OpenZeppelin | security

Synthetix Bfp-market Audit

SYNTHETIX

March 18, 2024

Table of Contents

Summary

Type	DeFi
Timeline	From 2023-09-11 To 2023-10-16
Languages	Solidity

Total Issues	31 (29 resolved, 1 partially resolved)
Critical Severity Issues	2 (2 resolved)
High Severity Issues	8 (8 resolved)
Medium Severity Issues	8 (7 resolved)
Low Severity Issues	5 (5 resolved)
Notes & Additional Information	7 (6 resolved, 1 partially resolved)
Client Reported Issues	1 (1 resolved)

Scope

We audited the Synthetixio/bfp-market repository at commits 6353c55 and 6705ae5.

Note that as the audit was underway at commit 6353c55, the codebase was updated partway, causing the audit to continue from commit 6705ae5 for the remainder of the audit period. While part of the codebase changed from one commit to the other, the filenames and directory structure stayed the same.

In scope were the following contracts:

```
contracts/
├── Proxy.sol
├── external/
│   ├── IMarket.sol
│   ├── ISpotMarketSystem.sol
│   ├── ISynthetixSystem.sol
│   └── pyth
│   ├── IPyth.sol
│   ├── IPythEvents.sol
│   └── PythStructs.sol
├── interfaces/
│   ├── IBasePerpMarket.sol
│   ├── ILiquidationModule.sol
│   ├── IMarginModule.sol
│   ├── IMarketConfigurationModule.sol
│   ├── IOrderModule.sol
│   ├── IPerpAccountModule.sol
│   └── IPerpMarketFactoryModule.sol
├── modules/
│   ├── CoreModule.sol
│   ├── FeatureFlagModule.sol
│   ├── LiquidationModule.sol
│   ├── MarginModule.sol
│   ├── MarketConfigurationModule.sol
│   ├── OrderModule.sol
│   ├── PerpAccountModule.sol
│   └── PerpMarketFactoryModule.sol
├── storage/
│   ├── Margin.sol
│   ├── Order.sol
│   ├── PerpMarket.sol
│   ├── PerpMarketConfiguration.sol
│   └── Position.sol
└── utils/
    ├── ErrorUtil.sol
    └── MathUtil.sol
```

Commit changes

Initially, the audit was conducted at commit 6353c55. However, in the fourth week of the audit, significant changes were made to the codebase. These changes were reflected in commit 6705ae5. As such, the audit continued at this new commit for the remainder of the fourth and fifth weeks.

As a result of this commit change, some issues found in the original commit (6353c55) were fixed. These issues are reported here for posterity, with accompanying "update" statements describing how they were dealt with by the Synthetix team.

System Overview

At its core, the bfp-market protocol is a perpetual futures market. It allows for opening low-leverage short and long positions on Synthetix "synth" assets. Currently, the market is designed for opening leveraged positions on Synthetix Wrapped Staked Ether (swstETH) against Synthetix USD (snxUSD), using either token as collateral. However, the system is generally designed to be expandable to any asset in the future provided that it is a synth asset and has a corresponding Chainlink and Pyth price feed.

Delayed Order Settlement

The system uses a delayed settlement scheme for orders, whereby orders are first "committed" by the user and then later "settled" by anyone on behalf of that user. Settlement cannot change the order parameters such as size or limit price, and can only occur during the interval between a system-wide `minOrderAge` and `maxOrderAge` time after commitment. As a design feature, each user may only have one active order at a time, which is created upon order commitment and is deleted when the order expires or is settled. Order settlement uses this scheme, along with Pyth oracles (which have "publish times") in order to counter oracle update frontrunning (described in further detail below).

Liquidation Windows

The system is designed to have an MEV-resistant liquidation scheme. The general design is to liquidate positions in small chunks. This gives external markets enough time to absorb and arbitrage the liquidation transactions and avoid creating profitable "sandwich" opportunities for MEV searchers. The design utilizes a liquidation window, whereby within a set time period, a certain amount of liquidation can take place before being halted until the next window.

In order to counter the possibility of underwater users avoiding liquidations by filling the window before their positions could be liquidated, a "flagging" functionality is used. Flagging has no rate limiting and is intended to mark a position for liquidation. Once a position is flagged, it cannot be modified and is available to be liquidated.

There is also a "relief valve" for liquidations, which allows the window functionality to be bypassed by whitelisted liquidators. It allows liquidations to proceed beyond the maximum amount per window under certain conditions, namely when the market is not skewed beyond a certain threshold, the position is not too large, and/or the last liquidation was not in the current block.

Skew

The system defines the "skew" as the aggregate size of all long positions minus the aggregate size of all short positions. The skew is limited by a configurable "max market size" parameter and thus must be between (`-maxMarketSize, maxMarketSize`).

The system incorporates a "funding rate" like most perpetual markets. The funding rate is tabulated on all positions and paid by one side while being earned by the other. Importantly, the funding rate is paid and earned at equal rates by each side, regardless of the market's skew. As such, all shorts may be earning X% funding, while all longs are paying that same X%.

A funding rate velocity is determined based on the skew of the market and represents how fast the funding rate changes over time. It is linearly proportional to the skew with configurable bounds for a maximum and minimum funding rate velocity.

The system also incorporates a premium or discount based on the skew. This is to encourage traders to open positions which counteract the skew and balance the market. The premium or discount is added to the oracle-reported price and is used as the entry/exit price at the time of order settlement.

Price Feeds

The Pyth network is used for prices when filling orders and utilizes a non-standard pull updates pattern. This is done in order to avoid the problem of price-update frontrunning.

Pyth prices are constantly being aggregated off-chain, and are condensed and posted on-chain as needed. Pyth and Synthetix have utilized this structure in order to prevent oracle update frontrunning in Synthetix's Perps v2 protocol.

The system also utilizes Chainlink price feeds for estimating market debt values, determining liquidateability, converting fixed fees to volatile token amounts, and spot-checking Pyth prices when needed.

Integration Assumptions

Bfp-market relies on the Synthetix v3 system and specifically its spot market. For this engagement, it is assumed that the Spot market prices will be generally accurate. Since the Synthetix v3 system holds the assets deposited into the protocol, it is also assumed that the system will be able to accept and produce assets when deposited/withdrawn by the bfp-market protocol.

Synths are assumed to be well-behaved (compliant) ERC-20 tokens and to reasonably hold their peg to the value of the assets they represent. Since profits made by traders generate debt for SNX stakers within the overall Synthetix system, it is assumed that the Synthetix system will be able to mint tokens in order to pay the traders during regular operation.

The Pyth and Chainlink oracles are expected to be accessible and produce accurate prices. In times of high volatility, it is assumed that prices may somewhat diverge from the reported prices. In the case of failure of the oracles, the mode of failure is assumed to match the documented patterns from Pyth and Chainlink, rather than some unknown failure mode. For example, it is assumed that a price of `0` is impossible and corresponds to an error code, but any other value is assumed to be a legitimate price.

Although it was expressed that the perps markets may enable cross-margining, there appeared to be no evidence of cross-margining in the audited code. Hence, it is assumed that the codebase does not allow cross-margining at the audited commit and that significant changes will be needed to enable this functionality.

The Pyth, Chainlink, and Synthetix v3 codebases have not been audited in this engagement. They are generally assumed to work as documented.

Critical Severity

C-01 Partial Liquidation Incorrectly Updates Total Market Size and Skew

When `liquidatePosition` is called on a position that exceeds the liquidation cap for a single transaction, only the maximum capacity will be liquidated. However, the entire size of the position, rather than just the liquidated portion, is subtracted from the total `skew` and `size` of the market. This leads to an incorrect accounting of the market size and skew, as the entire position size will be subtracted multiple times - once for each partial liquidation.

Consider subtracting only the liquidated amount from the skew.

Update: *Resolved at commit 6705ae5. Please note that the Synthetix team also found this vulnerability independently during the engagement.*

C-02 Incorrect Tracking of Liquidation Window Utilization

MEV and sandwiching attacks on the bfp-market were a concern identified in SIP-2005 and SIP-337. The proposed solution, defined in SIP-337, was to pre-define a liquidation window and only allow a certain value to be liquidated during this window.

The protocol's implementation of this window makes use of a `lastLiquidationUtilization` variable, which is supposed to track the amount that has already been liquidated during the current window. However, its value increases with every liquidation and is never reset.

Once enough liquidations are processed, such that `lastLiquidationUtilization` exceeds `maxLiquidateableCapacity`, the protocol can end up in a state where only the first liquidation of each liquidation window would succeed, regardless of size. This is a result of using the max capacity in the first liquidation per liquidation window, and subsequently reverting on further liquidations when trying to do a `uint` subtraction that results in a number less than `0`. This is because `lastLiquidationUtilization` would then exceed `maxLiquidateableCapacity` due to not having been reset from the previous window.

This opens up the possibility for a griefing attack, whereby any further liquidations beyond the first one in a given liquidation window are blocked. This effectively disables a core functionality of the protocol. One consequence of such a protocol state is that the affected accounts could be kept in a semi-permanently flagged state, preventing any further action by the user such as withdrawing collateral. To prevent this, consider resetting `lastLiquidationUtilization` at the start of a new liquidation window.

Update: *Resolved at commit 6705ae5. Please note that the Synthetix team also found this vulnerability independently during the engagement.*

High Severity

H-01 Liquidation Window Incorrectly Implemented

MEV and sandwiching attacks on the bfp-market were a concern identified in SIP-2005 and SIP-337. The proposed solution was to define a liquidation window and only allow a certain value to be liquidated during the window. This would limit the price impact of liquidations and allow arbitrageurs to rebalance prices.

However, in the most recent audited commit, the liquidation window does not rate-limit the sale of collateral. This is because the sale of collateral has been moved into the `flagPosition` function. This is in contrast to the perpetual position which is sold off in multiple transactions with a minimum cooldown period between each sell-off.

This means that large amounts of collateral can be sold off at once, opening the door for MEV via sandwiching attacks.

Note, that while some parts of the liquidation process still follow the liquidation window logic, those are limited largely to the internal accounting of those liquidations, rather than the sale of collateral. Specifically, the following are done step-wise:

- Market size update
- Skew update
- Debt correction update
- Internal position tracking per account

Consider modifying the code in `LiquidationModule.sol` such that the liquidation window logic is applied to the `flagPosition` function. Additionally consider partially selling collateral on the spot market during partial liquidations and large order settlements, rather than selling all the collateral in one transaction. Alternatively, consider making the sale of assets part of the same execution flow as the liquidation window logic. By doing this, sandwiching attacks can be mitigated in severity and the design of the protocol will better adhere to the specification.

Update: *Resolved in pull request #45. Collateral is now distributed directly to LPs upon flagging of positions instead of being sold into the Synthetix v3 Spot Market.*

H-02 No `minAmountReceived` Specified for `sellExactIn`

In `Margin.sol`, there are a few `sellExactIn` calls that are made to the Synthetix core system. These calls perform a swap and, based on the swap's size and market conditions, are subject to some skewing in price.

These calls have the `0` value specified for the `minAmountReceived` parameter. It must be noted that the minimum amount being `0` is dangerous as it does not limit the skew at all. Price manipulation could be used to skew the price substantially and leave the bfp-market protocol insolvent. This danger is exacerbated by the fact that these sales occur when orders are settled or when positions are liquidated, which can both be initiated by users other than the account owner. Thus, attackers can exploit the protocol by triggering swaps at prices significantly different from the prevailing market rate, and earn arbitrage profits while creating bad debt in the bfp-market.

Consider implementing some `minAmountReceived` parameter in order to prevent collateral from being sold at an unfair rate and protect the health of the protocol. As an example, this parameter can be in the form of a configurable maximum percentage difference compared to the oracle price.

Update: *Resolved in pull request #45.*

H-03 No Incentive to Flag Small Positions

In order to incentivize timely liquidations of underwater positions, entities which flag liquidatable positions are rewarded with a liquidation reward.

The calculation performed for `liqReward` does not have a lower bound. The reward is set as a percentage of the size of the position being liquidated. As such, this can result in situations where liquidators have no incentive to flag a position as the gas costs exceed the liquidation reward. However, this is inconsistent with the calculation of `keeperFee` which does implement a lower bound ensuring that it exceeds the gas price. In SIP-80, the Synthetix team refers to this potential issue as "small account bloat".

Consider implementing a lower bound based on `liqReward` or position size to ensure there is an incentive to liquidate small positions.

***Update:** Resolved in pull request #36 at commit e0c9db8.*

H-04 Incorrect Check of Initial Margin in Trade Validation

When opening or modifying a position, it is important to check whether a sufficient margin for an account exists. Inside the `validateTrade` function, multiple checks are done to ensure that margin requirements are met for an account, both before and after the trade.

However, one of the checks performed ensures that the existing position before the trade (if one exists) meets the initial margin requirement. This check is incorrect as it disallows actions which would de-risk a position but still leave the margin below the maintenance margin.

The system should always allow de-risking when a position is not liquidatable. Later in the same function, there is a special logic block to allow for de-risking when initial margin requirements are not met after a trade is completed. This contradicts the intention of the initial margin check for the "before" position.

Consider implementing the initial margin check only when a trade increases the leverage in a position, such as decreasing collateral or increasing the absolute size of a position.

***Update:** Resolved in pull request #38 at commit 01f3a96.*

H-05 Faulty Enforcement of Market-Wide Collateral Limits

In order to be able to open a new position, or de-risk an existing one, users need to deposit collateral into the system. If the value of that collateral decreases to the point of not fulfilling maintenance margin requirements, the position it collateralizes is marked for liquidation.

Assets naturally vary in their volatility, on-chain liquidity, and in their risk of price manipulation. As such, in the interests of systemic risk management, the protocol attempts to enforce market-wide limits on exposure to certain collateral assets.

To implement such a limit, the system has a `maxAllowable` variable in the `Margin` storage library. It stores the global maximum allowed amount of collateral per collateral type. However, when this limit is checked upon depositing collateral, only the individual account's total deposited collateral is checked to be lower than the `maxAllowable` variable. Instead, it is the market-wide deposited collateral that should be checked to be lower than the `maxAllowable` amount.

Consider tracking the total deposited amount of collateral per collateral type and checking against this variable whenever collateral is deposited.

Update: *Resolved at commit 6705ae5.*

H-06 Liquidation Reward Not Proportional to Liquidated Size

In order to incentivize timely liquidations, entities are rewarded for flagging liquidatable positions. This liquidation reward should be proportional to the size of the position being liquidated.

However, the computed liquidation reward is proportional to the entire size of the position, rather than the actual liquidated size. These values can diverge in case of partial liquidations. If a small position is liquidated in many chunks, the overall liquidation reward may be substantially higher than anticipated. The current logic incentivizes extending liquidations by liquidating as little as possible, many times, to maximize extractable value.

Consider updating the liquidation reward computation to be proportional to `liqSize`.

Update: *Resolved in pull request #29 at commit bbcbc67.*

H-07 Price Manipulation Can Trigger Liquidations

In order to determine whether a position is liquidatable, the current margin of an account is computed and checked against minimum margin requirements. An account's margin consists of the value of its collateral and its position's PnL, including funding and fees. In order to prevent positions from artificially being pushed into the liquidation range, one must use manipulation-resistant oracle price feeds.

The bfp-market uses Synthetix's Spot Market prices to check the value of a user's collateral. This is problematic, as the price quoted by the Spot Market is susceptible to sandwiching due to applying a `skewFee`. By selling the synth in question on the Spot Market, users can be pushed into the liquidation range by reducing the value of an account's collateral, thereby creating MEV in the form of liquidation rewards.

Consider introducing additional checks which ensure that the quoted price by the Spot Market is not too far from the oracle price of the synth.

Update: *Resolved in pull request #45 at commit 6545a7e. The collateral valuation logic now relies on an oracle instead of a spot price.*

H-08 No Way to Cancel Orders

For each account, there can only be one order pending settlement at a time. Until `settleOrder` is called for a given account, or until `validateOrderAvailability` is reached and an order is older than the `maxOrderAge`, an order is still considered "pending" and will prevent the creation of new orders and the depositing or withdrawing of collateral.

There are many situations where a normal user may want to cancel an order, such as accidentally or intentionally placing an order far from the current price, or when needing to quickly deposit more funds to shore up an unhealthy account.

It is imperative that users are able to make their accounts healthier at any time before liquidation. Consider creating a "cancel order" functionality to allow users to de-risk their positions more effectively when needed. Alternatively, consider not preventing the depositing of more collateral when an order is pending. Furthermore, consider allowing orders to be cancelled if their price differs substantially from the market rate.

Update: *Resolved in pull request #41 at commit e8455d9.*

Medium Severity

M-01 Allowances Never Reset in `setCollateralConfiguration`

In `setCollateralConfiguration`, the old collateral configuration is cleared and a new set of collateral configurations is assigned.

When setting up a new configuration, the `globalMarketConfig.synthetix`, `globalMarketConfig.spotMarket`, and `address(this)` addresses are approved for the maximum amount of tokens. However, this approval is never revoked when this configuration is reset, though it appears that this is intended in future versions.

Consider reconfiguring the revocation portion of collateral configuration so that it removes the infinite approval from the Synthetix system. Although the system is presumed to be trusted, we cannot be sure there will never be an exploit whereby tokens are stolen from the bfp-market. For maximum safety, unneeded approvals should always be revoked.

Update: *Resolved in pull request #37 at commit da7b77.*

M-02 Avoid Using `decreaseAllowance`

In `setCollateralConfiguration`, a call is made to `decreaseAllowance` for some IERC20 token. However, `decreaseAllowance` is not a standard ER-C20 function, and should not be presumed to exist for all ERC-20 tokens.

The referenced code has more than one issue, and the developers should note the related issues **M-06: Confusion of `owner` with `globalMarketConfig.synthetix`** and **L-01: Do Not Self-Approve.** Regardless, the apparent intent of the referenced lines is to reduce the allowance to `0`. In this case, `decreaseAllowance` has no practical advantage over simply setting the allowance to `0`. Consider changing this call to `approve` instead of `decreaseAllowance`, and setting the value to `0`.

Update: *Resolved in pull request #37 at commit da7b778.*

M-03 Disallowed Tokens Will Be Unwithdrawable

If a collateral is allowed and then disallowed by calling `setCollateralConfiguration`, users will not be able to withdraw this collateral.

This is because both the `supported` and `supportedSynthMarketIds` mappings will be deleted for the old collateral configuration. This will set `maxAllowable` for the collateral to `0`.

In `modifyCollateral`, this will prevent withdrawals of the previously allowed collateral. Additionally, since `withdrawAllCollateral` only iterates over `supportedSynthMarketIds`, this collateral will not be withdrawable through `withdrawAllCollateral` either. This makes withdrawing a previously allowed collateral impossible, once it has been disallowed.

Consider changing this logic within the `modifyCollateral` and `withdrawAllCollateral` functions to allow withdrawals of previously allowed collaterals. This may be accomplished by checking whether a collateral modification is a withdrawal or deposit and creating a mapping of "previously allowed" collaterals. In addition, consider avoiding open token withdrawals (allowing any token to be withdrawn) as attackers may use it to call malicious contracts which appear like tokens to the protocol.

Update: *Resolved in pull request #42 and pull request #46.*

M-04 Incorrect Bounding of Liquidation Keeper Fee

The `getLiquidationKeeperFee` function returns the fee that is used to compensate accounts for performing liquidations. This fee is bounded by the configurable variables `globalConfig.minKeeperFeeUsd` and `globalConfig.maxKeeperFeeUsd`.

The bounding functions `MathUtil.max` and `MathUtil.min` are in the wrong order, and thereby always return the maximum fee.

Consider switching the order of the `MathUtil.max` and `MathUtil.min` functions, so that the correct fee is returned.

Update*: Resolved in pull request #36 at commit 1439c9c.*

M-05 Confusion of `owner` with `globalMarketConfig.synthetix`

The `setCollateralConfiguration` function is only callable by the `owner`, which appears to be considered the same as the `globalMarketConfig.synthetix` value, which represents the Synthetix main system.

However, in case the `owner` is updated (ownership is transferred), the use of `msg.sender` may result in revoking an incorrect allowance and leaving the Synthetix main system with an open allowance. Consider changing both instances to the same target, either `msg.sender` or `globalMarketConfig.synthetix`.

Update: *Resolved at commit 6705ae5.*

M-06 Incorrect Casting Within `getRemainingLiquidatableSizeCapacity`

There is an incorrect use of casting when computing `remainingCapacity` in `PerpMarket.sol`. Specifically, since `maxLiquidatableCapacity` and `lastLiquidationUtilization` are both `uint128` values, the result will be a `uint128` and will revert when the result is less than `0`.

Based on this, the `.toInt()` cast and the use of `max(...,0)` are unnecessary. It appears that the `toInt()` cast should instead be used twice, both on `maxLiquidatableCapacity` and `self.lastLiquidationUtilization` *before* the subtraction. This way, the subtraction results in an `int` value which can be less than `0`, and the `max(...,0)` call will serve a purpose.

Consider also that this error may result in unintended reverts in case `maxLiquidatableCapacity` increases due to some parameter change, such that it is higher than `self.lastLiquidationUtilization`.

Consider casting both `maxLiquidatableCapacity` and `self.lastLiquidationUtilization` to `int` before the subtraction on line 287.

Update: *Resolved at commit 6705ae5.*

M-07 Incorrect Target for Allowance Decrease

In `setCollateralConfiguration`, when removing an old configuration, immediately after checking the allowance of the `msg.sender`, the `allowance` value is used to decrease the allowance of `address(self)`.

It appears the intended target of the allowance decrease is actually `msg.sender`, and calling `decreaseAllowance` for the wrong target may revert if the allowance at the target is not high enough. Additionally, it leaves the intended allowance intact, which is dangerous even if the `msg.sender` is a trusted address.

Consider changing the target of the allowance decrease call to `msg.sender`.

Update: *Resolved at commit 6705ae5.*

M-08 Inconsistent Use of Price Feeds May Allow for Immediate Liquidation

When committing an order, and as well as when settling an order, checks for immediate liquidatability are performed. In order to ensure that positions are not immediately liquidatable, the protocol has to utilize the same price feed for these checks as when checking whether a position can be flagged for liquidation.

In the case of committing orders, `fillPrice` is used for checking whether an existing position is liquidatable, as well as for checking whether the resulting new position is liquidatable. This is also the case for settling orders, where `runtime.params` contains the fill price, which is used for checking liquidatability.

However, when checking for liquidatability in `flagPosition`, `oraclePrice` is used. This is problematic, as `fillPrice` differs from `oraclePrice` by a premium/discount. In certain cases, where the fill price diverges significantly from the oracle price, either naturally, or by malicious skew manipulation, a position might be up for liquidation immediately after being opened.

Consider consistently using the oracle price whenever checking for the liquidatability of a position in order to avoid unforeseen liquidations.

Update: *Acknowledged, not resolved. The Synthetix Team stated:*

> This is intended behavior and we believe it was implemented correctly. `fillPrice` describes a position's equity/average entry while `oraclePrice` is used to trigger

> liquidations in a tamper-proof way (adjusted with expected exit premium/discount cost). Additionally, the buffer between initial margin requirements (where positions can be opened) and maintenance margin requirements (where positions are liquidated) always prevents positions from being instantly liquidatable.

Low Severity

L-01 Do Not Self-approve

In the `MarginModule.sol` contract, there are many instances of misuse regarding self-approvals and `transferFrom` calls where the sender is `address(this)`. Generally, a self-approval should not be used as it needlessly complicates the codebase and there may be tokens which do not allow it. Some relevant instances have been identified:

- On line 251, `decreaseAllowance(address(this))` is called.
- On line 273, `approve(address(this))` is called.
- On line 101, `transferFrom` is called with `address(this)` as the sender.

Consider removing the self-approval instances, and consider changing the instance of `transferFrom(address(this), ..., ...)` to the ERC-20 standard function `transfer`. Note that some ERC-20 contracts may prevent self-approvals or calls to `transferFrom` where the sender is `address(this)`.

Update: *Resolved in pull request #40 at commit 02affc3. Note that self-approvals for snxUSD are needed to successfully call* `depositMarketUsd`*.*

L-02 Floating Pragma

Pragma directives should be fixed to clearly identify the Solidity compiler version with which the contracts will be compiled.

Throughout the codebase, there are multiple floating pragma directives. For instance, the following files have the `solidity >=0.8.11 <0.9.0` floating pragma directive:

- `Order.sol`
- `Position.sol`
- `PerpMarketConfiguration.sol`
- `IPyth.sol`

- `Margin.sol`
- `PythStructs.sol`
- `IPythEvents.sol`
- `PerpMarket.sol`

Consider using a fixed pragma version.

Update: *Resolved in pull request #32 at commit 44d4e5b.*

L-03 Gas Inefficiency in `recomputeFunding`

In `recomputeFunding`, there is a call to `getNextFundingAccrued` right after a call to `getCurrentFundingRate`. Within `getNextFundingAccrued`, there is another call to `getCurrentFundingRate`.

While this is currently needed to get the `fundingRate` value within `recomputeFunding`, internal functions could be used to encapsulate the repeated logic and remove the need to call `getCurrentFundingRate` twice within the same execution flow.

Update: *Resolved in pull request #34 at commit d97d2b7.*

L-04 Repeated Instances of `validateOrderAvailability`

There are two instances of `validateOrderAvailability`, one in `MarginModule.sol`, and one in `OrderModule.sol`. The one in `OrderModule.sol` is directly copied from the one in `MarginModule.sol`.

They both are currently identical and there appears to be no advantage to having instances within each file.

Having duplicated code bloats the system and makes it difficult to understand. Additionally, there is a chance that the code is updated in one spot and not the other, unintentionally introducing bugs. For maximum safety and efficiency, consider refactoring the codebase so that there is only one instance of this function.

Update*: Resolved in pull request #41 at commit 864acd7.*

L-05 `liquidationMaxPd` Can Be Exceeded

The protocol limits the size of liquidations that can happen within a certain time frame. It does this by limiting the sum of liquidations in a rolling time window. Under certain conditions, however, the maximum liquidatable capacity per liquidation window is allowed to be bypassed.

These conditions are checked in the `validateLiquidation` function of `Position.sol`, and consist of checking that liquidations can only occur when `skew`, divided by the `skewScale` is less than the `liquidationMaxPd`, and the current position size divided by the `skewScale` is also less than `liquidationMaxPd`.

However, these two checks do not ensure that `liquidationMaxPd` will not be bypassed. The liquidation itself could increase the `skew` such that the `skew` divided by `skewScale` exceeds `liquidationMaxPd`, even though the size of the liquidation and current skew are low enough to pass the identified checks.

Consider either setting `maxPdScaled` to half of the desired max skew (scaled), or making sure that the liquidation will not increase the `skew` divided by `skewScale` beyond `maxPdScaled`.

Update: *Resolved in pull request #35 in commit d2e64d0. One of the relevant checks has been removed. Statement from the Synthetix Team:*

> We believe this was implemented correctly. `maxLiquidationPd` is only intended to serve as a sandwich detection readout and thus only needs to be enforced before a liquidation goes through, not after.

Notes & Additional Information

N-01 `getPositionDebtCorrection` Can Be Removed

The `getPositionDebtCorrection` function is not used anywhere in the codebase. Since it is `internal`, it cannot be called from outside the codebase.

Consider removing this function to improve the clarity of the codebase.

Update: *Resolved in pull request #41 at commit e8455d9.*

N-02 Naming Issues

There are a few places in the codebase that would benefit from clearer naming:

- In line 115 of `Position.sol`, consider renaming the function `validateNextPositionIsLiquidatable` to `validateNextPositionIsNotLiquidatable`.
- In line 106 of `PerpMarketFactoryModule.sol`, consider renaming `totalDebt` to `totalReportedDebt` in order to avoid confusion with the concept of `totalDebt` in the core system.
- In lines 79 and 80 of `Order.sol`, consider renaming `takerSize` and `makerSize` to `takerSizeProportion` and `makerSizeProportion`, as these are proportions, not absolute sizes.
- In line 73 of `PerpMarketConfiguration.sol`, consider renaming `maxMarketSize` to `maxMarketSizeOneSide` in order to adhere to the meaning of `marketSize` outlined in SIP-80.

Update*: Partially resolved in pull request #24 and pull request #45. The first two bullet points identified here have been fixed. The Synthetix Team stated:*

> Will not be addressing (3) and (4). (3) is pretty clear, there's a comment right above it which describes the how and changing the variable name here adds no value. I also think (4) is clear, `maxMarketSize` refers to just one side. It's named `maxMarketSize` in PerpsV3, `maxMarketValue` in PerpsV2, and there is documentation to clarify that this is one side.

N-03 Non-Explicit Imports

The use of non-explicit imports in the codebase can decrease the clarity of the code, and may create naming conflicts between locally defined and imported variables. This is particularly relevant when multiple contracts exist within the same Solidity files or when inheritance chains are long.

Throughout the codebase, global imports are being used. For instance:

- Line 4 of `IMarket.sol`
- Lines 4, 5, and 6, Line 7 of `ISynthetixSystem.sol`

- Lines 4 and 5 of `IPyth.sol`

Following the principle that clearer code is better code, consider using named import syntax (*`import {A, B, C} from "X"`*) to explicitly declare which contracts are being imported.

Update: *Resolved in pull request #48 at commit 3e14756.*

N-04 Wrong Parameter Value on Custom Error

The second parameter of the custom error `ErrorUtil.InsufficientCollateral` should return the available amount of collateral a margin account has. However, in line 222 of `MarginModule`, the custom error reverts with the total available collateral across the bfp-market as its second parameter.

Consider using `accountMargin.collaterals[synthMarketId]` as the second parameter of the `ErrorUtil.InsufficientCollateral` custom error.

Update: *Resolved in pull request #48 at commit 3e14756.*

N-05 Gas Optimizations

- In `withdrawAllCollateral`, at line 154 of `MarginModule.sol`, an account's collateral can be directly set to `0` as `available` is set to the same value on line 147.
- The check in line 198 of `MarginModule.sol` can be moved to the top of its function to save gas when reverting and better follow the checks-effects-interactions pattern.
- The check in line 203 of `MarginModule.sol` can be moved to the top of its function to save gas when reverting and better follow the checks-effects-interactions pattern.

Consider making these changes to save gas and follow best practices across the codebase.

Update: *Resolved in pull request #48 at commit 658d6ee.*

N-06 Typographical Errors

- In line 74 of `MarginModule.sol`, "an collateral withdraw" should say "a collateral withdrawal".
- In line 143 of `PerpMarket.sol`, the sentence ends early and should be completed.
- In line 228 of `Margin.sol` "if skewed is expanded" should say "if skew is expanded"

Consider correcting the typographical errors identified above to improve the readability and quality of the codebase.

Update: *Resolved in pull request #48 at commit 3e14756.*

N-07 TODO Comments

The following instances of TODO/Fixme comments were found in the codebase.

These comments should be tracked in the project's issue backlog and resolved before the system is deployed:

- The `TODO` comment on line 269 in `MarginModule.sol`
- The `TODO` comment on line 236 in `OrderModule.sol`
- The `TODO` comment on line 26 in `PerpMarketConfiguration.sol`
- The `TODO` comment on line 120 in `PerpMarket.sol`

During development, having well-described TODO/Fixme comments will make the process of tracking and solving them easier. Without this information, these comments might age and important information for the security of the system might be forgotten by the time it is released to production.

Consider removing all instances of TODO/Fixme comments and instead tracking them in the issues backlog. Alternatively, consider linking each inline TODO/Fixme to the corresponding issues backlog entry.

Update: *Resolved at commit 6705ae5.*

Client-Reported

CR-01 Incorrect Target for Deposits of snxUSD

In the flow for depositing collateral into the system, the call to `depositMarketUsd` within the `transferAndDeposit` function has an incorrect parameter.

The parameter `address(this)` should be `msg.sender`, as the `depositMarketUsd` function will burn tokens from that address. Thus, when attempting to deposit tokens, a burn would be triggered attempting to burn tokens held by the bfp-market contract, instead of those held by the user depositing.

***Update**: Resolved in pull request #27.*

Conclusion

During this five-week engagement, the audit team took an in-depth look at this codebase and found several critical and high issues along with some lower-severity findings. The auditors appreciated the Synthetix team's responsiveness and sense of partnership as we worked together to improve the system.

The codebase could benefit from better documentation and generally lowering the amount of redirection or sub-calls needed to execute the code. Due to the number of changes proposed, and changes made to the mechanism design of the system during fix review, we recommend increasing test coverage, including simulations for "edge case" events. We suggest considering the economic incentive design of the codebase and documenting the incentives for all key actions, such as liquidations. We also suggest updating the documentation to explain design choices and how they may relate to any SIPs.

Since the codebase hooks into the existing Synthetix system and can mint and burn assets, we recommend having a well-tested "shutdown" script for the bfp-market protocol. This script should minimize the risk of the protocol by being ready to go in the case of an incident. In addition, it should limit the protocol's ability to mint or move funds within the greater Synthetix system until any issues have been contained.