Competitive Security Assessment

zklink_bridge_update

Mar 27th, 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.

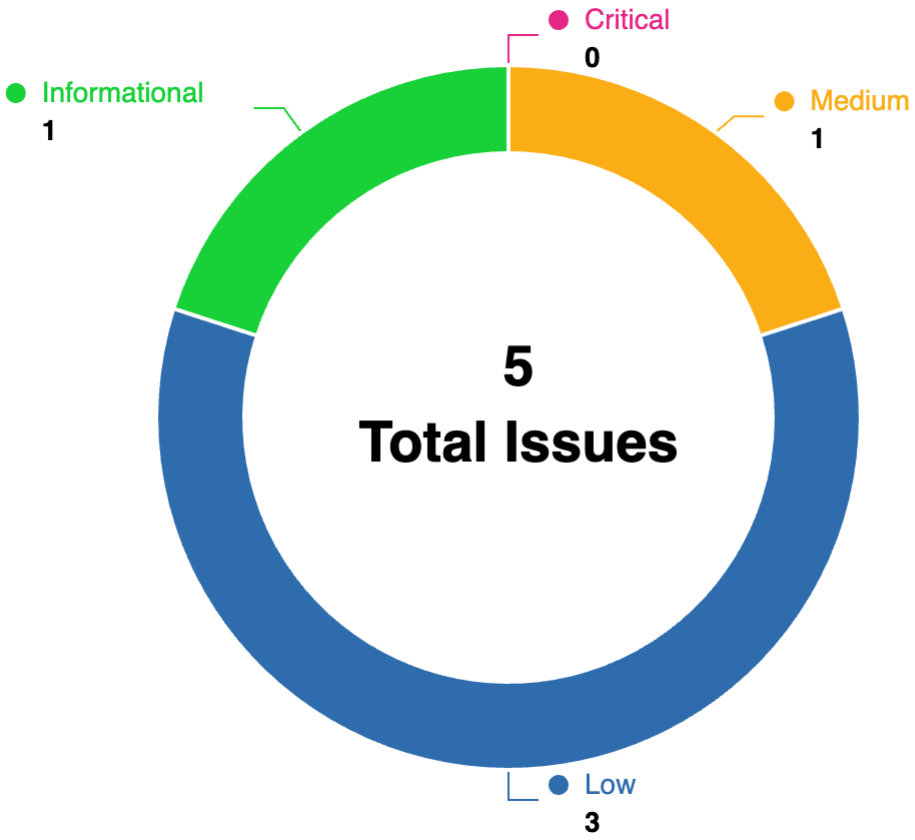
Overview

Project Name	zklink_bridge_update
Language	Solidity
Codebase	• https://github.com/zkLinkProtocol/era-contracts • audit version - 3bce278c653a5b8e928db138f35529d566f4af0f • final version - 4cb7aa42ff012abbf9a4ea2649214447d1d7a345
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Audit Scope

File	SHA256 Hash
l1-contracts/contracts/bridge/L1ERC20Bridge.sol	4c34e6f86862896612f9baecf3c1acf5ea43fc7b67edd4acdc24a4f89ea4f4cb
l1-contracts/contracts/bridge/L1WethBridge.sol	7433eb6ea89b20e78a615d0bff4fee7340cb30939394e1206fdc2d7bfbd9b54b
l2-contracts/contracts/bridge/L2ERC20Bridge.sol	2d828fdd10fe0e9108e32bd7cb3b1b6ec01b47e7bb9cce95f54c1cb9f8cec404
l2-contracts/contracts/bridge/L2WethBridge.sol	6194edc9725c9574beb44ae056cdaf6ccb779eaf7823c0c3ae8e1650ec5c78ea
l1-contracts/contracts/bridge/interfaces/IL1Bridge.sol	773fba7633d99ca7288445c5f5110ed1bb4151e040789793b2b97b7616869acc
l2-contracts/contracts/bridge/interfaces/IL2Bridge.sol	17780a3ded129da7198da6b7e774b1c2f3219d3354be9abf8e26f3f540bbe34b
l1-contracts/contracts/bridge/interfaces/IL2Bridge.sol	e8cae12ba7f5c0347a8af660811f1cd4481310c007fc85f794e93fbe6343712a
l2-contracts/contracts/bridge/interfaces/IMergeTokenPortal.sol	6b3b48d43cf3adaa883e5d1064988a332f141adfea1b2d3a7e9449dfab54f582

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
ZKB-1	`SafeERC20.safeApprove` reverts for changing existing approvals	DOS	Medium	Fixed	0xzoobi
ZKB-2	`safeApprove()` is deprecated	Logical	Low	Fixed	rajatbeladiya
ZKB-3	`approve()` / `safeApprove()` may revert if the current approval is not zero	Logical	Low	Fixed	rajatbeladiya
ZKB-4	Bridge Insolvency Due to Incorrect Calculation for Deflationary or Fee-on-Transfer Tokens in `withdrawFromMerge()` Function	Logical	Low	Acknowledged	rajatbeladiya
ZKB-5	`require` error messages are not verbose	Code Style	Informational	Acknowledged	0xzoobi

ZKB-1: `SafeERC20.safeApprove` reverts for changing existing approvals

Category	Severity	Client Response	Contributor
DOS	Medium	Fixed	0xzoobi

Code Reference

- code/l2-contracts/contracts/bridge/L2ERC20Bridge.sol#L120

```
120: IERC20Upgradeable(_l2Token).safeApprove(address(MERGE_TOKEN_PORTAL), _amount);
```

Description

0xzoobi: The **`SafeERC20.safeApprove()`** function is designed to revert when attempting to change a non-zero approval to another non-zero approval. In the context of the **`MERGE_TOKEN_PORTAL`**, if a user has any existing non-zero approvals with this contract, any attempt to change these approvals will always result in transaction reverting. This situation can create a form of Denial of Service (DoS), preventing users from successfully modifying their approvals.
Similar issues have been reported in the Sherlock Audit Contest, indicating the severity and importance of addressing this concern.
Referenced issue:

1. Sherlock Audit Contest - Issue #141

Recommendation

0xzoobi: Consider using **`safeIncreaseAllowance`** and **`safeDecreaseAllowance`** instead of **`safeApprove()`**.

Client Response

0xzoobi: Fixed - https://github.com/zkLinkProtocol/era-contracts/commit/4cb7aa42ff012abbf9a4ea2649214447d1d7a345

ZKB-2: `safeApprove()` is deprecated

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	rajatbeladiya

Code Reference

- code/l2-contracts/contracts/bridge/L2ERC20Bridge.sol#L120

```
120: IERC20Upgradeable(_l2Token).safeApprove(address(MERGE_TOKEN_PORTAL), _amount);
```

Description

rajatbeladiya: checkout: Openzeppelin's deprecation warning
It was deprecated in favor of **`safeIncreaseAllowance()`** and **`safeDecreaseAllowance()`**. If only setting the initial allowance to the value that means infinite, **`safeIncreaseAllowance()`** can be used instead. The function may currently work, but if a bug is found in this version of OpenZeppelin, and the version that you're forced to upgrade to no longer has this function, you'll encounter unnecessary delays in porting and testing replacement contracts.

Recommendation

rajatbeladiya: Use **`safeIncreaseAllowance()`** and **`safeDecreaseAllowance()`** instead of **`safeApprove()`**

Client Response

rajatbeladiya: Fixed - https://github.com/zkLinkProtocol/era-contracts/commit/4cb7aa42ff012abbf9a4ea2649214447d1d7a345

ZKB-3: `approve()` / `safeApprove()` may revert if the current approval is not zero

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	rajatbeladiya

Code Reference

- code/l2-contracts/contracts/bridge/L2ERC20Bridge.sol#L120

```
120: IERC20Upgradeable(_l2Token).safeApprove(address(MERGE_TOKEN_PORTAL), _amount);
```

Description

rajatbeladiya: - Some tokens (like the *very popular* USDT) do not work when changing the allowance from an existing non-zero allowance value (it will revert if the current approval is not zero to protect against front-running changes of approvals). These tokens must first be approved for zero and then the actual allowance can be approved.

- Furthermore, OZ's implementation of safeApprove would throw an error if an approve is attempted from a non-zero value (`"SafeERC20: approve from non-zero to non-zero allowance"`)

Recommendation

rajatbeladiya: Set the allowance to zero immediately before each of the existing allowance calls

Client Response

rajatbeladiya: Fixed - https://github.com/zkLinkProtocol/era-contracts/commit/4cb7aa42ff012abbf9a4ea2649214447d1d7a345

ZKB-4:Bridge Insolvency Due to Incorrect Calculation for Deflationary or Fee-on-Transfer Tokens in `withdrawFromMerge()` Function

Category	Severity	Client Response	Contributor
Logical	Low	Acknowledged	rajatbeladiya

Code Reference

- code/l2-contracts/contracts/bridge/L2ERC20Bridge.sol#L173

```
173: IERC20Upgradeable(mergeToken).safeTransferFrom(msg.sender, address(this), _amount);
```

Description

rajatbeladiya: The **`withdrawFromMerge()`** function in the **`L2ERC20Bridge`** contract contains a critical flaw that fails to correctly handle fee-on-transfer tokens. Fee-on-transfer tokens are a class of tokens that deduct a fee from the transferred amount during each transfer operation.
here, the contract assumes that the full **`_amount`** of tokens will be transferred from **`msg.sender`** to the contract. However, if the token deducts a fee on the transfer, the contract will not receive the full **`_amount`**. Then, the function attempts to withdraw the same **`_amount`** from the MERGE_TOKEN_PORTAL. However, since the contract received less than **`_amount`** due to the transfer fee, leads to it may not have enough tokens to perform all withdrawals.

Recommendation

rajatbeladiya: To fix this issue, the contract should account for balance of the token before and after the transfer, and using the actual received amount in the withdraw function call.

```
uint256 initialBalance = IERC20Upgradeable(mergeToken).balanceOf(address(this));
IERC20Upgradeable(mergeToken).safeTransferFrom(msg.sender, address(this), _amount);
uint256 finalBalance = IERC20Upgradeable(mergeToken).balanceOf(address(this));
uint256 actualReceived = finalBalance - initialBalance;

MERGE_TOKEN_PORTAL.withdraw(_l2Token, actualReceived, address(this));
```

Client Response

rajatbeladiya: Acknowledged - All tokens participating in the merge must be standard

ZKB-5: `require` error messages are not verbose

Category	Severity	Client Response	Contributor
Code Style	Informational	Acknowledged	0xzoobi

Code Reference

- code/l1-contracts/contracts/bridge/L1ERC20Bridge.sol#L89-L94
- code/l1-contracts/contracts/bridge/L1ERC20Bridge.sol#L249-L251
- code/l1-contracts/contracts/bridge/L1ERC20Bridge.sol#L346-L349
- code/l1-contracts/contracts/bridge/L1ERC20Bridge.sol#L400-L403

```
89: require(_l2TokenBeacon != address(0), "nf");
90:         require(_governor != address(0), "nh");
91:         // We are expecting to see the exact three bytecodes that are needed to initialize the br
idge
92:         require(_factoryDeps.length == 3, "mk");
93:         // The caller miscalculated deploy transactions fees
94:         require(msg.value == _deployBridgeImplementationFee + _deployBridgeProxyFee, "fee");
```

```
249: require(_amount != 0, "2T"); // empty deposit amount
250:         uint256 amount = _depositFunds(msg.sender, IERC20(_l1Token), _amount);
251:         require(amount == _amount, "1T"); // The token has non-standard transfer logic
```

```
346: require(proofValid, "yn");
347:
348:         uint256 amount = depositAmount[_depositSender][_l1Token][_l2TxHash];
349:         require(amount > 0, "y1");
```

```
400: require(_l2ToL1message.length == 76, "kk");
401:
402:         (uint32 functionSignature, uint256 offset) = UnsafeBytes.readUint32(_l2ToL1message, 0);
403:         require(bytes4(functionSignature) == this.finalizeWithdrawal.selector, "nt");
```

- code/l1-contracts/contracts/bridge/L1WethBridge.sol#L264
- code/l1-contracts/contracts/bridge/L1WethBridge.sol#L321

```
264: require(success, "vq");
```

```
321: require(msg.sender == l1WethAddress || msg.sender == address(zkSync), "pn");
```

- code/l2-contracts/contracts/bridge/L2ERC20Bridge.sol#L57-L59
- code/l2-contracts/contracts/bridge/L2ERC20Bridge.sol#L118-137

```
57: require(_l1Bridge != address(0), "bf");
58:         require(_l2TokenProxyBytecodeHash != bytes32(0), "df");
59:         require(_governor != address(0), "sf");
```

```
118: require(msg.sender == address(this), "Only bridge can call this function");
119:         IL2StandardToken(_l2Token).bridgeMint(address(this), _amount);
120:         IERC20Upgradeable(_l2Token).safeApprove(address(MERGE_TOKEN_PORTAL), _amount);
121:         MERGE_TOKEN_PORTAL.deposit(_l2Token, _amount, _l2Receiver);
122:     }
123:
124:     function _getExpectedL2Token(address _l1Token, bytes calldata _data) internal returns (addr
ess) {
125:         // Only the L1 bridge counterpart can initiate and finalize the deposit.
126:         require(AddressAliasHelper.undoL1ToL2Alias(msg.sender) == l1Bridge, "mq");
127:         // The passed value should be 0 for ERC20 bridge.
128:         require(msg.value == 0, "Value should be 0 for ERC20 bridge");
129:
130:         address expectedL2Token = l2TokenAddress(_l1Token);
131:         address currentL1Token = l1TokenAddress[expectedL2Token];
132:         if (currentL1Token == address(0)) {
133:             address deployedToken = _deployL2Token(_l1Token, _data);
134:             require(deployedToken == expectedL2Token, "mt");
135:             l1TokenAddress[expectedL2Token] = _l1Token;
136:         } else {
137:             require(currentL1Token == _l1Token, "gg"); // Double check that the expected value
equal to real one
```

- code/l2-contracts/contracts/bridge/L2WethBridge.sol#L122

```
122: require(msg.sender == l2WethAddress, "pd");
```

Description

0xzoobi: The code appears to resemble a fork of zksync bridge contracts. However, one aspect that could significantly improve is the clarity of the error messages in the require statements. Currently, it seems to use terse, two-letter words that lack clarity and can be challenging to interpret when functions revert due to these statements. As a best coding practice, it's essential to make these error statements verbose and descriptive to aid in debugging and understanding the reason for the revert.

Recommendation

0xzoobi: Improve the readability of the error messages in the `**require**` statements.

Client Response

0xzoobi: Acknowledged - We do not plan to modify the old error message of zksync era. We provide detailed error messages in the new code.

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.