vacuumlabs

Liqwid v1

Audit Report, v1.0

February 13, 2023

Contents

Revision table

Report version	Report name	Date	Report URL
1.0	Main audit	2023-02-13	Full report link

1 User 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.

This section contains the most important information that the user of the protocol should be aware of. For more in-depth discussion, read the Executive Summary and the Findings.

Short project overview

The Liqwid v1 project allows users to lend and borrow liquidity from pools without the requirement to find a peer. The basic protocol unit is a ***market***, where users can ***lend*** and ***borrow*** the ***underlying tokens***. As an example, Ada is deposited to and borrowed from an Ada market. Note that there can possibly be more markets with the same underlying token but with different configuration parameters. The most important parameters include the interest model, the compounding period, liquidation criteria, collateral weights.

The lenders deposit the market's underlying tokens and gain ***qTokens*** in return. They can later redeem the qTokens back for the underlying tokens, while earning interest, or sell them to someone else. This process is not risk-free, however some mitigations are implemented: each lending pool has a so called ***reserve***, which cannot be borrowed from and it is only used for redemption requests when the market does not have enough liquidity. The reserve is replenished only from the paid interest, according to the ***income ratio*** parameter.

The borrower has to deposit ***collateral*** when they want to borrow underlying tokens from a market. The value of the collateral is determined by an oracle, and if the collateral's price fluctuates, the users can lose this collateral (or a fraction of it) in a liquidation process. Users need to be careful about what tokens they use as collateral.

Short findings summary

The more critical issues we found were promptly fixed. The users should, however, be aware of issues that were acknowledged and those that are either hard to fix fully or are an intended part of the application design:

- **Centralization**: LIQV1-006, LIQV1-208, LIQV1-209.
  The markets have several essential properties changeable by a multi-signature scheme in which a few other companies operating on Cardano participate. The security of this solution depends on the correct implementation and the trusted members of the scheme. We recommend watching the reputation of the scheme members and also watching for updates. It also depends on the DAO votes and the distribution of the LQ governance token. Any update can pose an existential threat and should be thoroughly reviewed.

  Some parts of the application such as staking rewards and updating in-house oracles depend even on a single key. The oracles especially are a very important aspect of the security.

- **Oracle manipulation attacks**: LIQV1-203.
  An oracle is an essential part of any lending protocol. As a result, the protocol might be attackable by oracle manipulation attacks[1]. These attacks are generally not caused by a bug in smart contracts – they are economical attacks. The attacker can cause a price of a token to decrease by selling huge amounts of it, and then use the decreased price to perform some actions on the decentralized application. In this application, the attack vector was acknowledged and the team plans to take the volatility of assets into account when setting the collateral parameters. There is the *collateral weight* parameter which can make assets count as less collateral than they are actually valued, which could be used for volatile assets.

- **Possible availability issues**: LIQV1-205, LIQV1-207, LIQV1-303, LIQV1-304.
  In the Cardano eUTxO model, when multiple users try to spend the same UTxO, only one can succeed. In Liqwid, users interact with parts of the market directly. As a result, users need to be aware that during high activity spikes (or malicious denial of service attacks) it might be hard to interact with the application. See issue LIQV1-205 for more details. Similarly, issues LIQV1-207, LIQV1-303 and LIQV1-304 are about an attack that can slow down or block the application or cause unavailability of liquidity.

- **Code quality issues**: LIQV1-401 and others numbered 402 and up.
  There are multiple issues regarding naming, e.g. LIQV1-401 and LIQV1-403. These are not making the protocol vulnerable on their own, but they make the code harder

[1] https://coinmarketcap.com/alexandria/glossary/oracle-manipulation

to read, harder to audit and easier to make mistakes when changing a few lines of logic. Furthermore, the tests did not cover many of the possible market situations, e.g. they focused mostly on markets containing Ada and the non-Ada logic was tested sparingly.

2 Executive summary

Project overview

The Liqwid v1 project allows users to lend and borrow liquidity from pools without the requirement to find a peer. The basic protocol unit is a *market*, where users can *lend* and *borrow* the *underlying tokens*. As an example, Ada is deposited to and borrowed from an Ada market. Note that there can possibly be more markets with the same underlying token but with different configuration parameters.

When a lender deposits underlying tokens into a market, they get *qTokens* in return which represent the deposited tokens in a market. They can later redeem the qTokens back for the underlying tokens or sell them to someone else. The exchange rate from qTokens to the underlying tokens should increase with time. As users take on more loans, each qToken is gradually worth more underlying tokens as each loan accrues *interest*.

The borrower has to deposit *collateral* when they want to borrow underlying tokens from a market. The total value of the collateral has to be higher than the value of the borrowed asset. The value is calculated using rates from a simplified in-house *oracle*. Multiple white-listed asset classes can be deposited as collateral at the same time, but each asset class has its own parameters, in particular there is the *collateral weight* ratio used to multiply the value of the collateral. For example, for a very volatile asset with a collateral weight of 0.5, depositing $\$1000$ worth of this asset is only counted as $\$500$ worth of collateral. Note that other users can perform certain actions with a user's loan. They can repay a part of the loan or deposit more collateral. Naturally, they can not borrow more or withdraw collateral.

When the value of the borrowed asset compared to the collateral reaches an unhealthy ratio, the liquidators are able to *liquidate* the loan. There are two thresholds for the "healthiness" of a loan, the less healthy the loan the bigger fraction of the loan a liquidator can liquidate.

The underlying tokens deposited into a market are divided into the *supply*, *principal* and *reserve*. The supply can be borrowed by borrowers and once an amount is borrowed, it leaves the market and goes into the borrower's wallet. The borrowed amount is added to the market's principal and subtracted from the supply. Finally, when the supply can not cover a redemption request of a lender, they can use funds from the reserve to cover the requested amount of underlying tokens.

Some of the paid interest goes to the treasury, the dividends and the reserve, while the rest is paid back to the lenders in the form of an increased qToken exchange rate. For example, if the *interest rate* is a constant 10% per annum and 20% of the paid interest is

allocated to the treasury, the dividends and the reserve; the total interest yield for lenders is 8% per annum, as the outstanding 2% is allocated to the treasury, the dividends and the reserve. Apart from a DAO vote, the interest payments are the only way to increase the reserve.

The interest rate is dynamically changing, possibly multiple times a day, based on the current market utilization. If there are no outstanding loans in a market, the utilization is 0. If there is no supply left in the market, because it has been fully lent out, the utilization is 1. The utilization is used in the *interest model* formula to calculate the current interest rate.

The liquidity is split among a constant number of *Action UTxOs*. The users interact with them directly with no intermediary. The interest rate is updated during the *batching* process, when deposits and loans that happened since the last batching on single Action UTxOs are aggregated and synced together. The *market state* is updated in the final batching transaction marking the end of every batching process. In between the final batching transactions, the interest rate is out of sync with the actual state of the market. However, this design allows for a better throughput compared to a single UTxO holding the whole liquidity with no intermediary.

The most important parameters of the protocol are governed by a *decentralized autonomous organization* (DAO). The owners of the LQ token can vote (or delegate their votes and thus vote through a delegate) about changes in the protocol settings. The most important parameters include: the interest model, the compounding period, the number of action UTxOs, liquidation criteria, scripts used for treasury and dividends, etc. Any such update can pose an existential threat and should be thoroughly audited.

In case of an emergency, the same parameters can be updated via a multi-signature scheme. The initial participants of the scheme have been mentioned in a blog post but we recommend watching for updates and the reputation of the scheme members.

Glossary

- **Interest Model**: The interest model describes the function used to calculate the interest rate. If the utilization is high, the interest is higher too and vice versa.
- **Income Ratio**: Determines the proportions in which the paid interest is divided among the treasury, the supply, the dividends and the reserve.
- **Action UTxO**: Users interact with markets via Action UTxOs that each hold a portion of the whole market liquidity. It's marked by the Action Token (AT).
- **Loan UTxO**: Contains the collateral and information about the loan: the owner, principal, outstanding interest, etc. It's marked by a Borrow Token (BT).

- **Batching**: A regular market process that synchronizes the Action UTxOs, redistributes the liquidity and updates the market state. The process can potentially consist of multiple transactions that can be created and submitted by anyone. The last transaction of a single batching process is called the *Final batch* transaction.
- **Loan to Value**: The ratio between the total owed amount of a loan (i.e. the principal and interest) and the weighted value of the collateral.

Methodology

The first phase of our audit collaboration was a design review. We reviewed the high-level design and suggested improvements. The most notable improvement is the usage of an emergency multi-signature scheme instead of a single private key, see LIQV1-006.

We started the design review phase at commit `66e83f30236404fa6ed469ecddee199ab0af9859` .

After the design review, we conducted a deeper manual audit of the code and reported findings along with suggestions to the team in a continuous fashion, allowing the time for a proper remediation that we reviewed afterwards. We also supplied test cases demonstrating the vulnerabilities on multiple occasions.

Our manual process focused on several types of attacks, including but not limited to:

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

The audit lasted from 23 November 2022 to 11 February 2023. We interacted mostly on Slack and gave feedback in GitHub pull requests. The team fixed or mitigated all major and critical issues. Many of the rest were acknowledged.

Files audited

The files and their hashes reflect the final state at commit `62b1e463b80a7a7e0dca52b64c13c311220c8548` after all the fixes have been implemented.

SHA256 hash	Filename
a6b45...ac5be	liqwid-onchain/Liqwid/Market.hs
d76e8...177bd	liqwid-onchain/Liqwid/Market/Action.hs
4f197...d2356	liqwid-onchain/Liqwid/Market/Batch.hs
34da0...099a5	liqwid-onchain/Liqwid/Market/Loan.hs
c91f5...6d187	liqwid-onchain/Liqwid/Market/Params.hs
f35bb...9c89e	liqwid-onchain/Liqwid/Market/Scripts/Action.hs
650bb...75103	liqwid-onchain/Liqwid/Market/Scripts/ActionToken.hs
ddd17...df970	liqwid-onchain/Liqwid/Market/Scripts/Batch.hs
9f694...cc669	liqwid-onchain/Liqwid/Market/Scripts/BatchFinal.hs
dd900...fc15c	liqwid-onchain/Liqwid/Market/Scripts/BorrowToken.hs
b9d80...10e1c	liqwid-onchain/Liqwid/Market/Scripts/Helpers.hs
f66be...21abe	liqwid-onchain/Liqwid/Market/Scripts/Liquidation.hs
d365a...38756	liqwid-onchain/Liqwid/Market/Scripts/Loan.hs
00f88...c8c4c	liqwid-onchain/Liqwid/Market/Scripts/MarketState.hs

Continued on next page

a4f39...40ca1	liqwid-onchain/Liqwid/Market/Scripts/QToken.hs
098e6...940f2	liqwid-onchain/Liqwid/Market/Scripts/Staking.hs
e0001...cf7a5	liqwid-onchain/Liqwid/Models/Authorization.hs
ac319...1d6de	liqwid-onchain/Liqwid/Models/InterestIndex.hs
30af7...e94a7	liqwid-onchain/Liqwid/Models/InterestRate.hs
52fdd...39307	liqwid-onchain/Liqwid/Models/Loan.hs
2a75b...f3021	liqwid-onchain/Liqwid/Models/MarketState.hs
d120a...c1de1	liqwid-onchain/Liqwid/Models/QTokenRate.hs
b1a1c...6685f	liqwid-onchain/Liqwid/Optics.hs
f6c89...3a79d	liqwid-onchain/Liqwid/Plutarch.hs
d6441...3f5b3	liqwid-onchain/Liqwid/ScriptWrappers.hs
5d01e...0aefe	liqwid-onchain/Liqwid/Timing.hs
c96bb...802e9	liqwid-onchain/Liqwid/Units.hs
4f36d...0552c	liqwid-onchain/Liqwid/Units/IncomeRatio.hs
065df...019bf	liqwid-onchain/Liqwid/Units/LQ.hs
65599...bf7e3	liqwid-onchain/Liqwid/Units/Quantity.hs
38d53...8173e	liqwid-onchain/Liqwid/Utils.hs
ddfec...311f5	liqwid-onchain/PPrelude.hs

The code uses multiple functions from Liqwid-Plutarch-Extra package. We have audited functions used in the files listed above. The code also uses functions from the Agora-Pro repository which were part of a previous audit and also some added later after the audit. We audited the relevant additions to the Agora-Pro repository and the modified files with their hashes are listed below. The final commit of Agora-Pro was `ba24c6e1bad804b654eb8bccbb4769ad53bcc65c` .

SHA256 hash	Filename
`0e548...5c8ba`	`agora-pro/Agora/Pro/AuthorityToken/Check.hs`
`ea8b8...d572e`	`agora-pro/Agora/Pro/AuthorityToken/StateThread.hs`
`5acde...1cb77`	`agora-pro/Agora/Pro/Bootstrap.hs`
`7ecfd...becfd`	`agora-pro/Agora/Pro/MultiSig.hs`
`3b918...8b90a`	`agora-pro/Agora/Pro/MultiSig/Check.hs`
`1b612...d6976`	`agora-pro/Agora/Pro/MultiSig/Scripts.hs`
`37f87...f7e2c`	`agora-pro/Agora/Pro/MultiSig/StateThread.hs`
`af1dc...1a052`	`agora-pro/Agora/Pro/MultiSig/Utils.hs`
`e2168...74371`	`agora-pro/Agora/Pro/Utils.hs`

Please note that we did not audit the files not listed above that are a part of the commit hash. We did not review the protocol parameters. 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.

Market creation

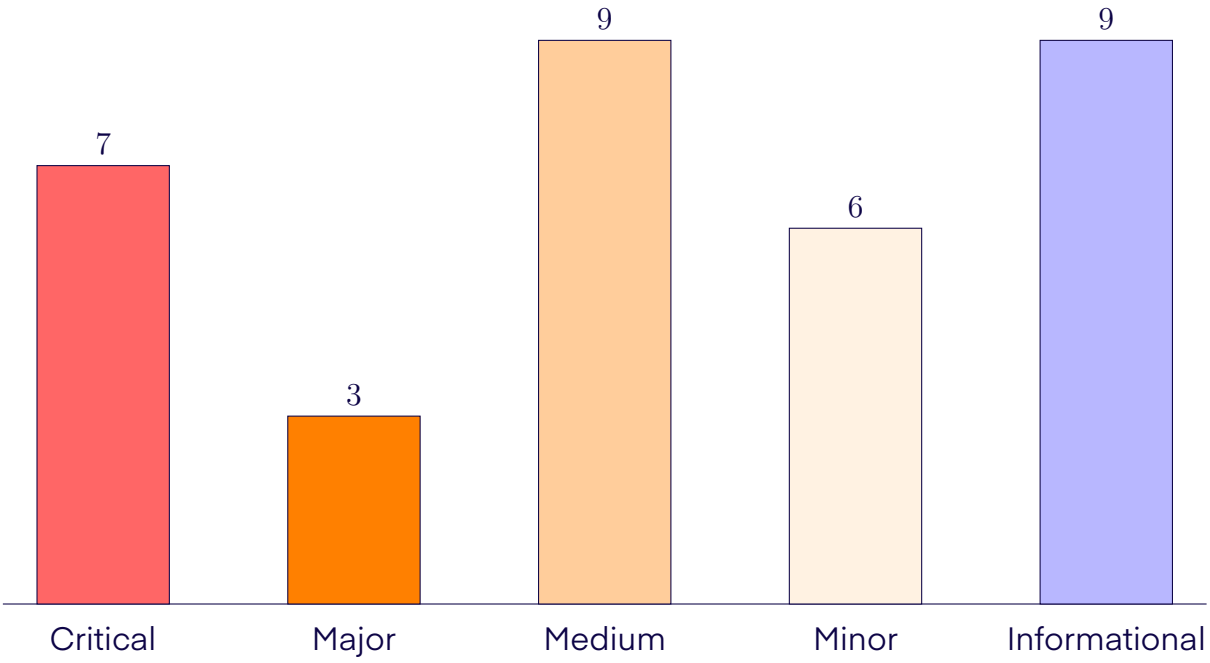
Note that we did not audit the Agora effect that is intended to be used for creating new markets. Also, some initial markets will be created by the Liqwid team and not by the Agora effect. One of the most important parameters of the protocol is the validator script that is used for the market parameters and collateral parameters. According to the team,

they will initially use the `freeMultiSigScriptValidator` function from the Agora-Pro repository. This validator only passes when the transaction is signed by enough members of the multi-signature scheme or if a voted-on (and passed) Agora effect wants to spend it. It means that the important market and collateral parameters can only be changed by a DAO vote or by the multi-signature scheme in case of emergencies. In fact, the choice of the script (initially `freeMultiSigScriptValidator`) can be overridden by a DAO vote.

Note that the UTxOs with the market and collateral parameters are marked by *market parameter tokens* and *collateral tokens*. These are unique tokens that ensure that there is exactly one UTxO with the market parameters and that there are collateral parameters only for the allowed collateral assets.

vacuumlabs

3 Severity overview



Findings

ID	TITLE	SEVERITY	STATUS
LIQV1-001	Retrieving the collateral without repaying allows for draining a market	CRITICAL	RESOLVED
LIQV1-002	Supply can be stolen while batching	CRITICAL	RESOLVED
LIQV1-003	Loan collateral can be stolen by overwriting the loan datum	CRITICAL	RESOLVED
LIQV1-004	Minting additional borrow tokens allows draining markets	CRITICAL	RESOLVED
LIQV1-005	Division by zero—collateral can be locked forever	CRITICAL	RESOLVED
LIQV1-006	Centralization	CRITICAL	PARTIALLY RESOLVED

Continued on next page

LIQV1-007	Market can be drained by tricking the `checkQTokenRate`	CRITICAL	RESOLVED
LIQV1-101	Incorrect reserve calculation can require liquidity deposit for batching	MAJOR	RESOLVED
LIQV1-102	A quick loan is without interest—staking rewards can be stolen	MAJOR	RESOLVED
LIQV1-103	Staking rewards can be withdrawn while redeeming liquidity	MAJOR	RESOLVED
LIQV1-201	Interest is accrued even after a loan is paid back	MEDIUM	RESOLVED
LIQV1-202	Negative interest is outstanding after a loan is paid back	MEDIUM	RESOLVED
LIQV1-203	Oracle exchange rate is a simple number	MEDIUM	ACKNOWLEDGED
LIQV1-204	Reserve only decreases	MEDIUM	RESOLVED
LIQV1-205	Attacker can block Action UTxOs	MEDIUM	ACKNOWLEDGED
LIQV1-206	Loan interest rounded down leads to an accumulated interest	MEDIUM	RESOLVED
LIQV1-207	Attacker can block the batching process – `minBatchSize` requirement is not enforced	MEDIUM	ACKNOWLEDGED
LIQV1-208	Staking delegation is controlled by a single key	MEDIUM	ACKNOWLEDGED
LIQV1-209	Oracle is controlled by a single key	MEDIUM	ACKNOWLEDGED
LIQV1-301	The `maxLoan` threshold can be exceeded	MINOR	RESOLVED
LIQV1-302	`compoundingPeriod` should be considered part of the interest model	MINOR	ACKNOWLEDGED
LIQV1-303	Attacker can slow down the batching process	MINOR	ACKNOWLEDGED
LIQV1-304	Attacker can hide liquidity during the batching process	MINOR	ACKNOWLEDGED

Continued on next page

vacuumlabs

LIQV1-305	Close factor below 100% causes never-ending liquidations	MINOR	RESOLVED
LIQV1-306	Batching can require batchers to pay min-ADA	MINOR	ACKNOWLEDGED
LIQV1-401	Variable name "supply" in some functions includes the reserve as well	INFORMATIONAL	ACKNOWLEDGED
LIQV1-402	Functions `checkBatchTime` appear twice with different semantics	INFORMATIONAL	ACKNOWLEDGED
LIQV1-403	Variable `liquidationDiscount` has a misleading name	INFORMATIONAL	ACKNOWLEDGED
LIQV1-404	Variable `maxBatchTime` has confusing name	INFORMATIONAL	ACKNOWLEDGED
LIQV1-405	Functions in `Helpers.hs` are not documented properly	INFORMATIONAL	ACKNOWLEDGED
LIQV1-406	The use of a `-Interest` suffix leads to unclear variable names	INFORMATIONAL	ACKNOWLEDGED
LIQV1-407	The variable `supplyDiff` has multiple meanings	INFORMATIONAL	ACKNOWLEDGED
LIQV1-408	Reserve field in Action datum may be imprecise	INFORMATIONAL	ACKNOWLEDGED
LIQV1-409	Other code best practices	INFORMATIONAL	ACKNOWLEDGED

LIQV1-001 Retrieving the collateral without repaying allows for draining a market

Category	Vulnerable commit	Severity	Status
Bug	`ee474e0b02`	CRITICAL	RESOLVED

Description

The loan UTxO delegates the validation to the borrow token minting policy, but the policy does not check the loan validator hash in the output. An attacker A can construct the following transaction (a `ModifyLoan` action):

- Spend a loan UTxO u which has been created by A beforehand. Burn its borrow token.
- Mint a new borrow token, put it into a fake script f (e.g. always accepting). Copy the datum from u and move most (99%) of the collateral from u to f, too. Move a tiny bit of collateral (1%) to A's wallet. In other words, do the action the proper way and just change the address to f.

As there is no check on the output validator hash, the attacker managed to borrow money but they got their collateral back in the following transaction (without repaying the debt). As a result, there is no existing record of their loan and they do not have to pay it back.

The maximum value stolen at once is limited by the value of the attacker's collateral, but they can repeat this many times and drain the markets completely.

Recommendation

Add a loan output address check to the `checkLoanOutput` function (the presence of the check is actually mentioned in the function description in a comment above it).

Resolution

Fixed in the pull request number 303.

LIQV1-002 Supply can be stolen while batching

Category	Vulnerable commit	Severity	Status
Bug	`d681e5648f`	CRITICAL	RESOLVED

Description

When a market is not Ada, the `pcheckActionValue` function only checks for `minAda` underlying tokens to be present in the action outputs. There's a typo in the definition of `expectedUnderlying`, which checks for `minAda` underlying tokens instead of the appropriate number of underlying tokens (the `underlying` variable in the code) in the default branch.

An attacker can execute the final batching transaction but only supply `minAda` underlying tokens in each action output and steal the rest.

Recommendation

Use `otherwise = underlying` in the function `pcheckActionValue` for the definition of `expectedUnderlying`. Also, update tests to reflect this.

Resolution

Fixed in the pull request number 314.

LIQV1-003 Loan collateral can be stolen by overwriting the loan datum

Category	Vulnerable commit	Severity	Status
Bug	`d681e5648f`	CRITICAL	RESOLVED

Description

The function `checkLoanOutput` does not check the loan output datum if the value of `modifying` is true, e.g. when there are no action UTxOs in the transaction. The attacker can do the following to steal the supply and reset his loan:

- Borrow from the market and create a loan UTxO in one transaction.
- In the following transaction:
  - Spend the loan UTxO.
  - Do not include any action UTxO, so that `actionChange` $= 0$ and `modifying` is true.
  - Write `principal` $= 0$ and `interest` $= 0$ into the output loan datum of the Loan UTxO u.
  - Move the whole collateral to u.
- In the last transaction, pay the loan u back. As `principal` $+$ `interest` is equal to zero, it means that the attacker can collect the collateral without having to repay the original loan.

Repeated many times over, the attacker is able to drain the market by repeatedly borrowing and unlocking the collateral.

Recommendation

Always check for loan output datum values, even when there's no change in the amount of underlying tokens (`validLoanDatum` should not depend on the `modifying` flag).

Resolution

Fixed in the pull request number 314.

LIQV1-004 Minting additional borrow tokens allows draining markets

Category	Vulnerable commit	Severity	Status
Bug	`a1a8071347`	CRITICAL	RESOLVED

Description

An attacker can mint additional borrow tokens which allow him to drain a market. He achieves it by the following transaction sequence:

1. Create a valid (small) loan UTxO u.
2. Pay it all back in the next transaction, creating a new loan UTxO v with `principal` $= 0$ and `interest` $= 0$.
3. In the next transaction, "pay back" v:
   - Spend the loan v.
   - Add an output UTxO w on a fake script address f (e.g. always accepting). Mint two new borrow tokens there.
   - Action UTxO does not have to be present as there is no principal and interest left to be paid.
   - Collateral from v can go anywhere.
4. In the following transaction, spend w and create a loan UTxO x. Set negative principal in x, e.g. $-10\,000$ Ada. Put only one of the borrow tokens from w into x. Loan UTxO x now looks as a valid loan UTxO, but there is negative principal written in its datum.
5. In the final transaction, "pay back" x. As the principal is negative, paying back x means getting the money from the Action UTxO.

The sequence can be repeated until the market is drained completely.

The crucial transaction number 3 passes, because the validators ignore UTxOs with 2 borrow tokens. Moreover, the function `countOwnTokens` calculates that there was 1 token minted, as $-1 + 2 = 1$, which is a valid value (although it should not be valid in this case).

Recommendations

In the function `findLoanAndActionInfo`, do not allow inputs or outputs with 2 or more borrow tokens.

Additionally, add a check for the number of borrow tokens in the input and output in the `handleModifyAction` function.

Finally, you could also check for the datum validity. In this example, it would help to protect against the negative principal value in the loan datum. It is not sufficient in itself, but rather a good additional measure.

Resolution

Fixed in the pull request number 338.

LIQV1-005 Division by zero—collateral can be locked forever

Category	Vulnerable commit	Severity	Status
Bug	`9625f05a90`	CRITICAL	RESOLVED

Description

It is possible to repay a loan but keep the loan UTxO with `principal` $= 0$ and `interest` $= 0$, since the `checkLoanOutputAndAuth` function does not enforce that the loan output is missing.

Once the loan UTxO has principal and interest equal to 0, any following spend of that UTxO results in a division by zero (in the `actualInterestIndex` function), a validation failure and thus the remaining collateral stays locked forever.

There are two situations in which this is dangerous:

- A user pays back their outstanding loans while keeping the collateral in the loan UTxO for later usage (e.g. for taking another loan in the future). They are locked out of such collateral, though.
- An attacker pays back someone else's loan but keeps the collateral in the loan UTxO. Again, the attacker has locked the collateral forever. The original owner is unable to access their collateral.

Recommendation

We suggest to either:

1. Make the state which causes the division by zero unreachable. This can be achieved by enforcing that the output loan UTxO is missing once the loan is repaid in full.
2. Make sure that the formula itself is robust against the division by zero.

This involves a business decision, as it might be convenient for users to keep the collateral in the loan UTxO, ready to be used for taking the next loan.

Resolution

Fixed in the pull request number 335 by following our second recommendation.

LIQV1-006 Centralization

Category	Vulnerable commit	Severity	Status
Design issue	`0b1f7fa162`	CRITICAL	PARTIALLY RESOLVED

Description

Some of the most important scripts such as the `MarketState`, the `qToken` policy and others depend on a single key pair whose owner is able to bypass the whole validation logic. This represents a single point of failure in the system.

Centralization is one of the most common issues within the crypto space, often resulting in massive losses. The users of the protocol need to trust that the private key is correctly secured and the owner of the private key is trusted which is hard to achieve in reality. A compromised key could have unforeseen consequences for the application, including draining the whole liquidity (e.g. by minting qTokens, manipulating the exchange rates, denying access to the protocol funds, etc.).

Recommendation

We understand that in the early stages of development the team wants to have a way of updating the market parameters. However, it is important to preserve a healthy balance so that the community can trust the protocol. Our recommendations are therefore to:

- Communicate this issue openly and provide a clear path to full decentralization for the community putting their money into the application.
- Limit the privileged owner role to the bare needed minimum instead of yielding the whole validation to a simple signature. Allow only for the necessary updates.
- Switch from a single key pair to a community-updatable multi-signature script.

Resolution

The single key has been replaced by a multi-signature scheme. The code has been updated in the pull request numbers 341 and 347.

The initial participants of the scheme have been mentioned in a blog post[2]. The security of this solution depends on the trusted members of the scheme. **It is important to**

[2] https://liqwidfinance.substack.com/p/liqwid-mainnet-launch-checklist

note that combined, they represent a backdoor to the whole protocol. Our second recommendation, limiting the possible downside, has not been addressed.

See other issues related to centralization:

- LIQV1-208 Staking rewards centralization
- LIQV1-209 Oracle centralization

LIQV1-007 Market can be drained by tricking the `checkQTokenRate`

Category	Vulnerable commit	Severity	Status
Bug	`25f8cad62d`	CRITICAL	RESOLVED

Description

The `checkQTokenRate` function takes the values `deltaUnderlying` (the change in liquidity in the transaction) and `deltaQTokens` (the number of minted or burned qTokens), and checks if an action happens in the correct ratio given by the `qTokenRate` saved in the market state. This function is supposed to work both with actions that redeem qTokens and that supply underlying tokens. These actions can be distinguished by the signs (plus or minus) of the `delta-` variables. Based on the sign of the `deltaUnderlying`, the correct part of the inequality is checked (when redeeming, an upper bound on the number of underlying tokens is checked; when supplying, a lower bound on the number of underlying tokens is checked - given the number of `deltaQTokens`).

An attacker can trick this by constructing a transaction that tries to do something unexpected: He can attempt to redeem the whole value of the Action UTxO (`deltaUnderlying` is negative) while minting qTokens (`deltaQTokens` is positive). This would pass the validation, as the inequality checked would translate to comparing that a negative number `inputRate` is less than or equal to a positive number. That is always true and there are no more checks on the amounts.

Recommendation

Make the `checkQTokenRate` function more robust by allowing only two scenarios: a valid redeem and a valid supply. Test this function also on the negative delta numbers and both the positive and negative combinations.

Note that currently, if the `deltaUnderlying` is zero, the function would also go to the wrong part of the equation. This is not possible to use in a full attack as it is stopped by the `minValue` check. However, we still believe that this crucial function should stop such an attempt as well and consider the stop by the `minValue` check a chance.

Resolution

The Liqwid team resolved this issue in two different ways, in pull requests 356 (reviewed commit hash `0a44238e17`) and 357 (reviewed commit hash `a00c39b1e7`) respectively. As the protocol was unfortunately already deployed on mainnet at the time this issue was found, a less invasive hotfix was used to patch the markets that were already operational. The proper fix (as suggested) is in the qToken policy. Fixing the policy in an existing market would mean changing the qToken asset class and thus it would be necessary to redistribute qTokens to all the users that already hold the old qTokens.

The team plans to use the proper fix for any new markets launched in the future. We have reviewed both fixes in the mentioned pull requests.

We have also reviewed an off-chain script verifying that the critical vulnerability was not exploited in the wild and to the best of our knowledge confirm the fact. The script uses data from Blockfrost.

LIQV1-101 Incorrect reserve calculation can require liquidity deposit for batching

Category	Vulnerable commit	Severity	Status
Bug	d681e5648f	MAJOR	RESOLVED

Description

The function `updateMarketState` has an incorrect calculation of the `toMarket` value. In case of an underflow, the expected reserve is not decreased according to the underflow. The attacker can do the following:

- Deposit liquidity into a market.
- Wait until most liquidity is lent out, so the supply of a market is close to 0.
- Redeem deposited liquidity. As the supply is close to 0, the reserve is used.
- Suppose we have `supply'` $= -200$ and `reserve` $= 1000$ in `updateMarketState`, then `reserveNew` $= 800$ and `supplyNew` $= 0$. This corresponds to 800 tokens that are left in the market as the reserve after the supply has been drained. However, `toMarket` is still 1000, which is the value that's expected to appear across the action outputs after the final batch.
- A user A doing the final batching has to provide the missing liquidity of 200 underlying tokens, so that there are 1000 tokens across the action outputs in total.

A is not compensated with qTokens for providing the liquidity, so there is a strong negative incentive to do the final batching. It means that the final batching will likely be significantly delayed. If the liquidity deposit requirement becomes too high, the batching might never happen.

Recommendation

Use `reserveNew` $+$ `supplyNew` to calculate the value of `toMarket`.

Resolution

Fixed in the pull request number 314.

LIQV1-102 A quick loan is without interest—staking rewards can be stolen

Category	Vulnerable commit	Severity	Status
Design issue	`0b1f7fa162`	MAJOR	RESOLVED

Description

If a borrower borrows money just after the batch final transaction and returns them before the `minBatchTime` is over, they will not pay any interest. This means that users can borrow money for the period of `minBatchTime` for 0 interest. In particular, looking at Ada market, an attacker can borrow as much Ada as possible just before the start of an epoch to earn staking rewards for the borrowed Ada, as staking rewards are calculated based on the amount of Ada at the start of the epoch. This has the following implications:

- The attacker gets staking rewards (*yield*) on an amount for which they did not pay interest on. As an example, an attacker owning 2 million worth of stablecoins can deposit those as collateral, borrow 1 million worth of Ada at the end of each epoch and earn staking rewards as if they really owned the Ada. They only have a short-term exposure towards the Ada price.
- The lenders depositing Ada into the market do not get the staking rewards as those are captured by the attacker. They also don't get any interest paid to them as the attacker does not have to pay any.

Recommendation

Either devise a "minimum interest" mechanism or round the interest up to the nearest compounding period whenever a loan is repaid.

Also, be especially cautious of Ada market where the staking rewards could be "stolen" by a borrower. Make sure that the interest paid in this case is at least that amount.

Resolution

Fixed in the pull request number 344.

LIQV1-103 Staking rewards can be withdrawn while redeeming liquidity

Category	Vulnerable commit	Severity	Status
Bug	`25f8cad62d`	MAJOR	RESOLVED

Description

The staking validator allows for withdrawing the rewards if there is exactly 1 qToken minted. The qToken policy then checks that if there is a value supplied to an Action UTxO, then the value is at least the amount withdrawn.

An attacker, partially making use of the faulty `checkQTokenRate` function as per the issue LIQV1-007, can withdraw his supply, mint the one qToken required and keep the staking rewards for himself as a bonus. The `checkQTokenRate` would pass (as per the other issue) and the `validWithdrawal` check would pass as well, as withdrawing supply would make it bypass the check altogether.

Recommendation

Remove the if-then-else branching in the `validWithdrawal` check. If there are ADA rewards withdrawn in the transaction, the amount supplied to the market should be at least that amount. This decouples the check from checking the proper `qTokenRate`. However, applying just the suggestions from the issue LIQV1-007 would invalidate this attack vector.

Resolution

Fixed in the pull request numbers 356 and 357. The same note about both the proper fix and the indirect hotfix mentioned in the LIQV1-007's resolution applies. Both are intended to be used on the mainnet.

LIQV1-201 Interest is accrued even after a loan is paid back

Category	Vulnerable commit	Severity	Status
Bug	`67d7c231b0`	MEDIUM	RESOLVED

Description

There is a bug in the interest calculation, causing a discrepancy between the loan UTxO and the market state.

- Create a loan of 1000 Ada.
- After some time, pay back the whole loan of $1000+10$ Ada, where 10 Ada is the interest.
- During the following batching, the market state will be updated. The paid interest of 10 Ada and the principal of 1000 Ada will be subtracted as it's part of the `-Diff` Action fields.
- However, the `totalOwed` in `updateMarketState` still contains the 1010 Ada that has been paid back. The interest is accrued for the whole time up to the batching (as opposed to when the loan was repaid). This results in a tiny discrepancy and an outstanding interest in the market state.
- This incorrectly calculated interest is a very small amount, because it is only accrued since the last batching, but over time, it will grow. E.g. if the time between the batches is 20 minutes, it will be up to the 20-minute interest for 1010 Ada. And it accumulates for every loan.

In particular, in a market with a single loan, there remains an outstanding interest even after the loan has been repaid. The value of `totalOwed` in `updateMarketState` should be 0 after all loans have been repaid.

An attacker can take out huge loans, then pay them back quickly to create more outstanding interest in the system, but he himself does not need to pay that interest. This will in turn increase the interest rate for other users.

Recommendation

Fix the `updateMarketState` function. Use the values of `actionPrincipalDiff` and `actionInterestDiff` earlier, e.g. when intializing the value of principal and `totalOwed`, respectively.

vacuumlabs

Resolution

Fixed in the pull request number 323.

LIQV1-202 Negative interest is outstanding after a loan is paid back

Category	Vulnerable commit	Severity	Status
Bug	`acbe6fda66`	MEDIUM	RESOLVED

Description

The `actionInterestDiff` is double-counted in the `updateMarketState` function, leaving the `interestNew` field smaller than it should be (even negative).

- Create a loan of 1000 Ada.
- After some time, pay back the whole loan of $1000+10$ Ada, where 10 Ada is the interest.
- During the following batching, the market state will be updated. The paid interest of 10 Ada and the principal of 1000 Ada will be subtracted as it is part of the Action fields with the `-Diff` suffix.
- Note that the value of `totalOwed` in `updateMarketState` is zero. The same is true for `totalOwedInterest`.
- For simplicity, say that this was the only loan. That means that `principalNew` is also zero, as well as the value `interest'` is zero.
- As `actionInterestDiff` is -10, `interestNew` $= -10$. This is the new `interest` field that is part of the next market state.

The `interest` field in the market state is not supposed to be negative and it breaks the model in multiple places. For example, the utilization rate computation, the following interest computation, etc.

Even if the `interest` was not straight negative, an attacker could make it smaller than it should be. That would also influence the above parts of the model.

Recommendation

Fix the `updateMarketState` function. Set `interestNew` $=$ `interest'` instead of subtracting the `actionInterestDiff` from it again.

Resolution

Fixed in the pull request number 328.

LIQV1-203 Oracle exchange rate is a simple number

Category	Vulnerable commit	Severity	Status
Design issue	`0b1f7fa162`	MEDIUM	ACKNOWLEDGED

Description

Cardano native tokens often do not have much liquidity and thus their price can be very volatile. When there is not enough liquidity in the markets, taking the exchange rate as a simple number may be misleading – any bigger buy/sell can move the price significantly. A liquidator is therefore not guaranteed to get the expected price for a large liquidated loan. Furthermore, the oracle would need to be updated very often and also somehow take this into account.

This could be also abused by a malicious attacker who could manage to get enough tokens to move the market from some other source (for example by taking a loan). Oracle price manipulation attacks are really common on other blockchains and often result in huge losses.

Recommendation

We recommend to carefully control which assets to list as collateral options, build sufficient reserves to counter a possible risk, limit bigger loans and carefully monitor the market across all sources available. For more options for managing risk, check the risk documentation from Aave[3]. Ideally, if feasible, also set the liquidation discounts such that it takes the volatility and collateral liquidity into account.

Additionally, as soon as other more robust oracles emerge on Cardano, migrate to them instead of the in-house oracle solution. A good oracle for the use case could come e.g. directly from a DEX.

Resolution

The team acknowledged the issue and is planning to address it by assessing the liquidity/volatility of new assets. They are also looking into the usage of a monitoring solution. The prices they plan to use for collateral assets also take the volatility into account to some degree.

[3] https://docs.aave.com/risk/

LIQV1-204 Reserve only decreases

Category	Vulnerable commit	Severity	Status
Bug	66e83f3023	MEDIUM	RESOLVED

Description

The `updateMarketState` function can only decrease the reserve. However, the `splitIncome` function also calculates the `toReserve` value which should be used to increase the reserve.

Recommendation

Use the `toReserve` value returned by the `splitIncome` function to increase the reserve in `updateMarketState`.

Resolution

Fixed in the pull request number 340.

LIQV1-205 Attacker can block Action UTxOs

Category	Vulnerable commit	Severity	Status
Design issue	`0b1f7fa162`	MEDIUM	ACKNOWLEDGED

Description

If multiple users try to interact with the same UTxO, only one can succeed. The others must re-create their transactions, re-sign them and resubmit them. This process can potentially happen multiple times. It could be especially painful for hardware wallet users.

If an attacker can predict which UTxO the user is going to interact with, he can use the UTxO himself and thus prevent the user from using it. If he cannot predict it exactly, he can try to interact with the UTxOs with the highest liquidity and thus effectively block potentially huge portions of liquidity.

Recommendation

The off-chain code for the application should try to choose UTxOs randomly and unpredictably. It should not compete against itself at the same time.

Another way would be to experiment with the number of Actions. The application is more robust if there are more action UTxOs and if the liquidity is distributed uniformly in them. The liquidity would need to be high enough to allow for this, though.

Resolution

The team acknowledged the design issue and will try to improve on the off-chain to limit the impact of this instead of an on-chain redesign.

LIQV1-206 Loan interest rounded down leads to an accumulated interest

Category	Vulnerable commit	Severity	Status
Bug	`0b1f7fa162`	MEDIUM	RESOLVED

Description

Suppose there are two loans in a market, each having `principal` $= 700$. After a while, both are repaid with an interest multiple of 0.4%. The function `totalOwedFromIndex` rounds the interest payment down (truncates) and $700 \cdot 0.004$ rounded down is equal to 2. Both interest payments are therefore worth 2 tokens, which is 4 tokens of interest in total. Plus 700 tokens of principal each are repaid, 1400 of principal in total.

However, the market registers a principal of 1400 for which the truncated interest is 5. After the repayment of both loans, the market state is updated with a total principal of 0 and a total interest of $5 - 4 = 1$. Due to the truncation, the market's outstanding interest remains positive even after both loans have been paid back. Due to compounding and new occurrences of this, it can lead to bigger and bigger inconsistency over time, making e.g. the `qTokenRate` artificially high.

Recommendation

Round up the interest payments in `totalOwedFromIndex` and also make sure that the paid interest does not make the market's outstanding interest negative. The rounded up interest can be considered as an additional interest income.

Resolution

Fixed in the pull request number 351.

LIQV1-207 Attacker can block the batching process – `minBatchSize` requirement is not enforced

Category	Vulnerable commit	Severity	Status
Bug	`25f8cad62d`	MEDIUM	ACKNOWLEDGED

Description

There is a protocol parameter called `minBatchSize` that represents the minimum number of actions that always need to be batched in a batch transaction. The requirement is not computed correctly though. To determine the number of actions batched, it counts in also the actions batched in previous transactions.

As a result, if `minBatchSize` actions are already batched, there is no need to batch any single additional action for the transaction to pass. This can be used as a denial of service attack for a motivated attacker. As there is only a single Batch UTxO, there is a clear contention on it.

In an example market scenario of 16 actions in total and `minBatchSize` $= 4$, this means that after 4 actions are batched, the rest of the batching process can be prolonged for as long as honest batching transactions would not be included in the blocks instead of malicious ones. Strictly speaking, it's impossible to guarantee that it would ever happen.

It is important to note that this can lead to significant market downtime as there are time guarantees that prevent interactions with Action UTxOs after a certain time if batching is not completed.

Recommendation

We would like to highlight the design change proposed as part of the LIQV1-303 issue that would remove the need for the Batch UTxO, hence removing the contention on it.

A more straightforward recommendation is to fix the bug directly in the Batch validator and enforce that either all actions are batched already or the `minBatchSize` is enforced in the correct way.

As the protocol is unfortunately already on mainnet at the time of writing this report, we suggested also a hotfix tweaking the protocol parameters that could mitigate this issue. It could negatively impact other contention related issues mentioned in this report, though.

LIQV1-208 Staking delegation is controlled by a single key

Category	Vulnerable commit	Severity	Status
Design issue	`25f8cad62d`	MEDIUM	ACKNOWLEDGED

Description

ADA staked is governed by a staking validator that is referred to by the market params. However, the currently used staking validator describes a single key that can change the delegation to any pool – the `adaRegistrationPubKey`.

Since a stake pool can take as much as 100% of the rewards as a fee, this represents a way for the private key holder to take the whole staking yield for himself.

Recommendation

Do not rely on a single key and use a more robust solution. This can be rather easily updated even over time. However, as of writing this report, this is still a present risk.

LIQV1-209 Oracle is controlled by a single key

Category	Vulnerable commit	Severity	Status
Design issue	25f8cad62d	MEDIUM	ACKNOWLEDGED

Description

A single private key is responsible for updating the oracle prices. Oracles, if compromised, pose an existential threat to any lending protocol. Therefore, there is a need for a more robust solution to mitigate the risk.

The key can be updated by the multisig scheme. The holder can still cause a lot of trouble in a short time if he is dishonest or his key is compromised.

There is also no expiration on the prices published by the oracles. If the key holder is not fulfilling his role, liquidations could still occur using the old prices.

Recommendation

Do not rely on a single key and use a more robust solution. When feasible, we recommend migrating to an independent oracle service.

Furthermore, we recommend a comprehensive monitoring being put in place.

LIQV1-301 The `maxLoan` threshold can be exceeded

Category	Vulnerable commit	Severity	Status
Bug	`28f0836af3`	MINOR	RESOLVED

Description

The function `checkValidMaxLoan` is double-counting the loan principal towards the total supply. The value of the new loan should not affect the value of `totalSupply` which represents the total supply plus the total principal at the time of the last batching. The attacker can take advantage of this to take a loan that exceeds the `maxLoan` ratio in the following way:

- Create a new loan, borrowing $\texttt{maxLoan}/(1 - \texttt{maxLoan})$ of the total supply (note that this is more than the supposedly allowed `maxLoan` fraction).
- E.g. if $\texttt{maxLoan} = 0.1$, the attacker can borrow $\sim 11\%$ of the total supply. For $\texttt{maxLoan} = 0.5$, the actual maximum is 100%.

Recommendation

Remove the `loanPrincipal` term from the calculation of `totalSupply` in the function `checkValidMaxLoan`.

Resolution

Fixed in the pull request number 326.

LIQV1-302 `compoundingPeriod` should be considered part of the interest model

Category	Vulnerable commit	Severity	Status
Code style	`9625f05a90`	MINOR	ACKNOWLEDGED

Description

Interest is typically understood as annualized. However, the interest rate in the Liqwid v1 app is meant to accrue every `compoundingPeriod`, meaning that a potential update to the `compoundingPeriod` parameter also affects interest payments. This parameter is currently separate from the `InterestModel` data record but it would be clearer for the users if it was a part of it. For example, halving the `compoundingPeriod` parameter at least doubles the interest, so it is an important parameter of the interest model. An attacker can create a DAO proposal for changing the `compoundingPeriod` in their favor – a lender would like to increase the interest rate, a borrower would like to decrease it. The DAO participants might not understand that this parameter also affects the interest rate and thus vote against their interests.

Recommendation

We suggest moving the `compoundingPeriod` field into the `InterestModel` data record, so that it's more clear that it influences the interest payments. Also, we suggest making the documentation and the model public so that the DAO participants can vote correctly with better information.

Finally, the name of the parameter is `compoundingPeriod` in the documentation while it's `compoundRate` in the code. We suggest sticking to `compoundingPeriod` for consistency.

LIQV1-303 Attacker can slow down the batching process

Category	Vulnerable commit	Severity	Status
Design issue	`0b1f7fa162`	MINOR	ACKNOWLEDGED

Description

In the protocol, anyone can batch transactions but there is only a single Batch UTxO. An attacker could use this UTxO and batch `minBatchSize` number of actions (the market parameter determining the minimum number of actions in a batch transaction). Because `numActions` actions must always be batched eventually, the duration of the batching process is brought up to $O(\texttt{numActions}/\texttt{minBatchSize})$. The attacker would still contribute to the batching process, but he would slow down the system.

Recommendation

We suggested a design change that would remove the need for the Batch UTxO completely. The supply would remain available while the batching is in progress and the batching itself could happen in parallel (with no Batch UTxO contention).

Resolution

The Liqwid team decided to address this issue in Liqwid v2. In this version, it is planned to be remedied by off-chain incentives for batchers to act honestly. However, no details were shared with us.

LIQV1-304 Attacker can hide liquidity during the batching process

Category	Vulnerable commit	Severity	Status
Design issue	`0b1f7fa162`	MINOR	ACKNOWLEDGED

Description

An attacker can begin the batching process by batching the Actions with the highest available liquidity in them. If the liquidity is not distributed uniformly, he could lock a big percentage of the whole liquidity for the rest of the batching process.

This is especially dangerous in combination with LIQV1-303 which enables the attacker to prolong the batching process.

Recommendation

Remove the need for the Batch UTxO and do the batching within Action UTxOs as suggested in the recommendation to LIQV1-303. The supply would remain available while the batching is in progress and the batching itself could happen in parallel (with no Batch UTxO contention).

Resolution

Liqwid decided to address this issue in Liqwid v2. In this version, it is planned to be remedied by off-chain incentives for batchers to act honestly. However, no details were shared with us.

LIQV1-305 Close factor below 100% causes never-ending liquidations

Category	Vulnerable commit	Severity	Status
Design issue	`9625f05a90`	MINOR	RESOLVED

Description

The close factor determines how big a fraction of the remaining loan a liquidator can liquidate. For example if the close factor is 50%, the first liquidator can liquidate half of the original loan, the second liquidator can liquidate $1/4$-th, the third can liquidate $1/8$-th, etc. Even after the n-th liquidation, there will be a small fraction of at least $1/2^n$ that won't be liquidated.

Recommendation

Similar protocols Aave and Compound allow liquidating the whole loan once certain conditions are met. You could add another threshold that unlocks the 100% close factor.

Resolution

The team agreed to set the initial close factor to 100%. Later, the DAO might override this setting which would reopen this issue.

LIQV1-306 Batching can require batchers to pay min-ADA

Category	Vulnerable commit	Severity	Status
Design issue	25f8cad62d	MINOR	ACKNOWLEDGED

Description

Anyone can participate as a batcher in the Liqwid v1 application. However, if there is any portion of interest supposed to go into the treasury or to be paid as dividends, the min-ADA amount for those UTxOs would likely need to be paid by the batchers themselves. It is especially a problem in non-ADA markets. This disincentivizes public to participate with practical implications to the centralization of the batchers.

Recommendation

We recommend updating the design. One option could be to write a treasury and dividends' scripts that would allow for their spend given the value contained within is increased. That would translate to paying min ADA just a few times and then batchers adding value into already existing UTxOs.

In any way, if the batchers are to be decentralized, it is necessary to not disincentivize them by requiring them to pay. Instead, it is expected that participants would earn on it.

LIQV1-401 Variable name “supply” in some functions includes the reserve as well

Category	Vulnerable commit	Severity	Status
Code style	`53beef5c2b`	INFORMATIONAL	ACKNOWLEDGED

Description

The functions `mkActionOutputChecks` and `checkActionOutput` use the word “supply” to name a variable, but the value includes the reserve which is not consistent with the other occurrences of the word.

- `nextSupply` in `mkActionOutputChecks` is in actually `nextSupply + nextReserve`.
- `supply` in `checkActionOutput` is similar.

Beware of inconsistent naming in critical functions. It makes any change to the code prone to errors.

Recommendation

Use other names for values including both the supply and the reserve, e.g. `underlying` or `expectedUnderlying` as is the case in other places in the code base.

LIQV1-402 Functions `checkBatchTime` appear twice with different semantics

Category	Vulnerable commit	Severity	Status
Code style	`60c3478060`	INFORMATIONAL	ACKNOWLEDGED

Description

There are two functions named `checkBatchTime` in the code base but they have different semantics. Putting it simply, one checks that an action other than batching can happen on an Action UTxO, and another checks that the batching can happen. This results in very different use cases and different checks. One of them is defined in the `Timing.hs` file suggesting a general utility, but it's apparent that other checks for the batching times are needed.

Recommendation

Rename the functions. Make the semantics clearer from the name itself and avoid using the same name for different checks. A potential candidate that would go in line with the rest of the code base could be `checkMinBatchTime` and `checkMaxBatchTime`. Other names suggesting the use case could be even clearer.

LIQV1-403 Variable `liquidationDiscount` has a misleading name

Category	Vulnerable commit	Severity	Status
Code style	4d878465c1	INFORMATIONAL	ACKNOWLEDGED

Description

The protocol parameter `liquidationDiscount` is used differently than what the name suggests. For example, in some files setting `liquidationDiscount` $= 0.95$ is explained in a comment as `discount` $= 5\%$. If it was instead equal to $1 -$ `liquidationDiscount` (its complement to 1), it would actually mean "discount".

Recommendation

Rename the parameter to "liquidation weight" or change the formulas so they use $1 -$ `liquidationDiscount` instead of `liquidationDiscount`.

LIQV1-404 Variable `maxBatchTime` has confusing name

Category	Vulnerable commit	Severity	Status
Code style	`4d878465c1`	INFORMATIONAL	ACKNOWLEDGED

Description

The protocol parameter named `maxBatchTime` represents the maximum time until when actions can be performed. After it, only the batching is possible on an Action UTxO. It does not control the time of the batching. Neither is it the maximum of that possible batching time period.

Recommendation

Rename the variable to a name that has more to do with the semantics of it, for example `maxActionTime`.

LIQV1-405 Functions in `Helpers.hs` are not documented properly

Category	Vulnerable commit	Severity	Status
Code style	`4d878465c1`	INFORMATIONAL	ACKNOWLEDGED

Description

Functions located in `Helpers.hs` do a lot of important logic, but their name and docstrings often do not describe it sufficiently. Examples of such functions are:

- `updateLoanLoanInput`
- `updateLoanNewLoan`
- `updateLoan`

Recommendation

Add more meaningful docstrings to the functions and/or rename them so that their semantics are clear from the names.

LIQV1-406 The use of a `-Interest` suffix leads to unclear variable names

Category	Vulnerable commit	Severity	Status
Code style	4d878465c1	INFORMATIONAL	ACKNOWLEDGED

Description

The suffix `-Interest` is used to mark quantities after the latest interest accrual. However, this leads to very unclear names such as `interestInterest` and `totalOwedInterest`. Also, `totalOwedInterest` sounds like it should mean the "total owed interest" but it does not, which is confusing for the readers of the code.

Recommendation

Use clear variable names that reflect the semantics of the variable usage.

LIQV1-407 The variable `supplyDiff` has multiple meanings

Category	Vulnerable commit	Severity	Status
Code style	4d878465c1	INFORMATIONAL	ACKNOWLEDGED

Description

The word "supply" in Action UTxO and Batch UTxO has a different meaning. After a deposit of 1000 Ada and a loan of 1000 Ada, the action UTxO has `supplyDiff` $= 1000$ and `principalDiff` $= 1000$. After batching, the batch UTxO has `supply` $= 0$, as the principal gets removed from the batch supply quantity. The action value of `supplyDiff` $= 1000$ would suggest that supply should be equal to 1000.

Recommendation

We suggest using `depositDiff` in the action datum instead of `supplyDiff` as it is a more fitting name.

LIQV1-408 Reserve field in Action datum may be imprecise

Category	Vulnerable commit	Severity	Status
Bug	`25f8cad62d`	INFORMATIONAL	ACKNOWLEDGED

Description

When distributing the supply and reserve in the `BatchFinal` transaction into Action UTxOs, the value that is distributed is rounded separately to the reserve amount that is written into the datum. However, the value is the sum of both the supply and the reserve. As a result, there can be rounding discrepancies between the reality and the *reserve* field put into the datum. In the extreme case, the reserve written into the datum can even exceed the underlying amount. That can happen thanks to the special handling for the last action.

Recommendation

We recommend changing the algorithm for the distribution of reserve tokens. First, create an initial assignment by rounding down the number of reserve tokens for each Action UTxO. Then complete the initial assignment by distributing the tokens that were not distributed due to rounding. A lot of approaches work, here is a simple greedy algorithm:

1. Add as much as possible to the reserve of any Action UTxO, not exceeding the total number of tokens distributed to this Action UTxO.
2. If there are still some left-over reserve tokens to be distributed, move to another (yet unused) Action UTxO and repeat the first step.

The reserve is always at most as big as the total number of tokens at Action UTxOs, so this algorithm will always result in a valid distribution.

LIQV1-409 Other code best practices

Category	Vulnerable commit	Severity	Status
Code style	25f8cad62d	INFORMATIONAL	ACKNOWLEDGED

Description

This issue summarizes a wide range of issues that were occurring throughout the codebase. It is not a full list.

- **Naming.** `pvalueLength` is the number of currency symbols only and ignores the number of token names; `backdoorScript r d` should be `backdoorScript d r`; `matchSingle` does not only match, it returns the quantity; `checkLoanValue` returns whether the collateral in the loan output is greater or equal to the loan input; `isSupplyAction` is true if and only if there is an action; `actionDatumI` and `actionDatum0` in the `getActionChange` function are not datums but values; `minValue` and `maxLoan` variables in market params are not descriptive of the reality; `isPolicy` should be a utility function and needs a more descriptive name.
- **Duplicate code.** Many validators use custom-made constructs just for them instead of reusing the same utility functions. An example is a utility function called `assertExactlyOne`. Many validators that delegate validation to any of a list of policies could use it. More often than not, however, they implement their own way of checking the same thing. A similar case is true of the `predicate` function compared to the `hasCredential`.
- **Dead code.** `placeholderStakeValidator`; `placeholderPolicy`; `deserialiseHex`; `ptxOut`; `preplicateDList`; `pappNTimes`; `stubValidator`; `stubPolicy`; `pleakAsError`; `pchooseData'`; `BatchRedeemer` is not *really* used; `actionB`, `actionC` validators (they are outdated either way); `interestIndexFromMultiplier`; `updateBorrowPrincipal`; `updateRepayPrincipal`; the `BorrowToken` validation path when actions are present in the transaction, `actionChange` $= 0$ and `modifying` is True: this is not a valid scenario due to the `minActionValue` check. It is not necessary to implement custom logic and variables for it, e.g. the `modifying` variable.
- **Legacy or imprecise comments.** The LQ minting link does not work; the docstring `maxBatchTime` is imprecise; there are still a number of comments or variable names referring to demand and supply actions, even though that design decision was aban-

doned a long time ago; Market Inbox is also legacy and still mentioned; some links to Notion documentation do not work anymore.

Recommendation

Polish the code based on the above and make a habit of refactoring the code to adhere to the best practices, examples of some of whose are outlined here.

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 Liqwid Labs LLC (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.

vacuumlabs

B Classification methodology

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.

C 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: 2023-02-13
Final commit: `62b1e463b80a7a7e0dca52b64c13c311220c8548`

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.

D About Us

Vacuumlabs has been building crypto projects since the day they became possible on the Cardano blockchain.

- Helped create the number 2 (TVL-based) decentralized exchange on Cardano - WingRiders
- We are the group behind the popular AdaLite wallet. It was later improved into a multichain wallet named NuFi which also integrates a decentralised exchange.
- We built the Cardano applications for hardware wallets Ledger and Trezor.
- We built a cutting-edge decentralized NFT marketplace JamOnBread on Cardano with truly unique features and superior speed of both the interface & transactions.

Our auditing team is chosen from the best.

- Ex-WingRiders, ex-NuFi developers
- Experience from Google, Spotify, traditional finance, trading and ethical hacking
- Medals and awards from programming competitions: ACM ICPC, TopCoder, International Olympiad in Informatics
- Passionate about Program correctness, Security, Game theory and Blockchain

Contact us:
audit@vacuumlabs.com