LiquidOps Pool Security Review

Auditors

Gerard Persoon, Lead Security Researcher

Eric Wang, Lead Security Researcher

Cryptara, Security Researcher

Christos Pap, Associate Security Researcher

Report prepared by: Lucas Goiriz

May 5, 2025

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Liquid Labs is committed to developing innovative software to advance decentralized finance protocols, empowering a more secure, inclusive, and decentralized Web3 future.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of LiquidOps Pool according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 5.5 days in total, LiquidOps engaged with Spearbit to review the liquidOps-pool protocol. In this period of time a total of **63** issues were found.

Summary

Project Name	LiquidOps
Repository	liquidOps-pool
Commit	bef1da36
Type of Project	DeFi, Lending
Audit Timeline	Mar 31st to Apr 5th
Fix period	Apr 15th to Apr 30th

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	3	3	0
Medium Risk	11	9	2
Low Risk	24	14	10
Gas Optimizations	3	3	0
Informational	22	14	8
Total	**63**	**43**	**20**

5 Findings

5.1 High Risk

5.1.1 Borrowing interest is calculated inaccurately and can be manipulated

Severity: High Risk

Context: interest.lua#L95-L119

Description: The system keeps track of each user's loan and the accrued interest individually. Each user has a corresponding last-update timestamp, which is compared with the current timestamp to calculate the newly accrued interest only when the user's position is updated, e.g., when borrowing funds or repaying debt.

The accrued interest during a period is calculated as:

```
interestAccrued = debt * (utilizationRate * baseRate + initRate) * delay / oneYearInMs
```

where `utilizationRate = TotalBorrows / (TotalBorrows + Cash)`, i.e., the market's current utilization rate. The `delay` variable represents the time difference between the user's current and last actions.

However, using the current utilization rate to calculate the interest in the past `delay` period is incorrect. It assumes that the rate has not been changed since the last user's action, which, however, is not always the case because other users' actions can arbitrarily change the market's state.

Consider the following example:

1. `t = 0`, User A borrows. The utilization rate is 50%.
2. `t = 100`, User B borrows. The utilization rate becomes 90%.
3. `t = 101`, User A repays. The interests are calculated using a rate of 90%, but it was 50% most of the time. User A ends up paying more interest.

Additionally, an attacker could exploit this design to lower the interest rate to a number close to `initRate`. Before repaying their debt, they could mint a large number of oTokens with another account to decrease the market's utilization rate as much as possible. After repaying the debt, they redeem the underlying tokens without incurring a loss.

Recommendation: Based on how the interest is calculated, the system should be designed to update every user's debt whenever the market's utilization rate or the state has changed. Compound V2's design is an example:

- Every user has a corresponding `interestIndex`, and the protocol keeps track of a global `borrowIndex`.
- Before performing any action that will change the market's state, the system accrues interest by increasing the `borrowIndex` using the formula above.
- The user's debt is calculated as `borrowBalance * borrowIndex / interestIndex`, which updates automatically when the `borrowIndex` increases.
- When a user borrows, their `borrowBalance` is first updated to include the accrued interest so far and then increased by the borrowed amount. Next, their `interestIndex` is set to the global `borrowIndex`. The process is similar when a user repays their debt.

LiquidOps: Fixed in PR 27.

Spearbit: Verified.

5.1.2 Incorrect Tag Parsing in Interest Sync

Severity: High Risk

Context: interest.lua#L168

Description: In the `interest.lua` contract, the `syncInterests` function incorrectly checks for a repayment action by evaluating `msg.Tags.Action == "Repay"`. However, according to the messaging conventions established by the protocol (specifically the `borrow-loan-interest-sync-dynamic` pattern), the repayment intent is signaled via the `X-Action` tag, not the `Action` tag. As a result, the intended conditional branch for interest synchronization on repayments is bypassed, and execution falls through to the default `else` block.

This default case uses `msg.From` as the interest update target, which in repayment flows refers to the collateral token process—not the borrower. Consequently, the system updates the interest balance for an incorrect user and skips the actual borrower, allowing them to underpay or bypass interest accrual entirely.

Additionally, there is no validation to ensure that `msg.Tags.Sender` or `msg.Tags["X-On-Behalf"]` is a properly formed address before updating interest. This opens the door to further inconsistencies or even execution errors if unexpected data is received in those fields.

Recommendation: The logic in `syncInterests` should be corrected to check `msg.Tags["X-Action"] == Repay`, in combination with Action == Credit-Notice and `msg.From == CollateralID` before falling back to `msg.Tags.Action`, ensuring that repayment actions are properly routed. When `X-Action` is `"Repay"`, the function should also validate that either `msg.Tags["X-On-Behalf"]` or `msg.Tags.Sender` is a valid address using the existing `assertions.isAddress` helper before proceeding with the interest update.

This ensures the correct borrower's interest is updated, prevents misuse via malformed tags, and aligns with the repayment flow conventions of the protocol.

LiquidOps: Code is removed in PR 27 due to redesign.

Spearbit: Verified.

5.1.3 Partial liquidations allow position management blocking and liquidation prevention due to queue mechanism implementation

Severity: High Risk

Context: *(No context files were provided by the reviewer)*

Description: The `LiquidOps` protocol allows partial liquidations where only a portion of a loan is liquidated. When a liquidation is initiated, the target user is added to a `LiquidationQueue` to prevent duplicate liquidations:

```lua
-- Add the target to the liquidation queue to lock further liquidations
table.insert(LiquidationQueue, target)
```

The user remains in this queue until the liquidation is complete, at which point they are removed:

```lua
-- Remove target from liquidation queue to allow further liquidations
LiquidationQueue = utils.filter(
  function (v) return v ~= target end,
  LiquidationQueue
)
```

While a user is in the `LiquidationQueue`, they are prevented from performing certain operations, as many protocol actions check if the user is queued:

```lua
-- check-queue handler
-- the user is queued if they're either in the collateral
-- or the liquidation queues
return msg.reply({
  ["In-Queue"] = json.encode(
    utils.includes(user, CollateralQueue) or
    utils.includes(user, LiquidationQueue)
  )
})
```

Due to lack of minimum liquidation size requirements, this implementation allows for two attack vectors:

1. An attacker could liquidate a minimal amount of a user's position to forcefully add them to the queue, temporarily preventing them from performing critical actions such as repaying loans or managing collateral.

2. Users at risk of liquidation could create multiple accounts to perform minimal self-liquidations (as direct self-liquidation is not allowed from the protocol), effectively preventing other liquidators from capturing the full liquidation incentives and potentially preventing the liquidation of undercollateralized positions.

Recommendation: Implement a minimum liquidation threshold as a percentage of the total outstanding loan (e.g., 20%) to make these attacks economically unfeasible. While this does not fully mitigate the issue, it makes the attack much more expensive to perform.

LiquidOps: Fixed in PR 26.

Spearbit: Verified, as the requirement of a 20% minimum liquidation threshold makes the attack vectors economically infeasible.

5.2 Medium Risk

5.2.1 No liquidation incentives if the discount interval has passed

Severity: Medium Risk

Context: controller.lua#L965-L969

Description: When a position is liquidated, a `discount` factor is calculated based on the time that has passed since the start of the auction. The auction follows a Dutch auction design, where the `discount` factor decreases linearly to zero if a configured period, `DiscountInterval` has passed.

However, it is better to keep a non-zero discount even if the period has passed, which would ensure that liquidators have an incentive to liquidate underwater positions in the system. Otherwise, without being liquidated, the position's debt could increase over time and potentially become bad debt, which puts the protocol at risk of insolvency.

Recommendation: Consider defining a variable as the minimum discount factor, which can be configurable by the controller or admin, and setting the `discount` variable to this value when a specific interval has passed.

LiquidOps: Fixed in commit 85e9fdb.

Spearbit: Verified.

5.2.2 Incorrect implementation of `supplyRate()`

Severity: Medium Risk

Context: interest.lua#L51-L75

Description: The `supplyRate()` function in `interest.lua` calculates the supply rate based on the current borrow and utilization rates. However, the implementation is incorrect and has two issues:

1. At interest.lua#L54, `borrowRateFloat` is calculated as `borrowRate / rateMul` and, therefore, does not need to be divided by `rateMul` again at interest.lua#L60.
2. At interest.lua#L61, the utilization rate is incorrectly calculated as `TotalBorrows / TotalSupply / 10 ^ CollateralDenomination`. It should be `TotalBorrows / (TotalBorrows + Cash)` by definition.

Recommendation: Consider modifying the `supplyRate()` function as suggested above.

LiquidOps: Fixed in PR 24.

Spearbit: Verified.

5.2.3 Incorrect rounding direction when calculating the seized collateral from the borrower

Severity: Medium Risk

Context: liquidate.lua#L160-L165

Description: The `liquidatePosition()` function in `liquidate.lua` transfers out the underlying token of a given `quantity`. It calculates how many `oTokens` the transferred underlying tokens are worth and subtract that value from the borrower's position. The value, `qtyValueInoToken`, is calculated as:

```
-- get supplied quantity value
-- (total supply / total pooled) * incoming
local qtyValueInoToken = bint.udiv(
 totalSupply * quantity,
 totalPooled
)
```

where `totalSupply` is the `oToken`'s total supply, and `totalPooled` is the sum of total borrows and cash in the market.

The `bint.udiv()` performs an integer division with the result rounded down, which benefits the borrower as their `oToken` balance is subtracted with a smaller value. The rounding error may be insignificant if `totalSupply` and `totalPooled` are large enough. However, in an empty-market attack scenario, where the attacker artificially inflates the market's exchange rate, the rounding error could become large enough to cause issues with the protocol.

For example, assuming an attacker can inflate the exchange rate such that 1 `oToken` is worth 100,000 underlying tokens. If the repay value is less than 100,000 underlying tokens, `qtyValueInoToken` will be 0. As a result, the borrower will not incur a loss while being liquidated. However, the `oToken` still transfers the `quantity` amount of underlying tokens to the liquidator.

Regarding how an attacker can inflate the exchange rate on an empty market, see the finding Security considerations of exchange-rate inflation in empty markets.

Recommendation: Consider implementing a helper function that rounds up the division result to calculate the `qtyValueInoToken` value. Also, consider implementing security measures and checks to mitigate potential exchange-rate inflation attacks on empty markets.

LiquidOps: Fixed in commit 767ffbd.

Spearbit: Verified.

5.2.4 Missing Invariant Check for Liquidation Threshold

Severity: Medium Risk

Context: pool.lua#L11-L14, config.lua#L61-L62, controller.lua#L235-L250

Description: The configuration logic found in both `controller.lua` and `config.lua` fails to enforce a critical invariant for lending safety: the `LiquidationThreshold` must be strictly greater than the `CollateralFactor`. This condition is necessary to ensure that borrowers are always at risk of liquidation before their collateral value drops below the borrowed amount, giving the protocol a safe buffer to cover debt positions during adverse price movements.

Without this invariant enforced, an administrator may configure the pool with a `CollateralFactor` equal to or greater than the `LiquidationThreshold`, effectively allowing users to borrow more than the system can safely reclaim in a liquidation event. This misconfiguration directly undermines the protocol's solvency assumptions and can lead to irrecoverable bad debt, especially during fast market downturns where prices can gap through the liquidation margin.

Furthermore, the current inline documentation in `config.lua` is misleading. It comments that the "liquidation threshold (should be lower than the collateral factor)", which is incorrect and contradicts the intended safety model. The correct relationship is that the `LiquidationThreshold` should always be strictly higher than the `CollateralFactor`.

Recommendation: The configuration validation logic should include an explicit check to enforce that `LiquidationThreshold > CollateralFactor` both in the main controller's token listing (`controller.lua`) and within the dynamic update routine (`config.lua`). If the condition is not met, the system should reject the update and return a descriptive error message to prevent misconfiguration.

For example, in both configuration paths:

```
assert(
  liquidationThreshold > collateralFactor,
  "Liquidation threshold must be greater than the collateral factor"
)
```

This preserves the health of the lending pool and ensures that all borrow positions remain recoverable in a liquidation scenario. Lastly, update the documentation line in `config.lua` to reflect the correct logic:

```
-- liquidation threshold (should be greater than the collateral factor)
```

LiquidOps: Invariant check was added in PR 24.

Spearbit: Verified.

5.2.5 Security considerations of exchange-rate inflation in empty markets

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: An empty market refers to a market with little or no liquidity, which can occur when a new market is created and listed without initial liquidity from the creator, etc. One of the most common vulnerabilities of empty markets is exchange-rate inflation, i.e., inflating the `totalPooled` amount while keeping `totalSupply` as a small number, which can be done through direct donations or stealth donations.

Specifically, stealth donations occur due to rounding errors when users mint or redeem tokens, which increases the exchange rate. For example, assuming an attacker holds 1 oToken (in the smallest unit), and the initial exchange rate is 2, i.e., 1 oToken is worth 2 underlying tokens. The attacker can:

1. Mint 0 `oToken` with 1 underlying token. The exchange rate becomes 3.
2. Mint 0 `oToken` with 2 underlying tokens. The exchange rate becomes 5.
3. Mint 0 `oToken` with 4 underlying tokens. The exchange rate becomes 9.

and so on. The attacker will be able to inflate the exchange rate exponentially. Although the attacker effectively donates their funds to the market through minting, they do not incur a loss as they hold the only oToken, which can redeem all the underlying tokens afterward.

If the markets have an initial exchange rate of 1, the attacker could first mint some oTokens and borrow the underlying tokens with some collateral in the other markets. After one block has passed, interest will be accrued, causing the exchange rate to exceed 1. The attacker then burns all the tokens except 1, resulting in 2 underlying tokens left in the market and, therefore, achieving the initial condition.

Exchange-rate inflation could lead to security issues when combined with other vulnerabilities, e.g., incorrect rounding directions. Compound V2 implemented an incorrect rounding direction in the redeem function, which has caused several incidents on Compound V2 forks, i.e., the empty market attack. The attackers were able to inflate the exchange rate first to increase the impact caused by the incorrect rounding direction and, therefore, redeem their donated tokens and left the position with a large debt.

For the reviewed protocol, the rounding direction when a user redeems an underlying token is correct. However, we identified an incorrect rounding direction in calculating the seized collateral from a borrower, which can potentially cause a more significant impact when combined with exchange-rate inflation. See the finding "Incorrect rounding direction when calculating the seized collateral from the borrower".

Every market should have sufficient liquidity to mitigate exchange-rate inflation attacks. Note that stealth donations are still possible even though the markets have enough liquidity. However, it would be inefficient or impractical for attackers to execute, and they are also more likely to incur a loss due to donations since they do not hold most of the oTokens.

Recommendation: Consider following this procedure when listing new markets:

1. Configure the collateral factor as 0 during the market's setup. Alternatively, the `pool.setup()` function could explicitly set `CollateralFactor` to 0.
2. Mint enough `oToken` (e.g., 10^9 or more) and keep them securely or send it to the "zero" address to make them unavailable permanently.
3. Set the collateral factor to a positive value to enable the market when necessary.

LiquidOps: Acknowledged, we will enforce this practice when listing new tokens.

Spearbit: Acknowledged.

5.2.6 Incomplete Validation in `isAddress()`

Severity: Medium Risk

Context: controller.lua#L889-L895, assertions.lua#L9-L15

Description: The `isAddress()` function in `assertions.lua` validates the input and returns whether the input is a valid address string. However, there are two issues in the implementation:

1. At assertions.lua#L10, the comparison `not type(addr) == "string"` should be `type(addr) ~= "string"` instead since the `not` operation has higher precedence than `==`.
2. At assertions.lua#L12, the regex check should be `string.match(addr, "^[A-z0-9_-]+$")`, i.e., including the `^` and `$` metacharacters, to ensure that all characters are valid.

Recommendation: Consider implementing the above suggestions.

LiquidOps: Fixed in commit a713065.

Spearbit: Verified.

5.2.7 Incorrect Update of Auction List after Liquidation

Severity: Medium Risk

Context: controller.lua#L672-L690

Description: When liquidating a position, the `removeWhenDone` variable indicates whether the position should be removed from the auction list after the current liquidation. According to the comment:

> The auction needs to be removed if there will be no loans left when the liquidation is complete.

Based on this comment, the comparison at L690 should be `== nil` instead. It is because if the position has no action loan after the liquidation, the `util.find()` function would return a `nil`.

However, instead of checking if the position has an active loan, a more accurate way is to check if the position would become healthy after liquidation. The position should be kept on the auction list if it is still unhealthy. Otherwise, it should be removed. The current method would still keep a healthy position with an active loan on the auction list.

Recommendation: Consider fixing the comparison at L690 or implementing the logic based on whether the position would become healthy.

LiquidOps: Fixed in commit 6e6e9db.

Spearbit: Verified.

5.2.8 Position enumeration in `sync-auctions` handler can lead to DoS when processing large numbers of positions

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: The `LiquidOps` protocol's `sync-auctions` handler performs a scan of all market positions to identify underwater positions eligible for liquidation. This handler executes a computationally intensive process that:

1. Fetches all token prices from the oracle.
2. Queries all positions across all markets.
3. Processes and aggregates every user position.
4. Evaluates each position's liquidation eligibility.

```lua
-- POSITION DATA COLLECTION
-- Prepare to query all positions across all markets (oTokens)
-- by creating message requests for each market in the system.
---@type MessageParam[]
local positionMsgs = {}

for _, token in ipairs(Tokens) do
  table.insert(positionMsgs, { Target = token.oToken, Action = "Positions" })
end

-- MULTI-MARKET POSITION FETCHING
-- Execute all the position queries in parallel and collect responses.
-- Each response contains all user positions in a specific market.
---@type Message[]
local rawPositions = scheduler.schedule(table.unpack(positionMsgs))
```

This position scanning process scales linearly with the number of positions (`O(n)` complexity), resulting in performance degradation as the position count increases. An attacker could create numerous small positions across different markets, forcing the system to process each one during liquidation checks. Since the handler iterates through all positions, this could significantly slow down the protocol's operations when the number of positions grows large.

Importantly, this scaling issue can arise not only from malicious activity but also naturally as the protocol gains adoption and more users create positions.

Additionally, the protocol doesn't clear positions when they're fully repaid, instead keeping them with zero values, which may lead to unnecessary processing of empty positions and wasted storage resources.

Recommendation: To mitigate these issues, consider implementing some of the following changes:

1. Enforce a minimum size for opening positions, and require that positions either remain above this threshold or be closed entirely.
2. When positions are fully repaid, remove them from storage entirely (set to `nil`) rather than keeping them with zero values.
3. Move the `sync-auction` handler into a separate process to avoid the controller being DoS'ed.
4. Implement an upper limit on the number of users that can be processed in a single message (for example process only `X` users at a time), allowing for batch processing that prevents excessive computational consumption and timeouts.
5. Perform stress tests to quantify the maximum number of positions the `sync-auctions` handler can process within reasonable time and resource limits under various load conditions.
6. Consider introducing a small fee for invoking the `sync-auctions` function to disincentivize spamming or excessive calls while evaluating if such a fee might negatively impact the protocol's health by discouraging legitimate liquidations when they are needed.

LiquidOps: Partially fixed in PR 24.

Spearbit: Verified. The fix now removes the value from storage entirely.

5.2.9 Tokens not always refunded

Severity: Medium Risk

Context: controller.lua#L453-L455, process.lua#L100-L107

Description: In `process.lua` tokens are refunded when `From ~= CollateralID`. However if transfers do have `From == CollateralID`, the `X-tag` could be specified incorrectly or could not be forwarded due to a bug in the token contract.

In `controller.lua` there is no code to refund tokens if the `X-tag` isn't `Liquidate`. This can happen with the supplied libraries, see finding `Tag "X-Action" = "Liquidate" missing in Javascript library`.

If the tokens aren't returned then they stay stuck in `process.lua` / `controller.lua` , although they can be recovery by the protocol via `batch-update` and other management actions.

Recommendation: Consider refunding tokens in more situations, for both `process.lua` and `controller.lua`.

5.2.10 `UpdateInProgress` only checked in a limited number of situations

Severity: Medium Risk

Context: controller.lua#L385-L390, controller.lua#L814, controller.lua#L852-L872

Description: `UpdateInProgress` is only checked in a limited number of situations: only during "Add-To-Queue", which is send in `Borrow()`, `Transfer()` and `Redeem()`. However the updates are powerful and can perform an arbitrary action on all `oTokens`. Such an update could influence all actions.

Recommendation: Consider checking `UpdateInProgress` for all actions. Also consider adding a check for `UpdateInProgress` in the function to handle "Check-Queue-For".

LiquidOps: Acknowledged. We will not remediate the entire issue here, since actions that don't influence user positions would just run normally. Additionally, actions that happen instantly and don't use coroutines will just run on the new version (fine for us, since they arrived later) and any other action should use the queue.

Spearbit: Acknowledged.

5.2.11 Potential race condition between repayments and liquidations due to missing queue check, may allow incorrect accounting

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: The LiquidOps protocol has a potential race condition between loan repayments and liquidations that can lead to accounting inconsistencies. The issue exists because the `repay.handler` function does not check if the user is in the controller's `LiquidationQueue` before processing a repayment.

While the initial transfer handler is wrapped with `queue.useQueue()` which performs queue checks, the internal `repay.handler` that processes the `Credit-Notice` with `X-Action = "Repay"` lacks this protection:

```
Handlers.advanced({
  name = "borrow-repay",
  pattern = {
    From = CollateralID,
    Action = "Credit-Notice",
    ["X-Action"] = "Repay"
  },
  handle = repay.handler,
  errorHandler = repay.error
})
```

In the `repay.handler` function, no queue check is performed:

```lua
function repay.handler(msg)
  assert(
    assertions.isTokenQuantity(msg.Tags.Quantity),
    "Invalid incoming transfer quantity"
  )

  -- quantity of tokens supplied
  local quantity = bint(msg.Tags.Quantity)

  -- allow repaying on behalf of someone else
  local target = msg.Tags["X-On-Behalf"] or msg.Tags.Sender

  -- check if a loan can be repaid for the target
  assert(
    repay.canRepay(target, msg.Timestamp),
    "Cannot repay a loan for this user"
  )

  -- execute repay
  local refundQty, actualRepaidQty = repay.repayToPool(
    target,
    quantity,
    true
  )

  -- Rest of the function...
}
```

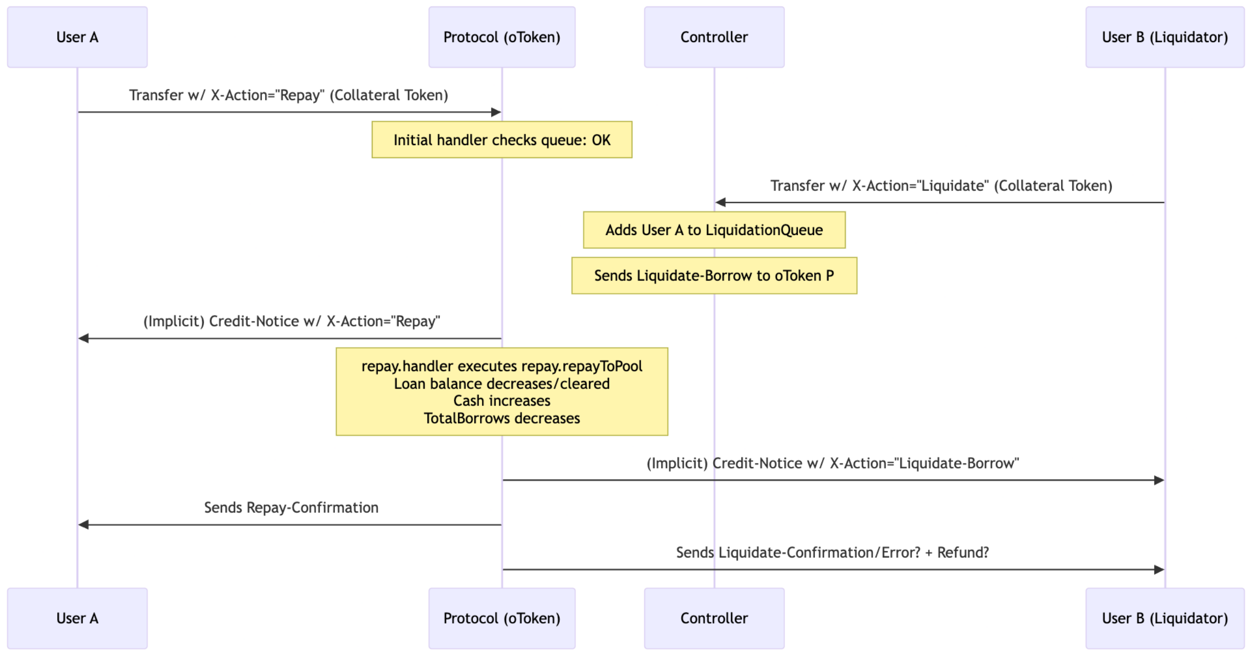
Meanwhile, the controller's liquidation process adds users to the `LiquidationQueue` during liquidation:

```lua
-- Add the target to the liquidation queue to lock further liquidations
table.insert(LiquidationQueue, target)

-- ... liquidation processing ...

-- Remove target from liquidation queue after completion
LiquidationQueue = utils.filter(
  function (v) return v ~= target end,
  LiquidationQueue
)
```

Impact: This race condition can lead to a an incorrect accounting scenario where both repayment and liquidation succeed simultaneously and is as follows:


User A
Protocol (oToken)
Controller
User B (Liquidator)
Transfer w/ X-Action="Repay" (Collateral Token)
Initial handler checks queue: OK
Transfer w/ X-Action="Liquidate" (Collateral Token)
Adds User A to LiquidationQueue
Sends Liquidate-Borrow to oToken P
(Implicit) Credit-Notice w/ X-Action="Repay"
repay.handler executes repay.repayToPool
Loan balance decreases/cleared
Cash increases
TotalBorrows decreases
(Implicit) Credit-Notice w/ X-Action="Liquidate-Borrow"
Sends Repay-Confirmation
Sends Liquidate-Confirmation/Error? + Refund?
User A
Protocol (oToken)
Controller
User B (Liquidator)

Recommendation: Apply queue checks to all handlers that modify user positions, including the internal `repay.handler`.

LiquidOps: Fixed in PR 25.

Spearbit: Fix looks good as it:

1. Merges `CollateralQueue` and `LiquidationQueue` into a single `Queue` which simplifies the logic.
2. It wraps the `repay.handler` and `mint.handler` with `queue.useQueue(oracle.withOracle(borrow))`.

5.3 Low Risk

5.3.1 Insufficient input validation in `isCollateralizedWithout()`

Severity: Low Risk

Context: assertions.lua#L77-L80

Description: The `isCollateralizedWithout()` function returns whether a position is still collateralized if a given `capacity` is removed from the position. The function should ensure `removedCapacity` is less than or equal to `position.capacity`. Otherwise, the subtraction would result in a negative value, causing incorrect results from the `bint.ule()` function since `bint.ule()` considers both inputs as unsigned integers.

Recommendation: Consider changing the function implementation as follows. The difference is in the first condition:

```
function mod.isCollateralizedWithout(removedCapacity, position)
 return bint.ult(removedCapacity, position.capacity) and
 bint.ule(position.borrowBalance, position.capacity - removedCapacity)
end
```

LiquidOps: Fixed in commit 7efb85d.

Spearbit: Verified.

5.3.2 Insufficient validation of returned values from `tonumber()`

Severity: Low Risk

Context: controller.lua#L259-L274

Description: Several uses of `tonumber()` are identified in this codebase to convert a value into a number. Since many of the input values are provided by the users, it would be necessary to validate them and ensure the results are valid numbers instead of special values to avoid unexpected results in subsequent operations.

Below are some edge cases of the `tonumber()` function:

- `tonumber("1e1000")` → `inf`.
- `tonumber("-1e1000")` → `-inf`.
- `tonumber(nan)` → `nan`. Note that the input `nan` is not a string.

These values, `inf`, `-inf`, and `nan`, pass the `~= nil` check, e.g., `inf ~= nil` is `true` in Lua.

Recommendation: Consider implementing a utility function that validates the input and the result of `tonumber()`, e.g.,

```
function stringToNumber(s)
  if type(s) ~= "string" then return false end
  value = tonumber(s)
  if type(value) ~= "number" then return false end

  -- Checks inf, -inf, nan
  if value == math.huge or value == -math.huge or x ~= x then return false end
  return value
end
```

Note that `assertions.isBintRaw()` , which is called from `isTokenQuantity()`, already ensures the result is not infinity or `nan`. However, the checks in, e.g., `controller.lua`, are less thorough, so it could be helpful to implement additional checks.

LiquidOps: Fixed in commit dc291a1.

Spearbit: Verified.

5.3.3 Potential error and precision loss in `getUSDDenominated()` due to use of `tostring()`

Severity: Low Risk

Context: controller.lua#L1148, controller.lua#L1163, oracle.lua#L148-L183

Description: The built-in `tostring()` function converts a value to a string. If the input is a floating point number, it does not necessarily output a string of the number in the decimal notation but possibly in the scientific notation. For example, `tostring(0.00001)` returns `"1e-05"` instead of `"0.00001"`.

The `getUSDDenominated()` function incorrectly assumes that the returned string is in the decimal notation, which would cause an error when converting the final string into `bint()`. Similarly, the `getFractionsCount()` could yield incorrect results.

Additionally the precision of the returned string is limited, note the missing decimals in this proof of concept:

```
val = 12345678.55554444
print(val) -- 12345678.555544
```

Recommendation: Consider implementing a helper function that uses format strings to convert numbers into strings. For example, `string.format("%0.16f", val)` would return a string in the decimal notation with 16 digits after the decimal point. Note that the precision is not arbitrary and has limitations since Lua numbers are stored as double-precision floating-point numbers, following the IEEE 754 standard.

Also, consider replacing the `tostring()` function with the helper function to ensure the converted string format is correct in other functions in the codebase. For example, the `utils.bintToFloat()` function performs string manipulation on a `tostring()` result, and therefore, it should be ensured that the initial string value has a correct format.

Note: check all instances, including `getFractionsCount()` and `getUSDDenominated()` in both `oracle.lua` and `controller.lua`.

LiquidOps: Fixed in commit a1277c8.

Spearbit: Verified.

5.3.4 Potential Division by Zero in Interest Calculation

Severity: Low Risk

Context: interest.lua#L109

Description: In the `updateInterest` function of the `interest.lua` contract, the yearly interest for a user is calculated using the expression:

```
local ownedYearlyInterest = bint.udiv(
  yieldingQty * mod.context.totalLent * mod.context.baseRate,
  mod.context.totalPooled * mod.context.rateMulWithPercentage
) + bint.udiv(
  yieldingQty * mod.context.initRate,
  mod.context.rateMulWithPercentage
)
```

This expression can result in a division by zero scenario when `mod.context.totalPooled` is zero. Although this is considered an edge case and unlikely under normal operations—since `totalPooled` is the sum of `TotalBorrows + Cash`—it's theoretically possible during the initial protocol bootstrap or after a mass withdrawal or liquidation event that zeroes out the pooled funds.

In such cases, the division operation will throw or produce undefined behavior due to an invalid denominator, which may lead to process failure or halt interest synchronization for users.

Recommendation: To ensure stability and robustness of the protocol, introduce a conditional check before the interest calculation. If `totalPooled == 0`, the dynamic interest component should be skipped, and only the fixed portion (`initRate`) should be applied. This protects the protocol from potential runtime errors during abnormal liquidity states.

LiquidOps: Fixed in PR 27.

Spearbit: Verified.

5.3.5 Incomplete Token Quantity Validation

Severity: Low Risk

Context: assertions.lua#L43-L53

Description: The `isTokenQuantity` function within `assertions.lua` is intended to validate token amounts before they are used in financial logic. However, its current implementation is overly permissive and fails to reject several invalid or potentially dangerous string formats.

Specifically, the function only checks if the first character is `"-"` when evaluating strings, but this allows malformed or misleading values like `"+-123"` or `"  -42"` to pass the validation. These values ultimately result in reverts or incorrect parsing when passed to `bint`, but the purpose of `isTokenQuantity` is to act as an early gatekeeper before such operations.

The function also accepts strings like `"asdf"` or `"123abc"` as valid inputs, as long as they do not start with `"-"`, due to the lack of strict numeric parsing.

Example of invalid acceptance:

```
assert.isTokenQuantity("+-123")   --> true (should be false)
assert.isTokenQuantity("asdf")    --> true (should be false)
assert.isTokenQuantity(" -1")     --> true (should be false)
```

These cases may cause subtle issues depending on the downstream logic or error handling of `bint`.

Recommendation: Strengthen the `isTokenQuantity` function by enforcing strict numeric validation using `tonumber` and explicit rejection of any malformed, negative, or non-numeric inputs.

This ensures only strictly positive, integer-formatted quantities are accepted and aligns the validation logic with the expectations of downstream arithmetic and financial processing.

LiquidOps: Fixed in PR 28.

Spearbit: Verified.

5.3.6 Redundant code in `borrow()`

Severity: Low Risk

Context: borrow.lua#L38, borrow.lua#L51-L67, borrow.lua#L72, interest.lua#L160-L162, interest.lua#L183, process.lua#L67-L92

Description: Function `borrow()` has code to initialize `Interests[]` and calculatie the acrued interest. However this code is already executed via the handler "borrow-loan-interest-sync-dynamic" that calls `interest.syncInterests()`.

Note: `updateInterest()` updates `TotalBorrows`, so the cached value of `local lent = bint(TotalBorrows)` on L38 should not be used on L72. However in `updateInterest()`, when the interest has just been updated, then `TotalBorrows` won't be updated.

Recommendation: Doublecheck the conclusion and remove the redundant code from `borrow()`.

LiquidOps: Fixed in PR 27.

Spearbit: Verified.

5.3.7 Queue mechanism lacks timeout and out-of-memory protection, risking permanent user locking

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: The `LiquidOps` protocol implements a queueing mechanism to prevent concurrent operations by the same user on their position. When a user initiates an operation like `borrowing`, `redeeming`, or transferring tokens, they are added to a queue until the operation completes.

```
function mod.useQueue(handle, errorHandler)
  return function (msg, env)
    -- default sender of the interaction is the message sender
    local sender = msg.From
    local isCreditNotice = msg.Tags.Action == "Credit-Notice"

    -- if the message is a credit notice, update the sender
    if isCreditNotice then
      sender = msg.Tags.Sender
    end

    -- update and set queue
    local res = mod.setQueued(sender, true).receive()

    -- if we weren't able to queue the user,
    -- then they have already been queued
    -- (an operation is already in progress)
    if not res then
      local err = "The sender is already queued for an operation"
```

The implementation uses `pcall` to catch errors during the operation and ensure users are removed from the queue afterward:

```
-- call the handler - here is the borrow, redeems and token transfers
local status, err = pcall(handle, msg, env)

-- unqueue and notify if it failed
mod
  .setQueued(sender, false)
  .notifyOnFailedQueue()
```

However, this approach may not catch all cases, as out-of-memory conditions or execution timeouts might not be caught by pcall, leading to execution termination before the unqueue operation runs. In such cases, users could remain permanently locked in the queue, unable to manage their positions.

Recommendation: Consider implementing a hybrid approach to address this issue while respecting message execution timing uncertainties (as delays in execution may occur):

1. When adding users to the queue, record the timestamp of when they were queued.
2. Users may call a function that identifies users who have been in the queue for an excessive period (like 2 hours).
3. Before forcibly removing a user from the queue, verify their current state in the system to ensure they're not in the middle of a valid operation.

Alternatively, an admin function can be created to manually release users from the queue in emergency situations, with appropriate safeguards and logging. This approach balances the need to prevent users from getting permanently stuck while respecting the asynchronous nature of message execution in the system.

LiquidOps: Acknowledged. Due to the nature of AOS, we can "admin remove" an user from the queue by executing a list filter manually.

```
Queue = utils.filter(
  function (addr) return addr ~= "address_to_remove" end,
  Queue
)
```

When transitioning to a DAO model, this will likely be subject to a vote.

Spearbit: Acknowledged.

5.3.8 Auction timer reset enables strategic discount advantage

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: The LiquidOps protocol implements a time-based discount system for liquidations. When a position first becomes eligible for liquidation, it is recorded in the Auctions table with the current timestamp, and liquidators receive a maximum discount. Over time, this discount decreases, incentivizing quick liquidations.

```
if bint.ult(position.liquidationLimit, position.borrowBalance) then
  local discount = 0

  -- AUCTION TRACKING
  -- Track when a liquidatable position is first discovered to enable
  -- time-based discount calculations. New auctions start with max discount.
  if Auctions[address] == nil then
    -- Record when this position became liquidatable
    Auctions[address] = msg.Timestamp
  elseif msg.Tags.Action == "Get-Liquidations" then
    -- Calculate the current discount based on time since discovery
    discount = tokens.getDiscount(address)
  end
```

However, the protocol also includes logic to remove positions from the Auctions table once they become healthy:

```
elseif Auctions[address] ~= nil then
  -- AUCTION CLEANUP
  -- If a previously liquidatable position is now healthy, remove it
  -- from the auctions list to ensure it's not incorrectly targeted.
  Auctions[address] = nil
end
```

This implementation creates a potential edge case that sophisticated liquidators could exploit:

1. A liquidator identifies a position with a reduced discount (auction has been active for some time).
2. The liquidator makes a small repayment to temporarily bring the position above the liquidation threshold.
3. This causes `Auctions[address]` to be set to `nil`.
4. When market conditions cause the position to fall below the threshold again, it gets a fresh timestamp and maximum discount.
5. The liquidator can then liquidate with the renewed maximum discount.

While this behavior doesn't compromise protocol security, it potentially allows strategic manipulation of the discount mechanism, especially during periods of price volatility.

Recommendation: Consider implementing one or more of the following measures:

1. Explicitly document this potential edge case in the protocol documentation.
2. Implement a cooling-off period before resetting the discount timestamp. For example, only reset the auction timer if a position has remained healthy for a minimum duration (e.g., 1 hour). This would prevent rapid cycling of positions in and out of liquidation status to game the discount system.

LiquidOps: Fixed in PR28.

Spearbit: The fix looks good. It uses `Handlers.remove` to safely cancel any pre-existing removal timer for the target, ensuring only the latest cooldown is active, and then employs `Handlers.once` to schedule the actual auction removal (`Auctions[target] = nil`) to execute only after the 3-hour cooldown has elapsed. Calling `Handlers.remove` when no handler exists for the target name results in no action being taken.

5.3.9 Missing oracle price data format validation

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: The LiquidOps protocol relies on external price data from an oracle to determine position health, calculate liquidation thresholds, and execute liquidations. The current implementation fetches price data but performs minimal validation on the returned data structure.

While the code checks if data is returned and validates timestamp freshness, it does not perform format validation on the actual price values. This could lead to issues if the oracle returns malformed price data, such as:

1. Multiple decimal points in price values.
2. Non-numeric characters in price strings.
3. Negative values or other invalid price inputs.

Invalid price data could lead to incorrect calculations in functions like position valuation, liquidation threshold determination, and collateral-to-debt ratio calculations, potentially causing unexpected behavior in the protocol.

```lua
-- Get price data for an array of token symbols
function oracle.sync()
  ---@type RawPrices
  local res = {}

  -- all collateral tickers
  local symbols = utils.map(
    ---@param f Friend
    function (f) return f.ticker end,
    Tokens
  )

  -- no tokens to sync
  if #symbols == 0 then return res end

  ---@type string|nil
  local rawData = ao.send({
    Target =  Oracle,
    Action = "v2.Request-Latest-Data",
    Tickers = json.encode(symbols)
  }).receive().Data

  -- no price data returned
  if not rawData or rawData == "" then return res end

  ---@type OracleData
  local data = json.decode(rawData)

  for ticker, p in pairs(data) do
    -- only add data if the timestamp is up to date
    if p.t + MaxOracleDelay >= Timestamp then
      res[ticker] = {
        price = p.v,
        timestamp = p.t
      }
    end
  end

  return res
end
```

Recommendation: Implement additional validation of oracle price data to ensure all values conform to expected formats before using them in calculations, including:

1. Each price has exactly one decimal point.
2. Validate against non-numeric characters in price values (except the one decimal point).

LiquidOps: Acknowledged. This is something we'll look into in the future.

Spearbit: Acknowledged.

5.3.10 Withdrawing the entire pool

Severity: Low Risk

Context: controller.lua#L623-L628, liquidate.lua#L167-L171, redeem.lua#L44-L48

Description: Function `Redeem()` prevents withdrawing the entire pool by using `ult()` but `liquidatePosition()` doesn't do that because it uses `ule()`. Leaving a small amount avoids division by zero errors.

Recommendation: Consider using `ult()` in `liquidatePosition()` too. In that case also consider adapting the `Liquidate` function in controller.lua#L626.

LiquidOps: Acknowledged. We'd prefer not to limit liquidations. We (as the protocol) will also have tokens in the pool by default anyway according to finding Security considerations of exchange-rate inflation in empty markets.

Spearbit: Acknowledged.

5.3.11 Handling of "Add-Friend" and "Remove-Friend" errors

Severity: Low Risk

Context: friend.lua#L10, friend.lua#L62, controller.lua#L313-L320, controller.lua#L370-L374

Description: The call to "Add-Friend" could potentially result in an error message. The "List" and "Unlist" functions in "controller.lua" don't catch these replies. This way the failure could escape attention and later on cause issues.

Recommendation: Consider catching the replies and sending an error message to the caller of "List" and "Unlist".

LiquidOps: Acknowledged.

Spearbit: Acknowledged.

5.3.12 List token check doesn't check everything

Severity: Low Risk

Context: controller.lua#L901-L916

Description: When "List"-ing a token it is also important that `X-..` tags are forwarded. This currently isn't verified.

Recommendation: If you think this is worth solving: it could be tested with a transfer, but then a balance of the token is required.

LiquidOps: Acknowledged. Unfortunately this cannot be verified, because transferring are the only action that support X- forwarded tags. We will manually check by verifying the token code when listing tokens.

Spearbit: Acknowledged.

5.3.13 Race condition in "List"

Severity: Low Risk

Context: controller.lua#L214-L228, controller.lua#L277, controller.lua#L307, controller.lua#L324-L330

Description: Suppose a token "List" is send and right after that a second "List" request, with different parameters is send. There is a duplicate check on L225-L228, however after that there are several external actions. On L325-L330 the token is inserted in the `Token` table. Between the check and the insertion the second "List" request could be accepted and it depends on timing what values are used. This scenario isn't very likely because "List" is an authorized function.

Recommendation: Consider having some kind of lock, however avoid lock situation where the token can't be listed anymore.

LiquidOps: Acknowledged.

Spearbit: Acknowledged.

5.3.14 Duplicate tickers cause issues

Severity: Low Risk

Context: controller.lua#L225-L228, oracle.lua#L42-L46

Description: The oracle uses `tickers` to retrieve information so it doesn't support duplicate tickers. However the `List` function does allow duplicate tickers, because it only does duplicate checks on addresses.

Note: `List` is authorized so normally duplicate tickers will not be listed.

Recommendation: Consider also checking for duplicate tickers in the "List" function of "controller.lua". Once a new version of an oracle exists that supports token address, consider changing the code to use that.

LiquidOps: Acknowledged. Once a better oracle that supports token addresses is available on AO, we'll switch to that. (We are aware of some projects that plan to do this).

Spearbit: Acknowledged.

5.3.15 `ValueLimit` not checked everywhere

Severity: Low Risk

Context: borrow.lua#L14-L18, repay.lua#L27-L31, mint.lua#L17-L21, redeem.lua#L38-L42

Description: `Repay()` doesn't check `quantity <= actualRepaidQty`. For comparison, the functions `mint()`, `redeem()` and `borrow()` do have such a check. Also refunds don't check this `ValueLimit`. This could be important in combination bugs in the code.

Recommendation: Consider adding a check for `actualRepaidQty` in `Repay()`. Consider adding a limit for refunds to prevent large refunds when a bug occurs.

LiquidOps: Acknowledged. The value limit was only added for the beta, so this will probably be revamped.

Spearbit: Acknowledged.

5.3.16 Unprotected `JSON` decoding can lead to unexpected errors in the protocol

Severity: Low Risk

Context: pool.lua#L39, controller.lua#L103, controller.lua#L1039, oracle.lua#L53

Description: The protocol contains instances where `json.decode` is used without error handling protection, which could lead to runtime errors if malformed JSON data is encountered:

```lua
-- In src/controller.lua
local data = json.decode(rawData)
```

```lua
-- In src/borrow/pool.lua
Friends = Friends or json.decode(ao.env.Process.Tags.Friends) or {}
```

```
-- In src/controller.lua
local marketPositions = json.decode(market.Data)
```

When `json.decode` encounters invalid JSON data, it throws an error that terminates the current execution flow. This can cause unexpected behavior, particularly in critical functions like price data processing in the oracle service or when initializing protocol configuration data.

While the impact of these issues may typically be limited since most inputs come from trusted sources, any error in those operations may disrupt protocol operations.

Recommendation: Use Lua's protected call (`pcall`) when parsing JSON to prevent potential runtime errors, which is already used in `oracle.lua:mod.setup()`.

LiquidOps: Fixed in PR 28 partially.

Spearbit: Verified.

5.3.17 Missing input parameter checks

Severity: Low Risk

Context: controller.lua#L470, controller.lua#L610, liquidate.lua#L20-L24, liquidate.lua#L40, liquidate.lua#L42, liquidate.lua#L95, liquidate.lua#L136, oracle.lua#L20

Description: Throught the code, input parameters are validated:

- `assertions.isAddress()` for addresses;
- `assertions.isTokenQuantity()` for quantities;

However this isn't done in all situations. For fields that are forwarded to other message, it is useful to check early to save processing power/gas.

Recommendation: Consider adding checks on input parameters:

- `assertions.isAddress()` checks:
  - `target` in `liquidatePosition()`.
  - `liquidator` in `liquidateBorrow()`.
  - `target` in `liquidateBorrow()`.
  - `msg.Tags["X-Reward-Market"]` in `liquidateBorrow()`.
  - `target` in `syncInterestForUser()`.
  - `Oracle` in `oracle.setup()`.
  - `liquidator` in "liquidate" of `controller.lua`.
  - `rewardToken` in "liquidate" of `controller.lua`.
  - `msg.Tags.Quantity` in "liquidate" of `controller.lua`.
- `assertions.isTokenQuantity()` checks:
  - `msg.Tags["X-Reward-Quantity"]` in `liquidateBorrow()`.
  - `msg.Tags.Quantity` in `liquidate.refund()`.

Note: doublecheck that any additional error checks are handled well with regards to refunds.

5.3.18 No check `inTokenData` / `outTokenData` / `availableRewardQty` are found

Severity: Low Risk

Context: controller.lua#L545-L580

Description: In function "liquidate" a scan is made over all positions in order to find values for `inTokenData` / `outTokenData` / `availableRewardQty`. However there is no check these values have indeed been found. If these values haven't been found the rest of the code executes with empty values.

Recommendation: Consider checking values for `inTokenData` / `outTokenData` / `availableRewardQty` have indeed been found.

LiquidOps: Fixed commit aef4de12.

Spearbit: Verified.

5.3.19 Missing default values in configuration

Severity: Low Risk

Context: pool.lua#L8, pool.lua#L39, pool.lua#L43, oracle.lua#L20, token.lua#L5, token.lua#L8

Description: In several situations, default values are defined in case a configuration parameter is missing. However this isn't done everywhere and this could lead to incorrectly configured code.

Recommendation: Consider using default values for the accompanying code. If no relevant default value can be determined consider stopping with an error.

LiquidOps: Fixed commit e097ca10.

Spearbit: Verified.

5.3.20 Function `updateInterest()` doesn't check `Loans[address] == nil`

Severity: Low Risk

Context: interest.lua#L102

Description: Function `updateInterest()` doesn't explictly check for the situation where `Loans[address] == nil`. This could cause an error in `bint()`. Although this situation normally shouldn't happen it is safe to make sure.

Recommendation: Consider handling the situation where `Loans[address] == nil`.

LiquidOps: Fixed in PR 27.

Spearbit: Verified.

5.3.21 Function `position()` doesn't check `TotalSupply == 0`

Severity: Low Risk

Context: position.lua#L37-L40

Description: The function `position()` doesn't explicitly check for `TotalSupply == 0`. Although this is unlikely, it is safe to check this to prevent a division by 0.

Recommendation: Consider handling the case where `TotalSupply == 0`.

LiquidOps: Fixed commit 50635d29.

Spearbit: Verified.

5.3.22 The `update()` function can execute any code

Severity: Low Risk

Context: updater.lua#L2-L16

Description: The `update()` function can execute any code. If there is a bug in the code it could make the `oToken` unusable and lock the tokens inside.

Recommendation: Consider adding some checks after `install()` to make sure basic functionality still works, perhaps the `update()` function itself.

LiquidOps: Acknowledged. We are planning to add extra security that prevents one wallet by itself to run an update and have a multisig-like system for it.

Spearbit: Acknowledged.

5.3.23 Testing suite limitations may affect codebase robustness

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: The current testing suite for the `LiquidOps` protocol has several limitations that may hinder comprehensive validation and may conceal potential vulnerabilities.

While unit tests exist for some functionalities, there are areas where the testing suite can be improved:

1. Untested Core Modules: Some functionalities remain untested. For example, the test file `__tests__/borrow.test.ts` explicitly marks entire sections related to Position, Repaying and Interests logic with `it.todo`, indicating a lack of tests for those modules.
2. Lack of Coverage Metrics: The AO/AOS ecosystem seems to lack tools for measuring test coverage. This absence makes it difficult to quantify and assess how much of the Lua codebase is covered by the existing tests, potentially masking untested code paths and logic within the tested files themselves.
3. Absence of Integration, Multi-Actor and Stateful Tests: The suite primarily consists of unit tests focusing on individual actions in isolation. It currently lacks *Integration*, *Multi-Actor*, and *Stateful Tests*, the absence of which may mean other vulnerabilities remain undiscovered. These test types are particularly crucial due to the unique message-passing architecture of AO and AOS, where operations frequently involve interactions across multiple different processes.

Recommendation: Enhance the testing suite significantly to provide greater assurance of the protocol's robustness and security. It's recommended to prioritize writing unit tests for the currently uncovered modules and to develop integration and multi-actor tests.

Furthermore, you can consider launching the protocol on a testnet or conducting a closed beta program as these expose the system to real-world usage patterns and may uncover edge cases that cannot be easily caught solely through testing.

LiquidOps: Acknowledged. We are planning to improve coverage and add a complete test suite once `AO` supports it. With their upcoming `HyperBeam` platform, this is already being implemented.

Spearbit: Acknowledged.

5.3.24 Input for `bint()` should not be `nil`

Severity: Low Risk

Context: controller.lua#L109-L110, controller.lua#L554, controller.lua#L558-L559, controller.lua#L610

Description: When converting to `bint()`, its important that the input value isn't `nil`, otherwise it will error. This is ensured in serveral places by using a default value, however on other places its missing.

Recommendation: Consider using default values for the accompanying code. If no relevant default value can be determined consider stopping with an error.

LiquidOps: Fixed commit 557c0665.

Spearbit: Verified.

5.4 Gas Optimization

5.4.1 Redundant check for `LiquidationQueue`

Severity: Gas Optimization

Context: controller.lua#L511-L522, controller.lua#L660-L666

Description: The "Liquidate" function checks the `LiquidationQueue` twice. Between the first and the second call there are no external messages send/received, so the second check isn't necessary.

Recommendation: Consider removing the second check.

LiquidOps: Fixed commit 84c7992e.

Spearbit: Verified.

5.4.2 Quantity can be verified before position data calculation to allow early failure in borrow and redeem functions

Severity: Gas Optimization

Context: borrow.lua#L14

Description: In the `LiquidOps` protocol's `borrow` function, the quantity validation occurs after fetching the user's position data:

```lua
-- get position data
local pos = position.globalPosition(account, oracle)

-- verify quantity
assert(
  assertions.isTokenQuantity(msg.Tags.Quantity),
  "Invalid borrow quantity"
)
```

The quantity validation is independent of the position data and doesn't rely on any calculated values from the position. This means the protocol is performing an expensive operation (position data calculation) before validating the input quantity, which could be invalid.

If the quantity validation fails, the protocol would have wasted resources calculating position data unnecessarily. By moving the validation earlier in the function flow, the protocol could fail fast when invalid input is provided, saving computational resources and improving gas efficiency.

Similar validation can be adjusted in other functions, like `redeem`, where the same pattern seems to exist.

Recommendation: Move the quantity validation before the position data calculation to allow early failure for invalid inputs:

```lua
-- verify quantity
assert(
  assertions.isTokenQuantity(msg.Tags.Quantity),
  "Invalid borrow quantity"
)

-- get position data
local pos = position.globalPosition(account, oracle)
```

LiquidOps: Fixed in PR 29.

Spearbit: Verified. The quantity is now checked before position data calculation.

5.4.3 Redundant boolean operation

Severity: Gas Optimization

Context: process.lua#L319

Description: In `process.lua`, at L319, the expression `msg.Action and msg.Action or "Unknown"` can be simplified to `msg.Action or "Unknown"` as they are equivalent.

Recommendation: Consider implementing the above suggestion.

LiquidOps: Fixed in commit 7a14723.

Spearbit: Verified.

5.5 Informational

5.5.1 Dead Code in Cooldown Enforcement Logic

Severity: Informational

Context: cooldown.lua#L42-L45, cooldown.lua#L62-L79

Description: In `cooldown.lua`, the cooldown enforcement logic in both the `gate` and `refund` functions contains conditional blocks that check if `msg.Tags.Action == "Credit-Notice"` in order to modify the sender address or to route specific refund behavior. However, these checks are effectively dead code due to how the cooldown system is integrated in the protocol.

The `controller-cooldown-gate` pattern is explicitly applied only to user-level actions such as `"Borrow"` and `"Redeem"`, as established in the protocol's handler routing layer. These actions never carry `Action = "Credit-Notice"`, which is reserved for token transfer messages.

Since `Credit-Notice` is not routed through the cooldown system, these blocks are unreachable and add unnecessary complexity.

Recommendation: Remove the `if msg.Tags.Action == "Credit-Notice"` branches from both the `gate` and `refund` functions in `cooldown.lua`. This will clean up the logic, reduce cognitive overhead for future maintainers, and eliminate confusion around dead paths.

LiquidOps: Fixed in PR 29.

Spearbit: Verified.

5.5.2 Incorrect Return Type Annotations in Middleware Functions

Severity: Informational

Context: queue.lua#L52, oracle.lua#L125

Description: In both `queue.lua` and `oracle.lua`, the higher-order functions `useQueue` and `withOracle` are incorrectly annotated with `@return HandlerFunction`. These functions return other functions (i.e., closures) that conform to the `HandlerFunction` type, meaning the correct type annotation should reflect that the return value is a function itself, not a direct handler result.

The current form:

```
---@return HandlerFunction
```

incorrectly implies that `useQueue` and `withOracle` immediately return a value compatible with `HandlerFunction`, rather than a function *that when called* conforms to `HandlerFunction`. This distinction is critical for accurate documentation, type checking, and static analysis.

Specifically:

- In queue.lua, useQueue wraps a handler with queuing logic and returns a middleware handler function.
- In oracle.lua, withOracle decorates a handler with real-time oracle data injection and returns a callable that fits HandlerFunction.

Recommendation: Update the return type annotations for both useQueue and withOracle to:

```
---@return function:HandlerFunction
```

This explicitly communicates that the return value is a function that matches the HandlerFunction type signature. It enhances tooling support and readability, ensuring middleware is correctly understood as a functional wrapper, not a direct handler execution.

LiquidOps: Acknowledged. Unfortunately this breaks the sumneko/Lua extension's types we use.

Spearbit: Acknowledged.

5.5.3 Use utils.filter() to Update the Handlers.coroutines Table

Severity: Informational

Context: process.lua#L335-L339

Description: In Lua, when an element is removed from an array table, the next element is shifted into that position. Therefore, if the table is being iterated at the same time, the next element will be skipped. For example:

```
T = {}
table.insert(T, { data = "X" })
table.insert(T, { data = "X" })

for i, x in ipairs(T) do
  if x.data == "X" then
    table.remove(T, i)
  end
end

T -- still has an element { { data = "X" } }
```

In process.lua, since the Handlers.coroutines table is being iterated while the code removes the elements, some elements may fail to be removed. Additionally, using Table.remove() while iterating the array has a time complexity of O(n^2), while the custom util.filter() function is O(n).

Recommendation: Consider using the utils.filter() function to remove the elements.

LiquidOps: Fixed in commit f9ed6c6.

Spearbit: Verified.

5.5.4 Unused Function spawnProtocolLogo

Severity: Informational

Context: controller.lua#L931

Description: Within controller.lua, the function spawnProtocolLogo is defined to dynamically generate a LiquidOps-themed SVG image incorporating a provided collateral logo. This function constructs a composite SVG and sends it to the AO process to be registered as a spawned asset. However, this function is never invoked anywhere in the codebase.

The intended use seems to be within the token listing path—presumably to decorate newly listed tokens with protocol-specific visuals—but in practice, the listing logic directly uses the msg.Tags.Logo or defaults to info.Tags.Logo, bypassing any dynamic rendering or invocation of spawnProtocolLogo.

This makes spawnProtocolLogo dead code, which adds unnecessary cognitive overhead and potential confusion for future maintainers. Moreover, maintaining unused logic in critical protocol components (like the controller) introduces an unnecessary surface area for audit and testing.

Recommendation: Remove the spawnProtocolLogo function from controller.lua as it is not used anywhere in the codebase and has no active call sites. If dynamic logo generation is needed in the future, it should be reintroduced in a separate utility module or injected explicitly into the token listing flow with a clear toggle or configuration flag.

LiquidOps: Fixed in PR 29.

Spearbit: Verified.

5.5.5 Potential division by 0 in function getUnderlyingWorth()

Severity: Informational

Context: rate.lua#L9-L27

Description: In function getUnderlyingWorth() there is a potential division by 0 if totalSupply==0. This is unlikely to occur in practice, because in that situation there are no tokens minted yet, however it better to be safe.

Recommendation: Consider check for totalSupply==0 and returning a plausible value.

LiquidOps: Fixed in commit 69263b7a.

Spearbit: Verified.

5.5.6 Inconsistent Type Handling in Interest Updates

Severity: Informational

Context: interest.lua#L129

Description: The updateInterest function in interest.lua performs arithmetic operations on Interests[address].value, which is sometimes treated as a string and other times cast directly into a bint. While this works in practice due to Lua's dynamic typing and coercion behavior, it leads to type inconsistency and unnecessary risk in critical arithmetic paths.

For instance, the following code appears to operate correctly:

```
Interests[address] = {
  value = tostring(Interests[address].value + interestAccrued),
  updated = timestamp
}
```

However, Interests[address].value is originally stored as a string (e.g., "1234"), which is implicitly treated as a number during the addition. This works as a proof of concept but introduces undefined behavior if the value format changes (e.g., stringified non-numeric values) or if future refactors tighten type safety.

The code also re-calls bint(Interests[address].value) after having already created a bint version earlier as interestQty, causing redundant conversions.

Recommendation: Standardize the data flow by ensuring all arithmetic values are explicitly cast and kept as bint objects throughout the calculation process. This improves reliability, avoids silent coercion bugs, and clarifies intent.

This change ensures type consistency and eliminates the ambiguity of treating .value as a numeric string. It also aligns the interest calculation logic with the strictness required for financial correctness.

LiquidOps: Fixed in PR 27 with the revamped interest accrual changes.

Spearbit: Verified.

5.5.7 Unnecessary default value after `bint()`

Severity: Informational

Context: interest.lua#L103

Description: Function `updateInterest()` assigns `interestQty` and uses a default value in case `bint(...)` returns `nil`. However `bint()` error if it can't convert the input value.

Recommendation: Consider removing the default value to simlify the code:

```diff
- local interestQty = bint(Interests[address].value) or mod.context.zero
+ local interestQty = bint(Interests[address].value)
```

LiquidOps: Fixed with the interest accrual revamp.

Spearbit: Verified.

5.5.8 Tag "X-Action" = "Liquidate" missing in Javascript library

Severity: Informational

Context: controller.lua#L453-L455

Description: In the library `LiquidOps-JS` the tag "X-Action" = "Liquidate" is missing. Due to this liquidations won't be triggered via this library and the send tokens will stay stuck in `controller`, see finding TBD.

Background information: when a liquidator initiates a liquidation, they transfer tokens to the controller. The transfer message is sent to the token that will be used to pay for the user's outstanding loan. Any messages with the "X-" prefix are forwarded, so the liquidator also needs to add an "X-Action" = "Liquidate" tag to their transfer. This is forwarded in the "Credit-Notice" message, so it must have "X-Action" = "Liquidate".

Recommendation: Add the tag "X-Action" = "Liquidate" to the library `LiquidOps-JS`.

LiquidOps: Solved in PR 10 of LiquidOps-JS.

Spearbit: Verified.

5.5.9 `Controller` doesn't handle "Debit-Notice" messages

Severity: Informational

Context: controller.lua#L704-L711, controller.lua#L735-L737, process.lua#L108-L113

Description: In `process.lua` there is a handler to ignore "Debit-Notice" messages. However `controller.lua` doesn't have this. A "Debit-Notice" message could be received after doing a transfer on L705 or L735.

Recommendation: Doublecheck if "Debit-Notice" messages should be handled by `controller.lua` too and if so add code to do that.

LiquidOps: It is not necessary to add a handler for `debit-notices`, simply to ignore them, because aos processes by default don't do anything with a message that doesn't match a handler. We only need this in the `oToken` code (`process.lua`), because we modified the handlers to send an error message if the incoming message did not match any handlers (the empty `debit-notice` handler prevents this).

Spearbit: Acknowledged.

5.5.10 Typos, Incorrect comments and missing function parameter documentation

Severity: Informational

Context: pool.lua#L54, controller.lua#L469, controller.lua#L958, controller.lua#L1133, liquidate.lua#L23, mint.lua#L85, rate.lua#L6, rate.lua#L15, utils.lua#L57, utils.lua#L333

Description: In the course of the security review, several instances of typos and incorrect comments were found in the codebase. The following instances were identified:

1. In controller.lua:L1133, "accouting" should be "accounting".
2. In rate.lua:L6, "wroth" should be "worth".
3. In rate.lua:L15, "than" should be "then".
4. In utils.lua:L57, "whetehr" should be "whether".
5. In utils.lua:L333, the comment is incomplete. It can be changed to: `Turn a floating point number into a bigint, scaled by an optional multiplier (default 100000).`
6. In controller.lua:L469, "posisition" should be "position".
7. In pool.lua:L54, "anc" should be "and".
8. In mint.lua:L85 there is no `---@param` for msg.
9. In controller.lua:L958 the comment should be `if the time passed is higher than the discount interval.`
10. In liquidate.lua:L23 "tartget" should be "target".

Recommendation: It's recommended to fix these instances of typos and incorrect comments.

For thorough documentation, ensure that all functions have:

- Accurate descriptions of what they do.
- Complete `@param` annotations for all parameters.
- Proper `@return` type annotations.
- Correct spelling throughout comments and documentation.

LiquidOps: Fixed in commit 7645fa3d. Note: for 8), it has `---@type HandlerFunction`, that should handle the Lua types/params/return for the entire function.

Spearbit: Verified.

5.5.11 Code in `getUSDDenominated()` can be simplified

Severity: Informational

Context: controller.lua#L1168-L1175, oracle.lua#L173-L180

Description: The code to calculate `denominated` in `getUSDDenominated()` can be simplified, which makes it easier to understand.

Recommendation: Consider changing the code in both `oracle.lua` and `controller.lua` to the following.

```lua
local wholeDigits = string.len(denominated) - fractions
denominated = denominated .. string.rep("0", denominator)
denominated = string.sub(denominated, 1, wholeDigits + denominator)
```

LiquidOps: Fixed in commit 3f4d26ef.

Spearbit: Verified.

5.5.12 Unused default value of `Denomination`

Severity: Informational

Context: token.lua#L10-L13

Description: When setting up a new oToken, the `CollateralDenomination` and `Denomination` values are set as follows:

```
-- the wrapped token's denomination
CollateralDenomination = tonumber(ao.env.Process.Tags["Collateral-Denomination"] or 0) or 0

Denomination = CollateralDenomination or 12
```

Note that if `CollateralDenomination` is set to a default value of 0, `Denomination` will be set to 0 as well since `0 or 12 == 0`. `Denomination` will never be set to a default value of 12.

Recommendation: Consider either changing token.lua#L11 to

```
CollateralDenomination = tonumber(ao.env.Process.Tags["Collateral-Denomination"] or 12) or 12
```

or changing token.lua#L13 to

```
if CollateralDenomination == 0 then Denomination = 12 else Denomination = CollateralDenomination
```

LiquidOps: Fixed in commit 876345a2.

Spearbit: Verified.

5.5.13 Several TODOs left in the code

Severity: Informational

Context: controller.lua#L725-L726, mint.lua#L47-L54, scheduler.lua#L61

Description: There are several TODOs left in the code.

Recommendation: Check the TODOs and resolve them.

LiquidOps: Acknowledged.

Spearbit: Acknowledged.

5.5.14 Several authorized functions are not called from `controller.lua`

Severity: Informational

Context: config.lua#L7, reserves.lua#L7, reserves.lua#L39, process.lua#L154-L158, process.lua#L177-L186

Description: Several authorized functions are not called from `controller.lua`. This means they will have to be called directly on the `oToken`. According to the project the plan is to introduce some kind of governance model, which would call these functions.

Recommendation: At least document the way these functions should be called.

LiquidOps: Fixed in commit a05206c5.

Spearbit: Verified.

5.5.15 Misleading function name

Severity: Informational

Context: repay.lua#L101-L114, liquidate.lua#L32-L36, liquidate.lua#L60-L62

Description: The function name `canRepayExact()` is misleading. It seem to require `quantity == borrowBalance + interestBalance` however the implementation is `quantity < borrowBalance + interestBalance`.

Recommendation: Check the intention of `canRepayExact()`. If it should ensure exact payment, update to logic to do so. Otherwise change the function name and the related comments in both `repay.lua` and `liquidate.lua`.

LiquidOps: Acknowledged, we will look into this in the future.

Spearbit: Acknowledged.

5.5.16 `Controller.lua` doesn't send `Raw-Error`

Severity: Informational

Context: controller.lua#L695-L702, controller.lua#L698-L702, controller.lua#L760-L769, utils.lua#L352-L356

Description: In error messages, the `oToken` code includes the `Raw-Error`. However `controller.lua` doesn't add/support that. Having the `Raw-Error` would make troubleshooting easier. It doesn't forward the `Raw-Error` from the loan liquidation result. It also doesn't add it in an error in "liquidate".

Recommendation: Consider also supporting `Raw-Error` messages in `controller.lua`.

LiquidOps: Acknowledged.

Spearbit: Acknowledged.

5.5.17 Use of `msg.From` could be made more readable

Severity: Informational

Context: repay.lua#L33-L41, repay.lua#L69-L74, liquidate.lua#L60-L70, liquidate.lua#L91-L97, process.lua#L220-L229, mint.lua#L69-L74

Description: Refunds of collateral tokens use `Target = msg.From`, which is somewhat difficult to understand.

Recommendation: Consider using `CollateralID` instead of `msg.From`, where it is sure `msg.From == CollateralID`.

LiquidOps: Fixed in commit 8c42e94d.

Spearbit: Verified.

5.5.18 Javascript library uses extra tags

Severity: Informational

Context: transfer.lua#L55-L59

Description: The javascript library adds the tag: `"Protocol-Name" = "LiquidOps"`. However the `LUA` code doesn't do this. To be consistent, it might useful to add `"Protocol-Name" = "LiquidOps"` to all messages.

Recommendation: Consider adding the tag: `"Protocol-Name" = "LiquidOps"` to all messages.

LiquidOps: Acknowledged.

Spearbit: Acknowledged.

5.5.19 Description of data fields incomplete

Severity: Informational

Context: pool.lua#L37-L39, controller.lua#L23-L25

Description: The description of several data field is incomplete.

Recommendation: Add documention to `Friends` and `Tokens` to indicate they have the following fields:

- `id:string (address)`.
- `ticker:string`.
- `oToken:process`.
- `denomination:number`.

LiquidOps: Fixed in commit 8366be1d.

Spearbit: Verified.

5.5.20 `ao.env.Process.Owner` could be made clearer

Severity: Informational

Context: queue.lua#L12, process.lua#L129-L143, process.lua#L154-L158, process.lua#L160-L167, process.lua#L177-L186

Description: `process.lua` and `queue.lua` use `ao.env.Process.Owner`, which is not directly clear. However `ao.env.Process.Owner` is the `controller`.

Recommendation: Consider replacing `ao.env.Process.Owner` with a variable like `Controller`.

LiquidOps: Fixed in commit 68842b8c.

Spearbit: Verified.

5.5.21 Code structure

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The code contains a lot of duplication which makes maintenance more difficult. This is mainly due to the fact that `controller.lua` is one large file because of deployment details. According to the project there are ways to split this file.

Recommendation: Consider splitting `controller.lua` and combine the code where possible with the code from `process.lua` and the `oTokens`.

LiquidOps: Acknowledged. This is planned in the future.

Spearbit: Acknowledged.

5.5.22 Documentation is limited

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The documentation is currently limited, which makes it more difficult to use, maintain and review the code. Especially the division between `controller.lua`, which is run on `AOS` and the rest of the code, which run in a seperate `WASM` module isn't clear.

Recommendation: Consider creating more documentation:

- Explain the division between `controller.lua` and the rest of the code;
- Add diagrams that explain the flows of message;
- Add specifications for the messages and error messages that can be received and sent;
- Add a table to show which actions map to which functions. See example below;
- An overview of the changes made to `ao.lua`;
- A list of the deployed code including addresses.

Name	Action	X-Action	From	Sender	Function	Error handler
liquidate-borrow	Credit-Notice	Liquidate-Borrow	CollateralID	controller	liquidateBorrow	refund
...	...	...	...	...	...	...

LiquidOps: Acknowledged. This is planned in the future along with a whitepaper/lightpaper.

Spearbit: Acknowledged.