Synthetix Alphard Release Smart Contract Audit

1. Introduction

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

- SIP-185 from 17 to 28 January 2022 with two auditors, consuming a total of 18 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 Alphard release.

SIP-185 introduced the `SynthetixDebtShare` contract as a new method to track stakers debt ownership in the system. The new architecture reduced the complexity of debt tracking and provided functionality to enable the transferral of debt to bridged chains.

One medium-risk issue was found which could prevent users from claiming their fees and rewards in certain conditions. This was resolved by Synthetix during the audit. Several low and informational issues were brought to the attention and were also resolved. The code was found to be of a high standard, and no issues identified were open at the conclusion of the audit.

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-185 smart contracts

Commit: f3537b9
Final commit: 9b54331
Files: SynthetixDebtShare.sol, BaseSynthetix.sol, FeePool.sol, Issuer.sol, ISynthetixDebtShare.sol, IFeePool.sol, IIssuer.sol

3.2 Methodology

The audit was conducted using a variety of techniques 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 discover security issues that could be exploited. 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
BaseSynthetix.sol	100	100	100	100	
FeePool.sol	100	79.17	97.83	95.98	... 653,654,655
Issuer.sol	100	93.59	98.61	99.56	307
SynthetixDebtShare.sol	100	81.25	100	100	
IFeePool.sol	100	100	100	100	
IIssuer.sol	100	100	100	100	
ISynthetixDebtShare.sol	100	100	100	100	

The coverage of branches was low. Additional test cases were provided by the iosiro test team to improve the coverage. It was also discussed with the team whether the `FeePool` contract could be simplified, reducing the number of logic branches. This was possible as the current configuration no longer required a variable history length. However, it was decided to maintain the current implementation.

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.

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. This specification is based on the implementation in the codebase. Any perceived points of conflict should be highlighted with the auditing team to determine the source of the discrepancy.

SIP-185 introduced the use of tokenized debt shares in the Synthetix system. This allowed debt to be tracked by minting ERC-20 tokens, each equivalent to 1 sUSD of debt incurred by stakers. A staker's percentage of the total system debt incurred could be calculated at any time as the quotient of their token balance over the total supply; this approach is simpler to implement and integrate than the previous debt ledger.

ERC-20 `SynthetixDebtShare` debt shares can be bridged to other chains, such as Optimism, through Synthetix bridge contracts. This allows stakers to manage their Synthetix portfolios on chains with cheaper gas fees.

As part of the deployment, Synthetix will take a snapshot of the system debt on the debt ledger and migrate an equivalent amount of debt shares to the

`SynthetixDebtShare` contract for users. Beyond this, the system will update itself by snapshotting the total system debt at the close of each fee period.

The `SynthetixDebtShare` ERC-20 token fields are tabulated below:

Field	Value
Symbol	SDS
Name	Synthetix Debt Shares
Decimals	18

5. Detailed findings

The following section details the findings of the audit.

5.1 High risk

No identified high-risk issues were open at the conclusion of the review.

5.2 Medium risk

No identified medium-risk issues were open at the conclusion of the review.

5.3 Low risk

No identified low-risk issues were open at the conclusion of the review.

5.4 Informational

No identified informational issues were open at the conclusion of the review.

5.5 Closed

5.5.1 Fees and rewards unclaimable if staker’s debt is reduced to zero (medium risk)

FeePool.sol#L555

Description

It was found that stakers who reduced their debt share to zero in the current fee period could not claim fees or rewards. The system incorrectly assumed that if a staker's current balance was zero, their balance at the end of the previous fee period was also zero.

The previous debt ledger implementation of `FeePool.feesByPeriod(address account)` made use of the staker’s `debtEntryIndex` to determine whether the staker historically had any debt. The implementation reviewed did not contain a similar check:

```
@@ -579,15 +545,14 @@ contract FeePool is Owned, Proxyable, LimitedSetup, Mi
     function feesByPeriod(address account) public view returns (uint[2][FEE_
         // What's the user's debt entry index and the debt they owe to the s
         uint userOwnershipPercentage;
-        uint debtEntryIndex;
-        FeePoolState _feePoolState = feePoolState();
+        ISynthetixDebtShare _debtShare = synthetixDebtShare();

-        (userOwnershipPercentage, debtEntryIndex) = _feePoolState.getAccoun
+        userOwnershipPercentage = _debtShare.sharePercent(account);

         // If they don't have any debt ownership and they never minted, they
         // User ownership can reduce to 0 if user burns all synths,
         // however they could have fees applicable for periods they had mint
-        if (debtEntryIndex == 0 &amp;&amp; userOwnershipPercentage == 0) {
+        if (userOwnershipPercentage == 0) {
             uint[2][FEE_PERIOD_LENGTH] memory nullResults;
             return nullResults;
         }
```

As can be seen in the diff above, if the `userOwnershipPercentage` for the current fee period was 0, the function would always return `nullResults`.

Recommendation

Exposing the number of balance entries for each staker would allow a similar check to the previous `debtEntryIndex` validation. If the balance array for the specific staker is zero, then it can be assumed that the staker has no debt history and would not be eligible for any fees or rewards.

Update

The issue was resolved in 49d8a69 by removing the function's early return for when the staker's current debt share is zero. Thus, the staker's balance history will always be used to calculate the fees and rewards that they are eligible to claim.

5.5.2 Missing events (low risk)

SynthetixDebtShare.sol#L141, *SynthetixDebtShare.sol#L145*

Description

Events were missing for `addAuthorizedBroker` and `removeAuthorizedBroker` .

Recommendation

Events should be emitted whenever there is a change to the set of `authorizedBrokers` . This will assist in monitoring the changes to the system.

Update

The recommended events were added in commit 49d8a69.

5.5.3 Design comments (informational)

Outdated comment

The comment above FeePool#L555 was outdated and still referenced `debtEntryIndex` .

Update

The comment was updated in 49d8a69.

Code duplication

SynthetixDebtShare.sol#L213, *SynthetixDebtShare.sol#L231*

`SynthetixDebtShare._deductBalance(address account, uint amount)` contained the following duplicate code:

```
balances[account].push(PeriodBalance(uint128(uint(balances[account][accountBalanceCou
```

`

To prevent unnecessary code duplication, the value should rather be calculated before the `if` statement. The repeated calculation was also present in `SynthetixDebtShare._increaseBalance(address account, uint amount)`.

Update

Fixed in 49d8a69 and 19eb553.

Zero address validation

SynthetixDebtShare.sol#L184

`SynthetixDebtShare.transferFrom` should add a zero address check for the recipient to prevent accidental transfers to an incorrect address.

Update

A zero address check was added in commit 49d8a69.

Redundant code

Due to the number of changes made, several legacy sources files, functions, variables, and constants became redundant:

- FeePoolState.sol was no longer used and could be removed.
- `FeePool.synthetix()` was no longer used and could be removed. This included the `CONTRACT_SYNTHETIX` constant returned included in the `resolverAddressesRequired` list and the interface import
- `Issuer.collateralManager()` is no longer used and can be removed. This includes the `CONTRACT_COLLATERALMANAGER` constant included in the `resolverAddressesRequired` list and the interface import.
- `SynthetixDebtShare.sharePercent(...)` could reuse `SynthetixDebtShare.sharePercentOnPeriod(...)` to avoid unnecessary code duplication. * `SynthetixDebtShare.balanceOfOnPeriod(...)` could internally be called by `SynthetixDebtShare.balanceOf(...)` to avoid unnecessary code duplication.

- The `FeePool.onlyIssuer` modifier is no longer used and should be removed.

Update

Fixed in 49d8a69 and 19eb553. The team decided to not refactor `SynthetixDebtShare.balanceOf(...)` .

Refactoring

A number of general code refactoring suggestions were made during the audit:

- The naming conventions for variables of type `ISynthetixDebtShare` were inconsistent. `sds` , `debtShare` and `_debtShare` were used at different points in the code.
- BaseSynthetix#L143 used the literal string "sUSD", when a constant had already declared and should be used instead.

Update

Fixed in 49d8a69.

Gas optimizations

- The slot layout of at SynthetixDebtShare.sol#L53 was neither optimally packed nor optimised for gas efficiency. It was more gas efficient to pack `currentPeriodId` , `decimals` and `isInitialized` together into a single storage slot.
- `FeePool.effectiveDebtRatioForPeriod` had a number of `require` statements that could be rewritten to return 0 instead, which could avoid unnecessary reverts.
- `FeePool._feesAndRewardsFromPeriod` could cache the `recentFeePeriodsStorage` result to reduce repeated SLOADs.

Update

Fixed in 49d8a69 and 19eb553.

Secure your system.

Request a service

START NOW →

ABOUT

CONTACT US

SMART CONTRACT AUDITING

PENETRATION TESTING

AUDIT REPORTS

PRIVACY POLICY

TERMS OF SERVICE