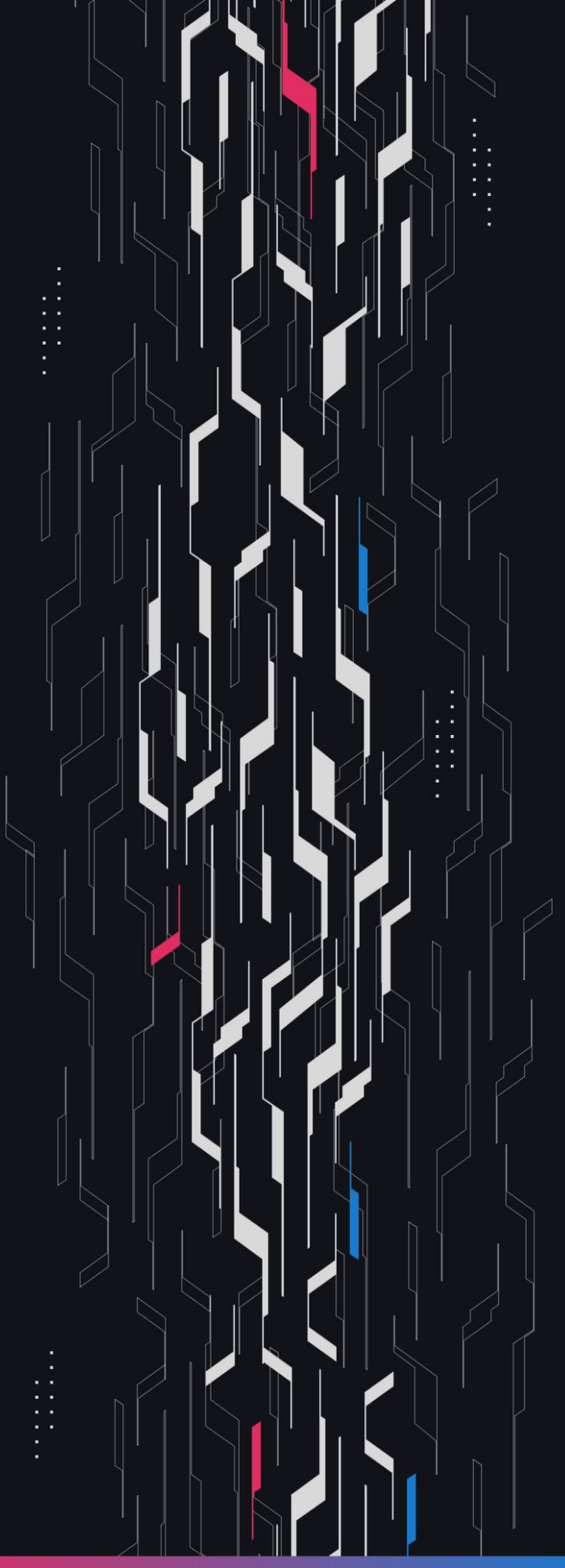
GA GUARDIAN

Foil

Stable Gas Pricing

Security Assessment

September 30th, 2024



Summary

Audit Firm Guardian

Prepared By Owen Thurm, Doğuş Köse, Cosine, Kiki, EnigmaticAuditor, Vladimir Zotov

Client Firm Foil

Final Report Date September 30, 2024

Audit Summary

Foil engaged Guardian to review the security of its virtual gas marketplace protocol. From the 26th of August to the 9th of September, 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 13 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 Foil protocol.

Security Recommendation Given the number of High and Critical issues detected, Guardian supports a secondary security review of the protocol at a finalized frozen commit.

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: bc80c3a7109299cfd43b9b116bdef6ccbb533200 Final Commit: 4a00c554338ac660076a7fb491442c3506d5bce0

Audit Summary

Delivery Date	September 30, 2024
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	9	0	0	0	0	9
High	4	0	0	0	0	4
Medium	14	0	0	4	0	10
Low	27	2	0	6	0	19

Audit Scope & Methodology

Vulnerability Classifications

Vulnerability Level	Classification
🔴 Critical	Easily exploitable by anyone, causing loss/manipulation of assets or data.
🟠 High	Arduously exploitable by a subset of addresses, causing loss/manipulation of assets or data.
🟡 Medium	Inherent risk of future exploits that may or may not impact the smart contract execution.
🟢 Low	Minor deviation from best practices.

Methodology

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 10,000,000+ 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.	✅	N/A	10M+
GLOBAL-02	The system should never revert with a InsufficientBalance error from the collateral token.	✅	✅	10M+
GLOBAL-03	There should never be any liquidity outside of the [min, max] range of an epoch.	✅	✅	10M+
GLOBAL-04	The amount of vETH in the system, position manager & swap router should equal the max supply	✅	✅	10M+
GLOBAL-05	The amount of vGAS in the system, position manager & swap router should equal the max supply.	✅	✅	10M+
TRADE-01	The debt of a position should never be > the collateral of the position.	✅	✅	10M+
TRADE-02	Long positions have their debt in vETH and own vGAS	✅	❌	10M+
TRADE-03	Short positions have their debt in vGAS and own vETH.	✅	✅	10M+
LIQUID-01	The debt of a position should not be > the collateral of the position.	✅	✅	10M+

Invariants Assessed

ID	Description	Passed	Remediation	Run Count
LIQUID-02	A open LP position should not own any vETH or vGAS.	✅	✅	10M+
LIQUID-03	After all LP positions have been closed, for the remaining trader positions: net shorts == net longs.	✅	✅	10M+
SETTLE-01	It should always be possible to settle all positions after the epoch is settled.	❌	N/A	10M+
STLESS-01	UniV3 and Foils getAmount0ForLiquidity should output the same value when given the same inputs.	❌	❌	10M+
POSITION-01	Discovery range size should not change	✅	✅	10M+
POSITION-02	Anchor liquidity stays the same post-drop	✅	✅	10M+
EPOCH-01	Floor reserves should not decrease post-drop within delta	✅	✅	10M+

Findings & Resolutions

ID	Title	Category	Severity	Status
C-01	increaseLiquidityPosition Uses The Wrong Id	Logical Error	Critical	Resolved
C-02	Liquidity Is Stuck After Epoch Is Settled	Logical Error	Critical	Resolved
C-03	borrowedVGas Is Set To 0 Before Subtraction	Logical Error	Critical	Resolved
C-04	Uniswap Pool Creation DoS	DoS	Critical	Resolved
C-05	Settled Trading Positions Can Be Closed	Logical Error	Critical	Resolved
C-06	Settlement Can be Impossible Due to Underflow	Underflow	Critical	Resolved
C-07	Undercollateralized Positions Can Be Created	Logical Error	Critical	Resolved
C-08	vETH Profit In Long Pos Deleted On Close	Logical Error	Critical	Resolved
C-09	Traders Profit can be stolen when closing	Logical Error	Critical	Resolved
H-01	collateralRequirementAtMaxTick Underflow	Underflow	High	Resolved
H-02	Required Collateral Invalid For Partial Closes	Logical Error	High	Resolved
H-03	LP Stuck Because of Underflow	Logical Error	High	Resolved
H-04	Exact Input Amount May Not Be Used	Validation	High	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
M-01	Mismatching Max Tick Boundary	Logical Error	Medium	Acknowledged
M-02	initializeMarket Can Be Front Run	Logical Error	Medium	Resolved
M-03	Unnecessary Fees When Closing Short	Logical Error	Medium	Resolved
M-04	Trades May Revert At Maximum Tick	Logical Error	Medium	Acknowledged
M-05	Settlement price frontrunning	Frontrunning	Medium	Resolved
M-06	Owner Can Bypass UMA Assertion Checks	Logical Error	Medium	Resolved
M-07	Underflow loanAmount Calculation	Underflow	Medium	Acknowledged
M-08	Collateral Can Be Stuck After Closing Position	Logical Error	Medium	Resolved
M-09	Rounding In Favor Of User	Rounding	Medium	Resolved
M-10	Market Updates Invalidate Previous Positions	Logical Error	Medium	Resolved
M-11	Uniswap Rounding Can Create Insolvent Positions	Rounding	Medium	Resolved
M-12	Fees Missing In Required Collateral Calc	Logical Error	Medium	Resolved
M-13	Missing onRecieved check	Best Practices	Medium	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
M-14	Traders can't close position pre settlement	Unexpected Behavior	Medium	Acknowledged
L-01	Unexpected Collateral Amount Used For LPs	Unexpected Behavior	Low	Acknowledged
L-02	Wrong Description For tokenByIndex	Code Quality	Low	Resolved
L-03	Unsafe Collateral Transfers	Validation	Low	Resolved
L-04	Misleading Error In submitSettlementPrice	Code Quality	Low	Resolved
L-05	Protocol Vulnerable To Reentrancy	Logical Error	Low	Resolved
L-06	Missing Checks In assertionDisputedCallback	Validation	Low	Resolved
L-07	Tokens With != 18 Decimals Are Not Supported	Validation	Low	Resolved
L-08	Revert On 0 Transfer Tokens Not Supported	Validation	Low	Resolved
L-09	swapTokensExactOut DoS In Edge Cases	Logical Error	Low	Resolved
L-10	Rebasing Tokens Are Not Supported	Logical Error	Low	Acknowledged
L-11	Missing Deadline Check	MEV	Low	Resolved
L-12	Misleading Function Name	Code Quality	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-13	tokenByIndex Off By One	Logical Error	Low	Resolved
L-14	Incorrect Comments in `modifyTraderPosition`	Documentation	Low	Resolved
L-15	Outstanding TODO Comments	Code Quality	Low	Resolved
L-16	submitSettlementPrice Overwrites assertionId	Logical Error	Low	Acknowledged
L-17	Lacking Min/Max Tick Validation	Validation	Low	Resolved
L-18	Disputes Prevent Settlement	Logical Error	Low	Acknowledged
L-19	Redundant refundAmountVGas Assignment	Optimization	Low	Resolved
L-20	Missing two step ownership change	Warning	Low	Resolved
L-21	Config lacks adequate opportunity for disputes	Warning	Low	Resolved
L-22	Constant Time Implementations	Warning	Low	Acknowledged
L-23	Possible to Open Epochs for Past	Validation	Low	Resolved
L-24	Zero Checks in updateValid and createValid	Validation	Low	Resolved
L-25	Bond currency should be constant	Warning	Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-26	Foils overestimates needed collateral	Logical Error	Low	Pending
L-27	LP turned to Trader will Encounter Price Impact	Documentation	Low	Pending

C-01 | increaseLiquidityPosition Uses The Wrong Id

Category	Severity	Location	Status
Logical Error	● Critical	EpochLiquidityModule.sol: 181	Resolved

Description

The `increaseLiquidityPosition` function calls the `NonfungiblePositionManager` with the Foil position ID instead of the Uniswap position ID. As there are liquidity and trade positions in Foil the IDs can differ.

Recommendation

Call Uniswap with the correct position ID.

Resolution

Foil Team: The issue was resolved in PR#47.

C-02 | Liquidity Is Stuck After Epoch Is Settled

Category	Severity	Location	Status
Logical Error	● Critical	EpochSettlementModule.sol 56-79	Resolved

Description

The only way to decrease liquidity of a position is through the `decreaseLiquidityPosition` function. This has a check `epoch.validateEpochNotSettled` which will revert if the epoch has settled.

The `settlePosition` function is supposed to allow you to exit liquidity positions through a branch which calls `_settleLiquidityPosition()`.

However, this calls the `collect` function in the `NonFungiblePositionManager` which will collect the tokens owed from fees and previous liquidity burns, but does not burn/decrease the liquidity still in the pool.

The user may still get their collateral back, but does not get the value of their liquidity if it exceeds the loan value.

It should be expected that the liquidity is worth than the loan value, since the LP positions are overcollateralized. When settling a liquidity position, consider first burning all the liquidity and the calling collect.

Recommendation

When settling a liquidity position, consider first burning all the liquidity and then calling collect.

Resolution

Foil Team: The issue was resolved in PR#47.

C-03 | borrowedVGas Is Set To 0 Before Subtraction

Category	Severity	Location	Status
Logical Error	● Critical	EpochLiquidityModule.sol 381	Resolved

Description

In `closePositionPosition`, the `vGasAmount` should be set to `collectedAmount0 - position.borrowVGas`. However, `position.borrowedVGas` is set to 0 before the subtraction.

This means that the user's `vGas` amount is overestimated and allows them to drain the protocol.

Recommendation

Reset `position.borrowedVGas` after rather than before the subtraction.

Resolution

Foil Team: The issue was resolved in PR#47.

C-04 | Uniswap Pool Creation DoS

Category	Severity	Location	Status
DoS	● Critical	Epoch.sol: 111-141	Resolved

Description PoC

Uniswap pool creation is done with three variables (`gasToken` address, `ethToken` address, and `feeRate`). These variables are all predictable even before creation of specified tokens.

Hence it is possible to frontrun pool creations happening in `Epoch.sol/createValid()`, which will lead to revert in epoch creation.

Recommendation

Use `CREATE2` to deploy the virtual tokens with a configurable salt so that pool creation cannot be permanently DoS, additionally be sure to use a private rpc to avoid being frontran to DoS individual epoch creations.

Resolution

Foil Team: The issue was resolved in PR#85.

C-05 | Settled Trading Positions Can Be Closed

Category	Severity	Location	Status
Logical Error	● Critical	EpochTradeModule.sol: 76-139	Resolved

Description PoC

The `modifyTraderPosition` function does not check if the given position is already settled.
This enables an attack vector to steal funds by closing an already settled position.

As the collateral & the owned tokens of the position are not set to 0 during the settlement process.

Recommendation

Always use the `validateNotSettled` validation in the `modifyTraderPosition` function as no trader activity should occur after an epoch end besides settlement.

If a trader wishes to close their position, they should use the `EpochSettlementModule` to do so.

Thus the `epoch.settled` case handling can be removed from the `swapTokensExactOut` and `swapTokensExactIn` swap functions as they will no longer be callable after the epoch has been settled.

Resolution

Foil Team: The issue was resolved in PR#87.

C-06 | Settlement Can Be Impossible Due To Underflow

Category	Severity	Location	Status
Underflow	● Critical	Position.sol 196	Resolved

Description

This Equation in the `settle` function of `Position.sol`:

`self.depositedCollateralAmount += self.vEthAmount - self.borrowedVEth;`
is slightly different from:

`self.depositedCollateralAmount = self.depositedCollateralAmount + self.vEthAmount - self.borrowedVEth;`

Because in the first equation, if `self.borrowedEth < self.vEthAmount`, then the equation will underflow, even though this position is fully collateralized.

This underflow makes certain positions impossible to settle.

Recommendation

Replace `self.depositedCollateralAmount += self.vEthAmount - self.borrowedVEth;`
With `self.depositedCollateralAmount = self.depositedCollateralAmount + self.vEthAmount - self.borrowedVEth;`

Resolution

Foil Team: The issue was resolved in PR#47.

C-07 | Undercollateralized Positions Can Be Created

Category	Severity	Location	Status
Logical Error	● Critical	Epoch.sol: 348	Resolved

Description

In this line, if `loanAmount0 > maxAmount0`, then the excess `loanAmount0` is not considered in the collateralization check:

`uint256 availableAmount0 = maxAmount0 > loanAmount0 ? maxAmount0 - loanAmount0 : 0;`

The `loanAmount0` entered into `collateralRequirementAtMinTick` is `loanAmount0 - tokensOwed0`.

Here is a sequence which leads to a state where `loanAmount0 > maxAmount0`:
1. Create a liquidity position when the current tick is below `tickLower`, so the loaned amount is 100% `token0`
2. Swap so the position is liquidity entirely `token1`
3. Remove most of the liquidity via `decreaseLiquidityPosition`.

Since the LP is now 100% `token1`, all the claimed tokens from `decreaseLiquidityPosition` would be `token1` (except for a tiny amount of LP fees). `DecreaseLiquidityPosition` reduced the `maxAmount0`, while `loanAmount0` is still the `amount0` required to create the initial position, so `maxAmount0 > loanAmount0`.

Now it is true that decreasing a liquidity position should make the collateral requirements lower, but this is already accounted for in `tokensOwed0` `tokensOwed1` being deducted from loan amounts.
1. Create a position that requires loaning `token0`
2. Swap so the position is entirely `token1`
3. `decreaseLiquidityPosition`.

All the claimed tokens from `decreaseLiquidityPosition` would be `token1` (except for a tiny amount of LP fees).

Recommendation

Consider converting `loanAmount0` to the corresponding `ETH` value and incorporating it into the collateral calculation rather than subtracting it from `maxAmount0`.

Resolution

Foil Team: The issue was resolved in PR#91.

C-08 | vETH Profit In Long Pos Deleted On Close

Category	Severity	Location	Status
Logical Error	● Critical	EpochTradeModule.sol: 403-412, 629	Resolved

Description

When a trader owns a long position that made a good amount of profit and the trader decreases the position so that the full loan is repaid the system will:
• calculate the excess `vETH` after fully repaying the `borrowedVEth` amount
• set the `borrowedVEth` to 0
• save the excess `vETH` as `vEthAmount` in the position by calling `updateBalance`

This is unusual as normally a long position has a `vGasAmount` amount > 0 and a `borrowedVEth` amount > 0, but the `vEthAmount` is usually 0.

This state is problematic when closing the long position as the flow of closing the position looks like the following:
• Swap the positions `vGasAmount` to `vETH`
• Increase the `depositedCollateralAmount` by the amount of `vETH` received from the swap
• Set the `borrowedVEth` to 0
• Call the `resetBalance` function to set the `currentTokenAmount`, `vEthAmount` & `vGasAmount` to 0

Therefore the trader's profit saved in the `vEthAmount` variable is deleted.

Recommendation

Increase the positions `depositedCollateralAmount` instead of the `vEthAmount` when decreasing a long position with profit > the borrowed `vETH` amount.

Resolution

Foil Team: The issue was resolved in PR#75.

C-09 | Traders Profit Can Be Stolen When Closing

Category	Severity	Location	Status
Logical Error	● Critical	EpochTradeModule.sol: 112	Resolved

Description

When a trader closes a position before settlement, there is currently no protection against slippage.

As long as the trader possesses sufficient `vEth/vGas` to settle their debts, the closure of the position will be successful.

This vulnerability could be exploited by an attacker to siphon off the trader's profits by manipulating the price of the pool, causing the trader to swap at a premium that would be covered by their profits.

By ensuring enough is returned to cover the trader's debt, the attacker can retain the profit minus fees.

Recommendation

Implement a parameter for closing a position that includes slippage protection to prevent potential exploitation by malicious parties.

Resolution

Foil Team: The issue was resolved in PR#92.

H-01 | collateralRequirementAtMaxTick Underflow

Category	Severity	Location	Status
Underflow	High	Epoch.sol 375	Resolved

Description

In `collateralRequirementAtMaxTick`, this line can revert due to underflow:
`return totalLoanAmountInEth - maxAmount1;`

It is possible `totalLoanAmountInEth` to be less than `maxAmount1`, as the loan amount of each token is `loanAmount - tokensOwed`. This means that the position is already overcollateralized by the `tokensOwed` + liquidity position.

Consider this scenario:
1. User opens liquidity position
2. They wash trade such that the fees paid for `token0` and `token1` exceed the loan amounts `loanAmount0` and `loanAmount1`

The fees are stored in `tokensOwed0` and `tokensOwed1`. Therefore the `loanAmount - tokensOwed` of both tokens are 0. This is logical because there's actually no collateral required to back a position who's loans is entirely backed by collected fees. However, since `maxAmount1` is greater than `0`, then the equation `return totalLoanAmountInEth - maxAmount1;` will underflow.

An example where `loanAmountInEth` becomes 0 was chosen to make the underflow obvious, but just a slight reduction in `loanAmountInEth` could make the underflow revert happen. This makes it impossible to increase or partially decrease liquidity for some liquidity positions.

Recommendation

In the `collateralRequirementAtMaxTick` function, consider returning `0` if `maxAmount1 > totalLoanAmountInEth`.

Resolution

Foil Team: The issue was resolved in PR#91.

H-02 | Required Collateral Invalid For Partial Closes

Category	Severity	Location	Status
Logical Error	High	Position.sol 150-156	Resolved

Description

In the `updateValidLp` function the `loanAmount0` and `loanAmount` are computed by deducting the respective `tokensOwed` from the loaned amount. However the amount credited to pay down the loan cannot exceed the loaned amount.

Consider the following scenario:
• Trader A opens a position which is initially all `vEth` liquidity
• Price moves downwards, trader A's position is now entirely `vGas` liquidity
• Trader A decreases their position and receives all vGas from reducing their liquidity
• Trader A's `tokensOwed0` are not reflected in a reduction of their loaned amount because they had no loaned `amount0` initially.
• Thus trader A does not receive any collateral back and instead must supply more collateral because the collateralization validation measures their position as being worth less.

As a result partial decreases are prevented for positions in this scenario as the `additionalCollateral` is hardcoded to 0.

Recommendation

Consider passing the `tokensOwed0` and `tokensOwed1` through to the `collateralRequirementAtMinTick` function and adding them to the `availableAmount0` and `availableAmount1` values respectively so that this value is not truncated to zero.

Resolution

Foil Team: The issue was resolved in PR#91.

H-03 | LP Stuck Because Of Underflow

Category	Severity	Location	Status
Logical Error	High	EpochLiquidityModule.sol: 387-396	Resolved

Description

How much `amount0` and `amount1` an LP provides initially and how much it receives while closing the position are subject to change based upon the pool price.

Considering LP's can provide liquidity amounts larger than their collateral to the pool, the following scenario is applicable in many situations:

LP's token 1 swapped to token 0 such that LP's `borrowedVEth - collectedVEth` amount will be bigger than LP's collateral which will result with underflow while closing the position.

This occurs when attempting to deduct the collateral in `_closeLiquidityPosition`: `position.depositedCollateralAmount = position.borrowedVEth - collectedAmount1;` In this case the LP is prevented from closing their position.

Recommendation

Allow excess debt that can't be covered to live on in the `position.borrowedVEth`.

Note that the case where `position.borrowedVEth` is left can only occur when the price of the pool has moved downwards through the LP relative to where the LP was first created.

Thus the LP will take on a long position after closing and it is expected and correctly handled when a nonzero `position.borrowedVEth` exists.

Resolution

Foil Team: The issue was resolved in PR#91.

H-04 | Exact Input Amount May Not Be Used

Category	Severity	Location	Status
Validation	High	EpochTradeModule.sol	Resolved

Description

In the SwapRouter.exactInputSingle function in Uniswap V3, the exact amountIn is not guaranteed to always be used. If the sqrtPriceLimitX96 is hit during the swap then the swap will complete and the exactInputSingle function call will pass.

https://github.com/Uniswap/v3-periphery/blob/0682387198a24c7cd63566a2c58398533860a5d1/contracts/SwapRouter.sol#L87

A sqrtPriceLimitX96 of 0 is used in the EpochTradeModule.swapTokensExactIn function, therefore the sqrtPriceLimitX96 is assigned to roughly the min or max tick upon performing the actual swap.This means if the swap are to go outside of the range of valid prices for the epoch the exact input amount will not be entirely used up.

In the context of a short, this can mean an overestimation of the amount borrowed which was not entirely used for the swap and causes immediate loss for the user. This is not an issue for the exactOutputSingle function as the amountOutReceived is validated to be exactly the amountOut:

https://github.com/Uniswap/v3-periphery/blob/0682387198a24c7cd63566a2c58398533860a5d1/contracts/SwapRouter.sol#L199.

Recommendation

There are a number of ways this edge case can be validated against:

• Consider reverting if the price of the Uniswap pool is outside of the valid range after a swap, or as an invariant check after all functions which interact with Uniswap.
• Consider validating whether the swap would put price outside of the valid range, and either reverting or using a partial fill if this is the case.
• Consider reverting if the balance used up by the swap is not exactly the amount specified, measured by the balance of address(this).

Resolution

Foil Team: The issue was resolved in PR#90.

M-01 | Mismatching Max Tick Boundary

Category	Severity	Location	Status
Logical Error	Medium	Epoch.sol 144	Acknowledged

Description

The `baseAssetMaxPriceTick` is the input parameter to `createValid` to set the maximum trading tick of the epoch.

However, this tick is not the same tick that is used in `epoch.sqrtPriceMaxX96` or `epoch.maxPriceD18` as the Uniswap `tickSpacing` is added to the tick.It is important to distinguish between ticks and `tickSpacing`.

Each tick is 0.01% price difference apart. A tickSpacing contains multiple ticks depending on the fee tier, and for a 1% fee pool this is 200 ticks which corresponds to a 2.02% price difference.

Therefore, `epoch.maxPriceD18` and `baseAssetMaxPriceTick` correspond to different ticks which are 2.02% price difference apart. The `maxPriceD18` is used to bound the settlement price, and is also the `highestPrice` during trades.

`baseAssetMaxPriceTick` is used in the `validateLp` function, which limits the range which liquidity is added.

Since the `baseAssetMaxPriceTick` is lower than `maxPriceD18`, it is impossible for liquidity to be added up to the maximum price, and consequently for traders to swap to that price.

Recommendation

Consider consistently using the `baseAssetMaxPriceTick` to derive the max tick and max price without adding a tick spacing.

If tick adjustment is necessary, consider adding a single `tick` rather than an entire `tickSpacing`.

Resolution

Foil Team: Acknowledged.

M-02 | initializeMarket Can Be Front Run

Category	Severity	Location	Status
Logical Error	Medium	EpochConfigurationModule.sol: 29-45	Resolved

Description

The owner of a market is set with the `initializeMarket` function, which is called after deploying the system and can be called by anyone.

This allows an attacker to take over the market by calling the function before the protocol calls it. A malicious owner would be able to configure malicious Uniswap contracts to steal user funds.

This also acts as a griefing attack as the protocol needs to pay gas for re-deploying the system.

Recommendation

Set the owner of the system at deployment.

Resolution

Foil Team: The issue was resolved in PR#74.

M-03 | Unnecessary Fees When Closing Short

Category	Severity	Location	Status
Logical Error	Medium	EpochTradeModule.sol 567-575	Resolved

Description

When closing a short position in the `_closePosition` function, `position.vEthAmount` is swapped to `vGas`. Next, if `position.borrowedVGas > tokenAmountVGas`, then `vEth` is swapped back to `vGas`.

Since tokens were swapped from `vEth` to `vGas` back to `vEth` the position closer had to pay extra fees for unnecessary swaps when they could just swapped a lower amount of `vETH` initially and skipped the second swap.

Recommendation

Consider first calculating the amount of `vETH` that needs to be swapped to `vGas` to close the position and then executing only a single swap.

Resolution

Foil Team: The issue was resolved in PR#75.

M-04 | Trades May Revert At Maximum Tick

Category	Severity	Location	Status
Logical Error	Medium	EpochTradeModule.sol 252	Acknowledged

Description

The `highestPrice` for a swap is the exchange rate at the maximum tick. However, this exchange rate does not account for fees.

Therefore, the `highestPrice` could prevent swaps which occur near the edge of the tick range, as the price after fees are included exceeds the `highestPrice`.

Recommendation

Account for fees when calculating the highest price.

Resolution

Foil Team: Acknowledged.

M-05 | Settlement Price frontrunning

Category	Severity	Location	Status
Frontrunning	Medium	EpochUMASettlementModule.sol: 53	Resolved

Description

Any user can grief the protocol by frontrunning the `submitSettlementPrice` function call and asserting a price directly to UMA.

UMA determines the assertion ID by taking in the following parameters:
`assertionId = _getId(claim, bond, time, liveness, currency, callbackRecipient, escalationManager, identifier);`

All of which an attacker can copy what Foil was going to use. When the attacker's transaction gets executed first, Foil's will revert shortly after with the following check:

`require(assertions[assertionId].asserter == address(0), "Assertion already exists");`

Recommendation

Since time is one of the parameters to create an assertion, submitting the transaction through a private mem-pool will be sufficient to prevent this attack.

Resolution

Foil Team: Resolved.

M-06 | Owner Can Bypass UMA Assertion Checks

Category	Severity	Location	Status
Logical Error	Medium	EpochConfigurationModule.sol 47-61	Resolved

Description

The `UmaSettlementModule` is a contract which ensures owner submits a valid settlement price.

However, the owner can change the address of the oracle at any time, and the address does not have to be a legitimate oracle.

The owner can call `updateMarket`, change the `optimisticOracleV3` address to themselves, and then call `assertionResolvedCallback()` to accept a malicious price.

Recommendation

Consider only allowing oracle updates when the epoch has not ended.

Resolution

Foil Team: The issue was resolved in PR#76.

M-07 | Underflow loanAmount Calculation

Category	Severity	Location	Status
Underflow	Medium	EpochLiquidityModule.sol: 346-347	Acknowledged

Description

In the `getCollateralRequirementForAdditionalTokens` the `loanAmount` is not capped at a minimum of 0 as it is done in the `updateValidLp` function.

This can lead to underflows if the borrowed amount plus the given increase amount is smaller than the `tokensOwed`.

Recommendation

Calculate the `loanAmount` as it is done in the `updateValidLp` function to prevent underflows.

Resolution

Foil Team: Acknowledged.

M-08 | Collateral Can Be Stuck After Closing Position

Category	Severity	Location	Status
Logical Error	Medium	EpochTradeModule.sol: 122	Resolved

Description

When calling `modifyTraderPosition` to close a position the system will not automatically withdraw all collateral of the position and instead withdraw based on the given `collateralAmount`.

This means when a trader makes the mistake of providing a positive `collateralAmount` to the `modifyTraderPosition` when closing the position, the closed position will still own collateral.

The user can't call the function again with a 0 `tokenAmount` and 0 `collateralAmount` to withdraw the remaining collateral, as this will call `_closePosition` again and perform a 0 token swap which will revert.

Therefore there are only two ways to withdraw the remaining collateral:
• Wait till the epoch is settled (which could take up to a month)
• Call `modifyTraderPosition` to open a new position and close it again (which costs trading fees)

Recommendation

Automatically withdraw all remaining collateral when closing a position.

Resolution

Foil Team: The issue was resolved in PR#75.

M-09 | Rounding In Favor Of User

Category	Severity	Location	Status
Rounding	Medium	Position.sol 190	Resolved

Description

There are instances where the protocol rounds in favor of the user. Even though these rounding errors are small, a user withdrawing even a slight amount more than they are entitled to can lead to insufficient funds to pay out the last withdrawer.

In the `settle` function of `Position.sol`: `self.borrowedVEth` rounds down: `self.borrowedVEth = (self.borrowedVGas * settlementPriceD18) / 1e18;`

Then in the next line `self.borrowedVEth` is subtracted as part of the user's collateral calculation: `self.depositedCollateralAmount = self.vEthAmount - self.borrowedVEth;`

Since `borrowedVEth` is lower than the exact value, then this makes the user's collateral slightly higher than it should be.

In `_afterSettlementSwapExactOut`, this equation calculates the `amountIn` a user needs to get a certain amount out.

Since it rounds down, the user can put in less than their required amount: `requiredAmountInVGas = amountOutVEth.divDecimal(epoch.settlementPriceD18);`

Recommendation

Consider substituting a division function which rounds in the instances where rounding down would be in favor of the user.

Resolution

Foil Team: The issue was resolved in PR#90.

M-10 | Market Updates Invalidate Previous Positions

Category	Severity	Location	Status
Logical Error	Medium	Market.sol 67-73	Resolved

Description

`updateValid()` allows the owner to change the Uniswap v3 `NonFungiblePositionManager`. However, changing this will invalidate the `tokenID`'s of all previous positions, among other problems.

Additionally, an update to `uniswapSwapRouter` will freeze previous positions as the tokens are approved to the old and not the new router.

Recommendation

Consider storing the variables such as the `uniswapPositionManager`, `uniswapSwapRouter` and `optimisticOracle` as part of the epoch parameters so that changes to market parameters only apply to future epochs.

Resolution

Foil Team: The issue was resolved in PR#76.

M-11 | Uniswap Rounding Can Create Insolvent Positions

Category	Severity	Location	Status
Rounding	Medium	Global	Resolved

Description

In Uniswap V3 the `getAmount0Delta` function rounds in the favor of the Uniswap protocol and against the user. Specifically, when supplying liquidity the amount in is rounded up and when burning liquidity the `amountOut` is rounded down. This behavior results in potentially insolvent positions as the collateralization requirement may not have the same rounding against the user.

For example, only one amount can be rounded by 1 wei since the position is assumed to be entirely in one asset. Additionally, the position may not be subject to precision loss at the max tick, but could be at the current tick.

Recommendation

Consider requiring an additional minimal amount of collateral to address any potential rounding from Uniswap that may occur. This amount could be as small as 2 wei.

Resolution

Foil Team: The issue was resolved in PR#91.

M-12 | Fees Missing In Required Collateral Calc

Category	Severity	Location	Status
Logical Error	Medium	Epoch.sol: 247-272	Resolved

Description

When calculating the required collateral for a trade position the uniswap fee to close the position is not included.

Therefore the trader might not be able to close the position with the deposited collateral as the fee was not accounted for.

The same could happen for liquidity positions which are converted to trade positions when they are closed.

Recommendation

Include the uniswap fees in the required collateral calculations.

Resolution

Foil Team: The issue was resolved in PR#47.

Guardian Team: As recommended in M-12 the fee is now added to the collateral requirement calculation, but this is only done if the epoch is settled. The fee is needed in this calculation before it is settled as before settlement trades happen in Uniswap. After settlement, the required collateral calculation is no longer needed in general. We recommend to always add the fee to the collateral requirement calculation.

M-13 | Missing onRecieved Check

Category	Severity	Location	Status
Best Practices	Medium	EpochLiquidityModule.sol	Resolved

Description

Functions `createLiquidityPosition` and `createTraderPosition` are minting position NFT's to `msg.sender`.

If the `msg.sender` is a contract and is not capable of handling NFT related actions and/or is not capable of calling other functions in the system, this position NFT's will stuck at the contract.

Considering all actions related to both LP's and Trader's have NFT ownership check, this can lead to locked funds for users.

Recommendation

Check if the caller of these functions can safely receive ERC721's via `checkOnErc721Received`.

Resolution

Foil Team: The issue was resolved in PR#62.

M-14 | Traders Can’t Close Position Pre Settlement

Category	Severity	Location	Status
Unexpected Behavior	Medium	EpochTradeModule.sol: 112	Acknowledged

Description

Traders may encounter difficulties in closing positions and, to a lesser extent, modifying positions.

But of utmost significance, traders may find themselves unable to close their positions before settlement if Liquidity Providers close their positions first.

As a result, traders may not be able to realize profit based on the current pool price and may have to wait until settlement, leading to temporarily locked funds and potential loss of yield for traders who are unable to close a profitable position promptly.

Recommendation

It is advised to document to users that the option to close trades before settlement is not guaranteed and is dependent on the availability of liquidity.

Resolution

Foil Team: Users in any market understand this is a possibility and are always able to trade out of any position with no slippage at expiration, so there is no additional risk compared to any other instrument/market.

L-01 | Unexpected Collateral Amount Used For LPs

Category	Severity	Location	Status
Unexpected Behavior	Low		Acknowledged

Description

In the `updateValidLp` function the `additionalCollateral` amount is provided by the user but not used as the amount of collateral tokens transferred in.

Instead the minimum required amount of collateral tokens are transferred in. Thus users will unexpectedly transfer in a lower amount of tokens than their provided `additionalCollateral` value.

Recommendation

Consider removing the use of `additionalCollateral` and the associated validation and always transfer in the required collateral, while making it clear to the user how much collateral is to be transferred.

Otherwise consider transferring in the entire `additionalCollateral` amount and using this full amount for the position's collateral.

Additionally, consider standardizing on a consistent behavior across LP positions and trader positions.

Such that either both LP positions and trader positions transfer in the minimum required collateral or both transfer in a specified amount of collateral from the user.

Resolution

Foil Team: Because we are unable to use desired collateral changes as inputs, both trade and LP functions specify a desired position size and slippage protection is implemented by setting limits on additional collateral requirements.

L-02 | Wrong Description For tokenByIndex

Category	Severity	Location	Status
Code Quality	Low	EpochNftModule.sol: 186	Resolved

Description

The description above the `tokenByIndex` function describes the behavior of the `totalSupply` function.

Recommendation

Update the description to match the behavior of the `tokenByIndex` function and move the description to the `totalSupply` function.

Resolution

Foil Team: The issue was resolved in PR#69.

L-03 | Unsafe Collateral Transfers

Category	Severity	Location	Status
Validation	Low	Global	Resolved

Description

Some ERC-20 tokens return a boolean instead of reverting therefore using `transferFrom` will not revert when the transfer fails.

This enables attack vectors in the system when such a token would be used as collateral.

Recommendation

Use `safeTransferFrom` instead of `transferFrom` or be aware not to add such tokens to the system.

Resolution

Foil Team: The issue was resolved in PR#73.

L-04 | Misleading Error In submitSettlementPrice

Category	Severity	Location	Status
Code Quality	Low	EpochUMASettlementModule.sol: 26	Resolved

Description

The `submitSettlementPrice` function checks if the epoch is already settled and returns a "Market already settled" error in that case.

This error is misleading because the epoch is settled, not the market.

Recommendation

Change the error message to "Epoch already settled".

Resolution

Foil Team: The issue was resolved in PR#61.

L-05 | Protocol Vulnerable To Reentrancy

Category	Severity	Location	Status
Logical Error	Low	EpochLiquidityModule.sol: 403-406	Resolved

Description

The `_closeLiquidityPosition` function updates the `depositedCollateralAmount` to 0 after withdrawing it. This pattern is vulnerable to a reentrancy attack if an ERC-777 token is used as collateral.

Also, there is a `ReentrancyGuard` inherited in the `EpochLiquidityModule` but never used.

Recommendation

Use a reentrancy guard in all state changing functions, or be aware not to add such tokens to the system.

Resolution

Foil Team: The issue was resolved in PR#66.

L-06 | Missing Checks In assertionDisputedCallback

Category	Severity	Location	Status
Validation	Low	EpochUMASettlementModule.sol: 105-121	Resolved

Description

The `assertionResolvedCallback` function checks if the given assertion exists & that the epoch is not settled yet, but the `assertionDisputedCallback` function does not.

Recommendation

Add these checks to the `assertionDisputedCallback` function to prevent unexpected state changes.

Resolution

Foil Team: The issue was resolved in PR#64.

L-07 | Tokens With 18 Decimals Are Not Supported

Category	Severity	Location	Status
Validation	Low	Global	Resolved

Description

The system assumes that the collateral token has the same precision as the `vETH` token (18 decimals). If a collateral token with a different precision is added, calculations will be incorrect.

Recommendation

Consider adding a check in the `initializeMarket` flow to ensure that the collateral token has 18 decimals.

Resolution

Foil Team: The issue was resolved in PR#77.

L-08 | Revert On 0 Transfer Tokens Not Supported

Category	Severity	Location	Status
Validation	Low	Position.sol: 94-97	Resolved

Description

The `updateCollateral` could transfer 0 tokens if the given `amt` equals the current `depositedCollateralAmount`.

Some tokens revert on 0 transfers, which can lead to DoS if such a token is added as collateral in the future.

Recommendation

Consider adding a if statement to check if the amount to transfer is greater than 0 before calling `IERC20.transfer`.

Resolution

Foil Team: The issue was resolved in PR#72.

L-09 | swapTokensExactOut DoS In Edge Cases

Category	Severity	Location	Status
Logical Error	Low	EpochTradeModule.sol: 800, 810	Resolved

Description

The `swapTokensExactOut` function checks at the end of the available amount in `vETH` or `vGAS` is bigger than the `amountIn` and reverts otherwise.

This means that it will revert if the available amount equals the `amountIn`.

Recommendation

Consider changing the condition to `availableAmount >= amountIn` to prevent unnecessary reverts.

Resolution

Foil Team: The issue was resolved in PR#75.

L-10 | Rebasing Tokens Are Not Supported

Category	Severity	Location	Status
Logical Error	Low	Global	Acknowledged

Description

As the system calculates within absolute amounts, rebasing tokens are not supported.

When a rebasing token would be used as collateral excess tokens would be stuck in the system if the supply increased, or it would not be possible to withdraw collateral in some cases if the supply decreased.

Recommendation

Handle the collateral amounts with a share price calculation instead, or be aware to not use rebasing tokens as collateral.

Resolution

Foil Team: The issue was resolved in PR#65.

L-11 | Missing Deadline Check

Category	Severity	Location	Status
MEV	Low	EpochTradeModule.sol, EpochLiquidityModule.sol	Resolved

Description

There are no deadline checks when performing swaps through Uniswap.

This can result in swaps occurring long after the transaction was initially submitted which means the user could have an unexpected price and their slippage parameters would be outdated.

Recommendation

Consider adding a deadline check to swaps and liquidity modification actions.

Resolution

Foil Team: The issue was resolved in PR#86.

L-12 | Misleading Function Name

Category	Severity	Location	Status
Code Quality	Low	Epoch.sol 206	Resolved

Description

The function `validateEpochNotSettled()` reverts if the the epoch has ended, regardless of whether it is settled, which contradicts the name

Recommendation

Consider changing the name of the function to reflect it's behaviour

Resolution

Foil Team: The issue was resolved in PR#60.

L-13 | tokenByIndex Off By One

Category	Severity	Location	Status
Logical Error	Low	ERC721EnumerableStorage.sol 46	Resolved

Description

Whenever a token is minted in `createLiquidityPosition()`, it's index is set to `totalSupply() + 1`.

That means that the first token minted has an index of `1`, not `0` and the highest indexed token has an index equal to the total supply.

The `tokenByIndex()` function reverts when `index >= totalSupply`. However it is incorrect to revert when `index == totalSupply` as that is the index of a token that has already been minted.

Recommendation

Consider changing `>=` to `>` in `tokenByIndex`.

Resolution

Foil Team: The issue was resolved in PR#63.

L-14 | Incorrect Comments In modifyTraderPosition

Category	Severity	Location	Status
Documentation	Low	EpochTraderModule.sol 95	Resolved

Description

In `modifyTraderPosition`, it says that "closing can happen at any time". However, positions positions cannot be closed between when the epoch has ended and when it is settled.

Recommendation

Consider removing the misleading comment, and correctly specifying when the position can be closed.

Resolution

Foil Team: The issue was resolved in commit 2d83d89.

L-15 | Outstanding TODO Comments

Category	Severity	Location	Status
Code Quality	Low	Global	Resolved

Description

Throughout the codebase there are outstanding TODO comments, some of which would address findings raised in this report.

Recommendation

Be sure to resolve all TODO comments.

Resolution

Foil Team: The issue was resolved in commit 4f864ab.

L-16 | submitSettlementPrice Overwrites assertionId

Category	Severity	Location	Status
Logical Error	Low	EpochUMASettlementModule.sol: 53	Acknowledged

Description

In the `submitSettlementPrice` function the `assertionId` is overwritten when the owner calls the function a second time before the first assertion has reached a terminal state.

As a result the previous submitted assertion will not be able to settle as the callback functions in the module will revert.

Additionally, a previously submitted assertion could resolve as disputed after a new valid assertion is submitted by the owner.

This would mark the `settlement.disputed` value as `true` and disallow the settlement of the valid true assertion.

Recommendation

Do not allow new assertions to be submitted by the owner until the existing assertion reaches a terminal state.

Resolution

Foil Team: For this iteration of the protocol, we intend to have only the owner address capable of asserting a settlement price. In the event that a false assertion is mistakenly submitted, the intent is that the owner address can immediately resubmit and overwrite the current assertion (resolving the bond sent to UMA separately).

L-17 | Lacking Min/Max Tick Validation

Category	Severity	Location	Status
Validation	Low	Global	Resolved

Description

When creating a new epoch an arbitrary `baseAssetMinPriceTick` and `baseAssetMaxPriceTick` are accepted and used for the respective epoch minimum and maximum prices.

Additionally, an arbitrary `epochParams.feeRate` is provided to determine the fee tier used in the Uniswap pool.

However there is no validation that the `baseAssetMinPriceTick` and `baseAssetMaxPriceTick` are even multiples that adhere to the associated tick spacing of the chosen fee tier.

This may lead to unexpected behavior and accounting issues.

Recommendation

Consider implementing validation in the `Market.createValid` and `Market.updateValid` functions to assert that the `baseAssetMinPriceTick` and `baseAssetMaxPriceTick` are indeed even multiples of the relevant tick spacing for the fee tier provided on the `epochParams`.

Resolution

Foil Team: The issue was resolved in PR#70.

L-18 | Disputes Prevent Settlement

Category	Severity	Location	Status
Logical Error	Low	EpochUMASettlementModule.sol: 98	Acknowledged

Description

In the `assertionResolvedCallback` the settlement is prevented if the `epoch.settled` value has been assigned to true.However this value will be assigned to true as soon as any dispute, valid or not, is submitted.

This is because the `assertionDisputedCallback` is called inside of the `disputeAssertion` function in the `OptimisticOracleV3` contract.

If the dispute is resolved to false, and the original claim is decided to be true, then the `settleAssertion` function will then call the `assertionResolvedCallback` with a value of true for `assertedTruthfully`.

However this call will not settle the epoch or set the settlement price in the Foil system as the `epoch.settlement.disputed` is true.

Recommendation

Use the `assertedTruthfully` value to decide whether the settlement should occur in the `assertionResolvedCallback` function rather than `epoch.settlement.disputed`.

Resolution

Foil Team: Acknowledged.

L-19 | Redundant refundAmountVGas Assignment

Category	Severity	Location	Status
Optimization	Low	EpochTradeModule.sol: 808	Resolved

Description

In the `swapTokensExactOut` function the `refundAmountVGas` is redundantly assigned twice when doing a gas for eth swap.

Recommendation

Remove the second assignment of the `refundAmountVGas` value.

Resolution

Foil Team: The issue was resolved in PR#75.

L-20 | Missing Two Step Ownership Change

Category	Severity	Location	Status
Warning	Low	EpochConfigurationModule.sol: 47-61	Resolved

Description

Only owner of the market can create epochs, update market variables, and submit settlement price.

One step ownership change that is happening with `updateMarket` call is error-prone and can lead to catastrophic results such as not being able to submit settlement price and locked funds.

Recommendation

Implement a two step ownership transfer mechanism, `Ownable2Step` libraries can also be used instead of custom ownership mechanism.

Resolution

Foil Team: The issue was resolved in PR#71.

L-21 | Config Lacks Adequate Opportunity For Disputes

Category	Severity	Location	Status
Warning	Low	Global	Resolved

Description

Dispute time for settlement prices configured as one hour. It is possible that users can not react to settlements within configured time frame.

Recommendation

Consider giving more time to disputers such that wrong prices can be prevented by anyone around the world in any settlement hour. We recommend configuring it to at least six hours.

Resolution

Foil Team: The issue was resolved in PR#68.

L-22 | Constant Time Implementations

Category	Severity	Location	Status
Warning	Low	Epoch.sol	Acknowledged

Description

It is possible to create epoch with variable lengths. Which is prone to mistake and can be guaranteed by code.

Recommendation

Make sure epoch times are 30 days by checking difference between `endTime` and `startTime`.

It is recommended to create epochs with just `startTime` as a variable and `creatingendTimevia` adding 30 days to `thestartTime`.

Resolution

Foil Team: We'd like to retain the ability to have different epoch durations (to fit to the calendar year, for example).

L-23 | Possible To Open Epochs For Past

Category	Severity	Location	Status
Validation	Low	Epoch.sol	Resolved

Description

It is possible to create epochs for the past. Which is prone to mistake.

Recommendation

Be sure to check `startTime` is at least `block.timeStamp`

Resolution

Foil Team: The issue was resolved in PR#67.

L-24 | Zero Checks In updateValid And createValid

Category	Severity	Location	Status
Validation	Low	Market.sol	Resolved

Description

Market can be created and updated with variables that are not putted as parameter which can lead to catastrophic effects.

Recommendation

Be sure to check parameters are not "0" in `createValid` and `updateValid` functions of `Market.sol`

Resolution

Foil Team: The issue was resolved in PR#77.

L-25 | Bond Currency Should Be Constant

Category	Severity	Location	Status
Warning	Low	EpochUMASettlementModul	Acknowledged

Description

A compromised admin has the ability to change the bond currency to an arbitrary token that only they have control over.

This action would enable them to submit any price without any opportunity for the price to be disputed.

Recommendation

It is advised to make the bond currency constant to ensure that the price can always be disputed.

Resolution

Foil Team: The bond currency is set at epoch creation (in EpochParams). A compromised admin would only be able to change it for an upcoming epoch, at which point market participants would have the option to leave the market.

L-26 | Foils Overestimates Needed Collateral

Category	Severity	Location	Status
Logical Error	Low	Epoch.sol 355	Pending

Description

In order to ensure accurate validation of a user's collateral for their LP position, it is essential to calculate the appropriate amount of gas for the position's liquidity.

This is achieved by utilizing the `getAmount0ForLiquidity` function, which has been customized based on `UniswapV3`'s original function.

However, the adjustment made to this function in Foil causes an underestimation of the `maxAmount0` for a position.

This results in the perceived value of the position being lower, leading to a greater collateral requirement to support the loaned amount.

As a consequence, users will be required to deposit slightly more collateral than necessary in some scenarios.

This rounding is fine from the protocol’s perspective because it ensures users are more collateralized instead of less collateralized.

Additionally, the rounding amount is trivial from the user’s perspective.

Recommendation

Be aware of this rounding and document it for users if deemed necessary.

Resolution

Foil Team: Pending.

L-27 | LP Turned To Trader Will Encounter Price Impact

Category	Severity	Location	Status
Documentation	Low	Global	Pending

Description

If an LP provide a liquidity to a ranger in order to short/long and their liquidity is used, hence they became a counter-party to the trader, their position closing will likely require two step:

- Closing the liquidity position which will turn the position to a trader.
- Closing the trader position.

Closing the trader position means swapping the same amount back (if epoch is not settled) which will encounter a price impact in the opposite direction this time.

Uninformed users about this might get surprised with the end result of their position closing before epoch is settled.

Recommendation

Be sure to inform users about all intricacies of being an LP and what their actions will result with exactly.

Resolution

Foil Team: Pending.

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