Neutrl Contracts Fix Review

Auditors

Kurt Barry, Lead Security Researcher

Report prepared by: Lucas Goiriz

September 11, 2025

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Neutrl is a market-neutral synthetic dollar designed to unlock untapped yield opportunities in OTC and altcoin markets. Neutrl leverages OTC arbitrage, funding rate inefficiencies, and DeFi-native market-neutral strategies to provide a single, high-yield access point for capital allocators.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of Neutrl Contracts Fix according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 2 days in total, Neutrl engaged with Spearbit to review the contracts protocol. In this period of time a total of **3** issues were found.

Summary

Project Name	Neutrl
Repository	contracts
Commit	c685d86d
Type of Project	Stablecoin, Yield
Audit Timeline	Sep 7th to Sep 9th

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	0	0	0
Gas Optimizations	1	1	0
Informational	2	1	1
Total	**3**	**2**	**1**

5 Findings

5.1 Gas Optimization

5.1.1 Structs Can Be Packed to Reduce Storage Operations

Severity: Gas Optimization

Context: AssetLock.sol#L9-L19

Description/Recommendation: The `AssetLock` contract defines the `UserLock` and `AssetInfo` structs as follows:

```
struct UserLock {
    uint256 amount;
    uint64 lockStartTime;
    uint64 lockEndTime;
}

struct AssetInfo {
    bool isSupported;
    uint256 maxLockCapacity;
    uint256 minLockAmount;
}
```

The use of smaller datatypes would allow packing the data of these structs into fewer storage slots, significantly decreasing the costs of writing and reading full structs. For example, if standardizing on `uint128` for token quantities, the structs can be changed to:

```
struct UserLock {
    uint128 amount;
    uint64 lockStartTime;
    uint64 lockEndTime;
}

struct AssetInfo {
    bool isSupported;
    uint128 maxLockCapacity;
    uint128 minLockAmount;
}
```

Reducing the number of slots used by each struct by one. `AssetInfo` could be further packed to use only a single slot:

```
struct AssetInfo {
    bool isSupported;
    uint120 maxLockCapacity;
    uint120 minLockAmount;
}
```

In the worst case, using a `uint120` gives a maximum value of order 10ˆ36, which for an 18 decimal token allows a whole token amount up to about 10ˆ18. This should be sufficient for all tokens in practice. Note that if this suggestion is implemented, safe (checked for overflow) casts should be used as a precaution to prevent any potential numerical exploits.

Neutrl: Fixed in commit 0dcfca2e.

Spearbit: Fix verified.

5.2 Informational

5.2.1 Refactor Lock Verification Logic Into a Single Function

Severity: Informational

Context: AssetLock.sol#L141-L154, AssetLock.sol#L182-L195

Description/Recommendation: Both `lockAsset()` and `lockAssetOnBehalf()` perform a significant amount of identical verification on the lock parameters. This code could be factored out into an `internal` function, reducing code duplication and making the code easier to maintain.

Neutrl: Fixed in commit f54f6b27.

Spearbit: Fix verified.

5.2.2 Consider Additional Getter Functions For User Lock Data

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description/Recommendation: Add the functions with the following behavior to improve on- and off-chain legibility of user locks:

- Fetch the total number of locks associated with a given (user, asset) pair.
- Fetch the lock data at a particular index of the lock array for a given (user, asset) pair.
- Fetch a range of indices of the lock array for a given (user, asset) pair (i.e. a paginated version of `getUserLocks()`). This will be especially useful in case a user accumulates a very large number of locks for some reason.

Neutrl: Acknowledged. We prefer to keep the API clean. The existing function gives us already what we need, and we mostly use offchain indexer that reproduce the UserLock positions, so even in case of very large numbers we should be good.

Spearbit: Acknowledged.