GUARDIAN

Foil

LP Vault

Security Assessment

January 13th, 2025

Summary

Audit Firm Guardian

Prepared By Daniel Gelfand, Nicholas Chew, Zdravko Hristov,
Osman Ozdemir, Mark Jonathas, Michael Lett

Client Firm Foil

Final Report Date January 13, 2025

Audit Summary

Foil engaged Guardian to review the security of its Vault, providing liquidity across the epoch's price range. From the 19th of November to the 27th of November, a team of 6 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Issues Detected Throughout the engagement 4 High/Critical issues were uncovered and promptly remediated by the Foil team. Several issues impacted the fundamental behavior of the protocol, following their remediation Guardian believes the protocol to uphold the functionality described for the Vault.

Security Recommendation Given the number of High and Critical issues detected, Guardian supports a secondary security review of the Vault at a finalized frozen commit. Furthermore, the Foil team should increase testing with various settlement scenarios which may present opportunities to DoS the Vault's operations.

For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

🔗 Blockchain network: **Ethereum**

✅ Verify the authenticity of this report on Guardian's GitHub: https://github.com/guardianaudits

📊 Code coverage & PoC test suite: https://github.com/GuardianAudits/foil-fuzzing

Table of Contents

Project Overview

Project Summary

Project Name	Foil
Language	Solidity
Codebase	https://github.com/foilxyz/foil
Commit(s)	Initial commit: 5b3416a28dfaa24ba3844e10081e55425d0a286a Final commit: faf7d3a296ad630ce0de70e84c2f067f59970286

Audit Summary

Delivery Date	January 13, 2025
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
● Critical	2	0	0	0	0	2
● High	2	0	0	0	0	2
● Medium	14	0	0	3	0	11
● Low	17	0	0	9	0	8

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: *High*	Impact: *Medium*	Impact: *Low*
Likelihood: *High*	● Critical	● High	● Medium
Likelihood: *Medium*	● High	● Medium	● Low
Likelihood: *Low*	● Medium	● Low	● Low

Impact

High	Significant loss of assets in the protocol, significant harm to a group of users, or a core functionality of the protocol is disrupted.
Medium	A small amount of funds can be lost or ancillary functionality of the protocol is affected. The user or protocol may experience reduced or delayed receipt of intended funds.
Low	Can lead to any unexpected behavior with some of the protocol's functionalities that is notable but does not meet the criteria for a higher severity.

Likelihood

High	The attack is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount gained or the disruption to the protocol.
Medium	An attack vector that is only possible in uncommon cases or requires a large amount of capital to exercise relative to the amount gained or the disruption to the protocol.
Low	Unlikely to ever occur in production.

Audit Scope & Methodology

Methodology

Guardian is the ultimate standard for Smart Contract security. An engagement with Guardian entails the following:

- Two competing teams of Guardian security researchers performing an independent review.
- A dedicated fuzzing engineer to construct a comprehensive stateful fuzzing suite for the project.
- An engagement lead security researcher coordinating the 2 teams, performing their own analysis, relaying findings to the client, and orchestrating the testing/verification efforts.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts. Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

Invariants Assessed

During Guardian’s review of Foil, fuzz-testing with Echidna was performed on the protocol’s main functionalities. Given the dynamic interactions and the potential for unforeseen edge cases in the protocol, fuzz-testing was imperative to verify the integrity of several system invariants.

Throughout the engagement the following invariants were assessed for a total of 5,000,000+ runs with a prepared Echidna fuzzing suite.

ID	Description	Passed	Remediation	Run Count
GLOBAL-01	The price of vGAS should always be in range of the configured min/max ticks.	✅	✅	5M+
GLOBAL-02	There should never be any liquidity outside of the [min, max] range of an epoch.	✅	✅	5M+
GLOBAL-03	The amount of vETH in the system, position manager & swap router should equal the max supply	✅	✅	5M+
GLOBAL-04	The amount of vGAS in the system, position manager & swap router should equal the max supply.	✅	✅	5M+
TRADE-01	The debt of a position should never be > the collateral of the position.	✅	✅	5M+
TRADE-02	Long positions have their debt in vETH and own vGAS	❌	✅	5M+
TRADE-03	Short positions have their debt in vGAS and own vETH.	❌	✅	5M+
TRADE-04	Trader should never have both borrowedVGas and borrowedVEth be non-zero.	✅	✅	5M+
TRADE-05	Trader's pending loss in ETH-worth should never exceed collateral put down (should never be in negative equity)	❌	✅	5M+

Invariants Assessed

ID	Description	Passed	Remediation	Run Count
TRADE-06	after creating/modifying trade position, the depositedCollateralAmount > debtValue - tokensValue	✅	✅	5M+
TRADE-07	After creating a trade position deposited collateral should be non-zero	✅	✅	5M+
TRADE-08	After user closes a trade position, no vGAS, vETH, borrowed vGAS, borrowed vETH	✅	✅	5M+
TRADE-09	After creating a trader position, positionSize is non-zero.	✅	✅	5M+
TRADE-10	createTradePosition should create a unique positionId	✅	✅	5M+
LIQUID-01	The debt of a position should not be > the collateral of the position.	✅	✅	5M+
LIQUID-02	A open LP position should not own any vETH or vGAS.	✅	✅	5M+
LIQUID-03	After all LP positions have been closed, for the remaining trader positions: net shorts == net longs.	✅	✅	5M+
LIQUID-04	Position.depositedCollateralAmount should be at least the required collateral for their position if their position turned into a Trade type.	❌	❌	5M+
LIQUID-05	QuoteLiquidityPositionTokens should match how many tokens are borrowed and how much liquidity is added after creating an LP position with createLiquidityPosition	✅	✅	5M+
LIQUID-06	After creating an LP position, liquidity in the Uni pool increases	✅	✅	5M+

Invariants Assessed

ID	Description	Passed	Remediation	Run Count
LIQUID-07	After increasing an LP position, liquidity in the Uni pool increases	✅	✅	5M+
LIQUID-08	After decrease an LP position, liquidity in the Uni pool decreases	❌	✅	5M+
LIQUID-09	After partial decrease an LP Position, should not get InsufficientColateral revert (unexpected in this case)	❌	✅	5M+
LIQUID-10	createLiquidityPosition should create a unique positionId	✅	✅	5M+
SETTLE-01	It should always be possible to settle all positions after the epoch is settled.	❌	✅	5M+
SETTLE-02	After settlement with settlePosition, position should not have any borrowedvETH nor borrowedVGAS, and no vGAS nor vETH (cleared out position)	✅	✅	5M+
SETTLE-03	Settlement should not revert with ERC20InsufficientBalance.	❌	✅	5M+
SETTLE-04	Settlement should not panic underflow	❌	✅	5M+
EPOCH-01	Position with non zero loan amount for lp should always have non-zero collateral required.	✅	✅	5M+
VLT-01	Vault functions should never revert with ERC20InsufficientBalance error	✅	✅	5M+
VLT-02	totalPendingDeposits should be sum of deposit requests - withdrawRequestDeposit(s)	❌	✅	5M+

Invariants Assessed

ID	Description	Passed	Remediation	Run Count
VLT-03	totalPendingWithdrawals should be sum of requestRedeem(s) - withdrawRequestRedeem(s)	❌	✅	5M+
VLT-04	pendingSharesToBurn should always be less than or equal to total supply of shares	✅	✅	5M+
VLT-05	Pending transaction requested epoch should never be greater than current epoch	✅	✅	5M+
VLT-06	Vault should not Panic	❌	✅	5M+
VLT-07	mint/deposit should decrease balance of shares in the Vault contract, total supply should stay the same	✅	✅	5M+
VLT-08	redeem/withdraw should decrease total supply	✅	✅	5M+

Findings & Resolutions

ID	Title	Category	Severity	Status
C-01	Total DoS Of Epochs	DoS	● Critical	Resolved
C-02	Bond Cannot Be Returned	Logical Error	● Critical	Resolved
H-01	tradeRatio Rounded In Wrong Direction	Rounding	● High	Resolved
H-02	Faulty Quoting With Small Amounts	Logical Error	● High	Resolved
M-01	Position With Zero Collateral	Logical Error	● Medium	Resolved
M-02	Inaccessible onlyOwner Functions	Access Control	● Medium	Acknowledged
M-03	DoS Via Deposit Before First Epoch	DoS	● Medium	Resolved
M-04	DoS Via Frontrunning Pool Creation	DoS	● Medium	Resolved
M-05	Gas Griefing Of Epoch Creation	Griefing	● Medium	Resolved
M-06	Positions With 0 Collateral	Logical Error	● Medium	Resolved
M-07	Fee Collector Can Hoard Fees	Logical Error	● Medium	Acknowledged
M-08	Decreasing LP May Require Collateral	Logical Error	● Medium	Acknowledged
M-09	vEth Credited When Closing A Position	Logical Error	● Medium	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
M-10	resolutionCallback Fails On Small Amounts	Logical Error	● Medium	Resolved
M-11	Cleared borrowedVEth	Logical Error	● Medium	Resolved
M-12	Resetting Does Not Refund Tokens	Logical Error	● Medium	Resolved
M-13	Collateral Of Epoch Ahead Can Be Stolen	Logical Error	● Medium	Resolved
M-14	Tick Modulus Hardcoded For Fee Tier	Logical Error	● Medium	Resolved
L-01	Single Vault Circuit Should Not Skip Iteration	Logical Error	● Low	Acknowledged
L-02	Overflow In DecimalPrice Library	Overflow	● Low	Resolved
L-03	Deposit/Withdraw On Behalf Of Others	Access Control	● Low	Acknowledged
L-04	New Vaults Cannot Be Added	Warning	● Low	Acknowledged
L-05	minCollateral Redeem Denomination	Validation	● Low	Acknowledged
L-06	Cheaper Settlement Delay	Logical Error	● Low	Acknowledged
L-07	Insufficient Balance For Last Withdrawer	Logical Error	● Low	Resolved
L-08	Consider Adding Exception Handling Mechanisms	Logical Error	● Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-09	minTradeSize For Liquidty Turned Trade	Logical Error	● Low	Acknowledged
L-10	Traders Unable To Close Profitable Position	Logical Error	● Low	Acknowledged
L-11	Epoch startTime Not Utilized	Logical Error	● Low	Resolved
L-12	Incorrect Error String	Logical Error	● Low	Resolved
L-13	Pending Functions Might Be Misleading	Informational	● Low	Resolved
L-14	Insufficient Trade Size Validation	Validation	● Low	Resolved
L-15	Fee Collectors Can Make Unbacked Trade Positions	Logical Error	● Low	Acknowledged
L-16	Negative Ticks Are Rounded Up	Logical Error	● Low	Acknowledged
L-17	Vault Is Not EIP Compliant	EIP	● Low	Resolved

C-01 | Total DoS Of Epochs

Category	Severity	Location	Status
DoS	● Critical	Vault.sol: 604-619	Resolved

Description

Users can create redemption requests for their vault shares using the requestRedeem function, which will increase the totalPendingWithdrawals variable. The only requirement regarding the request amount is that the users' balance must be sufficient.

Users’ shares are neither transferred nor burned at the creation of the request. Since these shares are transferable, a user can create a request using requestRedeem, transfer shares to another address, create another request, and repeat this process as many times as desired.

As a result, totalPendingWithdrawals will be inflated. This allows users to manipulate pendingSharesToBurn and totalSupply, or even cause a complete DoS in the system due to an underflow here in the _reconcilePendingTransactions function.

Recommendation

The redeem workflow should transfer tokens during the request creation process, similar to the deposit flow.

The requestRedeem function should transfer shares from the user to the vault. And then, the _redeemShares function should burn these shares from the vault instead of burning from the owner.

Resolution

Foil Team: The issue was resolved in PR#193.

C-02 | Bond Cannot Be Returned

Category	Severity	Location	Status
Logical Error	● Critical	Vault.sol: 133	Resolved

Description

When submitMarketSettlementPrice is called in Vault.sol, the vault is set as the asserter in the UMA oracle. Upon successful settlement of the assertion price, the bond is returned to the vault.

However, there is no mechanism to refund this bond to the user who submitted the price and paid for it. Additionally, there is no recovery function, causing the bond to remain permanently stuck in the vault.

Recommendation

In UMASettlementModule.submitSettlementPrice, allow the caller to specify an address to be set as the asserter. Then, in Vault.submitMarketSettlementPrice, ensure the caller's address is passed as the asserter to enable proper bond refunds.

Resolution

Foil Team: The issue was resolved in PR#181.

H-01 | tradeRatio Rounded In Wrong Direction

Category	Severity	Location	Status
Rounding	● High	TradeModule.sol	Resolved

Description

The recommendation of H-03 is to round up the trade ratio when going towards the long direction as a short. However, the fix implemented is the opposite - the trade ratio is being rounded down if `isLongDirection` and rounded up otherwise. Since the problem is not solved, the insolvency issue still exists.

Currently, tradeRatioD18 is used to compute both `closePnL` and `vEthAmount`/`borrowedVEth`. Foil's goal should always be to maximize `borrowedVEth` and minimize `closePnL` and `vEthAmount`. Because of this, different rounding directions should be used depending on what's being calculated.

Recommendation

The end goal should be to maximize the `borrowedVEth` and minimize the `vEthAmount` and `closePnL`. To accomplish this, you can have two different tradeRatios - one rounded down and one rounded up. You will also have three different `vEthToZero`.

The first one will be to calculate the `closePnL` and you will use the `tradeRatio` that's rounded down if the position is a long, otherwise use the rounded up one. The second `vEthToZero` will always use the `tradeRatio` that's rounded down and the third `vEthToZero` will always use the `tradeRatio` that's rounded up.

Next, you will also have two different `vEthFromZero` for each `vEthToZero`. Finally, in the `if/else` statement where you set `borrowedVEth` and `vEthAmount` you will choose the appropriate `vEthFromZero`.

For the `if` case you should use the `vEthFromZero` which absolute value is bigger to maximize borrow and for the `else` case you should use the `vEthFromZero` which absolute value is smaller to minimize the credited vETH.

Resolution

Foil Team: The issue was resolved in PR#198.

H-02 | Faulty Quoting With Small Amounts

Category	Severity	Location	Status
Logical Error	● High	LiquidityModule.sol	Resolved

Description PoC

When a new epoch is created, the Vault uses the assets in its reserves to deposit them as collateral in order to create a liquidity position. The vault will call `quoteLiquidityPositionTokens` to get the `amount0` and `amount1` that can be added as liquidity for the available collateral.

However, `Epoch.requiredCollateralForLiquidity()` now adds 1 to `loanAmount0` and `loanAmount1`. This means the actual required collateral for the position may exceed the available collateral in the vault.

In result, the transaction will revert because of `InsufficientCollateral()` and the epoch creation will not be successful. This issue can occur with non-trivial amounts, for example `1e17`.

Recommendation

Consider implementing higher minimum collateral amounts and documenting this behavior for clarity. Another option to consider is subtracting 1 wei from `amount0` and `amount1` when creating the `LiquidityMintParams`, which should account for the additional 1 wei.

Resolution

Foil Team: The issue was resolved in PR#197.

M-01 | Position With Zero Collateral

Category	Severity	Location	Status
Logical Error	● Medium	TradeModule.sol	Resolved

Description PoC

When a position is operating with small amounts, the required collateral for the position can be calculated to be zero due to rounding when calculating value of debt. Consequently, a user can modify their position to a size within a couple thousand wei and have to provide zero collateral.

All their prior deposited collateral would be returned, and their position would have no backing. In the original review, this issue was not possible since the minimum requiredCollateral was always at least 2 wei.

Recommendation

Have a minimum required collateral.

Resolution

Foil Team: The issue was resolved in PR#198.

M-02 | Inaccessible onlyOwner Functions

Category	Severity	Location	Status
Access Control	● Medium	Vault.sol	Acknowledged

Description

Since the `Vault` contract will be executing the `onlyOwner` `createEpoch()` function, it will be set as the owner of the foil system. The `ConfigurationModule.updateMarket()` function can be called by the Foil owner to update the market parameters.

However, this function is never called in the `Vault` contract. This means the market can never be updated once the ownership is transferred to the contract. There is also no call to `transferOwnership` in `Vault`, so you can't just use a new vault as the owner.

Recommendation

Add calls to `updateMarket` and `transferOwnership` in the vault.

Resolution

Foil Team: We are keeping everything immutable.

M-03 | DoS Via Deposit Before First Epoch

Category	Severity	Location	Status
DoS	● Medium	Vault.sol: 341	Resolved

Description

Deposits before the first epoch are possible, with a minimum deposit amount of 1e3. Any pending deposits before the first epoch are utilized to establish the initial liquidity position within the `_createNewLiquidityPosition` function.

This function deducts a dust amount of 1e4 from the deposited collateral amounts. If a user intentionally deposits an amount between 1e3 and 1e4 before the first epoch, and there are no other deposits, the initialization will fail due to underflow at this line.

Recommendation

Consider setting the minimum deposit amount higher than the dust. Alternatively, keep the codebase unchanged but externally deposit the difference if this situation occurs.

Resolution

Foil Team: The issue was resolved in PR#197.

M-04 | DoS Via Frontrunning Pool Creation

Category	Severity	Location	Status
DoS	● Medium	Epoch.sol: 145	Resolved

Description

After the implementation of the Vault, epoch settlements and the creation of the new epoch happens at the same transaction via callbacks. Because of this atomic behavior, failure of the pool creation for the next epoch will DoS the settlement of the previous epoch.

An attacker can precompute the virtual token addresses and create the Uniswap pool with these addresses as creating pools is permissionless in the Uniswap. This will cause `Epoch.createValid` function to revert while calling `IUniswapV3Factory.createPool` due to `require(getPool[token0][token1][fee] == address(0))` check in the factory.

The attack can cause complete blocking of the epoch settlements and creations. However, attackers must keep frontrunning and create new pools every time someone tries to `settleAssertion` in the optimistic oracle.

Recommendation

Check whether the pool already exists or not by calling the `getPool` in the factory instead of directly calling the `createPool`. If the pool already exists, check whether it was already initialized or not and set the starting price. Alternatively, always make sure to use a private RPC to prevent frontrunning.

Resolution

Foil Team: The issue was resolved in PR#209.

M-05 | Gas Griefing Of Epoch Creation

Category	Severity	Location	Status
Griefing	● Medium	Epoch.sol: 206	Resolved

Description

When a new epoch is created, block.timestamp is used as the salt for generating two virtual tokens. In _createVirtualToken, a loop probes for an available salt if a collision occurs.

However, the salt increments by 1 on each iteration, making it highly predictable and susceptible to front-running. An attacker can exploit this predictability to deliberately create collisions.

During testing, each iteration of the loop was found to cost approximately 600k gas, making it feasible for an attacker to force the epoch creation process to fail due to an Out-of-Gas error.

Recommendation

Consider using a less predictable and more robust mechanism for generating the salt, such as hashing with block variables. Alternatively, consider using CREATE3 which ensure that the address is only dependent on deployer and salt.

Resolution

Foil Team: The issue was resolved in PR#197.

M-06 | Positions With 0 Collateral

Category	Severity	Location	Status
Logical Error	● Medium	TradeModule.sol	Resolved

Description

When a position is operating with small amounts, the required collateral for the position can be calculated to be zero due to rounding when calculating value of debt. Consequently, a user can modify their position to a size within a couple thousand wei and have to provide zero collateral.

All their prior deposited collateral would be returned, and their position would have no backing. In the original review, this issue was not possible since the minimum `requiredCollateral` was always at least 2 wei.

Recommendation

Have a minimum required collateral.

Resolution

Foil Team: The issue was resolved in PR#198.

M-07 | Fee Collector Can Hoard Fees

Category	Severity	Location	Status
Logical Error	● Medium	LiquidityModule.sol	Acknowledged

Description

M-01 of the previous audit was not addressed. Fee collectors can create under-collateralized positions and collateralize them using `depositCollateral`.

Currently, there are no restrictions preventing a fee collector from creating an oversized liquidity position, which can monopolize all available liquidity and hoard fees, preventing other fee collectors from benefiting.

Recommendation

Impose limits on the size of liquidity positions that fee collectors can create to ensure fair distribution of fees.

Resolution

Foil Team: Acknowledged.

M-08 | Decreasing LP May Require Collateral

Category	Severity	Location	Status
Logical Error	● Medium	Epoch.sol	Acknowledged

Description PoC

The M-05's recommendation to let the user specify an amount of collateral to be added when decreasing a liquidity position has not been implemented which leaves the problem unsolved.

Recommendation

Allow the user to supply additional collateral when decreasing their position.

Resolution

Foil Team: Letting users know to decrease by larger than a few wei is acceptable due to this rounding issue.

M-09 | vEth Credited When Closing A Position

Category	Severity	Location	Status
Logical Error	● Medium	TradeModule	Resolved

Description PoC

When closing a position, the vEthToZero is calculated as initialSize * tradeRatio and should be equal to the signedTradedVEth. However, Solidity division truncates the result. Because of this, tradeRatio will be slightly off - both when rounded down or up - therefore vEthToZero as well.

Even though vEthFromZero should be roughly equal to targetSize * tradeRatio, the value assigned to it (for targetSize = 0) will be non-zero - positive or negative depending on the rounding. After that the absolute value of vEthFromZero will be assigned to vEthAmount.

In result, closed positions end up having positive vEthAmount, which is especially bad for long positions. This ultimately leads to an undercollateralized market, preventing the last user from settling.

Recommendation

Consider setting the vEthAmount of the new position to 0, if its size is 0 as well.

Resolution

Foil Team: The issue was resolved in PR#198.

M-10 | resolutionCallback Fails On Small Amounts

Category	Severity	Location	Status
Logical Error	● Medium	Vault.sol	Resolved

Description

Function _createEpochAndPosition passing is critical to the Vault's flow, since if the resolutionCallback fails the Vault's functionality is stopped. If the Vault has more collateral than the current minimum collateral, the Vault attempts to _createNewLiquidityPosition.

The issue is that even with enough collateral to meet the minimum threshold, is it not guaranteed that the liquidity to-be minted from the calculated amount0 and amount1 is greater than 0 due to Uniswap rounding down on small amounts, which would trigger a revert in UniswapV3Pool.mint: require(amount > 0);

Ultimately, the Vault will attempt to mint which will revert, causing the callback to fail and the mints/epoch creation will not occur.

Recommendation

Consider enforcing a higher minimum collateral.

Resolution

Foil Team: The issue was resolved in PR#197.

M-11 | Cleared borrowedVEth

Category	Severity	Location	Status
Logical Error	● Medium	TradeModule	Resolved

Description PoC

When a long position is being modified, its borrowedVEth is set to the absolute value of vEthFromZero. In some cases, it's possible to have a small amount of vGasAmount with 0 vEthFromZero due to the traded vETH matching the vEthToZero, primarily when operating with small position sizes and trade prices.

This leads to a long position that does not have a loaned amount. This leaves the Foil contract with less available collateral than it should have and in result, the last user will not be able to exit.

Recommendation

Validate that any opened long position has positive borrowedVEth: require(borrowedVEth > 0)

Resolution

Foil Team: The issue was resolved in PR#198.

M-12 | Resetting Does Not Refund Tokens

Category	Severity	Location	Status
Logical Error	● Medium	Vault: 649 + 530	Resolved

Description PoC

When a user calls withdrawRequestRedeem(), if their balance after is less than minimumCollateral then resetTransaction() will set their pending amount to zero. However, totalPendingWithdrawals will only be decremented by the amount of shares the user passes in.

This will lead to totalPendingWithdrawals being larger than the actual amount that is intended to be withdrawn. A malicious user could continuously call requestRedeem() in conjunction with withdrawRequestRedeem in order to inflate totalPendingWithdrawals to be larger than collateralFromPreviousEpoch plus totalPendingDeposits.

This will cause a DoS via underflow when _reconcilePendingTransactions() is called. Additionally, when a user calls withdrawRequestDeposit(), totalPendingDeposits is only decremented by assets. This will lead to the user's remaining tokens to be donated to other users of the protocol.

Recommendation

If the user's remaining amount is less than the minimumCollateral, then decrement totalPendingWithdrawals by the full amount or refund the remainder of their balance before calling resetTransaction(), depending on if it is a redeem or deposit.

Resolution

Foil Team: The issue was resolved in PR#197.

M-13 | Collateral Of Epoch Ahead Can Be Stolen

Category	Severity	Location	Status
Logical Error	● Medium	Vault.sol: 288	Resolved

Description

When updating share price after an epoch, if no collateral was received, the share price is set to `1e18`. This creates a significant issue as depositors can redeem their entire collateral even though no collateral was received after closing the liquidity position.

Effectively, this allows depositors to withdraw funds that belong to the next epoch's depositors, who have already transferred their collateral into the contract.

Recommendation

Initially, setting `sharePrice` to `0` instead of `1e18` was considered. However, this would affect the minting of new shares for the next epoch.

As a solution, if no collateral is received, set the `sharePrice` for the current epoch to `0` while ensuring the `sharePrice` for the next epoch is reset to `1e18`.

Resolution

Foil Team: The issue was resolved in PR#197.

Guardian Team: The issue was not fixed. The price of the epoch is hardcoded to 1e18 if no collateral is received.

Foil Team: Let’s halt the vault and allow a request deposit of something higher than 1e8 which will fix this.

M-14 | Tick Modulus Hardcoded For Fee Tier

Category	Severity	Location	Status
Logical Error	● Medium	Vault: 273, 276	Resolved

Description

`_calculateTickBounds()` uses modulus 200 in order to set the target tick value to the closest acceptable tick range. However, Foil is compatible with multiple fee tiers, but the value 200 is not.

For instance, the 0.3% fee tier uses a tick spacing of 60, which is not a divisor of 200. This will cause a revert when attempting to create the epoch.

Recommendation

Instead of hardcoding 200, use the appropriate value for the fee tier of the pool.

Resolution

Foil Team: The issue was resolved in PR#197.

L-01 | Single Vault Circuit Should Not Skip Iteration

Category	Severity	Location	Status
Logical Error	● Low	Vault.sol: 233	Acknowledged

Description

In `_calculateNextStartTime`, if there is a significant delay in resolving an epoch, the vault skips an entire `vaultCycleDuration` to maintain synchronization with other vaults in the circuit.

However, if only a single vault exists in the circuit, this synchronization is unnecessary. Skipping `vaultCycleDuration` in this scenario causes unnecessary downtime where no vaults are available.

Recommendation

Introduce a condition to check if only one vault exists in the circuit. In such cases, avoid skipping the `vaultCycleDuration` and instead start the next epoch immediately after resolution.

Resolution

Foil Team: Acknowledged.

L-02 | Overflow In DecimalPrice Library

Category	Severity	Location	Status
Overflow	● Low	DecimalPrice.sol	Resolved

Description

There is a comment left on L-02 that Foil now uses OpenZeppelin's code for its calculations, but the code in DecimalPrice is not changed - it's still possible for the result of the multiplication to exceed 2^256-1

Recommendation

Fix the issue.

Resolution

Foil Team: The issue was resolved in PR#202.

L-03 | Deposit/Withdraw On Behalf Of Others

Category	Severity	Location	Status
Access Control	● Low	Vault.sol	Acknowledged

Description

In the Vault contract, only the owners can request deposits and redemptions. However, claiming of these requests are external and anyone can claim on behalf of the owner.

Even though the owner created these requests, timing of the claim might matter for the owner and these actions should be access controlled.

Recommendation

Not allow other users to claim on behalf of owners.

Resolution

Foil Team: I don’t think there’s any advantage to claiming after the epcoh is settled, if anything, these functions not being gated gives us flexibility to force redemptions to clear any pending txns.

L-04 | New Vaults Cannot Be Added

Category	Severity	Location	Status
Warning	● Low	Vault.sol	Acknowledged

Description

totalVaults is stored as an immutable variable when Vault.sol is created. This implies that no new vaults can be added after the first batch of vaults. This may run counter to protocol design that new collateral types may be added.

Recommendation

Consider allowing for new vaults to be added.

Resolution

Foil Team: At least the plan right now is not to add any more vaults once a vault is initialized.

L-05 | minCollateral Redeem Denomination

Category	Severity	Location	Status
Validation	● Low	Vault.sol	Acknowledged

Description

Vault.requestRedeem requires the amount of shares being redeemed to be greater than minimumCollateral. However, minimumCollateral is denominated in assets, not shares.

Recommendation

Consider having different validation with the proper denomination.

Resolution

Foil Team: Maybe a rename of the variable would be better. Will do that.

L-06 | Cheaper Settlement Delay

Category	Severity	Location	Status
Logical Error	● Low	Global	Acknowledged

Description

As pointed out in this issue, anyone can dispute rightful assertions to delay the start of a given epoch by paying the bond of $5000.

Since now anyone can submit a price, the same entity can assert a rightful price and dispute it at the same time towards the end of the `assertionLiveness` period.

By doing so, they will receive half of their disputer bond. In result, the cost of the attack will be reduced from $5000 to $2500.

Recommendation

Be aware of the reduction in cost.

Resolution

Foil Team: Acknowledged.

L-07 | Insufficient Balance For Last Withdrawer

Category	Severity	Location	Status
Logical Error	● Low	SettlementModule.sol	Resolved

Description

The last user attempting to settle their position may be unable to do so if: `market.collateralAsset.balanceOf(address(this)) < withdrawableCollateral`.

This discrepancy can occur due to minor rounding errors during trade or liquidity activities, leaving the contract balance short by a few wei. As a result, the user cannot fully recover their collateral.

Recommendation

If `market.collateralAsset.balanceOf(address(this))` is less than `withdrawableCollateral`, consider transferring the remaining contract balance to the user instead.

This ensures the user can recover as much of their collateral as possible without leaving residual funds in the contract.

Resolution

Foil Team: The issue was resolved in PR#202.

L-08 | Consider Adding Exception Handling Mechanisms

Category	Severity	Location	Status
Logical Error	● Low	Global	Resolved

Description

With the implementation of the Vault contract, the settlement of the previous epoch and the creation of the next epoch happen in a single transaction.

Because of this, an unexpected failure at any step of the process (e.g., settlement, new epoch creation, quoting, or adding new liquidity) may cause the system to halt.

Recommendation

Consider implementing mechanisms like try/catch blocks along the transaction flow, allowing unexpected issues to be resolved externally and ensuring the system remains operational.

Resolution

Foil Team: The issue was resolved in PR#209.

L-09 | minTradeSize For Liquidty Turned Trade

Category	Severity	Location	Status
Logical Error	● Low	TradeModule.sol	Acknowledged

Description

When closing a liquidity position, it can turn into a `Trade` position if it cannot be repaid. If the amount left for the new `Trade` position is less than the `minTradeSize`, the owner of the position will not be able to directly close it.

They will have to make a bigger trade and close if after that. By doing so, they suffer losses because of price impacts.

Recommendation

Be sure to warn the users of Foil about this case.

Resolution

Foil Team: Acknowledged.

L-10 | Traders Unable To Close Profitable Position

Category	Severity	Location	Status
Logical Error	● Low	Global	Acknowledged

Description

L-20 of the previous audit was not addressed. Fee Collectors opened LP positions at the beginning of an epoch and deposit collateral after they've earned fees. This collateral could be streamed in periodically or provided in bulk at settlement.

Due to the under-collateralized LP positions, traders may find themselves unable to exit profitable positions until Fee Collectors deposit collateral. As Fee Collectors are expected to hold large LP positions, this may affect a large group of traders.

This leads to temporarily locked funds and potential loss of yield for traders who are unable to close a profitable position promptly.

Recommendation

Consider implementing a minimum deposit amount for fee collectors. Or else, document this risk for users.

Resolution

Foil Team: Acknowledged.

L-11 | Epoch startTime Not Utilized

Category	Severity	Location	Status
Logical Error	● Low	Epoch.sol	Resolved

Description

Epochs in Foil have `startTime`. However, liquidity and trades for a given epoch can be executed as soon as the epoch is created, no matter its `startTime`.

Recommendation

Be aware of this behavior.

Resolution

Foil Team: The issue was resolved in PR#202.

L-12 | Incorrect Error String

Category	Severity	Location	Status
Logical Error	● Low	Vault.sol: 642	Resolved

Description

"Previous deposit request is not in the same epoch" message in the withdrawRequestRedeem function (L642) should be "Previous withdraw request is not in the same epoch".

Recommendation

Change the error string in the require statement.

Resolution

Foil Team: The issue was resolved in PR#202.

L-13 | Pending Functions Might Be Misleading

Category	Severity	Location	Status
Informational	● Low	Vault.sol: 543, 662	Resolved

Description

Vault contract has `pendingDepositRequest` and `pendingRedeemRequest` functions. However, these functions do not check the transaction type of the pending request and directly return `userPendingTransactions[owner]`. `pendingDepositRequest` function can return a redeem request and vice versa.

Recommendation

Consider checking the transaction type in these functions, or implement a single function (e.g. `pendingRequest`) for all transaction types.

Resolution

Foil Team: The issue was resolved in PR#202.

L-14 | Insufficient Trade Size Validation

Category	Severity	Location	Status
Validation	● Low	TradeModule.sol	Resolved

Description

A minTradeSize configuration has been added to the Market in response to L-15. This works fine for createTraderPosition, but it's wrongly implemented in modifyTraderPosition. It calls _checkTradeSize(size) to ensure the trade size is bounded.

However, the argument passed is size (the final size), not deltaSize. Because of this, small trades (below the minTradeSize) will still be successfully executed.

Recommendation

Pass deltaSize instead of size to _checkTradeSize to ensure trades are beyond a minimum delta.

Furthermore, consider also validating the resulting size of the position, such that situations do not arise where a user creates a large position, decreases by position size-1, such that the delta trade size is large enough but the final position size is 1 wei.

Resolution

Foil Team: The issue was resolved in PR#198.

L-15 | Fee Collectors Can Make Unbacked Trade Positions

Category	Severity	Location	Status
Logical Error	● Low	LiquidityModule.sol	Acknowledged

Description PoC

Because the fee collector is not required to deposit collateral for their position, situations can arise where fee collectors close their liquidity position yet the the resulting position will become a Trade position with non-zero borrowed amounts but zero credit amounts.

This is because fee collectors will typically enter the following case `if (position.depositedCollateralAmount < collateralDelta)` due to no collateral requirements which will set a non-zero `borrowedvETH`.

Consequently, an unbacked Trade position may be created that cannot be directly decreased and closed, since the `deltaSize` would be zero and `Errors.DeltaTradeIsZero()` would be triggered.

Recommendation

Clearly document this behavior and even consider if `FeeCollectors` should close their LP positions before epoch settlement as this allows them to profit without ever depositing collateral.

Resolution

Foil Team: Acknowledged.

L-16 | Negative Ticks Are Rounded Up

Category	Severity	Location	Status
Logical Error	● Low	Vault.sol: 271	Acknowledged

Description

During epoch creation, in `_calculateTickBounds` positive ticks are rounded down but negative ticks are rounded up, which could lead to unexpected behavior.

Recommendation

Consider rounding down negative ticks for consistency or clearly documenting this behavior.

Resolution

Foil Team: Acknowledged.

L-17 | Vault Is Not EIP Compliant

Category	Severity	Location	Status
EIP	● Low	Vault.sol	Resolved

Description

Multiple functions in the Vault are not EIP compliant.

- `totalAssets`: Must not revert. However, it can revert if the `positionId` is not valid.
- `convertToShares`: Must not revert. However, it can revert if `totalAssets == 0`.
- preview functions: Must be as close as possible to on-chain conditions and must not revert based on vault specific user/global limits. May only revert that would also cause mint/withdraw etc. to revert too. However, these function are not supported at all.
- `deposit`: Mints shares by depositing exactly `assets` amount. However, the amount is ignored in the codebase.
- `mint`: Mints exactly the `shares` amount. However, the amount is ignored in the codebase.
- `withdraw`: Burns shares and sends exactly the `assets` amount. However, the amount is ignored in the codebase.
- `redeem`: Burns exactly the `shares` amount. However, the amount is ignored in the codebase.

The contract incorrectly signals supporting the ERC4626 interface with the `supportsInterface` function.

Recommendation

One option is trying to make the contract EIP compliant. However, based on what the contract wants to achieve, it might be best to not support ERC4626.

Consider removing `interfaceId == type(IERC4626).interfaceId` line from the `supportsInterface` function to prevent incorrect signaling for the external integrators.

Resolution

Foil Team: The issue was resolved in PR#202.

Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian’s position is that each company and individual are responsible for their own due diligence and continuous security. Guardian’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Guardian Audits

Founded in 2022 by DeFi experts, Guardian Audits is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian Audits upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit https://guardianaudits.com

To view our audit portfolio, visit https://github.com/guardianaudits

To book an audit, message https://t.me/guardianaudits