GUARDIAN

Synthetix

BFP Market

Security Assessment

April 18th, 2024

Summary

Audit Firm Guardian

Prepared By Daniel Gelfand, Owen Thurm, Wafflemakr, 0x3b, Giraffe, Kris Apostolov, Scourgedev

Client Firm Synthetix

Final Report Date April 18, 2024

Audit Summary

Synthetix engaged Guardian to review the security of its BFP market, which aims to allow stablecoin issuers and other DeFi protocols to create delta neutral perpetuals, utilizing ETH and ETH LSTs as collateral. From the 25th of March to the 18th of April, a team of 7 auditors reviewed the source code in scope. All findings have been recorded in the following report.

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

🔗 Blockchain network: **Ethereum Mainnet**

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

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

Table of Contents

Project Overview

Project Summary

Project Name	Syntheitx BFP Market
Language	Solidity
Codebase	https://github.com/Synthetixio/synthetix-v3/tree/main/markets/bfp-market
Commit(s)	33e4843d36ab4baaf8f6e56afd4f2b2a5c8d0f3a

Audit Summary

Delivery Date	April 18, 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	5	0	0	1	0	4
● Medium	21	0	0	2	1	18
● Low	24	0	0	4	1	19

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 BFP market, 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 140,000,000+ runs with a prepared Echidna fuzzing suite.

ID	Description	Tested	Passed	Remediation	Run Count
MGN-01	Position is never liquidatable after a successful margin withdraw	✅	✅	✅	140M+
MGN-02	A modify collateral call will always revert for an account with a flagged position	✅	✅	✅	140M+
MGN-03	A modify collateral call will always revert for an account that has a pending order	✅	✅	✅	140M+
MGN-04	If an account's collateral is 0, then the account's debt must also be 0	✅	✅	✅	140M+
LIQ-01	isPositionLiquidatable never reverts	✅	✅	✅	140M+
LIQ-02	If a position is flagged for liquidation before any function call, the position after is always either flagged for liquidation, or no longer exists	✅	✅	✅	50M+
LIQ-03	remainingLiquidatableSizeCapacity is strictly decreasing immediately after a successful liquidation	✅	✅	✅	50M+
LIQ-04	If a user gets successfully flagged, their collateral will always be 0	✅	✅	✅	50M+
LIQ-05	The sUSD balance of a user that successfully flags a position is strictly increasing	✅	✅	✅	50M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
LIQ-06	The sUSD balance of a user that successfully flags a position increases less or equal to maxKeeperFee	✅	✅	✅	50M+
ORD-01	If an account has an order commited, a subsequent commit order call will always revert	✅	✅	✅	140M+
ORD-02	The sizeDelta of an order is always 0 after a successful settle order call	✅	✅	✅	140M+
ORD-03	An order immediately after a successful settle order call, is never liquidatable	✅	✅	✅	140M+
ORD-04	If a user successfully settles an order, their sUSD balance is strictly increasing	✅	✅	✅	140M+
ORD-05	The sUSD balance of a user that successfully cancels an order for another user is strictly increasing	✅	✅	✅	50M+

Findings & Resolutions

ID	Title	Category	Severity	Status
C-01	User Debt Overwritten When Cancelling Orders	Logical Error	● Critical	Resolved
C-02	Splitting Positions Allows fromAccount To Go Below IM	Logical Error	● Critical	Resolved
C-03	reportedDebt Incorrectly Calculates Funding	Logical Error	● Critical	Resolved
C-04	Market Size Increased Indefinitely With Merge Accounts	Logical Error	● Critical	Resolved
C-05	Negative Price Impact Bypassed For Consistent Profits	Logical Error	● Critical	Resolved
C-06	Users Can Withdraw Their Collateral Without Paying Debt	Logical Error	● Critical	Resolved
C-07	LPs Can Claim Rewards While Avoiding Debt	Logical Error	● Critical	Resolved
C-08	Collateral Modification Abused To Levy LP Fees	Griefing	● Critical	Resolved
C-09	debtCorrection Not Updated Upon Realizing To Account Margin	Logical Error	● Critical	Resolved
H-01	minimumCredit Inaccurately Restricts Backing For Shorts	Logical Error	● High	Acknowledged
H-02	Liquidation Computes Utilization Before Updating Market Size	Logical Error	● High	Resolved
H-03	sUSD Drained From Vault When Liquidating Margin Only	Gaming	● High	Resolved
H-04	minimumCredit Backed By Trader sUSD Collateral	Validation	● High	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
H-05	minimumCredit Reserves Not Validated Upon Position Increase	Validation	● High	Resolved
M-01	Fill Price Causes Funding And Utilization Discrepancy	Logical Error	● Medium	Resolved
M-02	Full Utilization DoS	DoS	● Medium	Resolved
M-03	Wrong Feature Flag Used For Account Split	Validation	● Medium	Resolved
M-04	flagReward Incorrectly Based On marginUsd	Logical Error	● Medium	Resolved
M-05	entryPrice Used To Validate Initial Margin	Logical Error	● Medium	Resolved
M-06	Merging Accounts May Fail For Multicollateral Positions	Logical Error	● Medium	Resolved
M-07	Small Positions Accrue Bad Debt In The System	Logical Error	● Medium	Resolved
M-08	getFillPrice And validateLiquidation Revert Due To Division By 0	Arithmetic Error	● Medium	Resolved
M-09	Risk Free Trade With Merge Callbacks	Gaming	● Medium	Resolved
M-10	Keeper Gas Fee Is Fixed While Execution Gas Is Not	Incentives	● Medium	Partially Resolved
M-11	Non-Discounted Collateral Used To Validate IM	Logical Error	● Medium	Resolved
M-12	Actions May Be Completed When Accounts Are Liquidatable By Margin Only	Validation	● Medium	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
M-13	Lack of Pyth Confidence Interval Check	Logical Error	● Medium	Acknowledged
M-14	Endorsed Keeper May Receive Excessive Fees	Unexpected Behavior	● Medium	Acknowledged
M-15	Utilization Rate Not Bounded Below 1	Logical Error	● Medium	Resolved
M-16	Lacking Execution Incentive During Periods Of High Gas Fees	Incentives	● Medium	Resolved
M-17	Gas Griefing With Settlement Hooks	Gas Griefing	● Medium	Resolved
M-18	Order Cancellation Overcompensates Keepers	Incentives	● Medium	Resolved
M-19	Sudden Block Fee Increases May Cause Insolvent Liquidations	Warning	● Medium	Resolved
M-20	Risk Free Trade With Account Permissions	Gaming	● Medium	Resolved
M-21	Lacking Incentive To Repay Debt	Incentives	● Medium	Resolved
L-01	supportedSynthMarketIds DoS	DoS	● Low	Resolved
L-02	Misleading Comment	Documentation	● Low	Resolved
L-03	Typo	Typo	● Low	Resolved
L-04	reportedDebt Even Skew Optimization	Optimization	● Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-05	Modifying Fees Puts Users At Risk Of Liquidation	Warning	● Low	Acknowledged
L-06	Gas Optimizations For UpdateMarketPreLiquidation	Optimization	● Low	Resolved
L-07	Gas Optimization For getCurrentUtilizationRate	Optimization	● Low	Resolved
L-08	payDebt Always Deducts From Account Margin	Unexpected Behavior	● Low	Resolved
L-09	Unused updatePythPrice Function	Optimization	● Low	Resolved
L-10	getProportionalFundingElapsed Unnecessarily Performs Integer Division	Arithmetic error	● Low	Resolved
L-11	Endorsed Keeper Unable To Fully Liquidate Positions	Validation	● Low	Acknowledged
L-12	Keepers Doubly Incentivized To Flag Liquidatable Positions	Unexpected Behavior	● Low	Acknowledged
L-13	Withdrawals DoS'd By Merge Hook	DoS	● Low	Partially Resolved
L-14	Unnecessary Approvals	Optimization	● Low	Resolved
L-15	Position Mapping Not Cleared Upon Full Split	Unexpected Behavior	● Low	Resolved
L-16	Modify Collateral Permission Too Lenient	Access Control	● Low	Follow Up
L-17	Insolvent Positions DoS getWithdrawableMargin	DoS	● Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-18	Liquidator Fee Can Extract More Rewards Than Expected	Gaming	● Low	Acknowledged
L-19	Incorrect NatSpec	Documentation	● Low	Resolved
L-20	Typo	Typo	● Low	Resolved
L-21	Typo	Typo	● Low	Resolved
L-22	Typo	Typo	● Low	Resolved
L-23	Typo	Typo	● Low	Resolved
L-24	Positions Can Be Liquidated When Adding Collateral Prevented	Logical Error	● Low	Resolved

C-01 | User Debt Overwritten When Cancelling Orders

Category	Severity	Location	Status
Logical Error	● Critical	OrderModule.sol: 458	Resolved

Description PoC

When canceling an order with the cancelOrder function the keeper fee is directly accounted with updateAccountDebtAndCollateral, however if the user has no sUSD collateral updateAccountDebtAndCollateral reassigns the account's debt to the keeper fee, therefore overwriting any existing debt for the account.

Recommendation

Consider Including the accounts existing debt when charging the fee with the updateAccountDebtAndCollateral function, otherwise create a dedicated function to charge the keeper fee from the user's margin.

Resolution

Synthetix Team: The issue was resolved in PR#2079.

C-02 | Splitting Positions Allows fromAccount To Go Below IM

Category	Severity	Location	Status
Logical Error	● Critical	PerpAccountModule.sol: 267	Resolved

Description PoC

In the splitAccount function the toAccount which is created from a portion of the fromAccount is validated to meet the minimum initial margin requirement, however the fromAccount is not validated to still uphold the initial margin requirement after the split.

As a result it is possible for the fromAccount to circumvent the minimum initialMargin and create positions that are prone to insolvent liquidations and create bad debt. Additionally it is possible for a malicious actor to liquidate their small position that is left behind this way and wind up with a net profit from the flag and liquidation fee.

Recommendation

Validate that the fromAccount is still above the initial margin requirement after the split occurs.

Resolution

Synthetix Team: The issue was resolved in PR#2096.

C-03 | reportedDebt Incorrectly Calculates Funding

Category	Severity	Location	Status
Logical Error	● Critical	PerpMarketFactoryModule.sol: 116, 117	Resolved

Description

The reportedDebt function aims to report the net funding fees which have yet to be paid to or from traders, among other things.

Following from the documentation in PerpMarket.sol:

```
/// debtCorrection = positions.sum(p.collateralUsd - p.size * (p.entryPrice + p.entryFunding))
/// marketDebt      = market.skew * (price + nextFundingEntry) + debtCorrection
```

reportedDebt aims to compute the outstanding funding amount via market.skew * nextFundingEntry - positions.sum(p.size * p.entryFunding)

However in the implementation of the reportedDebt function, the unrecordedFunding is used as the nextFundingEntry in the equation above. Instead the currentFundingAccruedComputed + unrecordedFunding ought to be used as the nextFundingEntry. This clearly invalidates the computation of the outstanding funding amount as often the individual p.entryFunding values will be larger than the unrecordedFunding portion.

Recommendation

Use currentFundingAccruedComputed + unrecordedFunding instead of just unrecordedFunding when computing the outstanding funding fees of the market.

Resolution

Synthetix Team: The issue was resolved in PR#2091.

C-04 | Market Size Increased Indefinitely With Merge Accounts

Category	Severity	Location	Status
Logical Error	● Critical	PerpAccountModule.sol: 393	Resolved

Description PoC

Function mergeAccounts can merge any 2 accounts, adding their collateral and combining their positions. If the positions are opposite to each other the accounts will be merged with a reduced position size, however the market size will remain the same. This enables attacker to increase the size of the market as much as they want, while paying only order fees.

The impact from this is complex as size is taken into quite a few calculations and checks:

1. Increasing size increases utilization lowering PnL for other users.
2. Increasing size will reach max OI, which will prevent users from opening trades.
3. Utilizing 100% of the market will cause the LPs to be locked, as delegateCollateral will revert, which leads to minimumCredit checking if we are over the limit.

delegateCollateral -> _verifyNotCapacityLocked -> findMarketWithCapacityLocked -> isCapacityLocked -> getLockedCreditCapacity -> minimumCredit

Note that this doesn't need to be "exploited", as it will occur naturally with use of the protocol.

Recommendation

Check if the two positions are different and if true update the market size.

Resolution

Synthetix Team: The issue was resolved in PR#2095.

C-05 | Negative Price Impact Bypassed For Consistent Profits

Category	Severity	Location	Status
Logical Error	● Critical	PerpAccountModule.sol: 255	Resolved

Description PoC

Users can manipulate the skew to gain profit by making large orders and then merging them to use the fill price penalty as a way to gain a profit. A position is opened with the fill price, where depending on the skew the fill price can deviate from the actual price. That deviation is not realized when opening the position, but left as unrealized PnL.

When merging 2 positions mergeAccounts calculates the margin only for the to address, and assumes the from has no outstanding PnL. Then it adds its collateral and debt and sets the price to the current oracle price.

The above enables us to:
1. Have a small position - 10 USD.
2. Make a new bigger position - 100k USD, using 1% of the skew.
3. Use the hooks and merge this new position (deleting the unrealized loss from the fill price discount).
4. Our new position is at the current oracle price, but the skew is still there.
5. Close the position (lowering the skew) to claim the fill price incentive.

With the above example any user with enough capital can gain constant profits from the market, while only paying order and keeper fees. Furthermore, the disincentive to imbalance longs and shorts is bypassed.

Recommendation

Calculate any outstanding losses for the fromPosition based on the newly assigned fromPosition.entryPrice and account for these in the newly merged to position.

Resolution

Synthetix Team: The issue was resolved in PR#2112.

C-06 | Users Can Withdraw Their Collateral Without Paying Debt

Category	Severity	Location	Status
Logical Error	● Critical	MarginModule.sol: 43	Resolved

Description PoC

Users without sUSD deposited as collateral will accumulate losses in `debtUsd`. This happens in `updateAccountDebtAndCollateral`. In the scenario where a trader closes a position, and realizes a loss, there will be some collateral left and a pending `debtUsd` to pay.

The issue arises when a user tries to withdraw the deposited collateral without any open position. The `validatePositionPostWithdraw` will fail to do its job, as the `position.size` is 0, leading to `isLiquidateable` to return false and `im` to be 0. Therefore, the protocol will allow users to withdraw all their collateral, even with a pending debt to pay.

Recommendation

Validate the case where there is no open position, checking if the `discountedCollateralUsd < debtUsd`.

Resolution

Synthetix Team: The issue was resolved in PR#2100.

C-07 | LPs Can Claim Rewards While Avoiding Debt

Category	Severity	Location	Status
Logical Error	● Critical	LiquidationModule.sol: 90	Resolved

Description PoC

LP's which delegated their collateral can exit the pools between BFP flag and liquidate. On flag all of the collateral is distributed instantly, while keeping totalDebt the same (non-sUSD collateral) since the position still exists. However when liquidating totalDebt will change effectively assigning debt to the LP providers.

This allows LPs who have already staked for some time (above requiredMinDelegationTime) to flag a large position, claim the rewards in the payoutToken and then to decrease their exposure with delegateCollateral to 0, skipping the debt exposure when the position gets liquidated.

Consider the following scenario:
1) Bob and Alice both delegated equal amounts of collateral.
2) 10 ETH is distributed upon liquidatable position being flagged.
3) Alice claims her 5 ETH and removes her delegation.
4) The position is liquidated.
5) The debt is solely distributed to Bob's LP position, so the account's debt is increased.
6) Alice can re-delegate her collateral and all of the distributed debt is still on Bob's position.

Ultimately, Alice was able to avoid the socialization of debt while still gaining the same amount of rewards from the RewardDistributor.

Recommendation

This issue ultimately arises from awarding LPs with liquidated collateral before the liquidated position has been cleared from the reportedDebt. Consider avoiding the distribution of all collateral upon flagging. Instead, distribute collateral upon liquidatePosition, proportional with the amount of size being liquidated or only once the position has been entirely liquidated.

Resolution

Synthetix Team: The introduction of asynchronous delegation will resolve this.

C-08 | Collateral Modification Abused To Levy LP Fees

Category	Severity	Location	Status
Griefing	● Critical	MarginModule.sol: 231	Resolved

Description

When depositing and withdrawing sUSD collateral, a fee is accrued in the depositMarketUsd and withdrawMarketUsd functions in the MarketManagerModule. For example, in a deposit, the fee is subtracted from the amount and added to creditCapacityD18:

```
market.creditCapacityD18 += (amount - feeAmount)
```

However, these fees are handled only inside the Synthetix V3 system, whereas the BFP market adds the entire deposited amount to the trader balance, neglecting the fees:

```
accountMargin.collaterals[synthMarketId] += absAmountDelta
```

This way the trader does not experience the fees levied from their collateral deposit and withdrawal actions, rather the liquidity providers and potentially traders with unrealized profits are affected. This can be exploited to manipulate creditCapacityD18 and netIssuanceD18 by continuously calling modifyCollateral in order to increase netIssuanceD18 and decrease creditCapacityD18.

Anyone can batch deposit-withdraw calls in one TX using Morpho flashloans (0% fees) in order to maximize the impact with minimal capital requirements.

Increasing netIssuanceD18 will report more debt for the LPs than actually exists and lower their profits. Decreasing creditCapacityD18 could lead to traders getting stuck inside the BFP as withdrawMarketUsd reverts inside getWithdrawableMarketUsd and withdrawMarketCollateral can revert with newWithdrawableMarketUsd < 0.

Recommendation

Account for the fees being charged in the BFP market, reducing the amount of collateral traders are credited with upon deposits by the fee amount and reducing the amount of collateral traders ultimately redeem from the Synthetix V3 system by the fees.

Resolution

Synthetix Team: Fees will be removed with SIP-365.

C-09 | debtCorrection Not Updated Upon Realizing To Account Margin

Category	Severity	Location	Status
Logical Error	● Critical	PerpAccountModule.sol: 377	Resolved

Description PoC

In the mergeAccounts function the outstanding PnL for the to account is realized, however the debtCorrection is not updated to account for this realized PnL.

As a result every merge where the to account has outstanding PnL will result in a double counting of that PnL in the debtCorrection, as the skew still includes the old position and the account has now materialized it’s gain or loss into it’s collateral.

Recommendation

Update the debtCorrection for the settlement of the to account’s PnL.

Resolution

Synthetix Team: The issue was resolved in PR#2133.

H-01 | minimumCredit Inaccurately Restricts Backing For Shorts

Category	Severity	Location	Status
Logical Error	● High	PerpMarketFactoryModule.sol: 126	Acknowledged

Description

The minimumCredit function defines an amount which the available credit capacity cannot go below depending on the size of the market and the current index price. However this calculation perturbs the amount that ought to be reserved for shorts.

For example:
• A market has 10 index tokens as short open interest.
• The market size is 10.
• The price of the index token doubles.
• The minimum credit is now a ratio based on the 10 tokens of short open interest, now valued at twice the price.

This inaccurately represents the amount of backing liquidity that ought to be reserved for the market as shorts can only ever gain up to their cost-basis and when the price of the index token rises, shorts are in a loss.

Recommendation

Account for the long and short open interest separately when computing how much backing liquidity ought to be reserved.

For example, the GMX V2 system reserves liquidity based upon openInterestInTokens * price for longs and openInterest (position cost) for shorts.

Resolution

Synthetix Team: Acknowledged.

H-02 | Liquidation Computes Utilization Before Updating Market Size

Category	Severity	Location	Status
Logical Error	● High	LiquidationModule.sol L63	Resolved

Description

During liquidatePosition, updateMarketPreLiquidation is called to perform pre-steps and validation, including the recomputation of utilization. market.recomputeUtilization is incorrectly called before market.size is reduced by the liquidation size. The new utilization rate should be calculated with the new market size, similar to settleOrder in OrderModule.sol.

By calling recomputeUtilization before updating size, the liquidation leaves utilization rate unchanged when it should have reduced it, affecting all remaining traders.

Assume the utilization rate was very high, and a large position was just liquidated. This should in effect bring down utilization rate and improve the margins of all other traders. However, due to this error, the margins of all other traders remain unchanged, which could then lead to unfair liquidations.

Recommendation

Call recomputeUtilization after market size and skew are updated in the updateMarketPreLiquidation function.

Resolution

Synthetix Team: The issue was resolved in PR#2101.

H-03 | sUSD Drained From Vault When Liquidating Margin Only

Category	Severity	Location	Status
Gaming	● High	LiquidationModule.sol: 337	Resolved

Description PoC

In the isMarginLiquidatable function, accounts are determined to be liquidatable when they have a discountedMarginUsd of 0. However this puts the protocol at risk of taking on bad debt as there is no requirement that the margin is able to cover liquidator fees as well as potential instantaneous collateral price decreases.

Furthermore, an issue arises with low price collateral assets. An attacker can deposit a very small amount of collateral (1 wei), and the validation for isMarginLiquidatable will return true, as the discountedCollateralUsd will be 0 when collateral price is below 1e18:
discountedCollateralUsd += available.mulDecimal(discountedCollateralPrice);

The attacker will then liquidate the account and earn keeper fees, withdrawing sUSD from the V3 pool.

Recommendation

Change the definition of the isMarginLiquidatable function such that positions will be considered liquidatable by margin only if their discountedMarginUsd cannot cover liquidation keeper rewards, optionally as well as a safety bound for potential downward collateral price gaps.

Additionally, validate that positions hold enough collateral so that they are not liquidatable by margin only upon any action that would modify their position's collateral.

Resolution

Synthetix Team: The issue was resolved in PR#2116.

H-04 | minimumCredit Backed By Trader sUSD Collateral

Category	Severity	Location	Status
Validation	● High	(core) Market.sol: 277	Resolved

Description

In the BFP market, sUSD collateral is deposited by traders and counted towards the creditCapacity of the market. This creditCapacity is ultimately compared against the minimumCredit to validate that there is sufficient backing liquidity to safely operate the market.

However trader collateral should not be included in this validation as it cannot be locked in the event that the market's isCapacityLocked validation fails, failing to safely back the market with a minimum amount of liquidity and adequately back trader's positions. Additionally trader's profits which are settled to sUSD ought to always be able to be covered by backing liquidity, however this amount is not accounted for in the minimumCredit.

Recommendation

Consider accounting for the trader's deposited sUSD collateral when reporting the minimumCredit for the BFP market such that the creditCapacity must include the desired minimum amount in addition to the creditCapacity generated by the trader's sUSD collateral.

Since the additional sUSD depositedCollateral will include all settled profits and only some of the realized losses which can be covered directly by sUSD collateral, it may also be pertinent to reduce the minimumCredit by the net debt losses for all accounts. However the most conservative validation would be to count only the depositedColalteral value for sUSD.

Otherwise consider a larger refactor of the way trader sUSD collateral is handled, perhaps restricting it's withdrawal when the minimumCredit is breached or removing it from having an effect on the creditCapacity entirely.

Resolution

Synthetix Team: The issue was resolved in PR#2134.

H-05 | minimumCredit Reserves Not Validated Upon Position Increase

Category	Severity	Location	Status
Validation	● High	Position.sol: 202	Resolved

Description

Traders are allowed to open and settle orders even when the existing market positions cannot be adequately supported by the liquidity backing the BFP market. Even if the existing `minimumCredit` is too large for the backing `creditCapacity`, traders can continue to increase the market size and therefore increase the uncovered gap between the `minimumCredit` and the lacking `creditCapacity`.

As a result the BFP market can easily become insolvent in the event that traders continue to open positions without consideration for the backing liquidity. This leads to a market state where sUSD collateral withdrawals are DoS'd as well as any settlement for positions in a profit.

Recommendation

Validate that orders that would create new positions or increase existing positions do not invalidate the `minimumCredit` validation for the market.

Resolution

Synthetix Team: The issue was resolved in PR#2128.

M-01 | Fill Price Causes Funding And Utilization Discrepancy

Category	Severity	Location	Status
Logical Error	● Medium	Position.sol: 223	Resolved

Description

In the Position.validateTrade function the params.fillPrice is used to compute the marginValues, which include the funding and utilization fees based upon that price. However the funding and utilization fees should not be based upon the fillPrice, as this price includes a premium/discount according to how the trade affects the market skew.

This will lead to shorts paying less fees when they push the skew increasingly short, or longs paying more fees when they push the skew increasingly long.

Additionally, when the currentFundingAccruedComputed and currentUtilizationAccruedComputed is accounted for the market with the recomputeUtilization and recomputeFunding functions these values are based upon the pythPrice. As a result traders will experience a discrepancy in the amount of funding and utilization fees paid to the market's recorded funding and utilization accrued values.

Recommendation

Consider using the params.oraclePrice specifically for the funding and utilization fee calculations when computing the marginValues in the Position.validateTrade function.

Resolution

Synthetix Team: The issue was resolved in commit 7c5d2fa.

M-02 | Full Utilization DoS

Category	Severity	Location	Status
DoS	● Medium	PerpMarket.sol: 191	Resolved

Description

When computing the current utilization in the PerpMarket.getUtilization function if the backing liquidity is over-utilized, e.g. delegatedCollateralValueUsd < 0, then a utilization of 100% is returned.

However when the backing liquidity is exactly 100% utilized, e.g. delegatedCollateralValueUsd == 0 then the function will attempt to divide the lockedCollateralUsd by the delegatedCollateralValueUsd, resulting in a divide by 0 panic revert.

However, the risk of DoS is unlikely as the utilization is unlikely to be able to get to exactly 100%.

Recommendation

Modify the if condition on line 191 such that the PerpMarket.getUtilization function early returns if delegatedCollateralValueUsd <= 0.

Resolution

Synthetix Team: The issue was resolved in PR#2085.

M-03 | Wrong Feature Flag Used For Account Split

Category	Severity	Location	Status
Validation	● Medium	PerpAccountModule.sol: 149	Resolved

Description

In the splitAccount function the feature flag validation is performed with the Flags.MERGE_ACCOUNT feature, meanwhile the Flags.SPLIT_ACCOUNT feature ought to be used. This can allow an account which does not have permission to split their account to do so anyway.

Recommendation

Validate the feature flag based upon the Flags.SPLIT_ACCOUNT feature in the splitAccount function.

Resolution

Synthetix Team: The issue was resolved in PR#2084.

M-04 | flagReward Incorrectly Based On marginUsd

Category	Severity	Location	Status
Logical Error	● Medium	Position.sol: 191	Resolved

Description

In the validateNextPositionEnoughMargin function the Maintenance Margin is calculated through the getLiquidationMarginUsd function. However the invocation wrongly passes in the nextMarginUsd to compute the flagReward when it should be passing in collateralUsd.

The flagPosition function computes the flagReward based upon the collateralUsd of the position, therefore using nextMarginUsd to compute the flagReward inaccurately accounts for the amount in the liquidation check.

Recommendation

Provide the collateralUsd as the second to last parameter when invoking the getLiquidationMarginUsd function within the validateNextPositionEnoughMargin function.

Resolution

Synthetix Team: The issue was resolved in PR#2097.

M-05 | entryPrice Used To Validate Initial Margin

Category	Severity	Location	Status
Logical Error	● Medium	Position.sol: 152	Resolved

Description

In the `validateNextPositionIm` function the `getLiquidationMarginUsd` function is used to compute the initial margin value that the position must uphold. However the initial margin value is computed based upon the `newPosition.entryPrice` rather than the `oraclePrice`.

This is in direct contradiction to the price used to calculate and validate the maintenance margin for the position in the `validateNextPositionEnoughMargin` function, which uses the `oraclePrice`.

This leads to a discrepancy in the validation performed on a position when validating a trade. The initial margin is validated based upon the `entryPrice` while the maintenance margin is validated based upon the `oraclePrice`.

Recommendation

In the `validateNextPositionIm` function, call `getLiquidationMarginUsd` with the `oraclePrice` instead of `entryPrice`.

Resolution

Synthetix Team: The issue was resolved in PR#2097.

M-06 | Merging Accounts May Fail For Multicollateral Positions

Category	Severity	Location	Status
Logical Error	● Medium	PerpAccountModule.sol: 321	Resolved

Description PoC

When merging accounts, the function executes getMatchingMarketCollateral which returns the matching margin collateral equal to market. This function will return a matching synthMarketId and fromAccountCollateral.

If the fromId account has multiple collaterals, the getMatchingMarketCollateral function will only return the last collateral that matched. This means that the merge will transfer one collateral to the toId account, and leave the other 2 in the fromId account.

The issue is that the merge will then transfer a portion of the collateral but the whole position size. If the last collateral that matched is the smallest one in terms of USD, then the toPosition might invalidate the Initial Margin check and revert.

Recommendation

Transfer all collaterals with available balance to the toId account.

Resolution

Synthetix Team: The issue was resolved in PR#2095.

M-07 | Small Positions Accrue Bad Debt In The System

Category	Severity	Location	Status
Logical Error	● Medium	OrderModule.sol: 219	Resolved

Description

Users can set their own keeperFeeBufferUsd and limitPrice, potentially forcing bad debt into the system. This is because IM and MM have minimum values as follows:
IM = 2% * position.size + fixed = 2% * p.size + 50
MM = 1% * position.size + fixed + liqFlagReward + keeper reward = 1% * p.size + 50 + liqFlagReward + keeper reward
While the maximum value for keeperFeeBufferUsd is 100 USD.

Currently, it's possible to place an order with keeperFeeBufferUsd > collateral > IM && MM, choosing the maximum keeperFeeBufferUsd and an unrealistic limitPrice, using it to cancel your order later and accrue bad debt of keeperFeeBufferUsd - collateral.

Similarly users can submit orders that would decrease their position size by a trivial amount and avoid the liquidatable checks upon settlement, allowing the keeper fee to place their position in a liquidatable or even insolvent state.

Recommendation

Consider raising the minMarginUsd such that it would not be possible for a keeper fee to exceed an account's margin and cause bad debt to occur. Otherwise consider preventing orders from being created where the keeper fee would cause the account to ultimately become liquidatable.

Resolution

Synthetix Team: The issue was resolved in PR#2117.

M-08 | getFillPrice And validateLiquidation Revert Due To Division By 0

Category	Severity	Location	Status
Arithmetic Error	● Medium	Order.sol: 41, Postion.sol: 368	Resolved

Description

If `skewScale` is set to 0, arithmetic operations which divide by `skewScale` will revert.
This occurs in the `Order.getFillPrice` and `Position.validateLiquidation` functions.
While it is unlikely that `skewScale` will ever be set to 0, this is a possibility and is specifically handled in many areas of the codebase.

Recommendation

In `getFillPrice` and `validateLiquidation` handle the scenario for `skewScale == 0` and avoid division by 0.

Resolution

Synthetix Team: The issue was resolved in PR#2108.

M-09 | Risk Free Trade With Merge Callbacks

Category	Severity	Location	Status
Gaming	● Medium	Global	Resolved

Description PoC

Order execution with the settleOrder function requires that the priceUpdateData provided to parse the pythPrice is for the price update that satisfies the minimum and maximum times (commitmentTime + 12 seconds, commitmentTime + 60 seconds) as well as that the publish time of the price update that is sequentially previous to the provided price data took place before the minimum commitmentTime. Therefore only prices that satisfy these constraints may be used to execute an order while it is ready and not stale.

A malicious user may prevent an order from being executable in the block where the valid price data is accurate and only allow the order to go through once a significant period of time has passed and the user observes that the true current price of the index asset has moved in their favor. The order will only be executable with the outdated price, and therefore the user will realize a risk-free profit based upon how much price has diverged in the user's favor since then.

The malicious user may prevent their order from being executable by registering a mergeAccounts hook as a callback where the user's position as the fromAccount has sUSD collateral in addition to the market's index as collateral and therefore reverts. When the malicious user wishes their order to be executable, e.g. they have determined that price has moved in a direction that is favorable to them, they may remove the sUSD collateral with the payDebt function, assuming they have a pre-existing position with debt, and execute their order with the outdated price.

Recommendation

Consider increasing the orderFee percentage that is taken to dissuade from any risk-free short term trades. Otherwise consider removing the possibility for users to control whether or not their orders are executable by way of callbacks through a try/catch wrapper.

Resolution

Synthetix Team: The issue was resolved in PR#2095.

M-10 | Keeper Gas Fee Is Fixed While Execution Gas Is Not

Category	Severity	Location	Status
Incentives	● Medium	OrderModule.sol: 183	Partially Resolved

Description

Every keeper operation is rewarded with a specific keeper fee, where the gas units are fixed and the fee is calculated on the spot with: `gas * gasPrice + keeperProfit`. However, some operations can use much more gas than others.

Example:
• Settling a normal order versus one with 3 hooks.
• Flagging a position with 1 collateral versus one with 10 collaterals.
• Liquidating a position where you don't need to loop through the window for previous liquidations versus one where you need to loop through every block (using the `while`).

As a result some orders will be significantly more profitable than others, and some orders may end up being unprofitable entirely, even with a fee buffer, and as a result won't be executed.

Recommendation

Consider tracking the gas used during an execution with `gasLeft` and use this amount to compute the keeper’s fee with an added profit margin.

Resolution

Synthetix Team: Partially Resolved.

M-11 | Non-Discounted Collateral Used To Validate IM

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

Description

In the mergeAccounts and splitAccounts functions the IM is validated against the non-discounted margin value. However to be as conservative as possible the IM ought to be validated against the discounted margin value of these accounts

Recommendation

Validate the accounts in the mergeAccounts and splitAccounts functions against the IM based upon the discounted value of the margin.

Resolution

Synthetix Team: The issue was resolved in PR#2112.

M-12 | Actions May Be Completed When Accounts Are Liquidatable By Margin Only

Category	Severity	Location	Status
Validation	● Medium	Global	Resolved

Description

Throughout the BFP market actions are allowed to take place when an account involved is liquidatable by margin only, e.g. the account has a zero discounted margin value and a nonzero collateral value.

Orders can be settled and positions can be merged and split to accounts that are liquidatable by margin only. This can lead to users errantly creating orders for accounts that are about to be liquidated, and thus having the orders cancelled.

This behavior also introduces additional attack surface by allowing these accounts to be involved in such actions, which could potentially lead to unexpected scenarios.

Recommendation

Consider validating that accounts are not liquidatable by margin only in the `validateTrade`, `mergeAccounts`, and `splitAccount` functions.

Resolution

Synthetix Team: The issue was resolved in PR#2115,.

M-13 | Lack of Pyth Confidence Interval Check

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

Description

Currently the system uses `parsePriceFeedUpdatesUnique` to get the first unique price for the given time period.

Pyth provides instant prices, but because market price discovery takes time and happens gradually over all of the markets Pyth has implemented confidence in their system. For example, the returned price for ETH can be $2000 with confidence of +-20 USD. This means the real price of ETH can range from $1980 to $2020.

Currently there are no checks for confidence, which can lead to users not being liquidated in time, lowering the profits for LP providers, or putting them in debt.

Recommendation

Consider using the confidence intervals as described in Pyth’s best practices. If someone wants to open a derivative contract, their collateral may be valued at the lower price. However, if deciding whether someone's margin limits were violated, value their outstanding leveraged position at the higher price.

Resolution

Synthetix Team: Acknowledged.

M-14 | Endorsed Keeper May Receive Excessive Fees

Category	Severity	Location	Status
Unexpected Behavior	● Medium	Position.sol: 382	Acknowledged

Description

In validateLiquidation, liqKeeperFee is calculated based on liqSize which is expected not to exceed maxLiquidatableCapacity. Therefore, getLiquidationKeeperFee is expected to calculate iterations = 1 and return liquidationFeeInUsd * 1.

However, when an endorsed keeper performs a liquidation, liqSize may exceed maxLiquidatableCapacity. In this case, iterations could exceed 1, and the keeper will receive multiples of the keeperFee despite only performing one liquidation.

For example, if liqSize = 10 but maxLiquidatableCapacity = 2, then the keeper will receive five times the keeperFee.

Recommendation

Consider whether this is desired behavior. If it is not desired, then consider adding to getLiquidationKeeperFee:

```
if (ERC2771Context._msgSender() == globalConfig.keeperLiquidationEndorsed) {
      iterations = 1;
}
```

Resolution

Synthetix Team: Acknowledged.

M-15 | Utilization Rate Not Bounded Below 1

Category	Severity	Location	Status
Logical Error	● Medium	PerpMarket.sol: 195	Resolved

Description

The documentation for the getUtilization function states that the collateralUtilization is between zero and one, however this is not true as the lockedCollateralUsd / delegatedCollateralValueUsd ratio is not guaranteed to be below one.

While delegated collateral is locked when it's value drops below the minimumCredit (e.g. lockedCollateralUsd), this does not guarantee that the delegatedCollateralValueUsd will always be greater than the minimumCredit. The delegatedCollateralValueUsd may fall below the minimumCredit based on price action as well as increases in the minimumCredit by the introduction of new positions.

As a result the utilization can be above 1e18, which can lead to unexpected utilization fees as well as a DoS when computing the utilization rate with the getCurrentUtilizationRate function. If the utilization rate is allowed to hit > 3e18 then the highUtilizationRateInterest calculation is at risk of overflowing the uint128 and halting all order execution and liquidations in the BFP market.

Recommendation

Consider explicitly bounding the result from getUtilization below 1e18.

Resolution

Synthetix Team: The issue was resolved in PR#2123.

M-16 | Lacking Execution Incentive During Periods Of High Gas Fees

Category	Severity	Location	Status
Incentives	● Medium	OrderModule.sol: 177	Resolved

Description

In the test environment the maxKeeperFeeUsd is between $50 and $100, however this amount will be insufficient to incentivize the execution of orders and flagging of liquidatable positions during periods of high network usage and gas fees.

The maxKeeperFeeUsd ought to be raised to sufficiently incentivize timely order execution and liquidation.

Subsequently, some users may be unwilling to pay for extremely high execution costs during these times of high network usage. For these users it may be useful to have an order specific maximum keeper fee to limit the potential cost to their account's margin.

Recommendation

Consider raising the maxKeeperFeeUsd to a value that does not constrict the incentive for keepers to execute orders and flag positions for liquidation during periods of high gas fees. Based on the keeperSettlementGasUnits of 1.2 million and assuming an aggressive base fee of 40 gwei at a current price of $3,500 per ETH, the maxKeeperFeeUsd ought to be assigned to roughly $200+ to allow for appropriate incentives for order execution during periods of high network usage.

Additionally consider implementing a maximum keeper fee value that is configurable on a per-order basis to allow users to set their tolerance for network fees.

Resolution

Synthetix Team: Resolved.

M-17 | Gas Griefing With Settlement Hooks

Category	Severity	Location	Status
Gas Griefing	● Medium	OrderModule.sol: 374	Resolved

Description

Users can opt to use settle order hooks, called at the end of the `settleOrder` function.
These hooks will be mainly used for splitting and merging accounts when settling orders. These hooks require explicit permissions from the account holders: `_PERPS_MODIFY_COLLATERAL_PERMISSION`.

If a user commits an order with one of this hooks, they can front run the keeper order and remove the account permissions, reverting the transaction.

Recommendation

Be aware and clearly document that this can be an issue for the keepers.

Resolution

Synthetix Team: The issue was resolved in PR#2126.

M-18 | Order Cancellation Overcompensates Keepers

Category	Severity	Location	Status
Incentives	● Medium	OrderModule.sol: 455	Resolved

Description

Upon cancelling an order, the keeper is compensated with the same fee they would receive for executing the order. However the gas cost necessary to execute an order is far more than the gas cost to cancel an order.

Additionally, keepers will receive the `keeperFeeBufferUsd` as profit when cancelling an order, when this amount is meant instead to incentivize the execution of an order.
As a result keepers receive far more profit for cancelling orders rather than executing them. Therefore, keepers will not only be overcompensated for cancellation, but highly incentivized to front-run users who are cancelling their own orders to cancel them on the user's behalf and collect a fee from the user's margin.

Recommendation

Consider implementing a fee calculation that is specific to cancellation remuneration rather than overcompensating keepers for the cancellation with the same fee they would receive from execution.

Resolution

Synthetix Team: The issue was resolved in PR#2117.

M-19 | Sudden Block Fee Increases May Cause Insolvent Liquidations

Category	Severity	Location	Status
Warning	● Medium	Position.sol: 406	Resolved

Description

New positions are validated when the order is settled by checking if the margin minus the fees will satisfy the initial and maintenance margin checks. This fees include order, keeper, flag and liquidation fees.

The issue is that the calculations of these fees rely heavily on the `block.basefee` and eth price, the only parameters that the protocol can't control. This base fee can range from 10-15 Gwei in a normal market scenario, up to >600 Gwei when volatility is high.

Therefore, users can open LONG positions when the block base fee is low, and get liquidated a few blocks later if the base fee increases, even if the market price moves in their favor. Potentially, this can cause an insolvent liquidation for small accounts since the base fee jump might be an unpredictable stepwise change.

Recommendation

Consider increasing the market `minMarginUsd` to the point that it can cover the volatility of the base fee and reduce the risk of an insolvent liquidation.

Resolution

Synthetix Team: Resolved.

M-20 | Risk Free Trade With Account Permissions

Category	Severity	Location	Status
Gaming	● Medium	Global	Resolved

Description PoC

Similarly to M-09, it is possible to make a short term risk free trade by preventing keepers from executing an account's order by including a splitAccount callback where no keeper is granted the necessary `_PERPS_MODIFY_COLLATERAL_PERMISSION` for the `fromId` account.

As a result, a malicious user may prevent an order from being executable in the block where the valid price data is accurate and only allow the order to go through once a significant period of time has passed and the user observes that the true current price of the index asset has moved in their favor.

The order will only be executable with the outdated price, and therefore the user will realize a risk-free profit based upon how much price has diverged in the user's favor since then.

Recommendation

Consider increasing the `orderFee` percentage that is taken to dissuade from any risk-free short term trades. Otherwise consider removing the possibility for users to control whether or not their orders are executable by way of callbacks through a `try/catch` wrapper around the callbacks.

Resolution

Synthetix Team: The issue was resolved in PR#2126.

M-21 | Lacking Incentive To Repay Debt

Category	Severity	Location	Status
Incentives	● Medium	Global	Resolved

Description

In the BFP market users can leave accounts with realized debtUsd and enough collateral to support that debtUsd without ever repaying the debt. The debt inside these accounts will be reported in reportedDebt as profits for the LPs, however the sUSD will never be returned and never increase the creditCapacity for the market since the debt is not repaid.

Over time with many positions holding unpaid debt there is an increased risk of reducing the getWithdrawableMarketUsd to a point where traders are unable to withdraw their sUSD collateral or claim their sUSD profits.

Recommendation

Consider implementing an interest rate on unpaid debt, where traders pay a configurable rate on debt that has not yet been repaid in sUSD.

Resolution

Synthetix Team: The issue was resolved in PR#2170.

L-01 | supportedSynthMarketIds DoS

Category	Severity	Location	Status
DoS	● Low	Global	Resolved

Description

Throughout the codebase there are many instances where the `supportedSynthMarketIds` list is iterated over and often expensive operations are performed. When configuring the `supportedSynthMarketIds` list with the `setMarginCollateralConfiguration` function there is no validation on the maximum length of the list.

As a result it may be possible for the length of the `supportedSynthMarketIds` to be assigned such that many operations are extremely gas intensive or entirely DoS'd as they would exceed the block gas limit.

This could cause significant harm if e.g. an account liquidation would require too much gas to be executed. However it is unlikely that the `supportedSynthMarketIds` list would grow extremely long as currently only three synthetic assets are planned to be supported.

Recommendation

Consider implementing validation in the `setMarginCollateralConfiguration` function such that the `supportedSynthMarketIds` cannot exceed an acceptable length.

Resolution

Synthetix Team: The issue was resolved in PR#2084.

L-02 | Misleading Comment

Category	Severity	Location	Status
Documentation	● Low	LiquidationModule.sol: 370	Resolved

Description

In the liquidateMarginOnly function the comment on line 370 indicates that the keeperReward will go to pay the flagger, however this amount is not paid to the flagger instead it is paid to the msg.sender.

Recommendation

Update the comment to be clear that this fee is intended to be sent to the msg.sender, not the flagger.

Resolution

Synthetix Team: The issue was resolved in PR#2084.

L-03 | Typo

Category	Severity	Location	Status
Typo	● Low	PerpAccountModule.sol: 218	Resolved

Description

On line 218 in the `PerpAccountModule`, `toPosition` is misspelled as `toPostion`.

Recommendation

Correct it to `toPosition`.

Resolution

Synthetix Team: The issue was resolved in PR#2084.

L-04 | reportedDebt Even Skew Optimization

Category	Severity	Location	Status
Optimization	● Low	PerpMarketFactoryModule.sol: 111	Resolved

Description

In the reportedDebt function there is a short-circuit case which saves gas when market.skew == 0 && market.debtCorrection == 0 && market.totalTraderDebtUsd == 0.

The expensive operation being avoided is the market.getOraclePrice call, however this expensive operation can be avoided in the more general case where market.skew == 0.

Recommendation

In order to avoid the expensive market.getOraclePrice call in as many scenarios as possible, consider changing the short-circuit case to the following:

```
if (market.skew == 0)
    return totalCollateralValueUsd - market.debtCorrection.toUint() -
market.totalTraderDebtUsd;
```

Resolution

Synthetix Team: The issue was resolved in PR#2084.

L-05 | Modifying Fees Puts Users At Risk Of Liquidation

Category	Severity	Location	Status
Warning	● Low	Position.sol: 468	Acknowledged

Description

Changing any market parameters can instantly put users at risk of liquidation. This is because most parameters are closely tied to liquidation calculations, and altering just one can change the liquidation threshold for users. Examples include flag fees, maker and taker fees, and skew scale which affect the liquidation keeper fees, etc.

Recommendation

Consider introducing a timelock for modifying parameters that could cause positions to be at risk of liquidation.

Resolution

Synthetix Team: Acknowledged.

L-06 | Gas Optimizations For UpdateMarketPreLiquidation

Category	Severity	Location	Status
Optimization	● Low	LiquidationModule.sol: 86	Resolved

Description

In `updateMarketPreLiquidation` the following optimization can be made:

`market.updateDebtCorrection(market.positions[accountId], newPosition);`
can be changed to `market.updateDebtCorrection(oldPosition, newPosition);`

Recommendation

Consider implementing the suggested optimization.

Resolution

Synthetix Team: The issue was resolved in PR#2136.

L-07 | Gas Optimization For getCurrentUtilizationRate

Category	Severity	Location	Status
Optimization	● Low	PerpMarket.sol: 207	Resolved

Description

In the `getCurrentUtilizationRate` function the `utilization < utilizationBreakpointPercent` case can be optimized to account for the case where `utilization == utilizationBreakpointPercent`.

This is because when the `utilization` is the same as the `utilizationBreakpointPercent` the resulting value is entirely based upon the `lowUtilizationSlopePercent`.

Recommendation

Consider updating the low utilization case to: `utilization <= utilizationBreakpointPercent`.

Resolution

Synthetix Team: The issue was resolved in PR#2136.

L-08 | payDebt Always Deducts From Account Margin

Category	Severity	Location	Status
Unexpected Behavior	● Low	MarginModule.sol: 471	Resolved

Description

Permission can be granted for arbitrary addresses to `payDebt` on behalf of other users. However, when another user pays back the account's debt, the system will still deduct from the account's sUSD collateral.

The `accountId` is checked for any sUSD, and if found, the system reduces the paid amount by this sUSD while also reducing the account collateral in the sUSD market. This behavior may be unexpected for the user who is making the debt payment and can unintentionally place positions at a greater risk of liquidation than intended.

Recommendation

Be sure to document this behavior to users as it may be unexpected.
Otherwise consider adding a boolean parameter and use it to determine if sUSD from the account's margin should be used.

Resolution

Synthetix Team: The issue was resolved in PR#2119.

L-09 | Unused updatePythPrice Function

Category	Severity	Location	Status
Optimization	● Low	PerpMarket.sol: 110	Resolved

Description

The updatePythPrice internal function in the PerpMarket library is not used.
It seems this was meant to be used to manually update pyth prices without the need to create or cancel an order.

Additionally, this functions creates extra deployment costs, besides causing confusion to the reader of the contract.

Recommendation

Remove the unused internal function or add the external function with a proper access control so that pyth prices can be updated manually.

Resolution

Synthetix Team: The issue was resolved in PR#2119.

L-10 | getProportionalFundingElapsed Unnecessarily Performs Integer Division

Category	Severity	Location	Status
Arithmetic error	● Low	PerpMarket.sol: 298	Resolved

Description

The `getProportionalFundingElapsed` function performs `divDecimal` between the `block.timestamp - self.lastFundingTime` and `1 days`. Both of these values can be divided as `uint` values, however `toInt` is invoked on the time since the last funding. As a result the `divDecimal` overload accepting two `int256` variables is used.

This produces no unexpected behavior with the values being used, however to maximize the allowed domain space for these values, `uint` values ought to be used instead.

Recommendation

Consider modifying the `getProportionalFundingElapsed` definition such that the `divDecimal` overload accepting `uint` values is used:

```
return (block.timestamp - self.lastFundingTime).divDecimal(1 days).toInt();
```

Resolution

Synthetix Team: The issue was resolved in PR#2119.

L-11 | Endorsed Keeper Unable To Fully Liquidate Positions

Category	Severity	Location	Status
Validation	● Low	Position.sol: 352	Acknowledged

Description

keeperLiquidationEndorsed may not always be able to liquidate a user on the first attempt. This limitation arises because the condition allowing them to liquidate 100% of a user's position requires two criteria to be met: ERC2771Context._msgSender() == globalConfig.keeperLiquidationEndorsed and runtime.remainingCapacity == 0. This finding concerns the second criterion.

For the endorsed keeper to liquidate 100% of a user's position, remainingCapacity must be 0. However, this is not always the case as remainingCapacity restores (increases) with each block, due to the expiration of old liquidations.

In a busy market, it's common for there to be liquidations in every block. If our keeper is the first to attempt liquidation in the current block, remainingCapacity will **not be 0**. This results in the endorsed keeper liquidating only a tiny fraction of the user's position.

Consider the following example:
maxLiquidatableCapacity = 80 ETH with a refresh rate of 30 seconds on Base (15 blocks).
1. The market is volatile and busy, with significant MEV and front-running activity.
2. Small liquidations occur every block, depleting the capacity.
3. A large position needs immediate liquidation as prices move quickly.
4. The keeper, being first to call liquidate, but remainingCapacity has replenished with a few ETH.
5. The keeper liquidates only a small fraction of the order.

Under these circumstances, the endorsed keeper needs to call liquidate again in the same block, before remainingCapacity is replenished in the next.

Recommendation

Consider removing the runtime.remainingCapacity == 0 requirement for the endorsed keeper. If the endorsed keeper calls to liquidate a user, that user often must be liquidated in entirety in a timely manner.

Resolution

Synthetix Team: Acknowledged.

L-12 | Keepers Doubly Incentivized To Flag Liquidatable Positions

Category	Severity	Location	Status
Unexpected Behavior	● Low	Position.sol: 411, 418	Acknowledged

Description

When computing the liquidation flag reward in the `getLiquidationFlagReward` function the keeper receives a profit margin from two sources: the maximum of the `keeperProfitMarginPercent` or the `keeperProfitMarginUsd` as well as the `liquidationRewardPercent`.

This doubly incentivizes liquidators to flag positions, which may be intentional to ensure liquidations always occur in a timely manner, but raises an inconsistency with the way other keeper actions such as executing liquidations or settling orders are remunerated.

Recommendation

Consider if it is expected for liquidation flaggers to be compensated with profit margin twice, if it is not then adjust the `getLiquidationFlagReward` function accordingly.

Resolution

Synthetix Team: Acknowledged.

L-13 | Withdrawals DoS'd By Merge Hook

Category	Severity	Location	Status
DoS	● Low	PerpAccountModule.sol: 321	Partially Resolved

Description

The withdrawAllCollateral function prevents users from withdrawing their collateral if debtUsd is non-zero. If this user has previously given permissions to the merge settlement hook, a malicious user can front run a withdrawal, and settling an order with this hook, transferring some debtUsd (even 1 wei) to the target account, forcing the withdrawAllCollateral function call to revert.

Recommendation

Document this issue to users so they can revoke any permissions previously given to other contracts before they attempt to withdraw their collateral or otherwise accept the risk of this occurring.

Resolution

Synthetix Team: The issue was resolved in PR#2126.

L-14 | Unnecessary Approvals

Category	Severity	Location	Status
Optimization	● Low	MarginModule.sol: 122, 124	Resolved

Description

The owner address can configure the collaterals supported by the market using the `setMarginCollateralConfiguration`. This function will internally make an infinite approval to several addresses. Firstly an approval is made to the core Synthetix address, which is necessary to transfer market deposited collateral from traders.

Secondly, the spot market is approved however this approval amount is never utilized and unnecessarily introduces risk to the BFP market.

Thirdly, the BFP market address itself is approved to transfer all sUSD synths, however nowhere in the modules which will act on this address is this approval amount used.

Recommendation

If there is a reason for these approvals then clearly document them. Otherwise, remove the unnecessary approvals to the spot market and to the BFP market contract itself.

Resolution

Synthetix Team: The issue was resolved in PR#2119.

L-15 | Position Mapping Not Cleared Upon Full Split

Category	Severity	Location	Status
Unexpected Behavior	● Low	PerpAccountModule.sol: 143	Resolved

Description

Splitting an account with 100% proportion does not delete the `fromId` position from the `positions` mapping. Although this yields a position with size 0, the logic should coincide with the `settleOrder` logic, where the position is deleted from the `positions` mapping when size is zero.

Recommendation

Remove the from position from the positions mapping if it is found to have zero size after splitting.

Resolution

Synthetix Team: The issue was resolved in PR#2119.

L-16 | Modify Collateral Permission Too Lenient

Category	Severity	Location	Status
Access Control	● Low	MarginModule.sol: 452	Resolved

Description

Users can grant permissions to other addresses so that they can interact with their accounts. The issue is that `_PERPS_MODIFY_COLLATERAL_PERMISSION` may be too lenient for the address, as this role can now deposit/withdraw collateral, pay debt, split and merge accounts.

This prevents users to give specific permissions, for example, if they only want to allow a user to deposit or pay debt, but not withdrawing collateral. Furthermore, if an address has this permission it can open a new position for the user with `mergeAccount`, which can be unexpected for users.

Recommendation

Consider creating more granular permissions for account access. Otherwise, clearly document this access control behavior to users.

Resolution

Synthetix Team: The issue was resolved in PR#2140.

L-17 | Insolvent Positions DoS getWithdrawableMargin

Category	Severity	Location	Status
DoS	● Low	MarginModule.sol: 589	Resolved

Description

In the getWithdrawableMargin function when the size of the position is 0 the collateralUsd - debtUsd of the account is the uint value returned. However there is no guarantee that this resulting value will be a positive integer.

Ideally accounts will always remain solvent and be liquidated in a timely fashion before their debt is able to eclipse their collateral, but in the scenario where this does occur the getWithdrawableMargin function ought to not revert due to underflow and return a result of 0.

This way integrating systems relying on getWithdrawableMargin will not experience a DoS when positions enter an insolvent state.

Recommendation

Return the minimum of collateralUsd - debtUsd and 0.

Resolution

Synthetix Team: The issue was resolved in PR#2135.

L-18 | Liquidator Fee Can Extract More Rewards Than Expected

Category	Severity	Location	Status
Gaming	● Low	Position.sol: 382	Acknowledged

Description

The liquidator keeper fee is calculated based on a block base fee, eth price and some market and global configs. When a position is flagged, this fee is accounted for in the maintenance margin.
Depending on the position size and the `remainingCapacity` of the market for liquidations, a position can be either liquidated in a single transaction or multiple partial liquidation transactions. The iterations are calculated based on an ideal scenario where there is full capacity when liquidating the entire position.

Therefore, after a position is flagged, the keeper can actually earn multiple `liqReward` amounts:
Let's take an example of a 10 ETH position that needs to be liquidated
1. remainingCapacity in window = 1.
2. Position flagged and partially liquidated (10-1 = 9).
3. Next block, call `liquidatePosition` again, depending on prev market liquidations in window, keeper either liquidates fully or partially again.

This means that a keeper may earn 2 or more liquidation fee rewards, when only 1 was accounted for in the margin requirement. In the worst case, a small account happens to be liquidated on the precipice of the liquidation capacity for a single block, and must be liquidated over two blocks. However the maintenance margin for this account did not factor this in, and therefore the additional liquidation fees cannot be covered by the margin, resulting in bad debt.

Recommendation

Consider raising the minimum required collateral for a position such that small positions cannot inadvertently cause bad debt through experiencing more liquidation fee iterations than expected. Additionally, the margin requirement threshold could pessimistically assume that some liquidation capacity will have been taken up in the block when the account is liquidated and account for an additional liquidation iteration when the position is above a certain size.

Alternatively, consider solutions which would account for the current used liquidation capacity, factoring in if a position would require more than one liquidation given the amount of capacity that is currently used up. Though such a solution may be dissatisfactory as it introduces a potential liquidation manipulation vector.

Finally, consider the use of a monitoring system that checks the iterations and liquidation sizes for flagged positions. This way endorsed keepers could be used in outlier scenarios to prevent overpaying liquidation fees by liquidating smaller accounts in a single transaction when they would otherwise incur bad debt.

Resolution

Synthetix Team: Acknowledged.

L-19 | Incorrect NatSpec

Category	Severity	Location	Status
Documentation	● Low	OrderModule.sol: 157	Resolved

Description

The following functions have incorrect NatSpec:

- `recomputeUtilization` is using the documentation from the `recomputeFunding` function.
- `getSettlementKeeperFee` states that the fee is used for liquidations, which is incorrect.

Recommendation

Correct the NatSpec for these functions.

Resolution

Synthetix Team: The issue was resolved in PR#2119.

L-20 | Typo

Category	Severity	Location	Status
Typo	● Low	PerpMarket.sol: 179	Resolved

Description

In the comment describing the early return case in the getUtilization function, positions is misspelled as postions.

Recommendation

Correct postions to positions.

Resolution

Synthetix Team: The issue was resolved in PR#2119.

L-21 | Typo

Category	Severity	Location	Status
Typo	● Low	Margin.sol: 304	Resolved

Description

In the Margin library the docstring for the getOracleCollateralPrice misspells oracleManager as oraclerManager.

Recommendation

Correct it to oracleManager.

Resolution

Synthetix Team: The issue was resolved in PR#2119.

L-22 | Typo

Category	Severity	Location	Status
Typo	● Low	OrderModule.sol: 284	Resolved

Description

In the comment on line 283 in the settleOrder function getHealthData is misspelled as getHeathData.

Recommendation

Correct it to getHealthData.

Resolution

Synthetix Team: The issue was resolved in PR#2119.

L-23 | Typo

Category	Severity	Location	Status
Typo	● Low	Position.sol: 269	Resolved

Description

In the comment on line 269 in the validateTrade function, as is repeated twice.

Recommendation

Remove the second instance of the word as.

Resolution

Synthetix Team: The issue was resolved in PR#2119.

L-24 | Positions Can Be Liquidated When Adding Collateral Prevented

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

Description

The protocol is able to revoke access to the `DEPOSIT` feature, which prevents users from depositing more collateral. It is possible to be in a state such that adding collateral is prevented, yet liquidations can still occur. In such a scenario, users can be unfairly liquidated as they are unable to make their position healthy.

Recommendation

Consider also pausing liquidations for positions that become liquidatable or only allowing liquidations to be made by trusted addresses during this period. Otherwise, clearly document this behavior to users.

Resolution

Synthetix Team: Resolved.

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