Synthetix Alkaid Release Smart Contract Audit

1. Introduction

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

- SIP-120 from 5 to 9 July with two auditors, and one auditor on 13 July, 25 August, 1 November, 4 November, consuming a total of 14 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 Alkaid release.

SIP-120 introduced a new exchange function allowing users to atomically exchange assets without fee reclamation by pricing synths via a combination of Chainlink and DEX oracles. One high risk, one medium risk, and one low risk issue were identified and remediated during the audit.

In addition to the implementation of SIP-120, the removal of Inverse Synths from the system was also reviewed. No related issues were identified.

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-120 smart contracts

SIP-120

Commits: 97bdd78, 7d4e868, e06e164, 84c7658, ec86069
Files: BaseSynthetix.sol, ExchangeRates.sol, ExchangeRatesWithDexPricing.sol, Exchanger.sol, ExchangerWithFeeRecAlternatives.sol, ExchangerWithVirtualSynth.sol, MixinSystemSettings.sol, Synthetix.sol, SystemSettings.sol

Uniswap v3 Cross Pool Oracle

Commits: c348634
Files: UniswapV3CrossPoolOracle.sol

Uniswap v3 Spot TWAP Oracle

Commits: 8f9777a
Files: DexPriceAggregatorUniswapV3.sol

DexPriceAggregatorUniswapV3

Contract Address: 0xf120f029ac143633d1942e48ae2dfa2036c5786c

Inverse Synth Removal

Commits: 913d0b3

Files: ExchangeRates.sol, ExchangeRatesWithoutInvPricing.sol, Exchanger.sol, PurgeableSynth.sol, SynthUtil.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 discover 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.

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-120

The specification of SIP-120 was based on commit hash 52b3e2a.

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

No identified informational issues were present at the conclusion of the audit.

5.5 Closed

5.5.1 Function signature mismatch (high risk)

IDexPriceAggregator.sol#L8

Description

When using a mainnet fork to test IDexPriceAggregator with the UniswapV3CrossPoolOracle at 0xeaafd7547b781c60c71f0854a7da2c1ff23a7dd0 the tests would revert due to a function signature mismatch as the `twapPeriod` period being used in the interface was uint256, while in the implementation, it was represented by a uint32.

Recommendation

The function signatures should be updated to match to allow the interface to call the contract correctly.

Update

A new version of the DexPriceAggregatorUniswapV3 that made use of a uint256 to represent `_twapPeriod` was deployed to mainnet.

5.5.2 Outdated oracle library (medium risk)

Description

The DexPriceAggregator was found to use an outdated version of the Uniswap Oracle library. The main discrepancy between the versions was that the `baseAmount` variable in the deployed version was uint256, while the reference implementation made use of uint128.

Recommendation

The Oracle library should be updated to the latest available version from the Uniswap v3 Periphery repository.

Update

The new DexPriceAggregatorUniswapV3 made use of an updated Oracle library.

5.5.3 Avoid unbounded `while` loop (low risk)

ExchangeRatesWithDexPricing.sol#L103

Description

An unbounded while loop was used in the `synthTooVolatileForAtomicExchange` function, and the return value defaulted to true. While no obvious issue was identified with this approach, a cleaner and safer approach would be to use a fixed-length loop and default to `true` , which would imply that the synth is too volatile.

Recommendation

A code snippet to implement the suggested fixes is given below.

```
function synthTooVolatileForAtomicExchange(bytes32 currencyKey) external vie
        // sUSD is a special case and is never volatile
        if (currencyKey == "sUSD") return false;
        uint considerationWindow = getAtomicVolatilityConsiderationWindow(cu
        uint updateThreshold = getAtomicVolatilityUpdateThreshold(currencyKe

        if (considerationWindow == 0 || updateThreshold == 0) {
            // If either volatility setting is not set, never judge an asset
            return false;
        }

        // Go back through the historical oracle update rounds to see if the
        // updates in the consideration window than the allowed threshold.
        // If there have, consider the asset volatile--by assumption that ma
        // updates is a good proxy for price volatility.
        uint considerationWindowStart = block.timestamp.sub(considerationWin
        uint roundId = _getCurrentRoundId(currencyKey);
        for (updateThreshold; updateThreshold > 0; updateThreshold--) {
            // TODO: this assumes chainlink may be unreliable in reporting r
            // reliable in reporting times (always incrementing, roughly sim
            // block.timestamps). Is this a good assumption?
            (uint rate, uint time) = _getRateAndTimestampAtRound(currencyKey
            if (time != 0 &amp;&amp; time < considerationWindowStart) {
                // Round was outside consideration window so we can stop que
                return false;
            } else if (rate == 0 || time == 0) {
                // Either entire round or a rate inside consideration window
                // Consider the asset volatile
                break;
            }

            if (roundId == 0) {
                // Not enough historical data to continue further
                // Consider the asset volatile
                break;
            }
            roundId--;
        }

        // Safer to default to true
        return true;
    }
```

Update

Implemented in e06e164.