macro

Synthetix 3

Security Audit

March 2, 2023
Version 1.0.0

Table of Contents

Introduction

This document includes the results of the security audit for Synthetix's smart contract code as found in the section titled ‘Source Code’. The security audit was performed by the Macro security team from February 6, 2023 to February 10, 2023. This was the 2nd audit on this codebase, the first being a partial audit whose report can be found at Synthetix-2.

The purpose of this audit is to review the source code of certain Synthetix Solidity contracts, and provide feedback on the design, architecture, and quality of the source code with an emphasis on validating the correctness and security of the software in its entirety.

Disclaimer: While Macro’s review is comprehensive and has surfaced some changes that should be made to the source code, this audit should not solely be relied upon for security, as no single audit is guaranteed to catch all possible bugs.

Overall Assessment

The following is an aggregation of issues found by the Macro Audit team:

Severity	Count	Acknowledged	Won't Do	Addressed
High	2	-	-	2
Medium	3	-	-	3
Low	3	-	-	3
Code Quality	4	-	-	4
Informational	1	-	-	-

Synthetix was quick to respond to these issues.

Specification

Our understanding of the specification was based on the following sources:

- Discussions on Discord with the Synthetix team.
- Available documentation in the repository.

Source Code

The following source code was reviewed during the audit:

- **Repository:** synthetix-v3
- **Commit Hash:** `b50c83fde3d4967a366aedcb5c2d3e06246287c3`

Specifically, we audited the following contracts within this repository:

Contract	SHA256
contracts/interfaces/INodeModule.sol	`4a7d71969ace6a254d9bcbaf1885b5d0341e6e485c5bfe2d90ca88e1ea325507`
contracts/interfaces/external/IAggregatorV3Interface.sol	`6bb31e2643c0a95ecff1903d61b21f3609bb5a5f01fec9fa226105e3c91f0a23`
contracts/interfaces/external/IExternalNode.sol	`20dbb2d2fc5eee8f17641b381fc143b6ff799f656bc496fae332aaa553137eb3`
contracts/interfaces/external/IPyth.sol	`159b577f8dfc00b0cea383ca484d664ea3a99f12473796337251cbe97f83b70d`
contracts/interfaces/external/IUniswapV3Pool.sol	`1c28cb50c0e42f4cbf6d74724f5db06e02af88e22e3c903af97b42e35a2953e7`
contracts/modules/CoreModule.sol	`e61f3485048125c368650f6aeb285710910b5b96ca2dbc0a78017190270d0e1d`
contracts/modules/NodeModule.sol	`1166fd85eb570af80540ee27cfee7070e83e6b57af4a679229fd01582bf88fd9`
contracts/nodes/ChainlinkNode.sol	`438cf0f610bd20e315924b43b325fe540c29e7f57fbb8ae55772c9b82ce677c6`
contracts/nodes/ExternalNode.sol	`c78957d988fc938dd613d7c0bbd0f758bc5cda8401c72587601478d0d8d3d4b9`

Contract	SHA256
contracts/nodes/ExternalNodeUniswap.sol	`536f7139e4c35d123e06b6ef52be24bed6bfcf22fb5dce2e03cbf05dd82c74d3`
contracts/nodes/PriceDeviationCircuitBreakerNode.sol	`d510b681d34f5a32785646e8331b6afce0b65bd6234dbe08bfd9712772182da0`
contracts/nodes/PythNode.sol	`d2abe84fb1a19d3462d1ebd087ebe3a7b74c90544751be1872141ad8a102c480`
contracts/nodes/ReducerNode.sol	`d52933f8a33a59ecf1af330020b25ea58509d59ea899f93893c409b5e9ed7dc9`
contracts/nodes/StalenessCircuitBreakerNode.sol	`47442546b62b41dc8a0fb90ee6a11ad696c2d9e1be5014e4503f20da64879613`
contracts/nodes/UniswapNode.sol	`f36e394e642069a34814a56b1edad7dafc2c7ede470dfce94da9c3f3d38d5bc3`
contracts/storage/NodeDefinition.sol	`232a16c06b3ea103504829425e8df708ad8586fb8f2b6af30a8ec80940e84c93`
contracts/storage/NodeOutput.sol	`18b612549db0ce1c1682e18c862b90222fb8f2c8d7056099cc16d85ad607b7c6`
contracts/utils/FullMath.sol	`115a0473ef75cbad65de51cd67b2023c64096d5e81633fb9eab8edf28bcac593`
contracts/utils/TickMath.sol	`152cc5feab872139b4dcc7b1646c42d9653393dda54af656b9d3057fa1bfc5ec`

Note: This document contains an audit solely of the Solidity contracts listed above. Specifically, the audit pertains only to the contracts themselves, and does not pertain to any other programs or scripts, including deployment scripts.

Issue Descriptions and Recommendations

Click on an issue to jump to it, or scroll down to see them all.

Security Level Reference

We quantify issues in three parts:

1. The high/medium/low/spec-breaking **impact** of the issue:
    - How bad things can get (for a vulnerability)
    - The significance of an improvement (for a code quality issue)
    - The amount of gas saved (for a gas optimization)
2. The high/medium/low **likelihood** of the issue:
    - How likely is the issue to occur (for a vulnerability)
3. The overall critical/high/medium/low **severity** of the issue.

This third part – the severity level – is a summary of how much consideration the client should give to fixing the issue. We assign severity according to the table of guidelines below:

Severity	Description
(C-x) Critical	We recommend the client **must** fix the issue, no matter what, because not fixing would mean **significant funds/assets WILL be lost.**
(H-x) High	We recommend the client **must** address the issue, no matter what, because not fixing would be very bad, *or* some funds/assets will be lost, *or* the code's behavior is against the provided spec.
(M-x) Medium	We recommend the client to **seriously consider** fixing the issue, as the implications of not fixing the issue are severe enough to impact the project significantly, albiet not in an existential manner.
(L-x) Low	The risk is small, unlikely, or may not relevant to the project in a meaningful way. Whether or not the project wants to develop a fix is up to the goals and needs of the project.
(Q-x) Code Quality	The issue identified does not pose any obvious risk, but fixing could improve overall code quality, on-chain composability, developer ergonomics, or even certain aspects of protocol design.
(I-x) Informational	Warnings and things to keep in mind when operating the protocol. No immediate action required.
(G-x) Gas Optimizations	The presented optimization suggestion would save an amount of gas significant enough, in our opinion, to be worth the development cost of implementing it.

Issue Details

~~H-1~~ `PriceDeviationCircuitBreaker` does not handle negative prices correctly

TOPIC	STATUS	IMPACT	LIKELIHOOD
Input Ranges	Fixed	High	Medium

`PriceDeviationCircuitBreaker`'s logic currently does not handle negative prices correctly. The logic in the node fails to account for this on line 30:

```
int256 difference = abs(primaryPrice - fallbackPrice);
    // line 30 is line below
  if (deviationTolerance.toInt() < ((difference * 100) / primaryPrice)) {
      if (parentNodeOutputs.length > 2 && parentNodeOutputs[2].price != 0) {
          return parentNodeOutputs[2];
      } else {
          revert DeviationToleranceExceeded(difference / primaryPrice);
      }
  }
```

If the `primaryPrice` is negative, it will make the computed deviation negative and therefore always less than the deviation tolerance, even if the spread is larger.

Remediations to Consider:

- Using the absolute value of the `primaryPrice` instead of the raw value.

~~H-2~~ `ReducerNode`'s `max()` operation incorrect on only negative prices

TOPIC	STATUS	IMPACT	LIKELIHOOD

Input Ranges	Fixed	High	Low

`NodeOutput`'s `Data` allows for prices to be negative. Currently the `ReducerNode`'s `max()` will return zero if all of the input parent node's data is negative instead of the max negative price. This is due to the return variable being initialized to zero.

Remediations to Consider:

- Similarly to the `min()` function, initialize the output variable with data from one of the parent nodes.

~~M-1~~ `UniswapNode` does not validate that input tokens belong to the `pool`

TOPIC	STATUS	IMPACT	LIKELIHOOD
Interoperability	Fixed	High	Low

In NodeModule.sol, a `UniswapNode` can be registered by supplying the input parameters `pool`, `token0`, and `token1` address. Currently, there is no validation to ensure that `token0` and `token1` are the correct pair of the `pool`.

Since the quote amount depends on the address ordering of `token0` and `token1`, this can result in an incorrect final price calculation. It can also cause issues with the scaling of prices, as the decimals used for scaling are based from the tokens as well.

Remediations to Consider:

- Directly validating that `token0` and `token1` are the pairs used in `pool`.

~~M-2~~ `PriceDeviationCircuitBreaker` does not scale deviation parameter to 18 decimal places

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec	Fixed	Medium	High

The documentation for `PriceDeviationCircuitBreaker` states that the deviation parameter should be scaled to 18 decimal places. Currently the node does no scaling on the parameter. This does not allow for users to specify smaller deviations than 1%, which may not be granular enough for some use cases.

Remediations to Consider:

- Add scaling to the deviation tolerance check on line 30 of PriceDeviationCircuitBreakerNode.sol

---

~~M-3~~ `IExternalNode` is missing logic to validate parameters

TOPIC	STATUS	IMPACT	LIKELIHOOD
Use Cases	Fixed	Medium	Medium

ExternalNodes.sol and IExternalNode.sol do not enable implementations of `IExternalNode` to validate that a node's passed in parameters are valid. This is problematic because it raises the likelihood that users will create malformed external `nodeIds` by accident. This is due to a lack of implementation instead of an incorrect implementation.

Remediations to Consider:

- Adding some sort of `validate()` function to `IExternalNode` and passing parameters to it during ExternalNodes.sol's `validate()` function.

~~L-1~~ `ExternalNodes` lack the ability to have additional parameters

TOPIC	STATUS	IMPACT	LIKELIHOOD
Use Cases	Fixed	Low	High

The hardcoded nodes like, `UniswapNode` and `ChainlinkNode`, have parameters that are passed to the nodes which enable more complex behaviors. It is currently not possible to re-implement the same parameterized behavior of the hardcoded nodes through the `ExternalNode` interface, which is not the intended design. This is due to the ExternalNode.sol's `validate()` function limiting the parameters to only fit the `ExternalNode`'s mandatory contract address.

Remediations to Consider:

- Enabling `ExternalNode`s to have variable amounts of parameters.

~~L-2~~ `UniswapNode` cannot handle ERC20s with larger than 18 decimal places

TOPIC	STATUS	IMPACT	LIKELIHOOD
Input Ranges	Fixed	Medium	Low

ERC20s can have decimals in the range 0-255 (`uint8`'s range). Currently, the UniswapNode.sol's code is unable to process tokens that have decimals larger than 18. Reverts happen in two places:

- When adjusting for `token1`'s scale with `uint256 factor = (PRECISION - decimals1);`, this logic will underflow with any `token1` that has decimals larger than 18.
- When grabbing the price with `getQuoteAtTick()`, all `token0`s with decimals larger than 26 will overflow the `FullMath` calls in this function, and, `token0`s with decimals larger than 19 will overflow with tick values higher than `443637`. This is due to the line `uint256 baseAmount = 10 ** decimals0;`, this resulting value is too large for the `FullMath` code to handle.

ERC20s with higher than 18 decimal places are rare, but they do exist. Currently there are 8 on Ethereum. Even though there are not current Uniswap Pools operating for these 8 coins, we still recommend fixing this for the sake of future proofing the code.

Remediations to Consider:

- Fixing the code to accommodate larger decimals in the range of 0-255, or,
- Validate that created `UniswapNode`'s used tokens have decimals no larger than 18.

~~L-3~~ `PriceDeviationCircuitBreakerNode` does not allow for prices to be zero

TOPIC	STATUS	IMPACT	LIKELIHOOD
Input Ranges	Fixed	Medium	Low

The node `PriceDeviationCircuitBreakerNode` does not allow for prices to be zero, which is problematic as the `Node`'s price variable of `int256` includes zero in its valid range. This isn't documented. The node currently reverts when the `primaryPrice` is zero or if the third parent's price is zero.

The node's code can be modified to accommodate zero pricing. Some design decisions need to be made and should be well documented, as the size of something compared to zero is not a well-defined concept. Some implementation options are:

1. Keeping the code as is and warning developers that zero-prices are not accommodated in this node type.
2. Replacing the `primaryPrice` with the smallest amount (`1`) for comparison purposes if the `fallbackPrice` is also not zero.
3. Always returning the 3rd node parent or reverting if the `primaryPrice` is zero and the `fallbackPrice` is not zero.
4. Letting the node user toggle between options 2/3 with a flag set during node creation.

Remediations to Consider:

- Choosing an implementation option and documenting it clearly for developers.

~~Q-1~~ `UniswapNode`'s `token0` and `token1` naming is not user friendly

TOPIC	STATUS	QUALITY IMPACT
Readability	Fixed	Low

The names `token0` and `token1` for the `UniswapNode` parameters could be better named to help developers construct nodes properly. `token0` is supposed to be the target token, and `token1` is supposed to be the stable coin (or the denominator). Mixing up these tokens will give incorrect prices as creating nodes with them backwards will not give prices denominated in USD.

Remediations to Consider:

- Changing `token0` to `token` or `numerator`
- Changing `token1` to `stablecoin` or `denominator`

~~Q-2~~ ReducerNode.sol doesn't mention that `Operation.MEDIAN` returns the average when there are an equal amount of Parents

TOPIC	STATUS	QUALITY IMPACT
Documentation	Fixed	Low

The `ReducerNode`'s `median()` operation will return the average of the two most middle nodes when the length of the array is even. It would benefit developers to have this documented.

Remediations to Consider:

- Documenting this design choice so developers are aware of it.

~~Q-3~~ `ReducerNode`'s `div()` function will revert if a price is zero

TOPIC	STATUS	QUALITY IMPACT
Documentation	Fixed	Low

In ReducerNode.sol's `div()`'s logic, the function will revert without a good error message if a returned parent's node price is zero.

Remediations to Consider:

- Reverting with an error message instead of panicking the EVM by trying to divide by zero.

~~Q-4~~ `PriceDeviationCircuitBreakerNode`'s naming of `fallbackPrice` is misleading

TOPIC	STATUS	QUALITY IMPACT
Readability	Fixed	Low

The name of `PriceDeviationCircuitBreakerNode`'s variable `fallbackPrice` is misleading because the referenced price isn't the fallback price. That variable is the 2nd parent's price, which is just used to check the 1st parent's price. The 3rd parent is actually the fallback price.

Remediations to Consider:

- Renaming the variable to `comparisonPrice`

I-1 Geometric-Mean or Arithmetic-Mean ChainLink TWAP Options

TOPIC	IMPACT
Protocol Design	Informational ✳

In ChainLinkNode.sol, the TWAP is calculated using an Arithmetic-Mean. Using a Geometric-Mean instead is becoming popular in the DeFi space as it responds differently to types of oracle manipulation attacks. UniswapV3 chose to use the Geometric-Mean in their protocol.

There isn't a clean winner in terms of which option is better, as they have very similar behavior overall. See here for a good exploration of the differences in behavior.

Users could enjoy having the option to use a GM instead of a AM in the `ChainLinkNode`.

Disclaimer

Macro makes no warranties, either express, implied, statutory, or otherwise, with respect to the services or deliverables provided in this report, and Macro specifically disclaims all implied warranties of merchantability, fitness for a particular purpose, noninfringement and those arising from a course of dealing, usage or trade with respect thereto, and all such warranties are hereby excluded to the fullest extent permitted by law.

Macro will not be liable for any lost profits, business, contracts, revenue, goodwill, production, anticipated savings, loss of data, or costs of procurement of substitute goods or services or for any claim or demand by any other party. In no event will Macro be liable for consequential, incidental, special, indirect, or exemplary damages arising out of this agreement or any work statement, however caused and (to the fullest extent permitted by law) under any theory of liability (including negligence), even if Macro has been advised of the possibility of such damages.

The scope of this report and review is limited to a review of only the code presented by the Emergent team and only the source code Macro notes as being within the scope of Macro's review within this report. This report does not include an audit of the deployment scripts used to deploy the Solidity contracts in the repository corresponding to this audit. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. In this report you may through hypertext or other computer links, gain access to websites operated by persons other than Macro. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such websites' owners. You agree that Macro is not responsible for the content or operation of such websites, and that Macro shall have no liability to your or any other person or entity for the use of third party websites. Macro assumes no responsibility for the use of third party software and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.