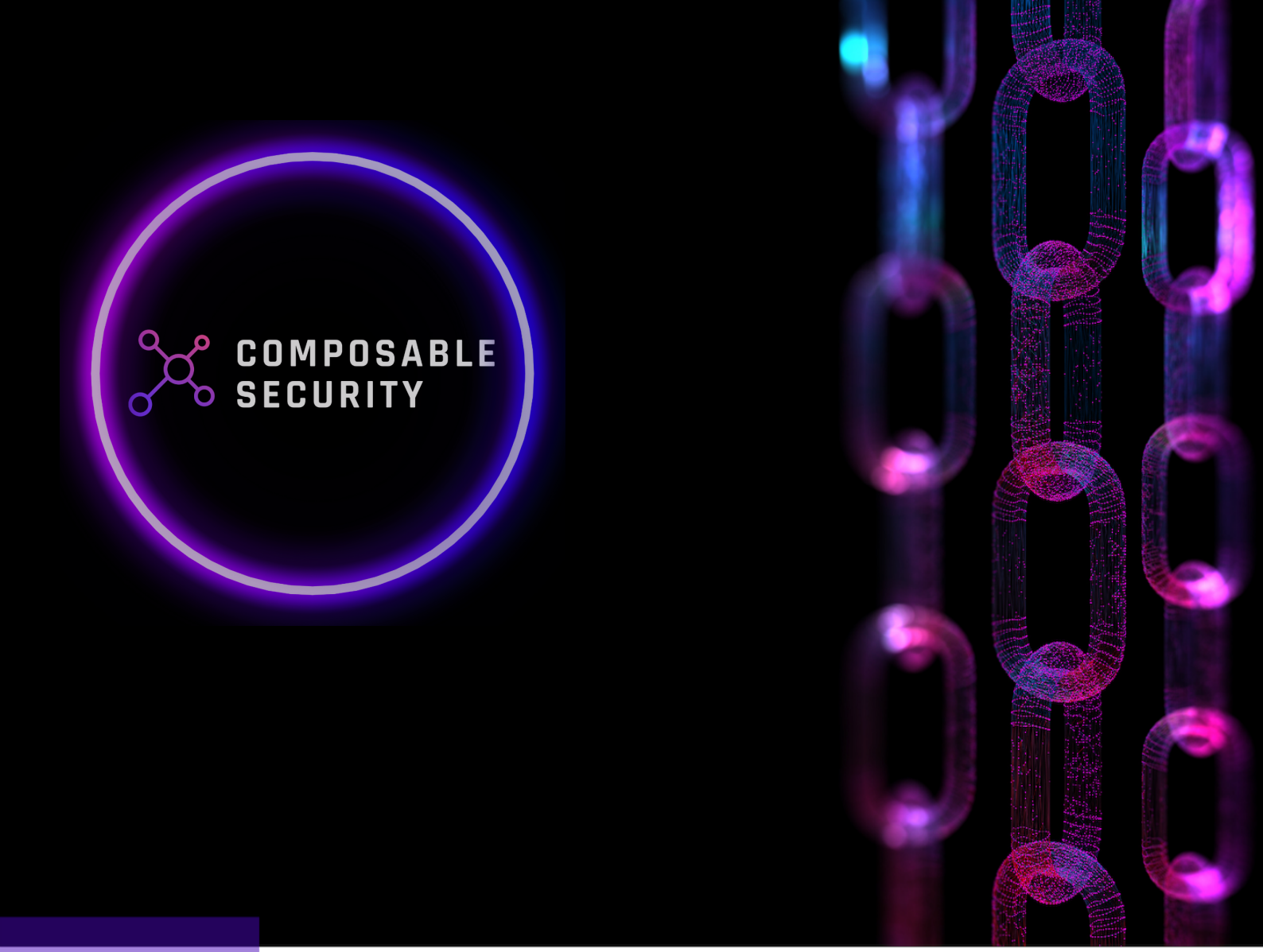


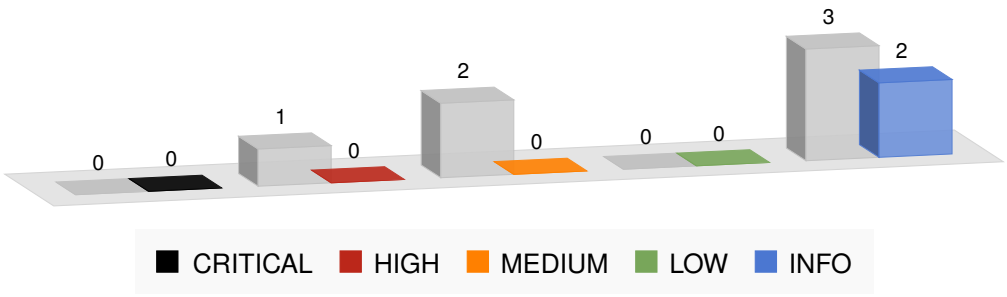
# PRELIMINARY REPORT

## Smart contract security review for Lido

Prepared by: Composable Security
Report ID: LDO-7582f002
Test time period: 2025-09-29 - 2025-10-09
Retest time period: 2025-10-09 - 2025-10-28
Report date: 2025-10-28
Version: 1.1
Visit: composable-security.com

# Contents

# 1. Retest summary (2025-10-28)




The description of the current status for each retested vulnerability and recommendation has been added in its section.

## 1.1. Results

The **Composable Security** team participated in verification of whether the vulnerabilities detected during the tests (between 2025-09-29 and 2025-10-09) were correctly removed and no longer appear in the code. The retest review also included the verification of the fix for the vulnerability submitted by the Lido team (LDO-7582f002-M02).

The current status of detected issues is as follows:

- The **high** vulnerability has been fixed.
- **Both** vulnerabilities with **medium** impact on risk have been fixed.
- **Three** security **recommendations** were handled as follows:
  - 1 has been implemented,
  - 2 have been acknowledged.
- The retest commit also includes a fix for a different issue regarding the dedicated gateway that has been addressed in a separate report.

## 1.2. Scope

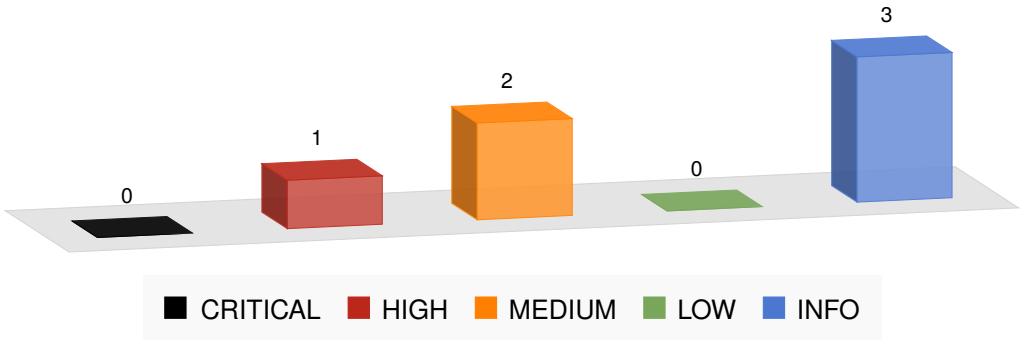
The retest scope included the changes, on a different commit in the same repository.

**GitHub repository:** https://github.com/lidofinance/lido-oracle/
**CommitID:** 0095fbe05955cd8832840c899c373711ebd2a7d3

# 2. Current findings status

| ID | Severity | Vulnerability | Status |
|---|---|---|---|
| LDO-7582f002-H01 | HIGH | Undercollateralized mint of stETH | FIXED |
| LDO-7582f002-M01 | MEDIUM | Invalid total value for predeposited validators | FIXED |
| LDO-7582f002-M02 | MEDIUM | Invalid CID for large files | FIXED |
| **ID** | **Severity** | **Recommendation** | **Status** |
| LDO-7582f002-R01 | INFO | Remove deprecated code | IMPLEMENTED |
| LDO-7582f002-R02 | INFO | Use expiration time for auth tokens | ACKNOWLEDGED |
| LDO-7582f002-R03 | INFO | Setting IPFS parameters explicitly | ACKNOWLEDGED |

# 3. Security review summary (2025-10-09)




## 3.1. Client project

Lido is the largest decentralized staking protocol that allows users to stake ETH and receive liquid staking derivative in return. The project successfully manages over $39,222,232,625 TVL supporting Ethereum growth and security since 2020.

Upcoming update to **Lido V3 introduces Staking Vaults (stVaults)** as a modular extension to the Lido protocol. These vaults allow stakers to delegate ETH to specific node operators (or DVT clusters) while still minting stETH with a reserve ratio. This design supports specialized staking setups yet preserves stETH liquidity, stability, and fungibility via overcollateralization.

Lido V3 includes the following key modifications:

- **Core protocol upgrade:** Extends support for minting stETH backed by external ETH holdings (i.e., ETH held in specialized vault contracts rather than the main Lido pool only), enabled EIP-7002 driven triggerable withdrawals support, still on with 0x01 validators within the Lido Core pool but 0x02 for stVaults.
- **Staking Vaults (stVaults):** New contracts that represent isolated vaults tied to a single operator or a DVT cluster, offering tailored staking arrangements. A central VaultHub contract coordinates these vaults, ensuring each remains sufficiently collateralized and healthy.
- **Off-Chain oracle enhancements:** Expands the existing oracle system to handle stVaults, reporting vault-specific data and managing updates asynchronously (so called Lazy Oracle for stVaults).
- **Node Operator Predeposit Guarantee:** Mitigates deposit frontrunning by requiring node operators to pre-commit deposit data on-chain, including on-chain BLS signature checks.

In this engagement, the Composable Security team focused on **the second iteration of Oracle upgrade for Lido V3 Staking Vaults**. Oracle is a core component of the Lido staking

ecosystem, responsible for aggregating, verifying, and relaying critical operational data that support the protocol's decentralized staking mechanism.

Below is an overview of its key components:

- **Accounting Oracle:** Tracks staking rewards, fees, and related financial metrics, ensuring the protocol's economics remain consistent and transparent.
- **Validators Exit Bus Oracle (VEBO):** Orchestrates exits of Lido validators when funds are needed to satisfy user withdrawals, generating exit requests and ensuring validator status changes are accurately reflected.
- **CSM Oracle:** Interfaces with the Community Staking Module (CSM) to report module-specific metrics—primarily the rewards produced by CSM validators.

The V3 adds off-chain oracle enhancements to include stVaults:

- **Vault-specific data reporting:** Each vault maintains its own state (e.g., validator balances, slashing reserve, accrued per-vault Lido fees) that the oracle must collect and submit on-chain. A modified `AccountingOracle` contract (and related modules) ingests vault-specific data as a Merkle root. On-chain logic then updates each vault's ETH balance, locked reserve, and related parameters lazily on demand.
- **Asynchronous update flow:** Different vaults may require updates at different times.
- **Integrity and consistency:** The sum of all vault balances plus the core pool balance must not exceed the total ETH managed by Lido.
- **Fee accounting:** The oracle upgrades must ensure that fees within each vault are recorded correctly and routed to the Lido DAO treasury.

This is a significant architectural change, as it impacts the core logic of the protocol - requiring calculations to consider not just the main Lido pool, but also isolated vaults operating in parallel.

The objectives of this project were established as follows:

- Confirm that partial or delayed updates do not break the system, and that cross-vault consistency is maintained.
- Validate that the updated oracle logic accurately calculates each vault fees, ensure no vault can bypass fees, and prevents double counting.
- Ensure that fees within each vault are accounted for correctly and routed to the Lido DAO treasury.

## 3.2. Results

The **Lido** engaged Composable Security to review security of **the second iteration for Oracle upgrade for Lido V3 Staking Vaults**. Composable Security conducted this assessment over 2 person-week with 2 engineers.

The summary of findings is as follows:

- no **critical** risk impact vulnerabilities were identified.
- 1 vulnerability with a **high** impact on risk was identified. Its potential consequence is:
  - Undercollateralized mint of stETH (LDO-7582f002-H01).
- 2 vulnerabilities with a **medium** impact on risk was identified.
- no vulnerabilities with a **low** impact on risk were identified.
- 3 **recommendations** have been proposed that can improve overall security, code clarity and help implement best practice.
- The team was very engaged and communication was excellent, including ongoing vulnerability triaging and solution consultations.
- The vulnerability LDO-7582f002-M02 has been detected and submitted by the Lido team.

Composable Security recommends that **Lido** complete the following:

- Address all reported issues.
- Extend unit tests with scenarios that cover detected vulnerabilities where possible.
- Consider whether the detected vulnerabilities may exist in other places (or ongoing projects) that have not been detected during engagement.
- Review dependencies and upgrade to the latest versions.

## 3.3. Scope

The scope includes the changes added to the Lido Oracle after latest audits of on-chain and off-chain components

The subjects of the test were changes from previously audited commit *8ad1b432943f0b589852d06b55f282327d240a5a* to the current implementation.

**GitHub repository:**
https://github.com/lidofinance/lido-oracle/
**CommitID:** 7582f00206f83f679312fe20060495a6e15857ee

The detailed scope of tests can be found in Agreed scope of tests.

# 4. Project details

## 4.1. Projects goal

The Composable Security team focused during this audit on the following:

- Perform a tailored threat analysis.
- Ensure that smart contract code is written according to security best practices.
- Identify security issues and potential threats both for **Lido** and their users.
- The secondary goal is to improve code clarity and optimize the code where possible.

## 4.2. Agreed scope of tests

The subjects of the test were selected files from the **Lido** repository.

**GitHub repository:**
https://github.com/lidofinance/lido-oracle/
**CommitID:** 7582f00206f83f679312fe20060495a6e15857ee

Files in scope:

```
├── __init__.py
├── constants.py
├── main.py
├── metrics/*.py
├── modules/*.py
├── providers/*.py
├── services/*.py
├── types.py
├── utils/*.py
├── variables.py
└── web3py/*.py
```

**Documentation:**

- Audit Scope
- Lido Staking Vaults (stVaults): Technical Design and Architecture

## 4.3. Threat analysis

This section summarizes the potential threats that were identified during the initial threat modeling performed prior to the audit. The tests were focused, but not limited to, finding security issues that could be exploited to achieve these threats.

Key assets that require protection:

- Report's submitted data, including:
  - stETH/ETH ratio components,
  - Vault total values,
  - Vault fees,
  - Predeposits,
  - Vault locked values (including slashing reserve),
- Protocols availability.

Threats and potential attackers goals:

- Invalid retrieval and manipulation of vaults' balances.
- Invalid calculation of locked reserves.
- Including the balances of a vault multiple times in the total balance.
- Manipulation of fees between vaults.
- Unauthorized forcing of vault rebalancing.
- Manipulation of the stETH/ETH rate to steal ETH.
- Withdrawal of locked ETH from vault.
- Bypassing business logic limits (e.g. the withdrawal request delay).
- Denial of Service (e.g. due to Oracle malfunction).

Potential vulnerable and attack scenarios to achieve the indicated attacker's goals:

- Retrieving invalid stages for new validators.
- Including the predeposit in the total value twice.
- Missing inclusion of pending deposits or externally deposited ETH.
- Applying vault data different that submitted in the report.
- Invalid retrieval and manipulation of vault balances.
- Invalid calculation of locked reserves.
- Including the balances of a vault multiple times in the total balance.
- Excluding validators or pending deposits from the total ETH balance calculation.
- Withdrawing ETH marked as locked.
- Disconnection of vault with locked ETH.
- Using deprecated or too fresh vault data when building the report.
- Front-running deposits to preset withdrawal credentials.
- Retrieving on-chain data from incorrect block via archive node.
- Intentional exit of validator by its operator to avoid fees.
- Missing events when calculating liquidity fees.
- Take advantage of arithmetic errors.
- Influence or bypass the business logic of the system.
- Inconsistency with documentation.
- Design issues.

- Uncaught exceptions.

## 4.4. Testing methodology

Security review was performed using the following methods:

- Q&A sessions with the **Lido** development team to thoroughly understand intentions and assumptions of the project.
- Initial threat modeling to identify key areas and focus on covering the most relevant scenarios based on real threats.
- Automatic tests.
- **Manual review of the code.**

## 4.5. Disclaimer

Smart contract security review **IS NOT A SECURITY WARRANTY**.

During the tests, the Composable Security team makes every effort to detect any occurring problems and help address them. However, it is not allowed to treat the report as a security certificate and assume that the project does not contain any vulnerabilities. Securing smart contract platforms is a multi-stage process, starting from threat modeling, through development based on best practices, security reviews and formal verification, ending with constant monitoring and incident response.

***Therefore, we encourage the implementation of security mechanisms at all stages of development and maintenance.***

# 5. Vulnerabilities

## [LDO-7582f002-H01] Undercollateralized mint of stETH

HIGH FIXED

**Retest (2025-10-10)**

The vulnerability has been fixed. Currently, the total value does not include the whole amount pre-deposited. In contrast, all new validators are checked for their pre-deposit stage and if it is PREDEPOSITED, the pre-deposit amount is added to the total value. Therefore, new validators that were deposited externally (not using pre-deposit flow) are not included until the validator is eligible for activation.

### Affected files

- staking_vaults.py#L103-L104

### Description

The vault's total balance calculation begins by retrieving its aggregated balance, which includes predeposited ETH. The Oracle then adds the validator's balance and any pending deposits if the validator meets the criteria for activation.

For a validator to be ready for activation, it must possess at least 32 ETH (from both balance and pending deposits) and have at least one pending deposit of at least 31 ETH. A malicious Node Operator can exploit this to have the Oracle count the predeposited ETH twice, allowing them to mint stETH based on this inflated amount.

### Attack scenario

Attackers could execute the following steps:

① The Node Operator predeposits a validator just before `refBlock` and immediately deposits 31 ETH via the `DEPOSIT_CONTRACT`.
② Another 31 ETH is staged in StakingVault, resulting in a total expenditure of 63 ETH (1 ETH predeposit + 31 ETH deposit + 31 ETH staged).
③ Assume no additional ETH is in the StakingVault and the Node Operator has only this one validator.
④ A new report is generated before the validator is confirmed. The total value calculated is: 0 (available) + 31 (staged) + 1 (predeposit) + 32 (validator ready) = 64 ETH.

⑤ Consequently, the Node Operator can mint stETH against 64 ETH, despite having only spent 63 ETH.

**Result:** The attack results in the minting of undercollateralized stETH, yielding an additional 1 ETH for every 63 ETH spent.

**Recommendation**

Exclude the predeposited ETH for validators that are ready for activation by tracking which validators have active predeposits and deducting that amount from their total balance.

### References

1. SCSVS G4: Business logic

## [LDO-7582f002-M01] Invalid total value for predeposited validators

MEDIUM FIXED

**Retest (2025-10-28)**

The vulnerability has been fixed as recommended.

### Affected files

- lazy_oracle.py#L131
- staking_vaults.py#L105
- staking_vaults.py#L119

### Description

The `get_validator_stages` function gets as parameter the public keys of validators and checks their status in the predeposit flow. It creates a dictionary where each public key serves as the key and its corresponding stage as the value.

Currently, the dict key is obtained using the `str(pk)` expression, where `pk` is of `HexBytes` type. This conversion results in a binary format representation (e.g., `b'\x86-S\xd9\xe413t\xd2'`) instead of the expected hexadecimal format prefixed with `0x`.

When querying the resulting dictionary, the search uses the `0x` prefixed hexadecimal format. Consequently, no validators are found, resulting in the `NONE` stage being returned for all checks.

```
inactive_validator_stages = self._get_non_activated_validator_stages(validators,
    vaults, block_identifier) # this is the dict returned by
   get_validator_stages function
(...)
validator_pubkey = validator.pubkey.to_0x_hex()
(...)
stage = inactive_validator_stages.get(validator_pubkey, ValidatorStage.NONE)
```

### Vulnerable scenario

The following sequence of actions leads to the identified problem:

① A Node Operator predeposits the first validator using withdrawal credentials associated with the vault.
② The Oracle computes the total value for the vault, which should amount to 1 ETH due to the validator being in the PREDEPOSITED stage.
③ The `get_validator_stages` function accurately identifies the PREDEPOSITED stage for the validator, but records it under the incorrect binary format key.
④ The Oracle then attempts to retrieve the `0x` prefixed hex public key from the dictionary, fails to find it, and defaults to the NONE stage.
⑤ As a result, the Oracle reports a total value of 0 ETH instead of the correct 1 ETH.

**Result:** The total value does not account for predeposited ETH.

**Recommendation**

Modify the key conversion from `str(pk)` to `pk.to_0x_hex()`.

### References

1. SCSVS G4: Business Logic

## [LDO-7582f002-M02] Invalid CID for large files

MEDIUM FIXED

**Retest (2025-10-28)**

The vulnerability has been removed. Now, the CID generation first splits the content into 256kB chunks and generates a CAR with multiple blocks.
Additionally, the testing suite has been extended to cover both small and large files and in case there are different CIDs returned by providers, firstly the Oracle searches

for the CID which was returned by the majority of providers. If there is no majority, the priority list of providers is taken into account and the CID returned by the first successful provider is selected, starting with Storacha.

## Affected files

- converter.py#L86-L135

## Description

The `CARConverter` class incorrectly generates a CID for files larger than 256kB. It does not split the file contents into chunks, leading to a different CID than other providers (e.g. Pinata).

Consequently, the CID generated for the same file is different depending on the provider and cannot be queried, leading to block of report generation.

**Note:** The vulnerability has been detected and shared by the Lido team.

## Vulnerable scenario

The following sequence of actions leads to the identified problem:

① The Oracle creates new report.
② The Oracle uses some provider to upload report data (e.g. pinata) and get CID.
③ The Oracle validates the CID from the provider against the CID generated by the oracle.
④ The CID generated by the Oracle is different and an error is raised.

**Result:** Denial of Service via inability to generate the report.

**Recommendation**

Calculate the CID the same way for Storacha as for other providers, including splitting the data into chunks.

## References

1. SCSVS G4: Business Logic

# 6. Recommendations

## [LDO-7582f002-R01] Remove deprecated code

INFO IMPLEMENTED

**Retest (2025-10-10)**

The recommendation has been implemented as recommended.

### Description

The `calculate_gross_core_apr_old` function is an outdated implementation that was previously used to calculate the core APR for the Lido protocol. This function is no longer in use.

**Recommendation**

Remove the unused function.

### References

1. SCSVS G1: Architecture, design and threat modeling

## [LDO-7582f002-R02] Use expiration time for auth tokens

INFO ACKNOWLEDGED

**Retest (2025-10-28)**

The recommendation has not been implemented. Tokens are deliberately set to be nonexpirable to avoid issues if they are not rotated promptly, which can impact the report. The team can rotate a token if it becomes compromised.

### Description

The authentication tokens used for IPFS services (LidoIPFS and Storacha) currently lack an expiration date. This configuration raises concerns regarding potential token leakage and unauthorized use, which could ultimately lead to the depletion of available storage space within the services.

**Recommendation**

Implement expiration dates for authentication tokens and establish a regular rotation schedule. Tokens should be rotated with every Oracle update.

### References

1. SCSVS G1: Architecture, design and threat modeling

## [LDO-7582f002-R03] Setting IPFS parameters explicitly

INFO ACKNOWLEDGED

**Retest (2025-10-28)**

Lido IPFS nodes automatically pin files locally and propagate them to other nodes internally, allowing p2p sharing across the network. This is taken care of on a cluster setup level.

### Description

The IPFS Cluster utilized by the LidoIPFS provider accepts multiple parameters for adding a new file. One of these, `local`, ensures that the file is pinned to the local node and is set to `false` by default. This configuration might store the file on other peers and leads to longer storage times. Another parameter, `no-pin`, which is also set to `false` by default, may potentially require adjustment to ensure the file is pinned accurately, particularly if default settings are altered.

**Recommendation**

Explicitly set the `local` parameter to `true` and the `no-pin` parameter to `false`.

### References

1. SCSVS G1: Architecture, design and threat modeling

# 7. Impact on risk classification

Risk classification is based on the one developed by OWASP [1], however it has been adapted to the immutable and transparent code nature of smart contracts. The Web3 ecosystem forgives much less mistakes than in the case of traditional applications, the servers of which can be covered by many layers of security.

Therefore, the classification is more strict and indicates higher priorities for paying attention to security.

| OVERALL RISK SEVERITY | | | | |
|---|---|---|---|---|
| Impact on risk | HIGH | CRITICAL | HIGH | MEDIUM |
| | MEDIUM | MEDIUM | MEDIUM | LOW |
| | LOW | LOW | LOW | INFO |
| | | HIGH | MEDIUM | LOW |
| | | Likelihood | | |

[1] OWASP Risk Rating methodology

# 8. Long-term best practices

## 8.1. Use automated tools to scan your code regularly

It's a good idea to incorporate automated tools (e.g. slither) into the code writing process. This will allow basic security issues to be detected and addressed at a very early stage.

## 8.2. Perform threat modeling

Before implementing or introducing changes to smart contracts, perform threat modeling and think with your team about what can go wrong. Set potential targets of the attacker and possible ways to achieve them, keep it in mind during implementation to prevent bad design decisions.

## 8.3. Use Smart Contract Security Verification Standard

Use proven standards to maintain a high level of security for your contracts. Treat individual categories as checklists to verify the security of individual components. Expand your unit tests with selected checks from the list to be sure when introducing changes that they did not affect the security of the project.

## 8.4. Discuss audit reports and learn from them

The best guarantee of security is the constant development of team knowledge. To use the audit as effectively as possible, make sure that everyone in the team understands the mistakes made. Consider whether the detected vulnerabilities may exist in other places, audits always have a limited time and the developers know the code best.

## 8.5. Monitor your and similar contracts

Use the tools available on the market to monitor key contracts (e.g. the ones where user's tokens are kept). If you have used code from another project, monitor their contracts as well and introduce procedures to capture information about detected vulnerabilities in their code.


## Damian Rusinek

Smart Contracts Auditor

@drdr_zz

damian.rusinek@composable-security.com



## Paweł Kuryłowicz

Smart Contracts Auditor

@wh01s7

pawel.kurylowicz@composable-security.com