SALUS SECURITY

OCT 2024

CODE SECURITY ASSESSMENT

ETHSTORAGE

Overview

Project Summary

- Name: EthStorage
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/ethstorage/storage-contracts-v1
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	EthStorage
Version	v2
Type	Solidity
Dates	Oct 30 2024
Logs	Oct 25 2024; Oct 30 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	2
Total informational issues	2
Total	4

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Incompatible boolean rlp decoding	Low	Configuration	Resolved
2	Unimplemented refund function	Low	Redundancy	Acknowledged
3	Missing two-step transfer ownership pattern	Informational	Business Logic	Resolved
4	Use of floating pragma	Informational	Configuration	Resolved

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Incompatible boolean rlp decoding

Severity: Low	Category: Configuration
Target: - contracts/library/RLPReader	

Description

contracts/library/RLPReader:L2-L5

```
/*
 * @author Hamdi Allam hamdi.allam97@gmail.com
 * Please reach out with any questions or concerns
 *
https://github.com/hamdiallam/Solidity-RLP/blob/e681e25a376dbd5426b509380bc03446f05d0f97
/contracts/RLPReader.sol
 */
```

The `RLPReader` contract is an open source contract on github. However, there is an issue with the version used in this contract.

Because the project uses geth for data storage, RLP encoding would be used very frequently. Although this is considered an edge case, we still recommend that projects that make heavy use of RLP address this issue.

Recommendation

It is recommended to choose the version after the problem is fixed.

Status

The team has resolved this issue in commit c4ca613.

2. Unimplemented refund function

Severity: Low	Category: Redundancy
Target: - contracts/DecentralizedKV.sol	

Description

contracts/DecentralizedKV.sol:L238 - L240

```
function removeTo(bytes32 _key, address _to) public virtual {
    revert("DecentralizedKV: removeTo() unimplemented");
}
```

The `removeTo` function in the `DecentralizedKV` contract is defined but unused, only inherited and implemented within the test contract.

Recommendation

Consider implementing a user exit feature.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

3. Missing two-step transfer ownership pattern

Severity: Informational	Category: Business logic
Target: - contracts/DecentralizedKV.sol	

Description

The `DecentralizedKV` contract inherits from the `OwnableUpgradeable` contract. This contract does not implement a two-step process for transferring ownership. Thus, ownership of the contract can easily be lost when making a mistake in transferring ownership.

Recommendation

Consider using the Ownable2Step contract from OpenZeppelin instead.

Status

The team has resolved this issue in commit 0aded1d.

4. Use of floating pragma

Severity: Informational	Category: Configuration
Target: - All	

Description

```
pragma solidity ^0.8.28;
pragma solidity ^0.8.20;
pragma solidity ^0.8.0;
```

All contracts use a floating compiler version.

Using floating pragma statements is discouraged, as code may compile to different bytecodes with different compiler versions. Use a locked pragma statement to get a deterministic bytecode. Also use the latest Solidity version to get all the compiler features, bug fixes and optimizations.

Recommendation

It is recommended to use a locked Solidity version throughout the project. It is also recommended to use the most stable and up-to-date version.

Status

The team has resolved this issue in commit 9c7fc37.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 8acd50e:

File	SHA-1 hash
contracts/DecentralizedKV.sol	ea796ff3d68aa722da65650b381579ef47b59646
contracts/EthStorageContract.sol	93ac795fdfaa350fad94eb61264c05b25f7f4231
contracts/EthStorageContract2.sol	1c53074509e141399bcb86e3e68d57371f69282b
contracts/EthStorageContractL2.sol	c2af6b48dadb94c6df8884a73ef9ff991f16d95e
contracts/EthStorageUpgradeableProxy.sol	9858b582952be406b85a0374460fd78b36e1dcd3
contracts/StorageContract.sol	b88553b961320b1a01f9615e59a5f8bdd64d411b
contracts/libraries/BinaryRelated.sol	acf522d927e46a58ff7c24d4fd9fd2d1aa1108b9
contracts/libraries/MerkleLib.sol	8c743e90d2677ccf5a840ed68941e863d2323cd6
contracts/libraries/MiningLib.sol	24c0b092ff9812d461ea5c0d4cd1472496df4eb8
contracts/libraries/RLPReader.sol	b41a0f06eb04ea96b6866a6938c45518bdf23014
contracts/libraries/RandaoLib.sol	01266bd26cc143e1e189e05be91767413c8ed309