate, partial theft of funds/yield, etc.
MINOR	Contract fails to deliver promised returns, but does not lose user funds.
INFORMATIONAL	Best practices, code style, readability, documentation, etc.

Resolution status

The following table explains the different resolution statuses.

Resolution status	Description
RESOLVED	Fix applied.
PARTIALLY RESOLVED	Fix applied partially.
ACKNOWLEDGED	Acknowledged by the project to be fixed later or out of scope.
PENDING	Still waiting for a fix or an official response.

Categories of issues

The following table explains the different categories of issues.

Category	Description
Design Issue	High-level issues in the design. Often large in scope, requiring changes to the design or massive code changes to fix.
Logical Issue	Medium-sized issues, often in between the design and the implementation. The changes required in the design should be small-scaled (e.g. clarifying details), but they can affect the code significantly.
Code Issue	Small in size, fixable solely through the implementation. This category covers all sorts of bugs, deviations from specification, etc.
Code Style	Parts of the code that work properly but are possible sources of later issues (e.g. inconsistent naming, dead code).
Documentation	Small issues that relate to any part of the documentation (design specification, code documentation, or other audited documents). This category does not cover faulty design.
Optimization	Ideas on how to increase performance or decrease costs.

E Report revisions

This appendix contains the changelog of this report. Please note that the versions of the reports used here do not correspond with the audited application versions.

v1.0: Main audit

Revision date: 2024-09-04
Final commit: `32c371b5d1cacedb264db8acae09dc0413d733c3`

We conducted the audit of the main application. To see the files audited, see Executive Summary.

Full report for this revision can be found at url.

F About us

Vacuumlabs has been building crypto projects since the early days.

- We are behind the popular AdaLite wallet. It was later improved into a multichain wallet NuFi.
- We built the Cardano applications for the hardware wallets Ledger and Trezor.
- We built the first version of the cutting-edge decentralized NFT marketplace Jam On Bread on Cardano with truly unique features and superior speed of both the interface and transactions.

Our auditing team is chosen from the best.

- Talent from esteemed Cardano projects.
- Rich experience across Google, traditional finance, trading and ethical hacking.
- Award-winning programmers from ACM ICPC, TopCoder and International Olympiad in Informatics.
- Driven by passion for program correctness, security, game theory and the blockchain technology.

Contact us:
audit@vacuumlabs.com