Veridise Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Shib Original Vision Locker

Veridise Inc.

May 24, 2023

| Prepared For:

Shib Original Vision

| Prepared By:

Benjamin Mariano
Xiangan He

| Contact Us: contact@veridise.com

| Version History:

May 19, 2023	Initial Draft
May 23, 2023	V1
May 24, 2023	V2

Contents

1 Executive Summary

From May 16, 2023 to May 19, 2023, Shib Original Vision engaged Veridise to review the security of the Shib Position-NFT Locker, a "locker" contract that is used to lock up Uniswap V3 LP tokens which still enable the LP fees to be extracted at any time. Veridise conducted the assessment over 2 person-week, with 2 engineers reviewing code over 1 week on commit N/A. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual auditing.

Code assessment. The Shib Original Vision developers provided the source code of the Shib Position-NFT Locker contracts for review. No other documentation was provided, nor were any test cases. However, the code was quite small, consisting of a single contract with just over 200 LOC.

Summary of issues detected. The audit uncovered 3 total issues. Most of the code consisted of interfaces and the actual contract logic consisted of no more than 50 lines. As a result of the simplicity of the protocol, minor issues were reported. Namely: a concern on the effect of a possible re-entrancy attack, and incorrect event emission on withdrawals.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

Project Dashboard 2

Table 2.1: Application Summary.

Name	Version	Type	Platform
Shib Position-NFT Locker	N/A	Solidity	Ethereum

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
May 16 - May 19, 2023	Manual & Tools	2	2 person-week

Table 2.3: Vulnerability Summary.

Name	Number	Resolved
Critical-Severity Issues	0	0
High-Severity Issues	0	0
Medium-Severity Issues	0	0
Low-Severity Issues	2	2
Warning-Severity Issues	1	1
Informational-Severity Issues	0	0
TOTAL	3	3

Table 2.4: Category Breakdown.

Name	Number
Reentrancy	1
Missing/Incorrect Events	1
Logic Error	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of Shib Original Vision's smart contracts.

In our audit, we sought to answer the following questions:

- Can the liquidity of a locked NFT be decreased?
- Can an NFT get stuck in the Locker contract?
- Can a user transfer a token while it is deposited in the locker (i.e., before it is withdrawn)?
- Can an NFT be withdrawn before the deadline has passed?
- Can an NFT that is held in the Locker contract be stolen from its original owner through the Locker (e.g., another user who is not the original owner withdraws the NFT)?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved a combination of human experts and automated program analysis & testing tools. In particular, we conducted our audit with the aid of the following technique:

- *Fuzzing/Property-based Testing.* We also leverage fuzz testing to determine if the protocol may deviate from the expected behavior. To do this, we formalize the desired behavior of the protocol as [V] specifications and then use our fuzzing framework OrCa to determine if a violation of the specification can be found.

Scope. The scope of this audit is limited to the `Locker.sol` contract provided by the Shib Original Vision developers, which contains the smart contract implementation of the Shib Position-NFT Locker.

Methodology. Veridise auditors inspected provided tests, and read the Shib Position-NFT Locker documentation. They then began a manual audit of the code assisted by tooling.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Vulnerability Report

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowleged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-SHB-VUL-001	Possible reentrancy concern	Low	Fixed
V-SHB-VUL-002	Bad event argument	Low	Fixed
V-SHB-VUL-003	Liquidity can update on locked position	Warning	Acknowledged

4.1 Detailed Description of Issues

4.1.1 V-SHB-VUL-001: Possible reentrancy concern

Severity	Low	**Commit**	N/A
Type	Reentrancy	**Status**	Fixed
File(s)	`Locker.sol`		
Location(s)	withdraw()		

After an NFT has been deposited and the desired funds can be extracted, the following call to `withdraw` can be used to withdraw a user's NFT:

```
function withdraw(uint256 tokenId) external {
    require(deposits[tokenId].unlockTime < block.timestamp, "Not allow");
    nonfungiblePositionManager.safeTransferFrom(address(this), deposits[tokenId].
    owner, tokenId);
    delete deposits[tokenId];
    emit Withdrawn(deposits[tokenId].owner, tokenId);
}
```

The call to `safeTransferFrom` can possibly reenter, which could allow a malicious user to manipulate the event log.

Impact At the time of writing, the worst exploits of this reentrancy the auditors conjured are calls to `deposit` in the reentrant call that could manipulate the event log or even result in a token getting stuck in the contract if `deposit` is called with the same `tokenId` of the token being withdrawn.

Recommendation Move the call to `safeTransferFrom` to the end of the function, such that all state has been appropriately updated before the possible reentrancy.

Developer Response The developers elected to add a `nonReentrant` modifier to all functions. Technically speaking, this could cause concerns if any outside applications depend on the public value of `deposits`, however, developers indicated that this is not a concern for them.

4.1.2 V-SHB-VUL-002: Bad event argument

Severity	Low	**Commit**	N/A
Type	Missing/Incorrect Eve	**Status**	Fixed
File(s)	`Locker.sol`		
Location(s)	withdraw()		

After a `withdraw`, an event is emitted as follows:

```
function withdraw(uint256 tokenId) external {
    require(deposits[tokenId].unlockTime < block.timestamp, "Not allow");
    nonfungiblePositionManager.safeTransferFrom(address(this), deposits[tokenId].
    owner, tokenId);
    delete deposits[tokenId];
    emit Withdrawn(deposits[tokenId].owner, tokenId);
}
```

The call to `deposits[tokenId].owner` will always be the 0 address because the entry is deleted on the previous line.

Impact All `Withdrawn` will have the wrong owner.

Recommendation Save the owner before deleting the entry or emit the event before the deletion.

4.1.3 V-SHB-VUL-003: Liquidity can update on locked position

Severity	Warning	**Commit**	N/A
Type	Logic Error	**Status**	Acknowledged
File(s)	`Locker.sol`		
Location(s)	N/A		

For Uniswap-V3 position NFTS, the following function can be invoked to increase the liquidity of a position:

```
function increaseLiquidity(IncreaseLiquidityParams calldata params)
        external
        payable
        override
        checkDeadline(params.deadline)
        returns (
            uint128 liquidity,
            uint256 amount0,
            uint256 amount1
        )
```

This function is not restricted to the owner of the NFT and therefore the liquidity in a position held by the `Locker` contract can increase

Impact Locked NFTs can increase liquidity.

Recommendation Clarify this as desired behavior or adjust to disallow liquidity increases.

Developer Response The developers are aware of this behavior and intend to keep it as is.

5 Fuzz Testing

5.1 Methodology

Our goal was to fuzz test the Shib Locker to assess its functional correctness i.e, whether the implementation deviates from the intended behavior. We used OrCa, our in-house fuzzer, to verify invariants – logical formulas that should hold after every transaction. For each invariant, we wrote a harness which executed a random sequence of relevant external calls and then asserted the invariant should hold after the calls. We prioritized invariants which had a higher security impact e.g, if violated would allow someone to steal funds. For all invariants, we ran OrCa for 10 minutes with 3 simulated users.

5.2 Properties Fuzzed

The following table describes the invariants we fuzz-tested. The first column states which contract the invarianted is associated with. The second describes the invariant informally in English and the last column notes whether we found a bug when fuzzing the invariant (✗ indicates no bug was found and ✗ means fuzzing this invariant revealed a bug). We ran OrCa for 10 minutes when fuzz-testing each invariant.

Table 5.1: Invariants Fuzzed.

Contract	Invariant	Bug
Locker	If NFT is deposited and never withdrawn, liquidity should never decrease.	✗
Locker	Withdrawing a token from the locker should remove the deposit.	✗
Locker	Transferring ownership should not be possible if no tokens have been deposited.	✗
Locker	Withdrawing should not be possible if no tokens have been deposited.	✗
Locker	It should not be possible to deposit the same token twice without withdrawing.	✗
Locker	NFT should only be able to be withdrawn if the specified deadline has passed.	✗
Locker	NFT owner by the Locker contract should always have a non-zero owner.	✗

As shown in the table above, no violations of the invariants above were found through fuzz testing. Additional manual checking of these invariants similarly found no violations.