Lido on Kusama/Polkadot Liquid Staking Delta Audit

Smart Contract Security Assessment

September 27, 2022

MixBytes()

ABSTRACT

Dedaub was commissioned to perform a security audit on Lido on Kusama/Polkadot liquid staking contracts by MixBytes.

This report focuses exclusively on the recent changes (on multiple contracts) in the protocol. The scope of the audit included the most recent updates and fixes to the contracts in the repository Lido KSM/DOT liquid staking, from the commit `07da59e89e023be7d1bc56dc98bd6a3e7a6d85eb` of the previous audit, up to commit `4d8860a2c08aa3124b38124866b237c027cc8acc`.

The audit focussed solely on the delta between these versions, the auditors did not re-audit the whole protocol. Although in terms of lines of code the delta is substantial, all of the changes are relatively straightforward in nature. The changes fix specific functionality issues of the previous version and, for all major ones, they are accompanied by corresponding tests. The auditors found that the changes properly address the corresponding issues and do not introduce vulnerabilities.

As a consequence, this report is light, only containing a few minor suggestions that we found worth mentioning.

SETTING & CAVEATS

Our earlier audits describe the setting and caveats for Lido on Kusama/Polkadot protocol. As a general warning, we note that an audit of small changes in a large protocol is necessarily out-of-context. We made a best-effort attempt to understand the changed lines of code and assess whether these changes are reasonable and do not introduce vulnerabilities. The audit, however, was restricted to the modified lines, and their interaction with the rest of the protocol is not always easy to assess.

The audit's main target is security threats, i.e., what the community understanding would likely call "hacking", rather than the regular use of the protocol. Functional correctness (i.e., issues in "regular use") is a secondary consideration. Typically it can only be covered if we are provided with unambiguous (i.e., full-detail) specifications of what is the expected, correct behaviour. In terms of functional correctness, we often trusted the code's calculations and interactions, in the absence of any other specification. Functional correctness relative to low-level calculations (including units, scaling and quantities returned from external protocols) is generally most effectively done through thorough testing rather than human auditing.

VULNERABILITIES & FUNCTIONAL ISSUES

This section details issues affecting the functionality of the contract. Dedaub generally categorizes issues according to the following severities, but may also take other considerations into account such as impact or difficulty in exploitation:

Category	Description
CRITICAL	Can be profitably exploited by any knowledgeable third-party attacker to drain a portion of the system's or users' funds OR the contract does not function as intended and severe loss of funds may result.
HIGH	Third-party attackers or faulty functionality may block the system or cause the system or users to lose funds. Important system invariants can be violated.
MEDIUM	Examples: 1) User or system funds can be lost when third-party systems misbehave. 2) DoS, under specific conditions. 3) Part of the functionality becomes unusable due to a programming error.
LOW	Examples: 1) Breaking important system invariants but without apparent consequences. 2) Buggy functionality for trusted users where a workaround exists. 3) Security issues which may manifest when the system evolves.

Issue resolution includes "dismissed" or "acknowledged" but no action taken, by the client, or "resolved", per the auditors.

CRITICAL SEVERITY:

[No critical severity issues]

HIGH SEVERITY:

[No high severity issues]

MEDIUM SEVERITY:

[No medium severity issues]

LOW SEVERITY:

ID	Description	STATUS
L1	Missing Reentrancy Guards	**ACKNOWLEDGED**

The publicly callable functions of the `Lido` contract have no reentrancy guards. Although we have not identified any possible points of reentrancy, such points could be easily introduced in future versions of the code. As a consequence, we suggest, when possible, moving any interactions with external contracts after all effects have been applied (or, alternatively, adding reentrancy guards).

For instance, `Lido::deposit` calls `VKSM.transferFrom` before updating its internal state:

```
function deposit(uint256 _amount) external whenNotPaused returns
(uint256) {
    require(fundRaisedBalance + _amount < depositCap, "LIDO:
DEPOSITS_EXCEED_CAP");

    VKSM.transferFrom(msg.sender, address(this), _amount);

    // ...

    fundRaisedBalance += _amount;
    bufferedDeposits += _amount;
    _mintShares(msg.sender, shares);
```

Although `VKSM.transferFrom` does not currently transfer the control to the adversary, it could be conceivable in the future to introduce ERC777-like hooks or similar functionality that notifies the sender in case of a transfer. If the adversary were able to reenter into `Lido::deposit` from `VKSM.transferFrom`, he could employ such nested calls to effectively bypass the `depositCap` limit. Simply making the transfer after updating the state prevents any such issues.

OTHER/ ADVISORY ISSUES:

This section details issues that are not thought to directly affect the functionality of the project, but we recommend addressing them.

<table>
<tr><th>ID</th><th>Description</th><th>STATUS</th></tr>
<tr><td>A1</td><td>Optimization suggestions</td><td><b>INFO</b></td></tr>
<tr><td colspan="3">In the <code>RelayEncoder.sol</code> contract the function <code>encode_withdraw_unbonded()</code> uses several arithmetic operations with numbers that can be expressed as powers of 2. Thus, the multiplications and the divisions can be replaced with bitwise operations for more efficiency and maintainability.
Furthermore, in <code>Encoding.sol::scaleCompactUint:45</code> the <code>0xFF</code> can be removed since the <code>uint8()</code> casting will give the same result even without the AND operation.</td></tr>
<tr><td>A2</td><td>Tests for minor changes</td><td><b>ACKNOWLEDGED</b></td></tr>
<tr><td colspan="3">The auditors appreciated the inclusion of tests for all major changes. It would be beneficial to include tests also for smaller changes that seem to be missing (for instance we could not find a test for the case <code>totalXcKSMPoolShares == 0</code> and <code>totalVirtualXcKSMAmount != 0</code>). Although this check is minor, the fact that it was missing in the previous version makes it worthy of a test.</td></tr>
<tr><td>A3</td><td>Compiler known issues</td><td><b>INFO</b></td></tr>
<tr><td colspan="3">The code is compiled with Solidity 0.8.0 or higher. For deployment, we recommend no floating pragmas, i.e., a specific version, to be confident about the baseline guarantees</td></tr>
</table>

offered by the compiler. Version 0.8.0, in particular, has some known bugs, which we do not believe affect the correctness of the contracts.

DISCLAIMER

The audited contracts have been analyzed using automated techniques and extensive human inspection in accordance with state-of-the-art practices as of the date of this report. The audit makes no statements or warranties on the security of the code. On its own, it cannot be considered a sufficient assessment of the correctness of the contract. While we have conducted an analysis to the best of our ability, it is our recommendation for high-value contracts to commission several independent audits, a public bug bounty program, as well as continuous security auditing and monitoring through Dedaub Watchdog.

ABOUT DEDAUB

Dedaub offers significant security expertise combined with cutting-edge program analysis technology to secure some of the most prominent protocols in DeFi. The founders, as well as many of Dedaub's auditors, have a strong academic research background together with a real-world hacker mentality to secure code. Protocol blockchain developers hire us for our foundational analysis tools and deep expertise in program analysis, reverse engineering, DeFi exploits, cryptography and financial mathematics.