Synthetix Didpha Release Smart Contract Audit

1. Introduction

iosiro was commissioned by Synthetix to conduct a smart contract audit of their Didpha Release, which included the following components:

- SIP-80
- SIP-165

The audit of SIP-80 was performed over several iterations starting from 2 to 14 December 2021 with three auditors, continuing from 10 to 25 February 2022 with two auditors, and concluding from 7 to 14 March 2022 with two auditors, consuming a total of 48 resource days.

The audit of SIP-165 was performed from 1 to 11 March 2022 with two auditors, consuming a total of 8 resource days.

This report is organized into the following sections.

- **Section 2 - Executive summary:** A high-level description of the findings of the audit.
- **Section 3 - Audit details:** A description of the scope and methodology of the audit.
- **Section 4 - Design specification:** An outline of the intended functionality of the smart contracts.
- **Section 5 - Detailed findings:** Detailed descriptions of the findings of the audit.

The information in this report should be used to understand the smart contracts' risk exposure better and as a guide to improving the security posture of the smart contracts by remediating issues identified. The results of this audit reflect the in-scope source code reviewed at the time of the audit.

The purpose of this audit was to achieve the following:

- Identify potential security flaws.
- Ensure that the smart contracts function according to the documentation provided.

Assessing the off-chain functionality associated with the contracts, for example, backend web application code, was outside of the scope of this audit.

Due to the unregulated nature and ease of transfer of cryptocurrencies, operations that store or interact with these assets are considered high risk from cyber attacks. As such, the highest level of security should be observed when interacting with these assets. This requires a forward-thinking approach, which takes into account the new and experimental nature of blockchain technologies. Strategies that should be used to encourage secure code development include:

- Security should be integrated into the development lifecycle, and the level of perceived security should not be limited to a single code audit.
- Defensive programming should be employed to account for unforeseen circumstances.
- Current best practices should be followed where possible.

2. Executive summary

This report presents the findings of a smart contract audit performed by iosiro of Synthetix's Didpha release.

SIP-80 introduced a futures market to the Synthetix ecosystem, which allowed users to trade various synthetic assets with leverage. Traders could take a long or short position on a base asset by providing margin in the form of sUSD. The debt of the leveraged positions was subsidized by the Synthetix debt pool, while liquidated positions contributed to the pool. A funding rate was charged from positions that created a market skew and paid to positions that opposed it as a mechanism to balance the market and limit the risk exposure of the Synthetix debt pool. A more comprehensive description of the system is given in the SIP in the design specification below.

One low-risk issue and four informational issues were identified during the audit; the low-risk issue was closed during the assessment. The code of this system underwent significant changes over the course of the audit, which both simplified the code and improved its quality. The code was of a high standard by the end of the audit.

SIP-165 added functionality to allow the Synthetix ecosystem to use an oracle to query the debt ratio and total amount of issued synths on its chain. The addition of these debt oracles moves the system toward being able to query the debt across all Synthetix deployments across various chains. This SIP will be implemented in a phased manner, with the current phase implementing the `SingleNetworkAggregators` to preserve current functionality by querying debt information through an internal oracle. The next phase will use PDAO functionality to make the system query Chainlink oracles for the debt information to enable synth and debt share fungibility across chains. As part of the second phase, the functions that still depend on internal functions to query debt and synth balances will be updated to use the current debt. Initially, the Chainlink oracles will be configured with a 1% trigger and a 1-hour heartbeat for L1, and a 0.5% trigger and a 20-minute heartbeat for L2.

One low-risk issue and two informational issues were identified during the audit, with the low-risk issue being closed during the assessment. Overall the code quality was found to be of a high standard.

3. Audit details

3.1 Scope

The source code considered in-scope for the assessment is described below. Code from all other files was considered to be out-of-scope. Out-of-scope code that interacts with in-scope code was assumed to function as intended and not introduce any functional or security vulnerabilities for the purposes of this audit.

3.1.1 Synthetix SIP-80 smart contracts

Project Name: Synthetix
Initial commit: d7b8c12
Review commits: f88c9f3, 63fa3f1, f665639, 60fe55d, 87c7d29, 4ca54bb, 17a48ae
Final commit: 4d520d7
Files: FuturesMarketBase.sol, FuturesMarket.sol, FuturesMarketManager.sol, FuturesMarketData.sol, FuturesMarketSettings.sol, MixinFuturesMarketSettings.sol, MixinFuturesNextPriceOrders.sol, MixinFuturesViews.sol, EmptyFuturesMarketManager.sol

3.1.2 Synthetix SIP-165 smart contracts

Project Name: Synthetix
Initial commit: 657443e
Final commit: e45b6d9
Files: BaseSingleNetworkAggregator.sol, FeePool.sol, Issuer.sol, BaseSynthetixBridge.sol, OneNetAggregatorDebtRatio.sol, OneNetAggregatorIssuedSynths.sol, SynthetixBridgeToBase.sol, SynthetixBridgeToOptimism.sol

3.2 Methodology

A variety of techniques were used in order to perform the audit. These techniques are briefly described below.

3.2.1 Code review

The source code was manually inspected to identify potential security flaws. Code review is a useful approach for detecting security flaws, discrepancies between the specification and implementation, design improvements, and high-risk areas of the system.

3.2.2 Dynamic analysis

The contracts were compiled, deployed, and tested in a test environment, both manually and through the test suite provided. Manual analysis was used to confirm that the code was functional and to identify security issues that could be exploited.

3.2.3 Automated analysis

Tools were used to automatically detect the presence of several types of security vulnerabilities, including reentrancy, timestamp dependency bugs, and transaction-ordering dependency bugs. Static analysis results were reviewed manually and any false positives were removed. Any true positive results are included in this report.

Static analysis tools commonly used include Slither, Securify, and MythX. Tools such as the Remix IDE, compilation output, and linters could also be used to identify potential areas of concern.

SIP-80 Test Coverage

The coverage report of the provided tests as on the final day of the audit is given below.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
EmptyFuturesMarketManager.sol	100	100	0	0	... 31,32,33,37
FuturesMarket.sol	100	100	100	100	
FuturesMarketBase.sol	100	93.06	100	99.2	354,535
FuturesMarketData.sol	100	100	100	100	
FuturesMarketManager.sol	100	91.67	100	100	
FuturesMarketSettings.sol	100	100	100	100	
MixinFuturesMarketSettings.sol	100	100	100	100	
MixinFuturesNextPriceOrders.sol	100	100	100	100	
MixinFuturesViews.sol	100	100	100	100	

The lines of code not covered in `FuturesMarketBase.sol` would have been addressed with the completion of the no-opped test cases. Beyond this, the coverage for the other files was acceptable.

SIP-165 Test Coverage

The coverage report of the provided tests as on the final day of the audit is given below.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
BaseOneNetAggregator.sol	100	100	85.71	87.5	54
BaseSynthetixBridge.sol	100	100	100	100	
FeePool.sol	100	83.33	100	98.94	452,484
Issuer.sol	100	93.62	98.7	99.23	365,749
OneNetAggregatorDebtRatio.sol	100	100	100	100	
OneNetAggregatorIssuedSynths.sol	100	50	100	80	25
SynthetixBridgeToBase.sol	100	100	100	100	
SynthetixBridgeToOptimism.sol	100	100	100	100	

The coverage for this SIP missed a few lines of code; however, some of these lines were not directly related to the changes of the SIP. Nonetheless, test cases should be introduced to ensure no lines of code are missed.

3.3 Risk ratings

Each issue identified during the audit has been assigned a risk rating. The rating is determined based on the criteria outlined below.

- **High risk** - The issue could result in a loss of funds for the contract owner or system users.
- **Medium risk** - The issue resulted in the code specification being implemented incorrectly.
- **Low risk** - A best practice or design issue that could affect the security of the contract.

- **Informational** - A lapse in best practice or a suboptimal design pattern that has a minimal risk of affecting the security of the contract.
- **Closed** - The issue was identified during the audit and has since been satisfactorily addressed, removing the risk it posed.

4. Design specification

The following section outlines the intended functionality of the system at a high level.

4.1 SIP-80

The specification of SIP-80 was based on commit hash 7f0f063.

4.2 SIP-165

The specification of SIP-165 was based on commit hash d740e2a.

5. Detailed findings

The following section includes in-depth descriptions of the findings of the audit.

5.1 High risk

No high-risk issues identified during the audit were present at the conclusion of the audit.

5.2 Medium risk

No medium-risk issues identified during the audit were present at the conclusion of the audit.

5.3 Low risk

No low-risk issues identified during the audit were present at the conclusion of the audit.

5.4 Informational

5.4.1 Potential debt oracle front-running

SIP-165

Note: this issue does not pertain to the current implementation, but rather to the general move towards using oracles for reporting debt values.

As with most oracles, there will exist a delay between every debt oracle update. Any actions in the system affecting debt during the time between updates will create an error between the reported value and the real value. For the debt ratio and total issued synths, this will include burning and issuing synths, as well as exchange rates in the system. In both of these cases, changes in these parameters on any network will contribute to that error. As a result, it may be possible for an opportunistic user to front-run different parts of the debt system for profit.

For example, consider a user with 100 debt shares and each share is worth 10 sUSD, resulting in the user owing a total of 1,000 sUSD of debt. After the oracle posts these values on chain, there's a sudden dramatic increase in prices, which will result in each debt share being worth 20 sUSD. As the user can see this increase is about to happen, they burn enough sUSD to close their debt position and then wait for the oracle to post the new values. At this point, the user re-enters the system, resulting in the user only being responsible for 10 sUSD of debt, instead of the full 20 sUSD.

The main mitigating factors include:

- The settlement period and minimum staking time, which require the user to commit to their position for at least some period of time and take on some level of risk.
- Any profits will need to exceed incurred fees (exchange fees, mint fees, burn fees, etc.).

Two further mitigations that could be used include:

1. The oracle could report debt values per network. When calculating debt values on a specific network, the system could sum other networks and then use the live debt cache value of its own network, which would reduce the margin of error to some extent.
2. Users could commit to the next debt values (e.g. similar to the `nextPrice` pattern of SIP-80). Unfortunately, this would lead to a poor UX because it would require two transactions, but it would further narrow the ability to front-run debt changes.

Update

The Synthetix team indicated that they were content with existing mitigations.

5.4.2 Use a max skew rate

SIP-80

Description

When opening or modifying a position, the operation was validated to ensure that the total USD value of the market did not exceed the maximum that had been set for that market. This was a security measure to ensure that the market did not grow excessively large. However, this introduced an edge-case where a market's full capacity could be allocated to a single side of the market. As no additional positions could be opened at that point, it would not be possible to rebalance the market in the opposite direction, which would result in the full risk being borne by the debt pool. In this scenario, one mitigating factor was that the funding rate would aggressively diminish the heavily skewed positions, meaning that it would likely be unprofitable if held for more than a few days.

As this was still theoretically possible, one further mitigation to consider would be to enforce a maximum skew rate, which would prohibit opening new positions that exceeded the skew rate above a certain percentage, ensuring there is always some capacity for new positions to rebalance the market.

Update

The Synthetix team indicated that they were content with the reliance on the funding rate.

5.4.3 Issuance rate-limiter

SIP-80

The most significant risk presented by the introduction of futures markets into the Synthetix ecosystem was a new way to issue synths backed by the debt pool. As a result, if there was a problem relating to issuance that allowed for a large amount of synths to be issued (e.g. a broken oracle or implementation issue), this would be absorbed by the debt pool and be technically challenging to recover from.

While the exchange rate circuit breaker (SIP-65) was in place to protect against a particular class of oracle issues, it is recommended that an issuance rate-limiter is used as additional mitigation in the event of such an issue. The rate-limiter could throttle the amount of sUSD each market is allowed to issue each day, meaning that if there was an issue with a market, the effect of it would be capped to the issuance rate of that market.

Update

The Synthetix team indicated that they were interested in implementing a rate-limiter, but it would have to be integrated with other changes in the Issuer contract at a later stage.

5.4.4 Unvalidated casting

SIP-80

Explicit casting between `int` and `uint` was being used throughout the code without validating the operation through `SafeCast` or an equivalent function. While no related issues were identified during testing, care should be taken when adding new assets to ensure that the oracle is properly validated to fall within expected bounds. Furthermore, any changes to the code in the future should carefully consider these operations.

5.4.5 Use of `now`

SIP-165

Description

In Solidity version 0.7.0, the `now` keyword was deprecated. Developers are encouraged to use `block.timestamp` instead, to ensure forward compatibility. The `now` keyword was used on the following lines in the in-scope files:

- FeePool.sol#L252
- Issuer.sol#L262
- SingleNetworkAggregatorDebtRatio.sol#L28
- SingleNetworkAggregatorIssuedSynths.sol#L23

5.4.6 Missing test coverage

SIP-80

Description

While there was full line coverage of the code in scope, certain portions of the code were found to be covered implicitly. Explicit test cases help ensure test quality when testing specific areas of the code, which helps with the effectiveness of the tests and the maintainability of the codebase. The following test cases were being skipped:

Transferring margin:

- Transferring margin updates margin, last price, funding index, but not size
- Existing position:
- Increase margin
- Decrease margin
- Cannot decrease margin past liquidation point
- Cannot decrease margin past liquidation point
- Cannot decrease margin past max leverage
- Transferring margin realises profit and funding

Profit & Loss, margin, leverage:

- PnL:
- Zero profit on a zero-size position
- Remaining margin:
- profit and no funding:
  - positive profit

  - negative profit
- funding and no profit:
  - positive funding
  - negative funding
- funding and profit:
  - positive sum
  - negative sum
- Remaining margin is clamped to zero if losses exceed initial margin

Funding:

- Funding sequence:
- Funding sequence is recomputed by order submission
- Funding sequence is recomputed by order confirmation
- Funding sequence is recomputed by order cancellation
- Funding sequence is recomputed by position closure
- Funding sequence is recomputed by liquidation
- Funding sequence is recomputed by margin transfers
- A zero-size position accrues no funding

Market Debt:

- Market debt is the sum of remaining margins
- Liquidations accurately update market debt and overall system debt
- market debt incorporates funding flow:
- funding profits increase debt
- funding losses decrease debt
- market debt incorporates profits:
- profits increase debt
- losses decrease debt

- After many trades and liquidations, the market debt is still the sum of remaining margins
- Enough pending liquidation value can cause market debt to fall to zero, corrected by liquidating

Liquidations:

- Liquidation price:
- Liquidation price is accurate with funding with intervening funding sequence updates

It is recommended that the test cases mentioned above are implemented.

Update

The Synthetix team indicated that they are aware of the issue and will extend the unit tests in parallel with integration and fork tests.

5.5 Closed

5.5.1 Debt oracle price circuit breaker (low risk)

SIP-165

Description

Due to the tremendous risk presented by a faulty or compromised debt oracle result, it is recommended that a volatility check is implemented, similar to the exchange rate circuit breaker from SIP-65.

Update

Implemented in e45b6d9.

5.5.2 Potential overflow (low risk)

SIP-80
FuturesMarket.sol#L946

Description

The `_sameSide(int a, int b)` function was vulnerable to overflows if both parameters were very large.

The function should be reimplemented to ensure that both parameters are above or below 0 to determine whether they are on the same side.

Update

Implemented in f88c9f3.