Synthetix Shaula Release Smart Contract Audit

1. Introduction

iosiro was commissioned by Synthetix to conduct a smart contract audit on changes introduced in the Shaula release, including SIP-93, SIP-97, SIP-98, SIP-100 and SIP-103. The audit was performed between 20 November 2020 and 24 December 2020.

This report is organised 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 risk exposure of the smart contracts, and as a guide to improving the security posture of the smart contracts by remediating the issues that were identified. The results of this audit are only a reflection of the source code reviewed at the time of the audit and of the source code that was determined to be in-scope.

The purpose of this audit was to achieve the following:

- Ensure that the smart contracts functioned as intended.
- Identify potential security flaws.

Assessing the market effect, economics, game theory, or underlying business model of the platform were strictly beyond 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 very high risk with regards to 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. There are a number of techniques that can help to achieve this, some of which are described below.

- Security should be integrated into the development lifecycle.
- Defensive programming should be employed to account for unforeseen circumstances.
- Current best practices should be followed when possible.

2. Executive Summary

This report presents the findings of the audit performed by iosiro on the smart contract implementation of the Synthetix Shaula release. The following Synthetix Improvement Proposals (SIPs) were included in the audit.

SIP-93

SIP-93 implemented the Spartan Council delegated voting mechanism. Council members would be voted in by the community and then assigned an NFT token.

One medium risk, one low risk, and several informational issues were identified and closed during the audit. The medium risk issue related to a lack of validation for the `from` address when transferring, which could lead to certain edge-cases. The risk was largely mitigated due to the fact that the `transfer` function could only be called by the DAO.

SIP-97 and SIP-103

SIP-97 introduced the ability for users to borrow synthetic assets against ETH and ERC-20 collateral. Initially, users could borrow sUSD/sETH against ETH and sUSD/sBTC against renBTC, however, additional synths are planned to be made available for these loans. SIP-103 extended SIP-97 to include shorts. Shorts effectively performed a loan, however, the equivalent amount of sUSD was issued instead of the borrowed asset.

Six high risk issues and two medium risk issues were identified during the audit. The high risk issues related to users potentially bypassing the front-running reclaiming protection, incorrectly calculating amounts to issue and payout, improperly determining the collateralisation ratio, and potentially being vulnerable to front-running and flash loan attacks. One informational risk issue was still open at the conclusion of the audit.

SIP-98

SIP-98 implemented a double fee on any swing trade, i.e. any exchange that went between an `s` and an `i` Synth, excluding sUSD.

During the audit one low risk and one informational issue were identified and closed. The medium risk issue related to the fee not being correctly assigned.

SIP-100

SIP-100 introduced a modified address resolver pattern that simplified deployment and reduced gas costs.

No issues were identified during the audit. Overall, the implementation was of a high standard and accorded with the specification provided.

3. Audit Details

3.1 Scope

The source code considered in-scope for the assessment is described below. Code from all other files is considered to be out-of-scope. Out-of-scope code that interacts with in-scope code is assumed to function as intended and introduce no functional or security vulnerabilities for the purposes of this audit.

3.1.1 Synthetix SIP-93 Smart Contracts

Project Name: spartan-council
Commit: 705f48e
File(s): SpartanCouncil.sol

3.1.2 Synthetix SIP-97 and SIP-103 Smart Contracts

Project Name: Synthetix
Commit: d7671c9, 119164e, eb36644
File(s): Collateral.sol, CollateralErc20.sol, CollateralEth.sol, CollateralManager.sol, CollateralManagerState.sol, CollateralShort.sol, CollateralState.sol, DebtCache.sol, FeePool.sol, Issuer.sol, MultiCollateralSynth.sol, ShortingRewards.sol

3.1.3 Synthetix SIP-98 Smart Contracts

Project Name: Synthetix
Commit: 3b03563
File(s): Exchanger.sol

3.1.4 Synthetix SIP-100 Smart Contracts

Project Name: Synthetix
Commit: 83191da
File(s): AddressResolver.sol, BaseSynthetix.sol, BinaryOptionMarket.sol, BinaryOptionMarketFactory.sol, BinaryOptionMarketManager.sol, DebtCache.sol, Depot.sol, Depot.sol, EtherCollateral.sol, EtherCollateralsUSD.sol, ExchangeRates.sol, Exchanger.sol, FeePool.sol, FixedSupplySchedule.sol, Issuer.sol, Liquidations.sol, MintableSynthetix.sol, MixinResolver.sol, MixinResolver.sol, MultiCollateralSynth.sol,

PurgeableSynth.sol, Synth.sol, SynthetixBridgeToBase.sol, SynthetixBridgeToOptimism.sol, SystemSettings.sol, TradingRewards.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 Ganache test environment, both manually and through the test suite provided. Manual analysis was used to confirm that the code operated at a functional level, and to verify the exploitability of any potential security issues identified.

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. The static analysis results were manually analyzed to remove false-positive results. True positive results would be indicated in this report. Static analysis tools commonly used include Slither, Securify, and MythX. Tools such as the Remix IDE, compilation output, and linters are also used to identify potential issues.

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 addressed to a satisfactory level to remove the risk that it posed.

4. Design Specification

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

4.1 SIP-93

The specification of SIP-93 was based on commit hash 8bb9e74.

4.2 SIP-97

The specification of SIP-97 was based on commit hash e72d867.

4.3 SIP-98

The specification of SIP-98 was based on commit hash 1e5525a.

4.4 SIP-100

The specification of SIP-100 was based on commit hash d24b945.

4.5 SIP-103

The specification of SIP-103 was based on commit hash 2df93b8.

5. Detailed Findings

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

5.1 High Risk

No high risk issues remained open at the conclusion of the audit.

5.2 Medium Risk

No medium risk issues remained open at the conclusion of the audit.

5.3 Low Risk

No low risk issues remained open at the conclusion of the audit.

5.4 Informational

5.4.1 Interest accrual only approximates real-world values

SIP-97, SIP-103

Description

In order to make calculating interest rates on-chain feasible a pointwise approximation was used. This approach measured the current rate versus the last measured rate whenever certain functions were called, and then applied that rate across the delta between the times the two rates. While efficient, this approach could diverge from real-world values, as a discrete approach was used versus a continuous approach. Furthermore, there was no built-in correction mechanism, so it would be possible for errors to compound over time.

Remedial Action

It is recommended that the interest rates be monitored off-chain to determine whether this approach sufficiently tracks real-world values. If not, it may be necessary to implement some form of keeper functionality in order to incentivize storing rates more regularly.

5.5 Closed

5.5.1 Unsettled synths used for loan repayments (High Risk)

SIP-97, SIP-103

Collateral.sol#L296, Collateral.sol#L328, Collateral.sol#L457, Collateral.sol#L503

Description

Several loan repayment functions burnt synths without first determining whether the user had to first settle any pending rebates or reclaims.

SIP-37 introduced fee reclamations and rebates on user exchanges to prevent front-running. This mechanism ensures that any transfers or exchanges out of a synth first require that the user settle, which initiates the reclaiming and rebating process. However, the `closeInternal()`, `closeByLiquidationInternal()`, `liquidateInternal()` and `repayInternal()` loan functions directly burnt the user's synths without first settling pending rebates or reclaims. This allowed a user to potentially avoid synth reclaims in specific scenarios.

Remedial Action

It is recommended that the function revert if the user has a pending settlement. Alternatively, a user could be forced to first settle in the same transaction.

Update

Implemented in 92e0ab2.

5.5.2 Incorrect collateralisation ratio calculation (High Risk)

SIP-97: Collateral.sol#L132

Description

The `collateralRatio()` function calculated the collateralisation ratio incorrectly for non-sUSD loans.

A loan's collateralisation ratio is defined as the ratio of the value of the collateral to the value of the loan amount, including the loan's interest. The `collateralRatio()` did not evaluate the value of the loan's amount, but rather just the loan amount itself. This meant that the collateralisation ratio of non-`sUSD` loans were calculated potentially significantly higher than the actual ratio as the loaned synth quantity was not converted to `sUSD`.

Remedial Action

It is recommended to use `effectiveValue()` to determine the USD value of the loan debt before calculating the collateralisation ratio.

Update

Implemented in dc50685.

5.5.3 Incorrect loan and short amounts paid out (High Risk)

SIP-97: Collateral.sol#L589, Collateral.sol#L592, Collateral.sol#L656, Collateral.sol#L658

Description

`drawInternal()` and `_processPayment()` improperly calculated short and long positions as the incorrect variables were used.

`Collateral.sol#L589` and `Collateral.sol#L592` should use `amount` instead of `loan.amount`. An example of the issue is given below.

```
Drawing shorts
shorts before the 1 sETH loan: 0
shorts after the 1 sETH loan: 1
shorts after drawing another 1 sETH: 3
```

```
shorts after drawing another 1 sETH: 6
after shorting a total of 3 sETH, the total shorts are: 6
```

Similarly on Collateral.sol#L656 and Collateral.sol#L658 `loan.amount` should be `payment`. An example of the issue is given below.

```
Processing payments
this is the total sUSD longs before repayment: 100
this is the total sUSD longs after 10 sUSD repayment: 9.9999996766332842
```

Remedial Action

The `amount` variable should be used for `drawInternal()` while the `payment` variable should be used in `_processPayment()`.

Update

Implemented in b8d7c71.

5.5.4 Incorrect amount of synths issued (High Risk)

SIP-97: CollateralShort.sol#L45

Description

When opening a short, the amount of collateral claimed by the contract for the user was the total collateral minus the amount of issued synths. This approach was used to avoid having to transfer funds into the contract and then back to the user. However, doing so resulted in the contract itself not holding enough funds to pay out users fully as the synths were not fully issued when creating the short.

Remedial Action

It is recommended that users are issued synths in the same fashion as loans when the position is opened, and that the full collateral amount is claimed in full by the contract to ensure that it has the necessary collateral to close all of the positions.

Update

Implemented in b8d7c71.

5.5.5 Incorrect short collateral returned (High Risk)

SIP-97: Collateral.sol#L366

Description

The `closeInternal()` function should not subtract the loan amount from the collateral when returning a short. If the amount is subtracted, a successful short receives less than it should. In a test case, despite the user opening a short with 1 sETH at $100 and then the price dropping to $50, when closing the short the user was only able to retrieve their starting collateral without any profit.

Remedial

Collateral.sol#L366 should set `collateral = loan.collateral;` (i.e. not subtract `amount`).

Update

Implemented in b8d7c71.

5.5.6 No minimum loan time (High Risk)

Description

SIP-97, SIP-103

Loans and shorts could be opened and closed in the same block. As a result, they could be vulnerable to flash-loan attacks, as well as front-running attacks.

Flash loans

The interest rate for loans was dependent on the utilisation rate, which was defined as the ratio of non-SNX backed debt to SNX backed debt. An attacker looking to profit through the fees paid out could create a large flash-loan of non-SNX backed debt, and then perform some loan action, which would artificially spike the utilisation ratio, and therefore the interest rate.

Similarly for shorts, the market skew was defined as the difference between the total long and short positions of a synth. It was possible to skew the long positions to be greater than the short positions of a synth, waiving the short interest rate for the user.

Front-running

Due to the delay of retrieving off-chain price information, it was possible for front-runners to reliably profit from both creating and opening longs and shorts.

Remedial Action

It is recommended that a minimum time between performing any loan operations is enforced to protect against both of these attack vectors. While protecting against flash loans would only require a delay of only one block, this would not prevent a large whale from performing a similar attack. As such, in order to mitigate the effect of these attacks, it is recommended the delay be set to at least 30 minutes to ensure that the attackers would be exposed to a sufficient amount of risk to deter them from attempting either attack.

It should be noted that front-runners might still have some advantage in the event of a blackswan event, where they could enter at an earlier price if the event's effect on the market was expected to exceed the delay. As such, an alternative approach of integrating loans and shorts into the reclaim and rebate mechanism used for Synthetix exchanges might be considered.

Update

A delay was implemented in 92e0ab2.

5.5.7 Improper short rewards authorisation (High Risk)

SIP-103: ShortingRewards.sol#L179

Description

The Shorting Rewards contract made use of an `onlyShortContract` modifier to ensure that only the Short contract instance could call restricted functionality, such as increasing and decreasing the amount of tokens staked by a user to earn shorting rewards. While the modifier assigned a boolean value indicating whether the calling address was the Short contract instance, the modifier did not revert if the boolean was false. This would allow any account to arbitrarily increase or decrease any other account's stake, which would allow a malicious actor to steal the entirety of the shorting rewards.

Remedial Action

It is recommended that a `require` statement is used to validate the calling address to ensure that restricted functions can only be called by the Short contract instance.

Update

Implemented in a838649.

5.5.8 Fees incorrectly set (Medium Risk)

Description

SIP-97: Collateral.sol#L223

The `setCurrenciesAndNotifyManager()` function called `addShortCurrency()`, which pushed `0` as an interest rate for the currency and then updated the synth's `shortRatesLastUpdated`. As the function was public, anyone could call the function and reset the loan's interest rate. As a result, the last rate became 0 and the last updated time became the current time without accruing interest. Thus, calling this function in the same block as a repayment would cause the interest accrued to be 0.

Remedial Action

It is recommended that `addShortableSynth()` validate whether a synth has been initialized before calling `state.addShortCurrency(synthKey)`.

Update

Implemented in e24d9bf.

5.5.9 Lack of validation for stale synths (Medium Risk)

SIP-97

Description

During the audit, several instances were identified where rates were not being correctly validated to ensure that they were not stale. Using outdated rates could result in the system incorrectly calculating values and result in loss for both the system and users. The identified instances are given below.

1. The `getShortRate()` function did not return whether any rates are invalid, which could result in a stale rate being used.

2. The `accrueInterest()` function did not validate whether any rates are invalid before calculating and storing the latest interest rates.
3. `_totalIssuedSynths()` was found to incorrectly determine the `anyRateIsInvalid` variable, as the `totalLong` function did not return whether any of the rates were invalid.

Remedial Action

It is recommended that rates are validated before being used to ensure that they are current. Furthermore, `_totalIssuedSynths()` and `getShortRate()` should return whether any rates are invalid.

Update

1. Implemented in 92e0ab2.
2. Implemented in 92e0ab2.
3. Implemented in 6759c6f, however, a further issue was reported in the fix relating to the rate being reset in each iteration of the `for` loop. The fix to this issue can be found in 2117166.

5.5.10 No validation for transferring `from` the 0x0 Address (Medium Risk)

SIP-93: SpartanCouncil.sol#L71

Description

The transfer function was found to permit transferring tokens that had not yet been minted if the `from` address was 0x0. If the token ID were to then be burned it would pop off the wrong element from the array as the `for` loop would not find the ID. Additionally, the total supply (total number of council members) would be incorrect, which could impact on reaching consensus. This would also theoretically allow the 0 ID to be minted, which was assumed to not be possible.

Remedial Action

It is recommended that a validation is added to ensure that the `from` address is not the 0x0 address, i.e. `from != address(0)`.

Update

Implemented in f3a41e9.

5.5.11 Double fee not stored (Medium Risk)

SIP-98: a3395b6

Description

It was found that the result of doubling the fee was not being stored in the `exchangeFeeRate` variable that was returned in the `_feeRateForExchange()` function.

Remedial Action

It is recommended that the line be updated to `exchangeFeeRate = exchangeFeeRate.mul(2);` in order to correctly store the value.

Update

Implemented in fa7c10c.

5.5.12 Mint does not check if owner already owns a token (Low Risk)

SIP-93: SpartanCouncil.sol#L93

Description

It was possible to mint a token to an address that already owned a token, which could lead to inconsistencies in the system. The transfer function had `require(tokenOwned[to] == 0, "Destination address already owns a token");` while mint did not. Therefore the owner could mint a token to a user who already had one, which would overwrite their existing token in `tokenOwned` and add a new `ownerOf`. The half-owned token (correctly tracked in `ownerOf`, but incorrectly tracked in `tokenOwned`) could then be transferred to another user who did not have a token.

Remedial Action

It is recommended that the `mint` function check that `tokenOwned[to] == 0` to ensure that the address does not already own a token.

Update

Implemented in c13cae3.

5.5.13 The collateral rate may be invalid for loan operations (Low Risk)

SIP-97: Collateral.sol#L277, Collateral.sol#L317, Collateral.sol#L346, Collateral.sol#L373, Collateral.sol#L410, Collateral.sol#L474

Description

The loan operations `closeInternal()`, `closeByLiquidationInternal()`, `depositInternal()`, `withdrawInternal()`, `liquidateInternal()`, `repayInternal()` did not use the `CollateralRateNotInvalid` modifier. Without this modifier, it may be possible to perform loan operations while the collateral rate was flagged as invalid, or was considered stale.

Remedial Action

It is recommended to use the `CollateralRateNotInvalid` modifier on the aforementioned functions.

Update

Implemented in Collateral.sol#L288, Collateral.sol#L328, Collateral.sol#L384, Collateral.sol#L421, Collateral.sol#L485 and 75a4ddd.

5.5.14 Incorrect issue fee rate assignment (Low Risk)

SIP-97: Collateral.sol#L207

Description

The `setIssueFeeRate()` function in the Collateral contract did not correctly assign the passed `_issueFeeRate` parameter. This prevented `issueFeeRate` from being updated, leaving it at the default value of 0.

Remedial Action

It is recommended to assign the passed parameter `_issueFeeRate` to the `issueFeeRate` state variable.

Update

Implemented in a715993.

5.5.15 Issued synths may have an invalid rate (Informational)

SIP-97

Description

The development team indicated that additional functionality to loan other synths from ETH and ERC-20 collateral will be added. As such, the loaned synth should be validated to not have an invalid rate before being issued.

Initially, only sUSD/sETH can be loaned from ETH collateral, and sUSD/sBTC can be loaned from renBTC collateral. Since the `openInternal()` function validated that the collateral rate was not invalid, this was not an issue in the current version of contract.

Remedial Action

It is recommended to validate the rate of the synth before issuance to future-proof the contract for other loanable synths.

Update

Implemented in 6759c6f.

5.5.16 Unsafe arithmetic used (Informational)

SIP-97: accrueInterest()

Description

The `accrueInterest()` function made use of potentially unsafe mathematical operators.

Remedial Action

It is recommended that the SafeMath library be used to protect against potential issues.

Update

Implemented in 82cc39e.

5.5.17 Use of `transfer` function (Informational)

SIP-97: CollateralEth.sol#L77

Description

The `claim()` function made use of the `transfer()` function to send ether. While `transfer()` is commonly used to prevent reentrancy attacks due to its 2300 gas limit, it relies on the receiving contract to have a fallback function below this limit. As demonstrated in EIP-1884, which changed the gas cost of the `SLOAD` operation, gas costs can change. This could lead to a case where a contract has its fallback function increased above the 2300 limit, resulting in it becoming incompatible with the system. More information can be found in Consensys On Avoiding `transfer()`.

Remedial Action

It is recommended that the `call()` function be used to send ether instead of `transfer()`.

Update

Implemented in 6759c6f.

5.5.18 Design comments (Informational)

Actions to improve the functionality and readability of the codebase are outlined below.

Inefficient gas usage

SIP-97

The `accrueInterest()` function was found to have two potential gas issues.

1. The `getRates()` function called an external contract that returned an array that increased in size each time the `accrueInterest` function was called. As such, the cost of calling the function was found to grow rapidly. It is recommended that functions are implemented to allow retrieving only the necessary values from the array.
2. The call to the `updateLoan()` function was found to have a similar issue, as the users loans needed to be iterated through. As the number of open loans

increased, so did the amount of gas required to call the function. It is recommended that a maximum amount of loans per user is enforced or an alternative data storage pattern be used to store and retrieve this information.

Update

The `accrueInterest()` function was reworked in 6759c6f. A maximum amount of loans per user was enforced in 75a4ddd.

Total debt cap

SIP-97, SIP-103

As a safety precaution, it is recommended that a cap be enforced on the total debt that can be generated through issuing loans as a defense in-depth measure.

Update

Implemented in 75a4ddd.

Minimum loan size

SIP-97, SIP-103

It is recommended that a minimum loan size is implemented in order to mitigate the effect of griefing attacks.

Update

Implemented in 75a4ddd.

Reduce code duplication

SIP-93

The logic of the code could be simplified and the code size reduced. SpartanCouncil.sol#L110: `mint()` and `mintWithTokenURI()` could be refactored to share a single private `_mint()` function.

Update

Implemented in c13cae3.

Reentrancy guard

SIP-97

It is recommended that the `claim()` function implement the OpenZeppelin reentrancy guard to protect against reentrancy attacks. While no issues were identified due to the use of the checks-effects pattern, it is recommended that the modifier still be used as a defense in-depth measure.

Update

Implemented in 6759c6f.

Refactoring suggestions

It is recommended that portions of the code be refactored to improve readability and consistency, as indicated below.

1. SIP-93: SpartanCouncil.sol#L71 as the `transfer()` parameters are more similar to the traditional `transferFrom()` function, it may make sense to change the function name to `transferFrom()`.
2. SIP-97: Collateral.sol#L101 the comment indicated that the functions following the comment were external while they were internal.
3. SIP-97: CollateralManager.sol#L89 `synthByAddress()` could be called `hasSynth()`.
4. SIP-97: Collateral.sol#L241, CollateralManager.sol#L299, and CollateralManager.sol#L305 can remove unnecessary validation of `uint` parameters.
5. SIP-97: CollateralManager.sol `getLoanId()` should be `getNewLoanId()` to indicate that a new ID is being generated.
6. SIP-98: Exchanger.sol#L813 the sUSD constant variable should be used instead of using the equivalent string.

Update

1. Implemented in c13cae3.
2. Implemented in 0b1a1fc.
3. Implemented in 6759c6f.
4. Implemented in 7c1a789.
5. Implemented in 7c1a789.

6. Implemented in fa7c10c.

Fix typos, spelling and grammatical errors

Language mistakes were identified in the comments and revert messages in the codebase. Fixing these mistakes can help improve the end-user experience by providing clear information on errors encountered, and improve the maintainability and auditability of the codebase.

1. SIP-93: SpartanCouncil.sol#L15 `transfer` → `transferred`.
2. SIP-93: SpartanCouncil.sol#L22 `tokenx` → `tokens`.
3. SIP-97: Issuer.sol#L194 "Ether Collateral" → "non-SNX collateral".
4. SIP-97: Collateral.sol#L537 should read "8. Get the loan's last cumulative rate".

Update

1. Implemented in c13cae3.
2. Implemented in c13cae3.
3. Implemented in 6759c6f.
4. Removed in 6759c6f.