Nitro Audit Public Report

PROJECT: Nitro Project Audit

December 2020

Prepared For:

Nitro Project Team | Nitro Project

info@nitro.finance

Prepared By:

Jonathan Haas | Bramah Systems, LLC.

jonathan@bramah.systems

Table of Contents

Nitro Project Protocol Review

Executive Summary

Scope of Engagement

Bramah Systems, LLC was engaged in December of 2020 to perform a comprehensive security review of the Nitro Project smart contracts (specific contracts denoted within the appendix). Our review was conducted over a period of five business days by both members of the Bramah Systems, LLC. executive staff.

Bramah Systems completed the assessment using manual, static and dynamic analysis techniques.

Timeline

Review Commencement: December 14th, 2020

Report Delivery: December 21st, 2020

Engagement Goals

The primary scope of the engagement was to evaluate and establish the overall security of the Nitro Project, with a specific focus on trading actions. In specific, the engagement sought to answer the following questions:

- Is it possible for an attacker to steal or freeze tokens?
- Does the Solidity code match the specification as provided?
- Is there a way to interfere with the contract mechanisms?
- Are the arithmetic calculations trustworthy?

Contract Specification

Detailed specification was provided by the Nitro team upon the structure of the Nitro contract layout, flow control, and various aspects of the protocol.

In particular, the contracts surmise five primary functions:

1. Swapping
   a. Swapping aims to allow for two tokens to be swapped between holders, functionally similar to a bulletin board or ECN.
2. Price Verification
   a. Price verification aims to validate the authenticity of the quote that is provided. Price verification is paid for in the quote instrument as a function of volume, in which the caller must pay a fixed rate of the instrument in question as part of this verification. This resultantly mines N2, which is paid out to the various parties that are specified including the price provider. This allows for exchanges and market markers to benefit from on-chain activity, incentivizing participation in the protocol.
3. Farming
   a. The farming function converts deposited tokens within **_quoteInstrument** into their equivalent N2xyz token based on the volume in that instrument. For example, USDT as the **_quoteInstrument** results in the ERC20 of **N2USDT** being generated. Chainlink as the **_quoteInstrument** results in **N2LNK** being generated, and so on. As farming occurs with each verification, network participants are incentivized to participate.
4. Token Delivery / Factory Functionality
   a. Token delivery aims to allow for users to redeem and accept delivery of the underlying asset of their **N2ERC20**, differing from redemption (as one need not accept delivery to redeem, as is described below).
5. Redemption
   a. Redemption aims to allow users to redeem their farming share for the underlying asset without having to accept delivery. This results in a significantly cheaper redemption process, as delivery needn't be accepted (allowing the individual to continue to transact, lowering total gas expenditures).

Overall Assessment

Bramah Systems was engaged to evaluate and identify any potential security concerns within the codebase of the Nitro Project. During the course of our engagement, Bramah Systems found relatively few instances wherein the team deviated materially from established best practices and procedures of secure software development within DLT, as our report details.

These aside, the team otherwise used thoroughly reviewed and vetted components and provided details as to the token structure, economics, and intent, which helped Bramah highlight any potential concerns with their approach. In particular, Nitro's extensive test and function documentation made for very clear delineation of potential concerns versus intended behaviour. While minor deviations from best practices did occur, this extensive documentation made it very straightforward to debug potential areas of concern, including potential system invariants.

In addition, we felt Nitro gave extensive documentation as to their mitigations or responses to our findings. We applaud such dedication to their safety and successful operation of their protocol.

Disclaimer

As of the date of publication, the information provided in this report reflects the presently held, commercially reasonable understanding of Bramah Systems, LLC.'s knowledge of security patterns as they relate to the Nitro Project Protocol, with the understanding that distributed ledger technologies ("DLT") remain under frequent and continual development, and resultantly carry with them unknown technical risks and flaws. The scope of the review provided herein is limited solely to items denoted within "Scope of Engagement" and contained within "Directory Structure". The report does NOT cover, review, or opine upon security considerations unique to the Solidity compiler, tools used in the development of the protocol, or distributed ledger technologies themselves, or to any other matters not specifically covered in this report.

The contents of this report must NOT be construed as investment advice or advice of any other kind. This report does NOT have any bearing upon the potential economics of the Nitro Project protocol or any other relevant product, service or asset of Nitro Project or otherwise. This report is not and should not be relied upon by Nitro Project or any reader of this report as any form of financial, tax, legal, regulatory, or other advice.

To the full extent permissible by applicable law, Bramah Systems, LLC. disclaims all warranties, express or implied. The information in this report is provided "as is" without warranty, representation, or guarantee of any kind, including the accuracy of the information provided. Bramah Systems, LLC. makes no warranties, representations, or guarantees about the Nitro Project Protocol. Use of this report and/or any of the information provided herein is at the users sole risk, and Bramah Systems, LLC. hereby disclaims, and each user of this report hereby waives, releases, and holds Bramah Systems, LLC. harmless from, any and all liability, damage, expense, or harm (actual, threatened, or claimed) from such use.

Timeliness of Content

All content within this report is presented only as of the date published or indicated, to the commercially reasonable knowledge of Bramah Systems, LLC. as of such date, and may be superseded by subsequent events or for other reasons. The content contained within this report is subject to change without notice. Bramah Systems, LLC. does not guarantee or warrant the accuracy or timeliness of any of the content contained within this report, whether accessed through digital means or otherwise.

Bramah Systems, LLC. is not responsible for setting individual browser cache settings nor can it ensure any parties beyond those individuals directly listed within this report are receiving the most recent content as reasonably understood by Bramah Systems, LLC. as of the date this report is provided to such individuals.

General Recommendations

Best Practices & Solidity Development Guidelines

Non-utilization of ABIEncoderV2

Although still marked **experimental**, the ABIEncoderV2 pragma (per Solidity patch notes and an Ethereum Foundation blog post) is no longer considered to be experimental in nature and should be considered for mainline usage.

As the protocol heavily would benefit from the usage of this data structure (explicitly denoting the experimental nature and mitigations which we made in order to avoid its usage), we suggest reconsideration of usage of ABIEncoderV2, for both gas optimization and generalized readability of code.

Resolution: While the team has made a multitude of structural changes to the contract over the course of our engagement, changes to ABIEncoderV2 and the relevant gas costs that come along with this process were deemed to be overall a negative impact to the protocol's structure at this time.

The Nitro team has provided the following: "Nitro was originally written to entirely avoid use of ABIEncoderV2, for several reasons, first there had been previous security flaws introduced by the encoder, and second Nitro is heavily optimized to reduce or defer gas usage whenever possible. The ABIEncoderV2 had a worse gas usage profile in our testing. That being said we did end up using this encoder in the final contracts, specifically to enable multi-hop and multi-source ECN routing. Without the encoder the logic proved to be too difficult to follow, and so in the interest of code legibility it was employed.

That being said, there are many instances in the contract that were [not] upgraded to use structures externally, and still rely on slightly more cumbersome matching length lists of items. We have opted not to rewrite these functions post-audit, as the risk of introducing bugs or further complicating the launch of the product doesn't make sense."

Bramah concurs with this assessment.

SPDX declaration lacks license

If this code is to be licensed in a way that does limit its general distribution following publication to the blockchain, it should be declared as such within the SPDX declaration section of each file.

Resolution: The Nitro team has provided the following “Nitro's source code is available for perusal, primarily to alleviate security concerns, and make it easier for programmers and integrators to understand the inner workings of the protocol. It is Open, However it is not an open source project under a copyleft license. If you wish to fork Nitro for your own purposes, please contact us for licensing arrangements. The SPDX identifier does not explicitly include a reference to commercial licenses, as that runs contrary to the intent and scope of the system. We address this by using the LicenseRef- identifier which points to our non commercial all rights reserved License.md. Insofar as we can tell this is best practices with respect to commercial software.”

Bramah concurs with this assessment.

Events should be “emit” to differentiate between function calls

Events should be “emitted” to stand out with regular function calls, as per Solidity release notes:

General: Support and recommend using `emit EventName();` to call events explicitly.

In order to make events stand out with regards to regular function calls, `emit EventName()` as opposed to just `EventName()` should now be used to "call" events.

In particular, events that perform certain global sensitive actions (such as disabling swapping) should emit an event.

Occurrences:

NitroNexus.sol, Line 42

Resolution: The Nitro team has provided “The NitroNexus contract was missing "emit" on the contract registered and unregistered events. This has been remedied.

Further, an additional event was added when global swapping is disabled which is a reserved function for us (and ultimately the governance contracts) that prevents any swapping from [occurring] as a security precaution."

Bramah confirms events where applicable have been **emit**.

Snake case should be used for constants

Where present, constants should be in snake case as per Solidity's style guide, which borrows structural elements from PEP8.

Occurrences:

NitroSwap.sol, Line 30

Resolution: The Nitro team has provided the following: "Nitro uses all caps to denote constants. Although this differs from [established] solidity guidelines, it was an early decision that we are not going to change at this point. However, instances where capitalization was not employed properly for constants has been fixed."

Bramah has confirmed these fixes.

Functions can be renamed to more accurately reflect action

Function get_token_address should be renamed to generate_token_address to more accurately match action being performed. While a token address is returned, the token address may not exist previously, and is generated at the time of invoking the function.

Resolution: The Nitro team provides the following: "This is a matter of style. The example get_token_address versus generate_token_address. Generate to me sounds like an action that creates something, but get_token_address is a read-only view, that simply derives the token address from the source token. Perhaps a better name would be derive_token_address(), but at this point this won't be fixed."

Bramah understands this decision and agrees that this is a style preference.

Referenced endpoints should utilize HTTPS

Swap testing logic presently reaches out to a multitude of HTTP endpoints. As HTTP endpoint data can be intercepted and modified on certain networks, we suggest ensuring that all endpoints associated with the protocol's native operation utilize SSL/TLS.

Resolution: While a loss of privacy is possible through usage of HTTP, the man-in-the-middle concern is mitigated through the usage of signed web requests within the frontend.

The Nitro team notes: "The endpoints used in the nitro contract testing are not used directly by any of this code, they are just examples. As examples it would be best if they referenced https (SSL) channels, but there is no tangible [effect] one way or the other from a security perspective.

Further, although the backend endpoints do utilize https, it is primarily to preserve user privacy and has little to do with a man-in-the-middle attack. All pricing and execution data in Nitro is signed and verified, an MITM attack that changes any relevant structure would fail the secondary verification checks. This is an intended feature because the source of a quote may not be the endpoint that the user connects to, it could be relayed or otherwise encapsulated."

Bramah concurs with this assessment.

SafeMath library is available but not utilized

The SafeMath library is used to avoid cases of integer overflow among various arithmetic functions. While the SafeMath library is included within the contract imports and the **using** keyword is utilized (attaching library functions from the SafeMath library to the **uint256** type), the functions (**add**, **sub**, **mul**, **div**, **mod**) should be utilized.

Resolution: The Nitro team provides the following: "Originally Nitro was using entirely uint256 integers. When this was changed the intent was to manually verify for underflow and overflow using the constraint solver, thereby making the runtime (gas costing) checks [unnecessary]. This audit confirmed that approach was not sufficient, and as a result all math-using functions have been changed to utilize Safemath, intermediate values, and range checking."

Bramah concurs with this assessment.

Truncation from multiplication following division

There is a loss of precision resulting from multiplication following division in a number of places (as listed below).

Occurrences:

NitroSwap.sol, Line 372

NitroFeed.sol, Line 411

NitroFeedClientMock.sol, Line 55

Resolution: The Nitro team provides the following: “This occurs as part of the fee calculation, and the proportional calculations. The order has been switched to multiply first and divide last, that fixes the potential loss of precision. Both functions were also switched to use SafeMath correctly.”

Bramah concurs with this assessment.

Return value type declared but not utilized

Multiple functions within the protocol define a return type but do not return any value. This may result in assumptions made based on the default return type of the value.

Occurrences:

NitroFeedRegistry.sol, Lines 21-27

NitroFeedRegistry.sol, Lines 18-40

NitroFeedRegistry.sol, Lines 42-48

NitroFeed.sol, Lines 122-131

Resolution: The Nitro team provided the following: “There were instances of functions that had a return declared, but did not explicitly return. This has been resolved.”

Bramah concurs with this assessment.

Typographic errors in code comments

NitroFeed.sol specifies the following:

```
// Farming shares can be delivered to all parties as ERC20 tokens instantiates by proxy

// Delivery takes arrays of shares, so that a market maker can efficiently receive tokens

// across their mm activities.
```

// Anyone can initiate delivery of a share, it will always go to the originators of that share.

This comment should read instantiated by proxy, as the proxy entity instantiating the ERC20 tokens.

Resolution: Typographic errors found within code comments have been addressed appropriately.

Strings must be UTF-8 encoded

While strings are preferred to bytes32 (as strings do not have a practical size limit for the scope of this engagement), strings in Solidity must be UTF-8 encoded. This will limit domain characters which do not fall within this encoding (which while unlikely, are still possible). Feed registries should be explicitly aware of this limitation to avoid potential misconfiguration.

Resolution: The Nitro team provided the following: “Some URIs could be encoded incorrectly. This is addressed in two ways, [first] the backend filters URIs that are published but incorrectly formatted, so they will never propagate to an official Nitro client.

Second, the client side verification code tries to connect to URIs to do actually validate the feed and reconcile it with the published address. So in practice this is of limited concern. Changing to bytes32 allows anything to be used as a URI, but practically it also has to be validated to the same standards any http/https endpoint would be by the client or intermediate code.”

Bramah concurs with this assessment and feels these mitigations adequately address the concern.

Specific Recommendations

Unique to the Nitro Project Protocol

Public API functions are only accessible by owner

Despite being documented as public API functionality, these public functions can only be called by the designated Owner address.

Resolution: The Nitro team provided the following: “This is largely a documentation change, public or external functions that are owner only should be more explicit.”

Bramah concurs with this assessment.

Earlier swap routes are more likely to guarantee execution (non-atomic transactions)

The protocol takes advantage of numerous structured loop statements to iterate through existing instruments. As unbounded loops can have a theoretically infinite number of iterations (limited only by the length of the instruments array), later instruments in the loop are not necessarily guaranteed to execute (due to constraints involving gas).

While this can be limited by placing an upper bound on the number of instruments, we suggest evaluating potential measures by which each instrument is validated individually wherever possible, to prevent potential instances in which later participants are not guaranteed execution.

Since the transactions (liquidity sourcing, trading fee distribution, etc.) aren’t atomic, it is presently unclear what the intended behavior is if one or more steps can’t be completed (e.g., due to shortage of gas).

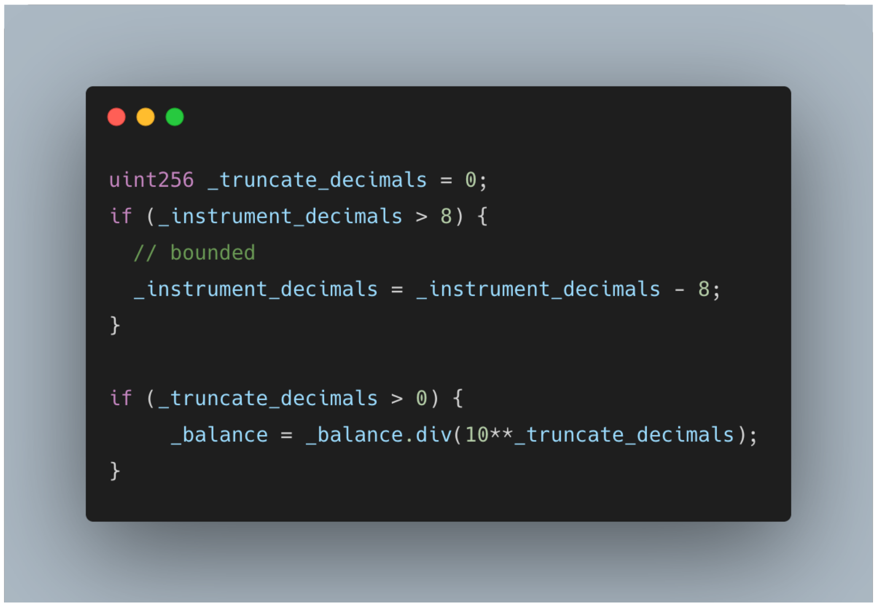
Resolution: The Nitro team provided the following: “This is a limitation in Ethereum itself, in that large or complicated swap routes may hit gas limits and fail to execute. Nitro is primarily designed to guarantee execution for Swap routes up to two hops. However, even that can be thwarted by exceptionally gas hungry token contracts. Even in a situation without iteration it is possible to hit these limits due to deployed code outside of the control of this project.

So, unfortunately we can only provide the facilities to enable multi-route or source swaps, but cannot guarantee they will execute. Best practice is to do a pre-execution check of every part of the swap path, as well as a gas estimate on the actual execution."

Bramah agrees with the sentiment expressed, and also notes that the structural limitations of Solidity are often a directly limiting factor on design choices.

Unreachable if statements

The function _calculate_proportional_share within NitroFeed.sol depends upon a calculation performed that is unreachable as it relies upon a value (_truncate_decimals) that is never reached (e.g. the value never exceeds zero).



```
uint256 _truncate_decimals = 0;
if (_instrument_decimals > 8) {
  // bounded
  _instrument_decimals = _instrument_decimals - 8;
}

if (_truncate_decimals > 0) {
     _balance = _balance.div(10**_truncate_decimals);
}
```

Resolution: The Nitro team provided the following:

"This is a bug in the _calculate_proportional_share function. It should truncate decimals when the decimalization of the instrument is greater than Nitros internal decimalization.

In the totalSupply function this is not an unreachable if statement. Most of the time the halving limit is not reached and this is executed, it covers partial tiers. So when the totalsupply of N2 falls between tiers this statement correctly calculates the difference. Similarly when the halving limit reaches 24 this is not executed (there is no partial tier as we have reached the hard limit).

While investigating this I noticed a loss of precision versus the elisp model for the totalSupply. This is the result of using floating point representations for the elisp based calculations, this results in a cumulative rounding error that isn't present (or rather is expressed differently in the Solidity execution). This loss of precision is less than 100 N2 units at the full halving, which seems reasonable and should not affect any aspect of the system so long as it is internally consistent."

Bramah has validated each of these statements independently and has confirmed each to be true. While dynamic analysis of the function was performed, our error in reference to totalSupply was a result of manual intervention.

Minimum waiting period can be reached but contract may not be added

The minimum waiting period must be exceeded, not simply reached when adding a contract, as current validation (**>**) is not inclusive of the minimum waiting period (**>=**).
Occurrences:

NitroNexus.sol, Line 40

Resolution: Nitro provided the following details: "Waiting period [now] accurately reflects the minimum instead of one less than the minimum." Bramah has validated this fix and believes it to be satisfactory.

Toolset Warnings

Unique to the Nitro Project

Overview

In addition to our manual review, our process involves utilizing static analysis and formal methods in order to perform additional verification of the presence of security vulnerabilities (or lack thereof). An additional part of this review phase consists of reviewing any automated unit testing frameworks that exist.

The following sections detail warnings generated by the automated tools and confirmation of false positives where applicable.

Compilation Warnings

No compilation errors are generated by the contract as of commit hash **0911240566b09ad04b64829c882479b890f81c5c**.

Test Coverage

The contract repository possesses substantial unit test coverage throughout. This testing traditionally provides a variety of unit tests which encompass the various operational stages of the contract, largely from the usage of a verbose test suite. This test suite includes a number of invariants throughout the test suite,

Static Analysis Coverage

The contract repository underwent heavy scrutiny with multiple static analysis agents, including:

- Securify
- MAIAN
- Mythril
- Oyente
- Slither

In each case, the team had either mitigated relevant concerns raised by each of these tools or provided adequate justification for the risk (such as adhering to the ERC-20 token standard).

Non-initialized return value

- File: contracts/NitroFeedRegistry.sol
    - Lines: 56-64
- File: contracts/NitroFeedRegistry.sol
    - Lines: 21-27
- File: contracts/NitroTokenRegistry.sol
    - Lines: 18-40
- File: contracts/NitroTokenRegistry.sol
    - Lines: 42-48
- File: contracts/NitroFeed.sol
    - Lines: 122-131
- File: contracts/mock/N2TokenFactoryMock.sol
    - Lines: 12-20

Local variable usage for gas efficiency within a loop:

- File: contracts/NitroSwap.sol
    - Lines: 197-212
- File: contracts/NitroTokenRegistry.sol
    - Lines: 35-39
- File: contracts/NitroTokenRegistry.sol
    - Lines: 29-33
- File: contracts/NitroFeed.sol
    - Lines: 281-300
- File: contracts/NitroFeed.sol
    - Lines: 315-324
- File: contracts/NitroFeed.sol
    - Lines: 128-130
- File: contracts/NitroFeed.sol
    - Lines: 102-104
- File: contracts/NitroFeed.sol
    - Lines: 232-265
- File: contracts/NitroNexus.sol
    - Lines: 78-80

Directory Structure

At time of review, the directory structure of the Nitro Project smart contracts repository appeared as it does below. Our review, at request of Nitro Project, covers the Solidity code (*.sol) as of commit hash **0911240566b09ad04b64829c882479b890f81c5c**.

```
├──── N2TokenFactory.sol
├──── NitroFeed.sol
├──── NitroFeedRegistry.sol
├──── NitroNexus.sol
├──── NitroSwap.sol
├──── NitroTokenRegistry.sol
├──── interfaces
│     ├──── INitro.sol
│     ├──── INitroFeed.sol
│     ├──── INitroFeedRegistry.sol
│     ├──── INitroNexus.sol
│     ├──── INitroRegistry.sol
│     └──── INitroTokenRegistry.sol
├──── mock
│     ├──── N2TokenFactoryMock.sol
│     ├──── NitroFeedClientMock.sol
│     ├──── NitroMock.sol
│     ├──── NitroNexusMock.sol
│     ├──── NitroSwapMock.sol
│     ├──── SimpleTokenMock.sol
│     ├──── SimpleTokenMock2.sol
│     ├──── SimpleTokenMock3.sol
│     └──── SimpleTokenMock4.sol
```

```
├── modified_zeppelin
│   ├── GSN
│   │   └── Context.sol
│   ├── access
│   │   └── Ownable.sol
│   ├── cryptography
│   │   └── ECDSA.sol
│   ├── math
│   │   └── SafeMath.sol
│   ├── token
│   │   └── ERC20
│   │       ├── ERC20.sol
│   │       ├── IERC20.sol
│   │       ├── N2ERC20.sol
│   │       └── SafeERC20.sol
│   └── utils
│       ├── Address.sol
│       ├── Pausable.sol
│       └── ReentrancyGuard.sol
├── nitro.org
├── shared
│   └── NitroVerifiedStruct.sol
└── spawner
    └── Spawner.sol

12 directories, 35 files
```