LIDO FINANCE STETH PRICE ORACLE SMART CONTRACT AUDIT

May 14, 2021

MixBytes()

CONTENTS

1. INTRODUCTION

1.1 DISCLAIMER

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Lido Finance. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 PROJECT OVERVIEW

The stETH price oracle is a set of smart contracts that allows you to store and update the stETH token price. The main feature of the project is the ability to extract variables for calculating the price from the block hash through the Merkle Patricia Proof Verifier. The stETH price oracle uses the well-known Curve stablecoin mechanic to calculate the price between tokens of the same value. In the new version of oracle, developers use the RLP library to optimize computation and reduce gas costs to update oracle state.

1.3 SECURITY ASSESSMENT METHODOLOGY

At least 2 auditors are involved in the work on the audit who check the provided source code independently of each other in accordance with the methodology described below:

01 "Blind" audit includes:

> Manual code study

> "Reverse" research and study of the architecture of the code based on the source code only

Stage goal:
Building an independent view of the project's architecture
Finding logical flaws

02 Checking the code against the checklist of known vulnerabilities includes:

> Manual code check for vulnerabilities from the company's internal checklist

> The company's checklist is constantly updated based on the analysis of hacks, research and audit of the clients' code

Stage goal:
Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flashloan attacks, etc.)

03 Checking the logic, architecture of the security model for compliance with the desired model, which includes:

> Detailed study of the project documentation

> Examining contracts tests

> Examining comments in code

> Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit

Stage goal:
Detection of inconsistencies with the desired model

04 Consolidation of the reports from all auditors into one common interim report document

> Cross check: each auditor reviews the reports of the others

> Discussion of the found issues by the auditors

> Formation of a general (merged) report

Stage goal:
Re-check all the problems for relevance and correctness of the threat level
Provide the client with an interim report

05 Bug fixing & re-check.

> Client fixes or comments on every issue

> Upon completion of the bug fixing, the auditors double-check each fix and set the statuses with a link to the fix

Stage goal:
Preparation of the final code version with all the fixes

06 Preparation of the final audit report and delivery to the customer.

Findings discovered during the audit are classified as follows:

FINDINGS SEVERITY BREAKDOWN

Level	Description	Required action
Critical	Bugs leading to assets theft, fund access locking, or any other loss funds to be transferred to any party	Immediate action to fix issue
Major	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.	Implement fix as soon as possible
Warning	Bugs that can break the intended contract logic or expose it to DoS attacks	Take into consideration and implement fix in certain period
Comment	Other issues and recommendations reported to/acknowledged by the team	Take into consideration

Based on the feedback received from the Customer's team regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The project team is aware of this finding. Recommendations for this finding are planned to be resolved in the future. This finding does not affect the overall safety of the project.
No issue	Finding does not affect the overall safety of the project and does not violate the logic of its work.

1.4 EXECUTIVE SUMMARY

A trustless oracle for the ETH/stETH Curve pool using Merkle Patricia proofs of Ethereum state.The oracle currently assumes that the pool's fee and A (amplification coefficient) values don't change between the time of proof generation and submission.This audit included an external contract https://github.com/hamdiallam/Solidity-RLP/blob/4fa53119e6dd7c4a950586e21b6068cd9520a649/contracts/RLPReader.sol, representing a solidity library for Ethereum's RLP decoding.

1.5 PROJECT DASHBOARD

Client	Lido Finance
Audit name	stETH Price Oracle
Initial version	ae093b308999a564ed3f23d52c6c5dce946dbfa7 4fa53119e6dd7c4a950586e21b6068cd9520a649
Final version	1033b3e84142317ffd8f366b52e489d5eb49c73f a2837797e4da79070701339947f32f5725e08b56
SLOC	720
Date	2021-04-26 - 2021-05-14
Auditors engaged	2 auditors

FILES LISTING

StateProofVerifier.sol	StateProofVerifier.sol
StableSwapStateOracle.sol	StableSwapStateOracle.sol
StableSwapPriceHelper.vy	StableSwapPriceHelper.vy
MerklePatriciaProofVerifier.sol	MerklePatriciaProofVe...
RLPReader.sol	RLPReader.sol

FINDINGS SUMMARY

Level	Amount
Critical	0
Major	0
Warning	2
Comment	5

CONCLUSION

Smart contract has been audited and several suspicious places were found. During audit no critical and major issues were identified. Several issues were marked as warnings and comments. After working on audit report all issues were fixed or acknowledged(if the issue is not critical or major) by client, so contracts assumed as secure to use according our security criteria. Pursuant to findings severity we also assume an initial commit `ae093b308999a564ed3f23d52c6c5dce946dbfa7` as secure. Final commit identifiers with all fixes: `1033b3e84142317ffd8f366b52e489d5eb49c73f`, `a2837797e4da79070701339947f32f5725e08b56`

2.FINDINGS REPORT

2.1 CRITICAL

Not Found

2.2 MAJOR

Not Found

2.3 WARNING

WRN-1	Block header incorrect input
File	StateProofVerifier.sol
Severity	Warning
Status	Fixed at **3d01ffac**

DESCRIPTION

In the function for extracting data from block header tx can fail without any information in case of incorrect input:
StateProofVerifier.sol#L65

RECOMMENDATION

We recommend to add following check:

```
require(headerFields.length > 11, "INCORRECT_HEADER");
```

CLIENT'S COMMENTARY

Solidity already provides array bounds checking so there is no way to supply an incorrect header in a way that would break the intended contract behavior, which is to fail the transaction in the case of any incorrect input.
That said, it's a good idea to add an explicit require statement as advised, this would make debugging failed reports easier.

WRN-2	Possible zero price for token
File	StableSwapStateOracle.sol
Severity	Warning
Status	Acknowledged

DESCRIPTION

In the new version of the oracle smart contract, the price for stETH = 0 until user calls the `submitState` function:
StableSwapStateOracle.sol#L268

RECOMMENDATION

We recommend to add following check:

```
require(stethPrice > 0, "PRICE_NOT_INITIALIZED");
```

CLIENT'S COMMENTARY

The contract for this function doesn't assume it fails when the price hasn't been reported yet: the function returns, among other values, the timestamp of the returned data, and this timestamp would be zero in the case the price is not set.

2.4 COMMENTS

CMT-1	Require without message
File	StateProofVerifier.sol StableSwapStateOracle.sol
Severity	Comment
Status	No issue

DESCRIPTION

In the following functions if revert occurs then user won't receive any information:
StateProofVerifier.sol#L100
StableSwapStateOracle.sol#L466
StableSwapStateOracle.sol#L474

RECOMMENDATION

We recommend to add message to require.

CLIENT'S COMMENTARY

We're not using messages in require calls to optimize the deployed bytecode size. The source code of the contract is verified on Etherscan so the exact location of any revert in a failed mainnet transaction can be inspected using free tools like Tenderly.

CMT-2	Bad comment for hash generation
File	StableSwapStateOracle.sol
Severity	Comment
Status	**Fixed** at **1fb349c0**

DESCRIPTION

It is impossible to check constant hash via following comment:
StableSwapStateOracle.sol#L95

RECOMMENDATION

We recommend to add more detailed comment.

CLIENT'S COMMENTARY

The comment was changed to use actual Solidity code so one can check the value easier.

CMT-3	Gas saving in price calculation
File	StableSwapPriceHelper.vy
Severity	Comment
Status	Acknowledged

DESCRIPTION

In the function for calculating price, you can save gas when calculating the variables `S_` and `c` in case if `i=1`, `j=0`, `N_COINS=2`: StableSwapPriceHelper.vy#L63-L72

RECOMMENDATION

We recommendto calculate variables using following formula:

```
S_ = x
c = D * D / (x * N_COINS)
```

CLIENT'S COMMENTARY

We intentionally made as few modifications to the original Curve pool code as possible:

- To avoid unintentionally introducing any behavior differences.
- To make it as easy as possible for someone to manually check that the code does exactly the same calculations over the pool state as the original pool contract's code.

CMT-4	Gas saving when copying memory
File	RLPReader.sol
Severity	Comment
Status	Fixed at a2837797, 1033b3e8

DESCRIPTION

In the function for copy memory when `len % WORD_SIZE == 0` it is possible to save some gas by adding simple check:
RLPReader.sol#L350-L355

RECOMMENDATION

We recommend to add following check:

```
if (len > 0)
{
    uint mask = 256 ** (WORD_SIZE - len) - 1;
    assembly {
        let srcpart := and(mload(src), not(mask)) // zero out src
        let destpart := and(mload(dest), mask) // retrieve the bytes
        mstore(dest, or(destpart, srcpart))
    }
}
```

CLIENT'S COMMENTARY

We've passed the comment to the library's author and will connect you with him shortly.

CMT-5	Unused variable
File	MerklePatriciaProofVerifier.sol
Severity	Comment
Status	**Fixed** at **63cbc0e5**

DESCRIPTION

Following smart contract contains unused variable:
MerklePatriciaProofVerifier.sol#L37

RECOMMENDATION

We recommend to remove unused variable.

CLIENT'S COMMENTARY

Fixed.

3.ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build open-source solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

BLOCKCHAINS

Ethereum

Cosmos

EOS

Substrate

TECH STACK

Python

Solidity

Rust

C++

CONTACTS

https://github.com/mixbytes/audits_public

https://mixbytes.io/

hello@mixbytes.io

https://t.me/MixBytes

https://twitter.com/mixbytes