

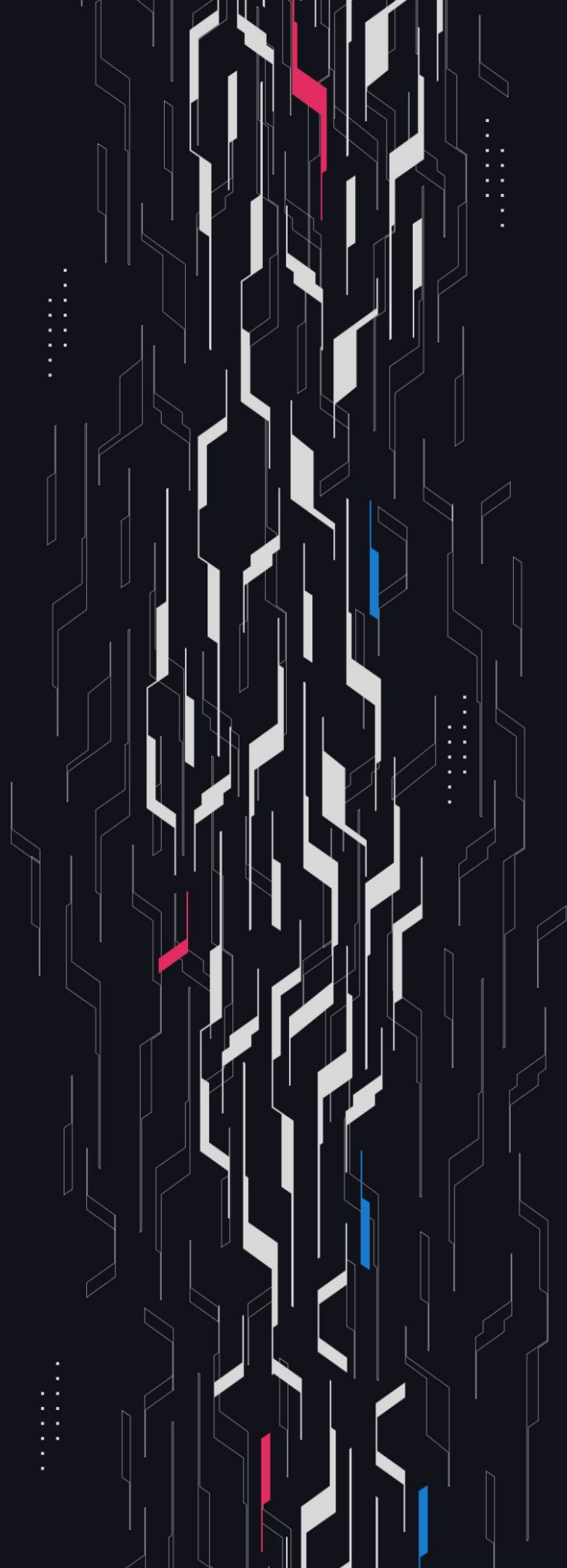
GA GUARDIAN

Synthetix

Perps V3

Security Assessment

October 28th, 2024



Summary

Audit Firm Guardian

Prepared By Daniel Gelfand, Owen Thurm, Dogus Kose, Andreas Zottl, Zdravko Hristov, Nicholas Chew

Client Firm Synthetix

Final Report Date October 28, 2024

Audit Summary

Synthetix engaged Guardian to review the security of its synthetix. From the 2nd of July to the 29th of July, 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 Synthetix team. Several issues impacted the fundamental behavior of the protocol, following their remediation Guardian believes the protocol to uphold the functionality described for the lending protocol product.

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



Blockchain network: **Ethereum, Base, Optimism**



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



Code coverage & PoC test suite: <https://github.com/GuardianAudits/perps-v3-fuzzing-minimal>

Table of Contents

Project Information

Project Overview 4

Audit Scope & Methodology 5

Smart Contract Risk Assessment

Invariants Assessed 7

Findings & Resolutions 11

Addendum

Disclaimer 74

About Guardian Audits 75

Project Overview

Project Summary

Project Name	Synthetix
Language	Solidity
Codebase	https://github.com/Synthetixio/synthetix-v3/tree/main/markets/perps-market
Commit(s)	Initial: fd4c562868761bdcafb1a3dc080c3465e4e4de76 Final: 7e9157e8eda2e9f3df84a9c6b65d701484bf3faa

Audit Summary

Delivery Date	October 28, 2024
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
● Critical	6	0	0	0	0	6
● High	7	0	1	1	0	5
● Medium	21	0	0	14	0	7
● Low	23	0	2	17	0	4

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: <i>High</i>	Impact: <i>Medium</i>	Impact: <i>Low</i>
Likelihood: <i>High</i>	● Critical	● High	● Medium
Likelihood: <i>Medium</i>	● High	● Medium	● Low
Likelihood: <i>Low</i>	● 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 Synthetix, 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 2,000,000+ runs with a prepared Echidna fuzzing suite.

ID	Description	Tested	Passed	Remediation	Run Count
ORD-01	If an account has an unexpired committed order, a subsequent commit order call will always revert	✓	✓	✓	2m
ORD-02	The sizeDelta of an order is always 0 after a successful settle order call	✓	✓	✓	2m
ORD-03	An order immediately after a successful settle order call is never liquidatable	✓	✗	✗	2m
ORD-04	If a user successfully settles an order, their sUSD balance is strictly increasing	✓	✓	✓	2m
ORD-05	The sUSD balance of a user that successfully cancels an order for another user is strictly increasing	✓	✓	✓	2m
ORD-06	The minimum credit requirement must be met after increase order settlement	✓	✗	✗	2m
ORD-07	Utilization is between 0% and 100% before and after order settlement	✓	✗	✓	2m
ORD-08	Non-SUSD collateral should stay the same after profitably settling order	✓	✓	✓	2m
ORD-09	Should always give premium when increasing skew and discount when decreasing skew	✓	✓	✓	2m




























Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
ORD-10	Market utilization rate is always between 0 and 100%	✓	✓	✓	2m
ORD-11	An account should not be liquidatable by margin only after order settlement	✓	✓	✓	2m
ORD-12	An account should not be liquidatable by margin only after order cancelled	✓	✓	✓	2m
ORD-13	Market size should always be the sum of individual position sizes	✓	✓	✓	2m
ORD-14	Position should not be liquidatable after committing an order	✓	✓	✓	2m
ORD-15	Position should not be liquidatable after cancelling an order	✓	✗	✗	2m
ORD-16	Open positions should always be added / removed from the openPositionMarketIds array	✓	✓	✓	2m
ORD-17	All tokens in the activeCollateralTypes array from individual accounts should be included in the global activeCollateralTypes array	✓	✓	✓	2m
ORD-18	Sum of the debt of all accounts == global debt	✓	✗	✓	2m
ORD-19	Debt should not vanish after settle another order	✓	✗	✗	2m
ORD-20	AsyncOrder.calculateFillPrice() should never revert.	✓	✓	✓	2m

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
LIQ-01	isPositionLiquidatable never reverts	✓	✓	✓	2m
LIQ-02	remainingLiquidatableSizeCapacity is strictly decreasing immediately after a successful liquidation	✓	✓	✓	2m
LIQ-03	A user can be liquidated if minimum credit is not met	✓	✗	✗	2m
LIQ-04	All account margin collateral should be removed after full liquidation	✓	✓	✓	2m
LIQ-05	Market deposited collateral should decrease after full liquidation by the account collateral that was liquidated	✓	✓	✓	2m
LIQ-06	User should not be able to gain more in keeper fees than collateral lost in liquidateMarginOnly	✓	✓	✓	2m
LIQ-07	If an account is flagged for liquidations the account is not allowed to have collateral or debt.	✓	✓	✓	2m
LIQ-08	MaxLiquidatableAmount can never return a value greater than requestedLiquidationAmount	✓	✓	✓	2m
LIQ-09	Calling LiquidationModule.liquidate after it has been previously called in the same block should not increase the balance of the caller	✓	✓	✓	2m
MGN-01	Position is never liquidatable after a successful margin withdraw	✓	✓	✓	2m
MGN-02	A modify collateral call will always revert for an account that has a pending order	✓	✓	✓	2m

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
MGN-03	If an account's collateral is 0, then the account's debt must also be 0				2m
MGN-04	depositedCollaterals array should be adjusted by amount of collateral modified (for WBTC)				2m
MGN-05	If sUSD collateral modified, minimumCredit should be updated by that amount				2m
MGN-06	Sum of collateral token values should be the totalCollateralValueUsd stored in the market				2m
MGN-07	User cannot withdraw more non-susd collateral than they deposited				2m
MGN-08	It should never happen that a user has an amount of collateral deposited with a token > 18 decimals precision and withdrawing lead to precision loss				2m
MGN-09	After modifying collateral, a trader should not be immediately liquidatable.				2m
MGN-10	After paying debt, a trader should not be immediately liquidatable.				2m
MGN-11	The sum of collateral amounts from all accounts should always equal the global collateral amount.				2m

Findings & Resolutions

ID	Title	Category	Severity	Status
C-01	All Debt Is Forgiven	Logical Error	● Critical	Resolved
C-02	Global Debt Is Not Changed On Debt Payment	Logical Error	● Critical	Resolved
C-03	Utilization Rate Manipulated Through Deposits	Logical Error	● Critical	Resolved
C-04	Existing Debt Can Be Erased When Settling Order	Logical Error	● Critical	Resolved
C-05	Wrong Collateral Discount	Logical Error	● Critical	Resolved
C-06	RewardDistributor Incompatible With Collaterals	Logical Error	● Critical	Resolved
H-01	validateRequest Errant Price Impact	Logical Error	● High	Resolved
H-02	Lacking minimumCredit Reserves Validation	Logical Error	● High	Resolved
H-03	Bad Debt Incurred During Liquidate Margin	Logical Error	● High	Resolved
H-04	Risk-Free Trades With payDebt	Gaming	● High	Resolved
H-05	LPs Can Game Liquidations Via mintUsd	Logical Error	● High	Acknowledged
H-06	Liquidations Errantly Adjust DebtCorrection	Logical Error	● High	Declined
H-07	Minimum Delegate Time Not Set	Logical Error	● High	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
M-01	MintUSD() Does Not Verify Market Capacities	Gaming	● Medium	Acknowledged
M-02	Price Impact Sandwich Attack	Logical Error	● Medium	Acknowledged
M-03	newRequiredMargin Uses fillPrice	Logical Error	● Medium	Resolved
M-04	Margin Liquidations Should Remove Pending Order	Logical Error	● Medium	Resolved
M-05	Order Fee Calculation Uses The Wrong Price	Logical Error	● Medium	Acknowledged
M-06	Allow Liquidations While Market Is Disabled	Logical Error	● Medium	Acknowledged
M-07	Liquidation Rewards Should Not Use Fill Price	Logical Error	● Medium	Acknowledged
M-08	Liquidations May Liquidate LPs	Logical Error	● Medium	Acknowledged
M-09	Liquidation Fees Not Always Covered	Logical Error	● Medium	Acknowledged
M-10	payDebt Should Update The interestRate	Logical Error	● Medium	Resolved
M-11	Interest Rates Not Accurate To Liquidity Updates	Logical Error	● Medium	Acknowledged
M-12	Stepwise Jump After Update	Logical Error	● Medium	Acknowledged
M-13	Partial Liquidators Not Fairly Rewarded	Logical Error	● Medium	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
M-14	validDistributorExists Not Checked	Validation	● Medium	Resolved
M-15	Fill Price Funding And Interest Discrepancy	Logical Error	● Medium	Acknowledged
M-17	Missing Access Control In payDebt	Validation	● Medium	Resolved
M-18	Max Collateral Can Be Exceeded Through payDebt	Logical Error	● Medium	Resolved
M-19	Account Can Be Made Liquidateable By Cancelling	Unexpected Behavior	● Medium	Acknowledged
M-20	Usage Of DEFAULT Price Tolerance	Gaming	● Medium	Acknowledged
M-21	Unsafe Collateral Amount Because Of Fees	Logical Error	● Medium	Acknowledged
M-22	Vaults With Zero Delegation Prevent Liquidatons	DoS	● Medium	Resolved
L-01	Unused Function	Optimization	● Low	Resolved
L-02	Interest Is Updated Differently	Unexpected Behavior	● Low	Declined
L-03	Wrong Event Emission	Events	● Low	Resolved
L-04	OrderFees May Change	Unexpected Behavior	● Low	Acknowledged
L-05	Incorrect NatSpec	Documentation	● Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-06	Incorrect Array Lengths	Optimization	● Low	Acknowledged
L-07	reportedDebt & minimumCredit Do Not Revert	Validation	● Low	Acknowledged
L-08	Precision Loss In L1 Gas Price Calculations	Precision	● Low	Acknowledged
L-09	Unused Variable In validateRequest	Optimization	● Low	Resolved
L-10	Keeper DoS Griefing Attack	DoS	● Low	Acknowledged
L-11	Liquidate Users By Manipulating Gas Costs	Logical Error	● Low	Acknowledged
L-12	Max Liquidation Windows May Be Under Estimated	Logical Error	● Low	Acknowledged
L-13	System Param Changes May Lead To Liquidations	Logical Error	● Low	Acknowledged
L-14	Utilization Rate Is Unbounded	Logical Error	● Low	Acknowledged
L-15	Precision Loss In valueInUsd Calculations	Precision	● Low	Acknowledged
L-16	Risk Reduced Trades By Fill Price Manipulation	Logical Error	● Low	Acknowledged
L-17	Traders Unfairly Punished During Liquidations	Logical Error	● Low	Invalid
L-18	Endorsed Liquidations Lower Liquidation Capacity	Unexpected Behavior	● Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-19	Funding Rate Is Rounded Down	Precision	<div><div></div></div> Low	Acknowledged
L-20	Funding Rate Lags	Unexpected Behavior	<div><div></div></div> Low	Acknowledged
L-21	Temporary DoS For Committing Orders	DoS	<div><div></div></div> Low	Acknowledged
L-22	Reward Distribution Is Not Future-Proof	Logical Error	<div><div></div></div> Low	Acknowledged
L-23	Immediately Cancelable Orders Are Permitted	Unexpected Behavior	<div><div></div></div> Low	Acknowledged

C-01 | All Debt Is Forgiven

Category	Severity	Location	Status
Logical Error	● Critical	PerpsAccount.sol: 266	Resolved

Description [PoC](#)

If `payDebt` is called with an amount that is larger than the existing debt, the function wipes all existing debt and then attempts to add the remaining amount as collateral to the account.

```
if (self.debt < amount) {self.debt = 0; updateCollateralAmount(self, SNX_USD_MARKET_ID, (amount - self.debt).toInt());}
```

The problem lies with setting `self.debt` to 0 before performing the collateral update. This results in the full amount being updated as collateral, and the previous debt was essentially repaid at no cost.

By just repaying 1 additional wei of debt, a user will have his entire debt repaid and his collateral increased by the debt amount + 1 wei.

Recommendation

Update collateral amount first before setting debt to 0.

Resolution

Synthetix Team: The issue was resolved in [PR#2202](#).

C-02 | Global Debt Is Not Changed On Debt Payment

Category	Severity	Location	Status
Logical Error	● Critical	PerpsAccount.sol: 252-270	Resolved

Description [PoC](#)

[PerpsAccount.payDebt\(\)](#) updates the debt value of individual users directly instead of calling [updateAccountDebt](#).

The user's debt will be updated, but this change will not affect the global market debt and as a result `reportedDebt` will report higher debt.

Recommendation

Don't update the debt directly and instead call `updateAccountDebt`.

Resolution

Synthetix Team: The issue was resolved in [PR#2202](#).

C-03 | Utilization Rate Manipulated Through Deposits

Category	Severity	Location	Status
Logical Error	● Critical	GlobalPerpsMarket.sol: 85	Resolved

Description [PoC](#)

Utilization rate is calculated in GlobalPerpsMarket.sol by `lockedCredit.divDecimal(delegatedCollateralValue)`.

The issue lies with `lockedCredit` which is taken from `minimumCredit` which includes deposited sUSD collateral. Intuitively, this deposited collateral should be excluded as it is not part of the market's positions.

By including it, an attacker can deposit collateral into the market to increase the utilization rate and result in higher interest charged to all traders.

Recommendation

Exclude deposited sUSD from minimum credit when calculating the utilization rate.

Resolution

Synthetix Team: The issue was resolved in [PR#2198](#).

C-04 | Existing Debt Can Be Erased When Settling Order

Category	Severity	Location	Status
Logical Error	● Critical	PerpsAccount.sol: 146	Resolved

Description [PoC](#)

The function `charge` is called when orders are settled to perform collateral and debt accounting.

The issue lies with the scenario where: 1) trader makes a loss, 2) leftover sUSD credit > 0 and 3) existing debt > 0. In that scenario, due to not accounting for existing debt, all debt is reset to 0.

Consider this scenario:

- Trader A deposits ETH collateral, makes a losing trade and incurs debt
- Trader A deposits sUSD collateral, makes another losing trade and has previous debt erased

Recommendation

In `PerpsAccount.charge`, account for existing debt as shown below:

```
if (leftoverCredit > 0) {
    updateCollateralAmount(self, SNX_USD_MARKET_ID, amount); newDebt = self.debt; // insert this
}
```

Also, consider reducing any existing debt when traders deposit sUSD collateral.

Resolution

Synthetix Team: The issue was resolved in [PR#2202](#).

C-05 | Wrong Collateral Discount

Category	Severity	Location	Status
Logical Error	● Critical	PerpsCollateralConfiguration	Resolved

Description [PoC](#)

The discounted collateral is computed in [valueInUsd](#). A discount is a value between `lowerLimitDiscount` and `upperLimitDiscount` that depends on the impact on the spot market.

The calculation is wrong as it has reversed the places of the min and max functions, which will result in wrong calculation. The discount will always be computed using the `upperLimitDiscount`, no matter the impact on the market.

In result, the users' total collateral value will be less than it has to be, which can lead to earlier liquidations.

Recommendation

Switch the ordering of the `min` and `max` functions.

Resolution

Synthetix Team: The issue was resolved in [PR#2211](#).

C-06 | RewardDistributor Incompatible With Collaterals

Category	Severity	Location	Status
Logical Error	● Critical	RewardDistributor	Resolved

Description

[RewardsDistributor.distributeRewards\(\)](#) supports only one collateral type. However, the Perps system [loops](#) over multiple collaterals and calls `distributeCollateral` for each one of them.

This operation will fail and will cause the whole transaction to revert which will DOS the perps market liquidation functionality.

Recommendation

Remove the checks from the `RewardDistributor` contract

Resolution

Synthetix Team: The issue was resolved in [PR#2165](#).

H-01 | validateRequest Errant Price Impact

Category	Severity	Location	Status
Logical Error	● High	AsyncOrder.sol: 316	Resolved

Description [PoC](#)

In the `validateRequest` function the `calculateStartingPnl` is computed as if the fill price affected the entire position, when in fact the `fillPrice` only causes a net change in the position’s margin for the newly added size from the current order.

For example:

- Current WETH price is \$5,000
- Bob has a long position with size 10 WETH at an entry price of \$5,000
- Bob opens an increase long for a size of 1 WETH and is negatively impacted to receive \$5,100 as a fill price
- The computed starting pnl is $11 * -100 = -\$1,100$
- However Bob only received a negative impact of \$100 on his order

As a result orders which build on top of existing positions will have their negative impact errantly accounted for the existing position size in the `currentAvailableMargin` validation, preventing valid orders from being executed.

Additionally, orders which flip the side of the position will not be validated for the entire negative price impact that they experience, as only the net position on the opposite side remains.

Therefore positions can be opened where they are in fact below the necessary margin since the entire negative price impact has not been accounted for.

Recommendation

Compute `calculateStartingPnl` based upon the size delta of the order which is currently being executed instead of the entire size of the new position.

Resolution

Synthetix Team: The issue was resolved in [PR#2198](#).

H-02 | Lacking minimumCredit Reserves Validation

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

Description [PoC](#)

Traders are allowed to open and settle orders even when the existing market positions cannot be adequately supported by the liquidity backing the Perps 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 Perps 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#2198](#).

H-03 | Bad Debt Incurred During Liquidate Margin

Category	Severity	Location	Status
Logical Error	● High	LiquidationModule.sol: 88	Resolved

Description [PoC](#)

Accounts are determined to be eligible for `liquidateMarginOnly` when their `availableMargin < 0`, which implies that the cost of their debt has exceed the value of their collateral.

The collateral discount acts as a buffer which would prevent bad debt from immediate price changes.

However, there is no requirement that margin is able to cover liquidation fees. These fees will always result in bad debt which will build up in the system over time.

Recommendation

When checking eligibility for margin liquidation in `isEligibleForMarginLiquidation`, subtract liquidation fees from `availableMargin`.

Resolution

Synthetix Team: The issue was resolved in [PR#2211](#).

H-04 | Risk-Free Trades With payDebt

Category	Severity	Location	Status
Gaming	● High	PerpsAccountModule.sol: 98-105	Resolved

Description

1. When a trader's collateral and positions are slightly above the threshold to allow creating the wished order, the order can be created, but trying to settle it a few blocks later will revert because interest is accrued every second.
 2. Users are not allowed to deposit funds while they have a pending order waiting to be settled, because of the `checkPendingOrder` check. This can be bypassed by calling the `payDebt` function instead.
 3. Orders can be executed after commit time + delay and till the expiry of the order. The price at commit time + delay is used even if the order is executed later than that (somewhere between this timestamp and the expiry timestamp)
- A malicious actor can abuse these conditions to create a risk-free trade:
- The attacker owns an account that has a big amount of collateral, some debt, and a small position that accrues interest every second
 - Attacker creates an order that requires exactly the available margin to be created
 - A small amount of interest accumulates till the order can be settled
 - Keepers are not able to settle the order because the attacker does not have enough available margin now
 - The attacker waits if the price changes so that the order would be instant profit and if so increases the available margin by calling `payDebt`

Recommendation

Add the `AsyncOrder.checkPendingOrder(account.id);` check to the `payDebt` function.

Resolution

Synthetix Team: The issue was resolved in [PR#2202](#).

H-05 | LPs Can Game Liquidations Via mintUsd

Category	Severity	Location	Status
Logical Error	● High	Global	Acknowledged

Description PoC

During liquidation, an account is flagged, and its positions are closed. However, if the liquidation window is small, some positions may remain open to be subsequently liquidated.

The issue lies with these 'ghost' open positions, which may incur PnL before they are closed. If the positions incur a loss, they temporarily reduce reported debt, and if they incur a profit, the reported debt is temporarily increased.

Additionally, if a position is not liquidated at all due to the capacity window validation, the position's pending PnL at the time of liquidation is not erased to offset the seizure of collateral.

SIP-366, Asynchronous Delegation, aims to address a scenario where LPs can target this intermediate liquidation mis-accounting to undelegate while their position's debt is deflated. LPs can specifically target this intermediate liquidation mis-accounting.

This addresses the vector where LPs would undelegate from the pool. However, LPs may still mint sUSD while they have artificially lowered debt. In this way, an LP could mint sUSD up to the c-ratio while their debt is suppressed.

Once the Perps-V3 account is liquidated, the LP's position would be immediately under the c-ratio by a stepwise and potentially significant amount.

In some cases, LPs may be able to mint more sUSD than they would lose from liquidation. As a result, the LP forces others in the vault to take on socialized debt and can even potentially make keeper fee profits from triggering the Perps-V3 liquidation and their own V3 liquidation in the same block.

Recommendation

Consider re-working the partial liquidation process, to avoid having these 'ghost' positions incur PnL before they are closed. These positions are technically liquidated and should not remain open.

Alternatively, ensure that c-ratios in the V3 core system are assigned such that it would not be possible for an LP to gain a net profit from this intermediate state.

Resolution

Synthetix Team: Acknowledged.

H-06 | Liquidations Errantly Adjust DebtCorrection

Category	Severity	Location	Status
Logical Error	● High	LiquidationModule.sol	Declined

Description [PoC](#)

When there is not enough liquidation capacity left, position liquidations will be done in multiple transactions while the account liquidation will occur in the first transaction.

In every subsequent liquidation for the position, the `debtCorrectionAccumulator` will be updated with the latest funding accrued for the `fundingDelta` and `fundingPnl`, however as the position does not realize the funding changes from the period [`flagPosition`, `liquidatePosition`], this `debtCorrectionAccumulator` adjustment is invalid.

As a result the `reportedDebt` of the market is perturbed.

Recommendation

In the case of a liquidation of a position which was previously flagged, do not use the latest funding changes that took place after the position’s flagging to adjust the `fundingPnl` value as this amount will not be realized to the account’s margin.

Resolution

Synthetix Team: When an account gets partially liquidated, LP’s take on the position and funding repercussions of that position (as well as the margin). So if an account has 1 ETH long and gets partially liquidated to 0.9 ETH long remaining. It’s not like at that point of time that funding gets suspended on that 0.9 ETH, but it simply shift from the trader to LP’s who take on that position.

H-07 | Minimum Delegate Time Not Set

Category	Severity	Location	Status
Logical Error	● High	Global	Resolved

Description

The core system has `setMarketMinDelegateTime()` function which has to be called by integrating markets to set their minimum waiting time for the LPs to withdraw their collateral after they have delegated it.

The `PerpsMarket` doesn't call this function which lets LPs delegate and undelegate collateral in the same block.

This can lead to some unexpected behaviors such as interest rate manipulation or risk-free yield opportunity by sandwiching a settlement to gain from fees by depositing a large amount of collateral and the withdrawing it.

Recommendation

Set some minimum delegate time.

Resolution

Synthetix Team: The issue was resolved in [PR#2276](#).

M-01 | MintUSD() Does Not Verify Market Capacities

Category	Severity	Location	Status
Gaming	● Medium	IssueUSDModule.sol: 52	Acknowledged

Description

The `mintUsd()` function decreases the `creditCapacity` of the markets connected to the pool. However, it does not verify if there is enough `creditCapacity` available to support the markets. As a result, by using `mintUsd`, LPs can potentially push the markets below their `minimumCredit`.

Recommendation

Similar to the practice in `delegateCollateral()`, the addition of the `_verifyNotCapacityLocked()` check to `mintUsd()` is suggested. This measure will prevent the minting of USD if there will be locked markets as a consequence of minting.

Resolution

Synthetix Team: If credit capacity is restricted by minting, then traders will get an incentive to unwind their positions via the assymetric funding rate is mainly the reason.

M-02 | Price Impact Sandwich Attack

Category	Severity	Location	Status
Logical Error	● Medium	AsyncOrder.sol: 295-300	Acknowledged

Description

Users who balance the skew can be sandwiched by an attacker who front runs and back runs the call:

- The skew is at 1000
- The user tries to balance the skew by shorting -1000
- Attacker front runs the call and shorts -1000 (receives a positive price impact)
- User call goes through and the user imbalances the skew by -1000 (receives a negative price impact)
- Attacker back runs the call and longs 1000 (receives a positive price impact)

Recommendation

Inform users about the risk of this attack so the slippage checks are set accordingly.

Resolution

Synthetix Team: Inform users that they can and should in many cases assign the priceLimit ahead of the market price when they would experience positive impact, thus not having that value extracted from them.

M-03 | newRequiredMargin Uses fillPrice

Category	Severity	Location	Status
Logical Error	● Medium	AsyncOrder.sol: 547	Resolved

Description

In `getRequiredMarginWithNewPosition`, `newRequiredMargin` is calculated with `fillPrice`. This is inaccurate as `fillPrice` includes a premium/discount, and therefore should only be used for PnL calculations.

Otherwise, a new position might be within the required margin (based on `fillPrice`) but immediately after settling it could be eligible for liquidation (which calculates margin based on oracle price).

This is also inconsistent with how `oldRequiredMargin` is calculated in the next step, which uses `currentPrice`.

Recommendation

Use `currentPrice` to calculate `newRequiredMargin`.

Resolution

Synthetix Team: The issue was resolved in [PR#2310](#).

M-04 | Margin Liquidations Should Remove Pending Order

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

Description

During `liquidateMarginOnly`, the liquidated account may have a pending order which is not removed. The order cannot be settled and if market price were to go outside of the `priceLimit` range, the order can now be cancelled.

However, as the account has been liquidated, the cancel order settlement reward is charged as debt to the account but may never be repaid. This debt will exist in the system as permanent bad debt.

Recommendation

Remove any pending order during margin liquidations.

Resolution

Synthetix Team: The issue was resolved in [PR#2304](#).

M-05 | Order Fee Calculation Uses The Wrong Price

Category	Severity	Location	Status
Logical Error	● Medium	AsyncOrder.sol: 409	Acknowledged

Description

- The fillPrice is used in the calculateOrderFee function to calculate the notional.
- As the orderFee is a percentage taken from the notional a higher fillPrice will lead to a higher orderFee and a lower fillPrice will lead to a lower orderFee.

This impacts long trades correctly, but short trades wrong:

- Long Trade:
 - With a positive price impact the fillPrice decreases
 - With a negative price impact the fillPrice increases
- Short Trade:
 - With a positive price impact the fillPrice increases
 - With a negative price impact the fillPrice decreases

Therefore a positive price impact on a short (trader balances the OI) increases the fillPrice and therefore also the orderFee of the trader and vice versa.

Recommendation

Use the orderPrice instead of the fillPrice.

Resolution

Synthetix Team: Given that limited repercussions of this imperfection in order fee, we will opt to acknowledge finding and accept it as is.

M-06 | Allow Liquidations While Market Is Disabled

Category	Severity	Location	Status
Logical Error	● Medium	Global	Acknowledged

Description

All state-modifying functions validate that the feature flag `PERPS_SYSTEM` is enabled before allowing transactions to proceed. This includes liquidations, which implies that if the feature flag is disabled, liquidations cannot be processed.

This is undesirable because open positions that become liquidatable while the feature flag is disabled can lead to significant bad debt in the system.

Additionally, traders should be allowed to close their open positions even while the market is disabled to avoid liquidations.

Recommendation

Consider allowing liquidations and the closing of positions while the market is disabled. This could be achieved with more finely-tuned feature flags that apply to different functions, similar to how the BFP market operates.

Resolution

Synthetix Team: Acknowledged.

M-07 | Liquidation Rewards Should Not Use Fill Price

Category	Severity	Location	Status
Logical Error	● Medium	AsyncOrder.sol: 566	Acknowledged

Description

getRequiredMarginWithNewPosition calculates the required margin for a new position by adding possible liquidation rewards to the required margin for the position.

The issue lies with how the liquidation rewards are calculated using fillPrice instead of an oracle price:

```
runtime.accumulatedLiquidationRewards = marketConfig.calculateFlagReward(
  MathUtil.abs(newPositionSize).mulDecimal(fillPrice));
```

- 1. This in contradiction with how isEligibleForLiquidation is calculated which uses an oracle price.
- 2. fillPrice includes a premium/discount according to how the trade affects the market skew. This would unfairly make shorts require a higher margin when they balance the market and less when they imbalance the market.

Recommendation

Use oracle price to calculate the liquidation rewards.

Resolution

Synthetix Team: Given that limited repercussions of this imperfection in order fee, we will opt to acknowledge finding and accept it as is.

M-08 | Liquidations May Liquidate LPs

Category	Severity	Location	Status
Logical Error	● Medium	Global	Acknowledged

Description

- When a trader is liquidated all of the trader's collateral is distributed to the LPs by reallocating it to the distributor where the funds do not contribute to the LP's health.
- After that the positions of the trader (with negative PnL) are closed.

The `totalDebt` is calculated with the following formula and can be negative if the LPs currently make a profit through trader losses: `totalDebt = reportedDebt - marketDepositedCollateral`

Here is an example of how a liquidation could influence this calculation:

- A trader owns \$1000 collateral and has a position with a \$700 loss
- Before the liquidation `totalDebt = 300 - 1000 = -700` (LPs make profit)
- After the liquidation `totalDebt = 0 - 0 = 0` (position is neutral)

As pools delegate to multiple markets it could be that other markets have a positive `totalDebt` and this negative `totalDebt` is needed to balance the LP's health.

Therefore this reallocation of debt to the distributor may cause some LP positions to become immediately unhealthy as they no longer are credited with the negative debt of the position (that was liquidated now).

Their debt and collateral would then be socialized amongst the other LP positions, however these positions would not receive the collateral rewards that would have gone to the liquidated account through the distributor this way.

Recommendation

Consider restructuring the method by which market deposited collateral is distributed to LPs upon liquidation.

Resolution

Synthetix Team: In future iterations, we have ideas that would help lean exactly on this risk, but for the current version, it will be just accepted and properly documented.

M-09 | Liquidation Fees Not Always Covered

Category	Severity	Location	Status
Logical Error	● Medium	LiquidationModule.sol: 340	Acknowledged

Description

- Traders can provide collateral in the sUSD token and also in other tokens.
- When a user is liquidated all of the users non sUSD collateral tokens are distributed among the LPs.
- The liquidator (keeper) receives the liquidation reward in sUSD

Therefore if the liquidated user did not deposit sUSD tokens and only used other tokens as collateral the liquidator will still be paid out in sUSD.

This will cause accrual of negative credit capacity. Once the credit capacity decreases large enough, `withdrawMarketUsd` will revert and traders will not get their collateral back.

Recommendation

Pay out the keeper in the given collateral assets.

Resolution

Synthetix Team: Centralized backup keepers will hopefully lean on this very very edge case.

M-10 | payDebt Should Update The interestRate

Category	Severity	Location	Status
Logical Error	● Medium	PerpsAccountModule.sol: 98-105	Resolved

Description

- 1. The interestRate is calculated based on the utilizationRate. This rate is calculated by comparing the delegated funds from the LPs to the given market, with the open interest of the market.
- 2. The payDebt function pays the LPs funds back that are automatically counted as delegated to the given market.

Therefore the interestRate after calling payDebt changes, but is not updated in the current implementation.

Recommendation

Update the interestRate at the end of the payDebt function.

Resolution

Synthetix Team: The issue was resolved in [PR#2202](#).

M-11 | Interest Rates Not Accurate To Liquidity Updates

Category	Severity	Location	Status
Logical Error	● Medium	Global	Acknowledged

Description

The utilization fees in the Perps-V3 market are dependent on the amount of credit capacity delegated to the market by the V3 core system, however the utilization rate does not update when the amount of backing liquidity changes.

For instance, a delegator can undelegate from a vault which provides credit capacity to the Perps-V3 market and increase the utilization ratio. However this utilization update is not reflected in the utilization rate until a `updateInterestRate` is triggered on the Perps-V3 side.

Therefore the interest rate that is charged can be misrepresentative of the actual amount of liquidity utilized during these periods before an interest rate update is triggered.

A malicious LP could abuse this by minting sUSD directly before a interest rate update in the Perps-V3 market, this way increasing the utilization ratio and forcing the the interest rate for all traders to be higher over the next period.

The LP may then backrun the interest rate update and burn their sUSD that was minted.

Recommendation

Consider calling `GlobalPerpsMarketModule.updateInterestRate` each time backing liquidity in the V3 core system changes.

Resolution

Synthetix Team: Acknowledged.

M-12 | Stepwise Jump After Update

Category	Severity	Location	Status
Logical Error	● Medium	MarketConfigurationModule.sol, GlobalPerpsMarketModule.sol	Acknowledged

Description

A trader's pending funding and interest are calculated based on an accrued value which is updated each time positions are updated based on the current rate and elapsed time.

The issue lies when admin updates funding or interest rate parameters without first realizing the accumulated funding/interest with the old parameters.

Any rate increase/decrease would directly affect the funding/interest that a user would have to pay (positively or negatively).

For example:

T0: User opens position, market caches `interestRate` = 2%

T5: Admin adjusts interest rate gradient, market interest rate should increase to 3% but `InterestRate.update` not performed so new rate is not stored.

T10: User closes position

Interest owed calculated using 2% rate -- (`calculateNextInterest` uses the last cached rate)
Instead user should have been charged 2% from T0-T5, and 3% from T5-T10.

Recommendation

Call `recomputeFunding` and `updateInterestRate` before updating funding or interest rate parameters respectively.

Resolution

Synthetix Team: Acknowledged.

M-13 | Partial Liquidators Not Fairly Rewarded

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

Description

Keepers that perform partial liquidations on flagged accounts may be responsible for liquidating multiple open positions.

However, they only receive a static `liquidateKeeperCost`, as opposed to `flagKeeperCost` (received by keeper that flags the account) which increases based on the number of open positions.

If the gas cost of liquidating multiple positions is greater than the reward, these partial liquidation keepers may not be incentivized to perform the liquidations at all which is detrimental to the system.

On the other hand, the keeper that flags the account for liquidation receives `flagKeeperCost` regardless of number of positions that were actually liquidated.

If the liquidation window happened to be small and only one out of many positions were liquidated, the keeper still receives the cost of liquidating all open positions.

Recommendation

Consider limiting the `flagKeeperCost` and increasing `liquidateKeeperCost` based on the number of open positions actually liquidated.

Resolution

Synthetix Team: Acknowledged

M-14 | validDistributorExists Not Checked

Category	Severity	Location	Status
Validation	● Medium	PerpsAccountModule.sol: 46	Resolved

Description

The `modifyCollateral` function calls `validDistributorExists` which returns a boolean that indicates if a distributor of the given collateral is set or not.

The `modifyCollateral` does not check if the returned boolean is true or false. Therefore if no distributor is set, the flow will continue.

Recommendation

Revert if the distributor was not set.

Resolution

Synthetix Team: The issue was resolved in [PR#files](#).

M-15 | Fill Price Funding And Interest Discrepancy

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

Description

In `_settleOrder` and `updatePositionData`, `oldPosition`'s PnL is called with `fillPrice`, which computes funding and interest based on `fillPrice`.

This is problematic for two reasons:

- a) it causes a discrepancy with `recomputeFunding` which uses `oraclePrice` (see `AsyncOrder.sol:289`).
- b) Funding and interest should not be based upon `fillPrice`, as this price includes a premium/discount according to how the trade affects the market skew.

As a result, shorts pay more fees when they balance the market and less when they imbalance the market. Consider this scenario:

- order price = 1000
- short trade (positive price impact), fill price = 1010 => more fees are paid
- short trade (negative price impact), fill price = 990 => less fees are paid

Recommendation

Consider changing to use `oraclePrice` in these two areas:

1. `_settleOrder`: `oldPosition.getPnl(runtime.fillPrice)`
2. `updatePositionData`: `oldPosition.getPnl(runtime.currentPrice)`

Resolution

Synthetix Team: Acknowledged.

M-16 | Missing Access Control In payDebt

Category	Severity	Location	Status
Validation	● Medium	PerpsAccountModule.sol	Resolved

Description

Except for the payDebt function, all other external, state-modifying functions in the Perps-V3 system have the access control: `FeatureFlag.ensureAccessToFeature(Flags.PERPS_SYSTEM)`.

This could lead to unexpected behavior in the event that the system is paused through the removal of access to flag feature.

Also, the payDebt function also does not check that the msg.sender is indeed the account owner, allowing anyone to reduce debt and increase collateral for another account. It is unclear if this is intended behavior.

Recommendation

Consider adding to the payDebt function:

```
FeatureFlag.ensureAccessToFeature(Flags.PERPS_SYSTEM) and
Account.loadAccountAndValidatePermission(accountId,
AccountRBAC._PERPS_MODIFY_COLLATERAL_PERMISSION)
```

Resolution

Synthetix Team: Resolved.

M-17 | Max Collateral Can Be Exceeded Through payDebt

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

Description

In the `modifyCollateral` function, the added collateral amount is validated through `globalPerpsMarket.validateCollateralAmount` to ensure that the maximum collateral limit is not exceeded.

However, in the `payDebt` function, this check is not performed, even though the function can increase an account's sUSD collateral (as any excess over the amount used to pay debt is added to the collateral).

Additionally, sUSD collateral may exceed the maximum amount through the realization of profits. Moreover, the `payDebt` function lacks the validation to ensure that the maximum number of collateral types per account is not exceeded, which is done through `PerpsAccount.validateMaxCollaterals`.

Recommendation

In the `payDebt` function, call: `globalPerpsMarket.validateCollateralAmount` and `PerpsAccount.validateMaxCollaterals` to ensure both max collateral amount and types are not exceeded.

Resolution

Synthetix Team: Resolved.

M-18 | Account Can Be Made Liquidateable By Cancelling

Category	Severity	Location	Status
Unexpected Behavior	● Medium	AsyncOrderCancelModule.sol	Acknowledged

Description

Cancellation charges account with keeper fee. Which this can put the account into a liquidatable state. This will result in immediate loss for user who could have withdrawn their funds otherwise.

Order’s cancel-ability can be controlled by manipulating the skew. In addition to the fee for cancellation, the liquidation fees provide an added incentive to cancel the order which could even cover the costs of the skew manipulation.

Recommendation

Validate that account won’t be liquidatable after cancellation.

Resolution

Synthetix Team: Acknowledged.

M-19 | Usage Of DEFAULT Price Tolerance

Category	Severity	Location	Status
Gaming	● Medium	Global	Acknowledged

Description

getCurrentPrice from PerpsPrice.sol has a different staleness tolerances for different cases. In the current cannon deployment, node's DEFAULT tolerance is settled as 1 hour.

Using a 1 hour price tolerance might lead to some attack vectors because user can maliciously choose best price for their purpose from the last one hour to use.

Followings are some functions that uses DEFAULT tolerance and possible attack vectors that can be used with them:

- 1. minimumCredit : This can be used to bypass checks (withdrawing collateral from market as an LP when the minimumCredit is in the border)
- 2. isEligibleForLiquidation : Accounts that are liquidatable can settle order via bypassing this check.
- 3. reportedDebt : Can return not accurate values
- 4. View functions: Might be problematic for integrators.

The reason for decreased severity is because it requires no recent update in Pyth price within the time period and because in order to create a large enough impact it requires good amount of price change within one hour window.

Recommendation

Be more strict for DEFAULT tolerance to decrease the possibility of mentioned issues and if possible use STRICT tolerance for specified functions. Also be sure to inform integrators about possible deviations if STRICT tolerance is not gonna be used for view functions.

Resolution

Synthetix Team: Acknowledged.

M-20 | Unsafe Collateral Amount Because Of Fees

Category	Severity	Location	Status
Logical Error	● Medium	AsyncOrder	Acknowledged

Description

When an order is settled, the account the order is committed for is charged `orderFees`. There are validations in [AsyncOrder.validateRequest](#) that the collateral value will not drop below the needed margin after paying the fees.

The following must be true: `currentAvailableMargin = getRequiredMarginWithNewPosition() + orderFees`.

However, [getRequiredMarginWithNewPosition](#) returns 0 when a position is being reduced. This makes the above expression equivalent to:
`currentAvailableMargin = orderFees`

This check is not sufficient. In the following scenario:

- `required margin = $500`
- `currentAvailableMargin = $550`
- `orderFees = 100`

The `currentAvailableMargin = orderFees` will pass, but after the account is charged the fees, its margin will fall below the required margin and their positions will instantly become liquidatable.

Recommendation

Validate the user is not liquidatable after settlement

Resolution

Synthetix Team: Acknowledged.

M-21 | Vaults With Zero Delegation Prevent Liquidations

Category	Severity	Location	Status
DoS	● Medium	LiquidationModule.sol	Resolved

Description

The `LiquidationAssetmanager` contract contains an array of addresses known as `poolDelegatedCollateralTypes`.

When an account undergoes liquidation, the distribution of its collateral takes place within the `distributeCollateral` function of this contract.

This particular function iterates over all pool collateral types, or vaults, and executes `distributeRewards` for each of them in proportion to the amount held by the vault.

By tracing the sequence of calls, we eventually arrive at the `distribute` function within `RewardDistributon.sol`. In this function, an attempt to distribute a reward to 0 vault delegators will result in an error and cause a revert.

Recommendation

To address this issue, we recommend implementing a check to skip the distribution process if either the vault's collateral amount or the reward amount is equal to 0. This way, the liquidation process will not be hindered by attempting to distribute rewards to vaults with zero collateral.

Resolution

Synthetix Team: The issue was resolved in [PR#2213](#).

L-01 | Unused Function

Category	Severity	Location	Status
Optimization	● Low	PerpsMarketFactory	Resolved

Description

The `PerpsMarketFactory.depositMarketUsd()` is not used.

Recommendation

Consider removing the function.

Resolution

Synthetix Team: The issue was resolved in [PR#2305](#).

L-02 | Interest Is Updated Differently

Category	Severity	Location	Status
Unexpected Behavior	● Low	PerpsMarket, GlobalPerpsMarket	Declined

Description

The interest is updated using STRICT price tolerance in GobalPerpsMarket, and using ONE_MONTH price tolerance in GlobalPerpsMarket. This can lead to unexpected differences between the results of the two updates.

Recommendation

Consider if ONE_MONTH is not a too big period.

Resolution

Synthetix Team: Declined.

L-03 | Wrong Event Emission

Category	Severity	Location	Status
Events	● Low	PerpsMarketFactoryModule	Resolved

Description

PerpsMarketFactoryModule.initializeFactory() can be called multiple times, but the factory will be initialized only once. However, the FactoryInitialized event is emitted on each call.

Recommendation

Emit the FactoryInitialized event only when initializing the factory.

Resolution

Synthetix Team: The issue was resolved in [PR#2305](#).

L-04 | OrderFees May Change

Category	Severity	Location	Status
Unexpected Behavior	● Low	AsyncOrder	Acknowledged

Description

The moment an order is committed, order fees are calculated. However, these fees are recalculated when the order is actually settled. If the difference between the fees is too much, the user may end up paying more than they wanted to.

Recommendation

Consider adding a parameter showing how much is the submitter of the request ready to pay for fees.

Resolution

Synthetix Team: Acknowledged.

L-05 | Incorrect NatSpec

Category	Severity	Location	Status
Documentation	● Low	ICollateralConfigurationModule.sol	Resolved

Description

The NatSpec of the `registerDistributor` function says the `distributor` parameter is the previous distributor and that it can be set to `address(0)`. This is wrong - the distributor is the new one and it cannot be set to `address(0)`.

Recommendation

Correct the NatSpec

Resolution

Synthetix Team: The issue was resolved in [PR#2305](#).

L-06 | Incorrect Array Lengths

Category	Severity	Location	Status
Optimization	● Low	KeeperCosts	Acknowledged

Description

`_processWithRuntime()` initializes the `runtimeKeys` and `runtimeValues` arrays with length of 4 when 2 is enough.

Recommendation

Change the length of the arrays to 2.

Resolution

Synthetix Team: Aknownledged.

L-07 | reportedDebt & minimumCredit Do Not Revert

Category	Severity	Location	Status
Validation	● Low	PerpsMarketFactoryModule.sol: 138 & 149	Acknowledged

Description

The `reportedDebt` and `minimumCredit` functions receive the `perpsMarketId` as a parameter and check if the given id equals the own one to ensure the correct contract/market was called.

If the check fails, the functions return 0 debt/credit instead of reverting. This is a dangerous practice as calling the wrong market will not revert the call, instead it goes on with the wrong data.

Recommendation

Revert instead of returning 0.

Resolution

Synthetix Team: Acknowledged.

L-08 | Precision Loss In L1 Gas Price Calculations

Category	Severity	Location	Status
Precision	● Low	OpGasPriceOracle.sol	Acknowledged

Description

In the function `getCostOfExecutionEth`, if `isEcotone` is true, the L1 gas cost for execution is calculated with a division before multiplication, which leads to precision loss:

```
uint256 l1GasPrice = (baseFeeScalar * l1BaseFee * 16 + blobBaseFeeScalar * blobBaseFee)
/ (16 * 10 ** decimals);

costOfExecutionGrossEth = ((gasUnitsL1 * l1GasPrice) + (gasUnitsL2 * gasPriceL2));
```

Recommendation

Consider moving the division of `(16 * 10 ** decimals)` to after `(gasUnitsL1 * l1GasPrice)` has been performed.

Resolution

Synthetix Team: Acknowledged.

L-09 | Unused Variable In validateRequest

Category	Severity	Location	Status
Optimization	● Low	AsyncOrder.sol: 281	Resolved

Description

In the function `validateRequest`, the variable `runtime.currentLiquidationReward` is obtained from `account.isEligibleForLiquidation` but is never used.

Recommendation

Remove the variable `runtime.currentLiquidationReward`.

Resolution

Synthetix Team: The issue was resolved in [PR#2305](#).

L-10 | Keeper DoS Griefing Attack

Category	Severity	Location	Status
DoS	● Low	LiquidationModule.sol: 70	Acknowledged

Description

The liquidation functions will revert if the given account can not be liquidated. As users can increase their collateral or repay debt at any time (as long as no other circumstances prevent it) the following attack path is enabled:

- Keeper tries to liquidate an account that does not have enough available collateral to back its open positions
- The account front runs the transaction and increases its collateral or repays debt
- The keeper wasted gas

Recommendation

Document this so keepers are aware of this griefing attack vector.

Resolution

Synthetix Team: Acknowledged.

L-11 | Liquidate Users By Manipulating Gas Costs

Category	Severity	Location	Status
Logical Error	● Low	PerpsAccount.sol: 504-505	Acknowledged

Description

- To compensate keepers for the gas costs of calling the liquidation functions every account that has open positions must hold the necessary amount of funds to pay these gas costs.
- Therefore if the gas price increases quickly a lot of accounts might be suddenly liquidatable. This can be abused by manipulating the `block.basefee` for example with block stuffing to liquidate users for the reward.

Recommendation

Consider increasing the `minimumPositionMargin` to the point that it can cover the volatility of gas and reduce the risk of liquidations.

Resolution

Synthetix Team: Acknowledged.

L-12 | Max Liquidation Windows May Be Under Estimated

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

Description

When calculating the possible liquidation rewards, function `getKeeperRewardsAndCosts` returns the max number of windows needed to liquidate one position.

The issue lies with the possibility of multiple positions requiring multiple liquidation windows. Just taking the highest number of liquidation windows for one position may not properly account for the gas needed to liquidate all positions.

Recommendation

Consider using the sum of liquidation windows across all positions instead of the max number of windows for one position.

Resolution

Synthetix Team: Acknowledged.

L-13 | System Param Changes May Lead To Liquidations

Category	Severity	Location	Status
Logical Error	● Low	GLOBAL	Acknowledged

Description

Updating system parameters like for example interest rate, funding rate, skew scale, margin requirements, can impact the health of the positions in the market and therefore lead to very fast or even instant liquidations.

Recommendation

Consider adding a timelock for important system parameter updates.

Resolution

Synthetix Team: Acknowledged.

L-14 | Utilization Rate Is Unbounded

Category	Severity	Location	Status
Logical Error	● Low	GlobalPerpsMarket.sol: 92	Acknowledged

Description

The function `utilizationRate` is used to calculate the percentage of delegated collateral is being used in the market, which is then used to calculate the interest rate.

`utilizationRate` is unbounded which means that if `delegatedCollateral` is a very small value, the rate returned and resulting interest could be excessively high.

Recommendation

Consider binding `utilizationRate` to a max value.

Resolution

Synthetix Team: Acknowledged.

L-15 | Precision Loss In valueInUsd Calculations

Category	Severity	Location	Status
Precision	● Low	PerpsCollateralConfiguration.sol	Acknowledged

Description

In the function `valueInUsd`, `impactOnSkew` is calculated with a division before multiplication which could lead to precision loss:

```
uint256 impactOnSkew = amount.divDecimal(skewScale).mulDecimal(self.discountScalar);
```

Recommendation

Perform the multiplication before division.

Resolution

Synthetix Team: Acknowledged.

L-16 | Risk Reduced Trades By Fill Price Manipulation

Category	Severity	Location	Status
Logical Error	● Low	AsyncOrder.sol: 314-347	Acknowledged

Description [PoC](#)

- Orders can be executed after commit time + delay and till the expiry of the order
- The price at commit time + delay is used even if the order is executed later than that (somewhere between this timestamp and the expiry timestamp)
- Therefore if the attacker can make settles revert during these two timestamps if the price does not go in the wished direction and can make it pass again if it goes in the wished direction the attacker can trade risk-free

The following attack path allows reduced risk trading:

- Attacker deposits collateral and creates an order that requires exactly the available margin to be created (this order increases the skew = the fill price is against the order)
- The attacker deposits collateral and creates an order with another account to increase the skew a bit further (in the same block)
- Attacker settles the second order before the first order (by paying more gas than the other keeper)
- Now the first order reverts because the new fillPrice after increasing the skew further ensures that the required margin for the order is bigger than the available one

Now the attacker has two options:

- If the price goes in the wished direction (makes instant profit) the attacker can create a third order to decrease the skew again so that the order goes through
- Else the attacker can wait till the order expires

Therefore the attacker was able to make one risk-free trade with another non-risk-free trade.

Recommendation

Cancel orders that lead to reverts because of price changes.

Resolution

Synthetix Team: Acknowledged.

L-17 | Traders Unfairly Punished During Liquidations

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

Description

During liquidation, all collateral is seized regardless of the amount trading losses and fees. This may be unfair to traders.

Consider this scenario:

Collateral value:

\$2,000

Required Maintenance Margin:

\$1,000

PnL:

-\$1,001

Trader is liquidated and loses all \$2,000 of collateral despite only losing \$1,001 in trading.

Consider this other scenario:

Collateral value:

\$2,000

Required Maintenance Margin:

\$1,000

PnL:

\$0

Collateral value falls to \$999 due to price drop. Trader is liquidated and loses all collateral despite having 0 trading losses.

Recommendation

Consider deducting only trading losses and fees, and return remaining collateral to the trader.

Resolution

Synthetix Team: Liquidations buffers are set as such in order to disincentivize activity that could lead to lp’s losing money. Given that price volatility and liquidity is fleeting. what is suggested here renders the system a lot more fragile.

L-18 | Endorsed Liquidations Lower Liquidation Capacity

Category	Severity	Location	Status
Unexpected Behavior	● Low	LiquidationModule	Acknowledged

Description

Whenever an endorsed liquidation happens, the amount liquidated is being reduced from the liquidation capacity for the relevant windows.

This creates a discrepancy between the amounts that can be liquidated by a normal user and endorsed user based on the order their transactions were executed.

Consider the following example:

- max amount to liquidate in window = 5000
- user A liquidates 4000
- endorsed liquidator liquidates 5000 more

The total liquidated amount is 9000, but if the endorsed liquidator's transaction was executed before the regular user's one, the second transaction would have reverted.

Recommendation

Consider not accounting for endorsed liquidations when calculating the current liquidation capacity

Resolution

Synthetix Team: Acknowledged.

L-19 | Funding Rate Is Rounded Down

Category	Severity	Location	Status
Precision	● Low	PerpsMarket	Acknowledged

Description

The calculations for the funding value use division multiple times and each one of them is rounding down the result. In addition, there are hidden divisions before multiplications, for example because of `propotionalElapsed()`, which cause further loses in the calculations.

Recommendation

Consider rounding up and possibly refactoring `propotionalElapsed()` to not divide before multiplying.

Resolution

Synthetix Team: Acknowledged.

L-20 | Funding Rate Lags

Category	Severity	Location	Status
Unexpected Behavior	● Low	PerpsMarket	Acknowledged

Description

Due to the way the funding rate is calculated, there may be a case where the market skew is inverted from positive to negative or vice versa, but the funding direction doesn't change.

For example, there are a lot of longs, so the funding rate is positive and the longs pay shorts. Then a user opens a lot of shorts and the skew becomes negative. However, the funding rate still charges longs instead of shorts.

Recommendation

Consider different calculation for the funding rate if longs should pay shorts only when the skew is positive and vice versa.

Resolution

Synthetix Team: This is intentional in the design, to encourage persistent arbitrage of the funding rate over time frames, rather instantaneous changes in the skew.

L-21 | Temporary DoS For Committing Orders

Category	Severity	Location	Status
DoS	● Low	AsyncOrder	Acknowledged

Description

Async orders cannot be committed or settled if the strategy they use is disabled. A problem may arise if the order's strategy is disabled after that order has been committed but not settled.

The AsyncOrder will be populated and not executable. It won't be cancellable as well (if the price is not exceeded). This will cause a temporary DoS - the user will have to wait until the order expires to commit a new one.

Recommendation

Allow users to cancel their order if the used strategy is disabled.

Resolution

Synthetix Team: In the extreme situation where the strategy is disabled then all orders should be suspended from action. This is unlikely to happen unless there is a looming situation that calls for it.

L-22 | Reward Distribution Is Not Future-Proof

Category	Severity	Location	Status
Logical Error	● Low	PerpsMarketFactory.sol: 127	Acknowledged

Description

- The currently implemented reward distribution flow for liquidations only allows to distribute to one pool per collateral token.
- Currently pool creation is done by governance only, but in the future, it will be permissionless.

New pools in the future will therefore not be able to receive liquidated non-sUSD collateral. This will either disincentivize pool creators to back any other market that they don't own (which is against SNX's vision), or a pool owner that is not aware of this situation will back those markets and LP's will lose funds.

Recommendation

Change how rewards are distributed and allow multiple pool IDs. While distributing skim all pool's collaterals and distribute accordingly.

Resolution

Synthetix Team: Acknowledged.

L-23 | Immediately Cancelable Orders Are Permitted

Category	Severity	Location	Status
Unexpected Behavior	● Low	AsyncOrderModule.sol: 31-88	Acknowledged

Description

Users can create orders that can be instantly canceled after creation under the current conditions, by providing a slippage threshold that is already exceeded at the time of order creation. This order will therefore be immediately canceled and the user will lose the keeper gas costs and reward fees.

Recommendation

Revert if such a slippage check is given.

Resolution

Synthetix Team: Acknowledged.

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>