Competitive Security Assessment

zkLink Nova Arbitrator Upgrade

Apr 13th, 2024

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

• Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.

• Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.

• Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.

• Verify the code base is compliant with the most up-to-date industry standards and security best practices.

• Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

Project Name	zkLink Nova Arbitrator Upgrade
Language	Solidity

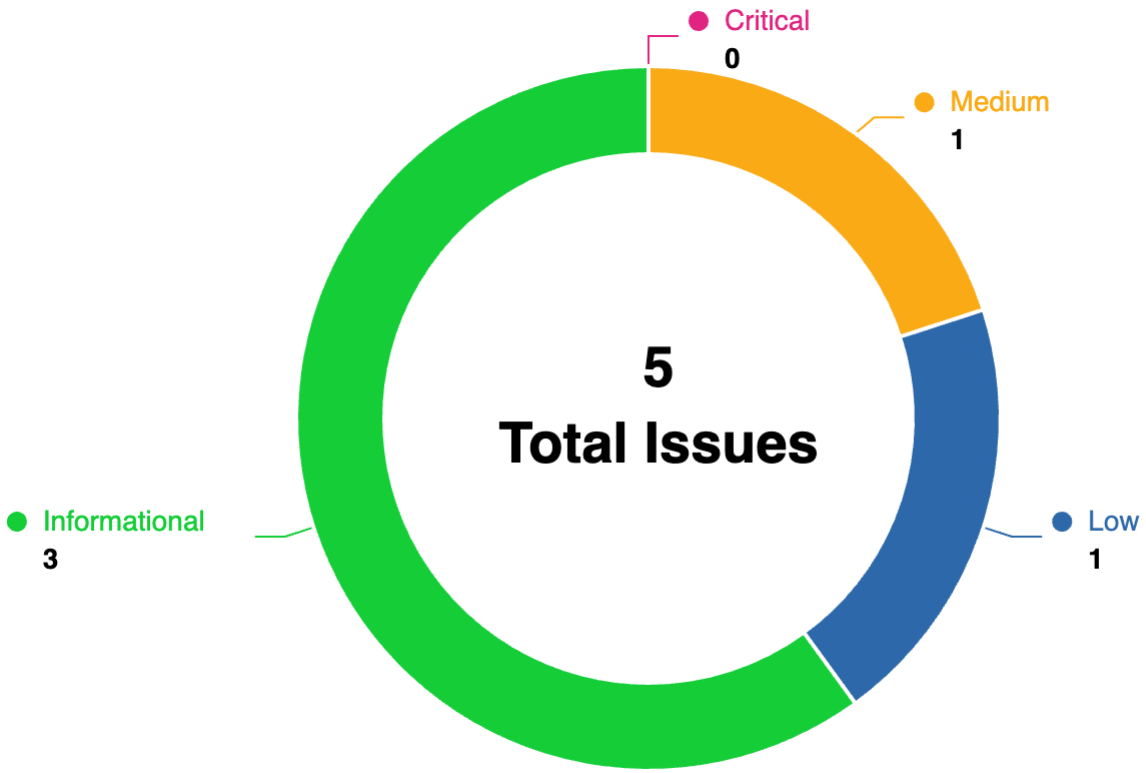
Codebase	• audit version
	  • https://github.com/zkLinkProtocol/era-contracts/pull/7/commits/5940c61d6d0854ed37d40915b879e752e3689ded
	  • https://github.com/zkLinkProtocol/zklink-evm-contracts/pull/90/commits/23a36a3b9cd2f63ba7e0658ac6c554b84291bb74
	• final version
	  • https://github.com/zkLinkProtocol/era-contracts/commit/899254af23c4314cf7fe1047a3e3bf4e26152f3b
	  • https://github.com/zkLinkProtocol/zklink-evm-contracts/commit/8badb5e313caa2e6b552b40aec1b3d1d589e1104

Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Audit Scope

File	SHA256 Hash
https://github.com/zkLinkProtocol/era-contracts	https://github.com/zkLinkProtocol/era-contracts/pull/7/commits/5940c61d6d0854ed37d40915b879e752e3689ded
https://github.com/zkLinkProtocol/zklink-evm-contracts	https://github.com/zkLinkProtocol/zklink-evm-contracts/pull/90/commits/23a36a3b9cd2f63ba7e0658ac6c554b84291bb74

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
ZKL-1	Ownership change should use two-step process	Logical	Medium	Acknowledged	Yaodao, 0xzoobi, biakia
ZKL-2	Specify the version of solidity as 0.8.24 to ensure that the tload/tstore operation will work.	Language Specific	Low	Fixed	biakia
ZKL-3	`public` functions not called by the contract should be declared `external` instead	Logical	Informational	Fixed	rajatbeladiya
ZKL-4	Use calldata instead of memory	Gas Optimization	Informational	Fixed	biakia
ZKL-5	Typographical Error in Variable Name	Logical	Informational	Fixed	BradMoonUESTC, danielt

ZKL-1:Ownership change should use two-step process

Category	Severity	Client Response	Contributor
Logical	Medium	Acknowledged	Yaodao, 0xzoobi, biakia

Code Reference

- code/zklink/zklink-evm-contracts/contracts/Arbitrator.sol#L5
- code/zklink/zklink-evm-contracts/contracts/Arbitrator.sol#L18

```
5: import {OwnableUpgradeable} from "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
```

```
18: contract Arbitrator is IArbitrator, OwnableUpgradeable, UUPSUpgradeable, ReentrancyGuardUpgradeable {
```

- code/zklink/zklink-evm-contracts/contracts/ZkLink.sol#L5
- code/zklink/zklink-evm-contracts/contracts/ZkLink.sol#L27-L36
- code/zklink/zklink-evm-contracts/contracts/ZkLink.sol#L32

```
5: import {OwnableUpgradeable} from "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
```

```
27: contract ZkLink is
28:     IZkLink,
29:     IMailbox,
30:     IAdmin,
31:     IGetters,
32:     OwnableUpgradeable,
33:     UUPSUpgradeable,
34:     ReentrancyGuardUpgradeable,
35:     PausableUpgradeable
36: {
```

```
32: OwnableUpgradeable,
```

Description

Yaodao: It is possible that the **`onlyOnwer`** role mistakenly transfers ownership to the wrong address, resulting in the loss of the **`onlyOnwer`** role.
0xzoobi: The contracts **`Arbitrator.sol`** and **`ZkLink.sol`** does not implement a 2-Step-Process for transferring ownership.
So ownership of the contract can easily be lost when making a mistake when transferring ownership.
Since the privileged roles have critical function roles assigned to them. Assigning the ownership to a wrong user can be disastrous.
So Consider using the **`Ownable2StepUpgradeable`** contract from OZ (https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/access/Ownable2StepUpgradeable.sol) instead.

biakia: The contract `ZkLink` and `Arbitrator` do not implement a two-step process for transferring ownership, so ownership of the contract can be easily lost when making a mistake when transferring ownership.

Recommendation

Yaodao: Consider implementing a two-step process where the onwer nominates an account and the nominated account needs to call an `acceptOnwership()` function for the transfer of the ownership to fully succeed.
0xzoobi: Implement 2-Step-Process for transferring ownership via `Ownable2StepUpgradeable`.
biakia: Consider Ownable2StepUpgradeable(https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/access/Ownable2StepUpgradeable.sol) instead.

Client Response

client response for Yaodao: Acknowledged
client response for 0xzoobi: Acknowledged
client response for biakia: Acknowledged

ZKL-2:Specify the version of solidity as 0.8.24 to ensure that the tload/tstore operation will work.

Category	Severity	Client Response	Contributor
Language Specific	Low	Fixed	biakia

Code Reference

- code/zklink/zklink-evm-contracts/contracts/Arbitrator.sol#L3
- code/zklink/zklink-evm-contracts/contracts/Arbitrator.sol#L164-L166
- code/zklink/zklink-evm-contracts/contracts/Arbitrator.sol#L211-L213

```
3: pragma solidity ^0.8.0;
```

```
164: assembly {
165:             tstore(finalizeMessageHash.slot, _finalizeMessageHash)
166:         }
```

```
211: assembly {
212:             _finalizeMessageHash := tload(finalizeMessageHash.slot)
213:         }
```

Description

biakia: Solidity 0.8.24 supports the opcodes included in the Cancun hardfork and, in particular, the transient storage opcodes TSTORE and TLOAD as per EIP-1153.
Currently, the `**Arbitrator**` contract is using `**tload()**`/`**tstore()**` but does not specify the version of solidity as 0.8.24:

```
assembly {
            tstore(finalizeMessageHash.slot, _finalizeMessageHash)
        }
```

```
 assembly {
            _finalizeMessageHash := tload(finalizeMessageHash.slot)
        }
```

```
pragma solidity ^0.8.0;
```

It is better to specify the version of solidity as 0.8.24 ensure that the tload/tstore operation will work.
Reference:
https://soliditylang.org/blog/2024/01/26/transient-storage/

Recommendation

biakia: Consider following fix:

```
pragma solidity 0.8.24;
```

Client Response

client response for biakia: Fixed - https://github.com/zkLinkProtocol/zklink-evm-contracts/commit/2c2d368331d64e8562d370a382d1a992a5dfbe85

ZKL-3: `public` functions not called by the contract should be declared `external` instead

Category	Severity	Client Response	Contributor
Logical	Informational	Fixed	rajatbeladiya

Code Reference

- code/zklink/era-contracts/l1-contracts/contracts/zksync/facets/Mailbox.sol#L62-L69
- code/zklink/era-contracts/l1-contracts/contracts/zksync/facets/Mailbox.sol#L142-L146

```
62: function proveL1ToL2TransactionStatus(
63:         bytes32 _l2TxHash,
64:         uint256 _l2BatchNumber,
65:         uint256 _l2MessageIndex,
66:         uint16 _l2TxNumberInBatch,
67:         bytes32[] calldata _merkleProof,
68:         TxStatus _status
69:     ) public view returns (bool) {
```

```
142: function l2TransactionBaseCost(
143:         uint256 _gasPrice,
144:         uint256 _l2GasLimit,
145:         uint256 _l2GasPerPubdataByteLimit
146:     ) public view returns (uint256) {
```

Description

rajatbeladiya: if a function is not called internally within a contract, it is more efficient to declare it as `external` rather than `public`.
here, `l2TransactionBaseCost()` and `proveL1ToL2TransactionStatus()` not called by the contract internally anywhere.

Recommendation

rajatbeladiya: mark `l2TransactionBaseCost()` and `proveL1ToL2TransactionStatus()` as `external` instead `public`

Client Response

client response for rajatbeladiya: Fixed - https://github.com/zkLinkProtocol/era-contracts/commit/899254af23c4314cf7fe1047a3e3bf4e26152f3b

ZKL-4:Use calldata instead of memory

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	biakia

Code Reference

- code/zklink/zklink-evm-contracts/contracts/ZkLink.sol#L490-L494

```
490: function openRangeBatchRootHash(
491:         uint256 _fromBatchNumber,
492:         uint256 _toBatchNumber,
493:         bytes32[] memory _l2LogsRootHashes
494:     ) external onlyValidator {
```

Description

biakia: The function `**openRangeBatchRootHash**` is used to unzip the root hashes in the range. The input param `**_l2LogsRootHashes**` is the l2LogsRootHash list in the range `**[_fromBatchNumber,_toBatchNumber]**`. When the `**_l2LogsRootHashes**` is a large list, it is better to use `**calldata**` instead of `**memory**` to save gas.

Recommendation

biakia: Consider using `**calldata**` instead of `**memory**`:

```
function openRangeBatchRootHash(
        uint256 _fromBatchNumber,
        uint256 _toBatchNumber,
        bytes32[] calldata _l2LogsRootHashes
    ) external onlyValidator {
```

Client Response

client response for biakia: Fixed - https://github.com/zkLinkProtocol/zklink-evm-contracts/commit/5918de27415dc608b8394aac2cf2b4915f829cea

ZKL-5:Typographical Error in Variable Name

Category	Severity	Client Response	Contributor
Logical	Informational	Fixed	BradMoonUESTC, danielt

Code Reference

- code/zklink/zklink-evm-contracts/contracts/ZkLink.sol#L473-L516

```
473: function syncRangeBatchRoot(
474:         uint256 _fromBatchNumber,
475:         uint256 _toBatchNumber,
476:         bytes32 _rangeBatchRootHash,
477:         uint256 _forwardEthAmount
478:     ) external payable onlyGateway {
479:         require(_toBatchNumber >= _fromBatchNumber, "Invalid range");
480:         require(msg.value == _forwardEthAmount, "Invalid forward amount");
481:         bytes32 range = keccak256(abi.encodePacked(_fromBatchNumber, _toBatchNumber));
482:         rangBatchRootHashes[range] = _rangeBatchRootHash;
483:         emit SyncRangeBatchRoot(_fromBatchNumber, _toBatchNumber, _rangeBatchRootHash, _forward
EthAmount);
484:     }
485:
486:     /// @dev Unzip the root hashes in the range
487:     /// @param _fromBatchNumber The batch number from
488:     /// @param _toBatchNumber The batch number to
489:     /// @param _l2LogsRootHashes The l2LogsRootHash list in the range [`_fromBatchNumber`, `_to
BatchNumber`]
490:     function openRangeBatchRootHash(
491:         uint256 _fromBatchNumber,
492:         uint256 _toBatchNumber,
493:         bytes32[] memory _l2LogsRootHashes
494:     ) external onlyValidator {
495:         require(_toBatchNumber >= _fromBatchNumber, "Invalid range");
496:         bytes32 range = keccak256(abi.encodePacked(_fromBatchNumber, _toBatchNumber));
497:         bytes32 rangeBatchRootHash = rangBatchRootHashes[range];
498:         require(rangeBatchRootHash != bytes32(0), "Rang batch root hash not exist");
499:         uint256 rootHashesLength = _l2LogsRootHashes.length;
500:         require(rootHashesLength == _toBatchNumber - _fromBatchNumber + 1, "Invalid root hashes
length");
501:         bytes32 _rangeBatchRootHash = _l2LogsRootHashes[0];
502:         l2LogsRootHashes[_fromBatchNumber] = _rangeBatchRootHash;
503:         unchecked {
504:             for (uint256 i = 1; i < rootHashesLength; ++i) {
505:                 bytes32 _l2LogsRootHash = _l2LogsRootHashes[i];
506:                 l2LogsRootHashes[_fromBatchNumber + i] = _l2LogsRootHash;
507:                 _rangeBatchRootHash = Merkle._efficientHash(_rangeBatchRootHash, _l2LogsRootHas
h);
508:             }
509:         }
510:         require(_rangeBatchRootHash == rangeBatchRootHash, "Incorrect root hash");
511:         delete rangBatchRootHashes[range];
512:         if (_toBatchNumber > totalBatchesExecuted) {
513:             totalBatchesExecuted = _toBatchNumber;
514:         }
515:         emit OpenRangeBatchRoot(_fromBatchNumber, _toBatchNumber);
516:     }
```

- diff/zklink_evm_contracts_diff.patch#L372
- diff/zklink_evm_contracts_diff.patch#L402
- diff/zklink_evm_contracts_diff.patch#L409
- diff/zklink_evm_contracts_diff.patch#L621

```
372: +    mapping(bytes32 range => bytes32 rangeRootHash) public rangRootHashMap;
```

```
402: +        rangRootHashMap[range] = _rangeRootHash;
```

```
409: +        bytes32 rangeRootHash = rangRootHashMap[range];
```

```
621: +        require(rangeBatchRootHash != bytes32(0), "Rang batch root hash not exist");
```

Description

BradMoonUESTC: In the provided smart contract function `**openRangeBatchRootHash**`, a typographical error is identified where the variable `**rangBatchRootHashes**` is used instead of the correct `**rangeBatchRootHashes**`. This discrepancy could lead to several issues depending on the broader contract context, including but not limited to compile-time errors, logic flaws, security vulnerabilities, data consistency problems, and maintenance challenges. Specifically, the error occurs in the line:

```
bytes32 rangeBatchRootHash = rangBatchRootHashes[range];
```

danielt: The message in below `**require**` statement is below:

```
require(rangeBatchRootHash != bytes32(0), "Rang batch root hash not exist");
```

It should be `**Range batch**` rather than `**Rang batch**`.
Similar, the variable `**rangRootHashMap**` intends to be `**rangeRootHashMap**`

Recommendation

BradMoonUESTC: it is recommended to correct the typographical error by replacing all instances of `**rangBatchRootHashes**` with the correct `**rangeBatchRootHashes**`. Here is the corrected version of the problematic line within the provided function:

```
bytes32 rangeBatchRootHash = rangeBatchRootHashes[range];
```

danielt: Recommend correcting the typos in the messages and variable.

Client Response

client response for BradMoonUESTC: Fixed - https://github.com/zkLinkProtocol/zklink-evm-contracts/commit/c77e28e57a775bc8cfb6e7a4d74d8569f534851e
client response for danielt: Fixed - https://github.com/zkLinkProtocol/zklink-evm-contracts/commit/c77e28e57a775bc8cfb6e7a4d74d8569f534851e

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3's prior written consent in each instance.

This report is not an "endorsement" or "disapproval" of any particular project or team. This report is not an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3's position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.