SALUS SECURITY

MAY 2025

CODE SECURITY ASSESSMENT

REDDIO

Overview

Project Summary

- Name: Reddio - EVM Bridge
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/reddio-com/evm-bridge
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Reddio - EVM Bridge
Version	v2
Type	Solidity
Dates	May 21 2025
Logs	Apr 29 2025, May 21 2025

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	7
Total Low-Severity issues	1
Total informational issues	2
Total	11

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	Missing access control in the relayMessage and relayMessageWithProof functions	High	Business Logic	Resolved
2	Missing gas limit validation	Medium	Business Logic	Resolved
3	Missing cross-chain fee withdrawal function	Medium	Business Logic	Resolved
4	MKR token does not work in evm bridge	Medium	Business Logic	Resolved
5	Missing gas refund mechanism	Medium	Business Logic	Resolved
6	Revisit bridge fee logic	Medium	Business Logic	Resolved
7	Centralization risk	Medium	Centralization	Acknowledged
8	Fee-on-transfer token does not work in evm bridge	Low	Business Logic	Resolved
9	Incorrect check in _registerERC1155Token	Informational	Business Logic	Resolved
10	Redundant code	Informational	Redundancy	Resolved

2.2 Notable Findings

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

1. Missing access control in the `relayMessage` and `relayMessageWithProof` functions

Severity: High	Category: Business Logic

Target:
- contracts/parentLayer/UpwardMessageDispatcherFacet.sol
- contracts/childLayer/DownwardMessageDispatcherFacet.sol

Description

In the `relayMessageWithProof` function, we update the execution status of a message on L2 to prevent potential replay attacks. However, this function currently lacks proper access control. As a result, a malicious actor could frontrun a legitimate message transaction and falsely mark the message as executed. This would cause the original transaction to revert, leading to a denial-of-service scenario.

contracts/parentLayer/UpwardMessageDispatcherFacet.sol: L82-L88

```
function relayMessageWithProof(
  uint32 payloadType,
  bytes calldata payload,
  uint256 nonce
) external
{
  ...
}
```

contracts/childLayer/DownwardMessageDispatcherFacet.sol: L106-L110

```
function relayMessage(
  uint32 payloadType,
  bytes calldata payload,
  uint256 nonce //whenNotPaused
) external {
  ...
}
```

Recommendation

It is recommended to add access control to the `relayMessage` and `relayMessageWithProof` functions.

Status

This issue has been resolved by the team with commit 398fdf2.

2. Missing gas limit validation

Severity: Medium	Category: Business Logic

Target:
- contracts/parentLayer/ParentBridgeCoreFacet.sol
- contracts/parentLayer/tokenMessages/ChildGasPriceOracle.sol

Description

When users transfer tokens from L1 to L2, we charge a bridge fee based on the provided `gasLimit` and the `l2BaseFee`. However, the current implementation lacks validation for the `gasLimit` parameter. This opens the door for malicious users to supply an artificially low gas limit in order to underpay the bridge fees, potentially leading to underfunded transactions on L2.

contracts/parentLayer/ParentBridgeCoreFacet.sol: L133-L148

```
function sendDownwardMessage(
  uint32 payloadType,
  bytes calldata payload,
  uint256 ethAmount,
  uint256 gasLimit,
  uint256 value
) external onlySelf whenNotPaused {
  // compute and deduct the messaging fee to fee vault.
  uint256 fee = estimateCrossMessageFee(gasLimit);
  ...
}
```

contracts/parentLayer/tokenMessages/ChildGasPriceOracle.sol: L72-L76

```
function estimateCrossDomainMessageFee(uint256 _gasLimit) external view returns
(uint256) {
  GasPriceOracleStorage storage ds = _gasPriceOracleStorage();

  return _gasLimit * ds.l2BaseFee;
}
```

Recommendation

Add `gasLimit` validation based on the message payload.

Status

This issue has been resolved by the team with commit 398fdf2.

3. Missing cross-chain fee withdrawal function

Severity: Medium	Category: Business Logic
Target: - contracts/parentLayer/ParentBridgeCoreFacet.sol	

Description

When a user sends a cross-chain message from L1 to L2, the protocol charges a bridge fee based on the specified gas limit. However, the current implementation does not provide any mechanism to withdraw the accumulated cross-chain fees. As a result, these funds remain locked in the contract and cannot be recovered or utilized.

contracts/parentLayer/ParentBridgeCoreFacet.sol: L133-L149

```
function sendDownwardMessage(
  uint32 payloadType,
  bytes calldata payload,
  uint256 ethAmount,
  uint256 gasLimit,
  uint256 value
) external onlySelf whenNotPaused {
  // compute and deduct the messaging fee to fee vault.
  uint256 fee = estimateCrossMessageFee(gasLimit);
  require(value >= fee + ethAmount, "Insufficient msg.value");

  appendCrossDomainMessage(payloadType, payload, gasLimit);

  emit DownwardMessage(payloadType, payload);
}
```

Recommendation

It is recommended to add a function for withdrawing cross-chain fees from the `diamond` contract.

Status

This issue has been resolved by the team with commit 398fdf2.

4. MKR token does not work in evm bridge

Severity: Medium	Category: Business Logic

Target:
- contracts/parentLayer/tokenMessages/ParentTokenMessageTransmitterFacet.sol

Description

The EVM bridge is designed to support bridging of various token standards—including ERC20, ERC721, and ERC1155—between L1 and Reddio L2. When transferring ERC20 tokens, the bridge constructs a payload based on the token's metadata, such as `name` and `symbol`. However, certain tokens like MKR do not strictly follow the ERC20 standard—specifically, their `name` and `symbol` functions return `bytes32` instead of `string`. This incompatibility causes the payload construction to fail, preventing the successful bridging of MKR from L1 to L2.

contracts/parentLayer/ParentBridgeCoreFacet.sol: L133-L149

```
function depositERC20Token(
  address tokenAddress,
  address recipient,
  uint256 amount,
  uint256 gasLimit
) external payable { // payable for gas fee.
  IERC20Token erc20Token = IERC20Token(tokenAddress);
  ...
  IParentBridgeCore bridgeCore = IParentBridgeCore(address(this));
  bytes memory payload = abi.encode(
    ParentERC20TokenLocked({
      tokenAddress: tokenAddress,
      tokenName: erc20Token.name(),
      tokenSymbol: erc20Token.symbol(),
      decimals: erc20Token.decimals(),
      parentSender: msg.sender,
      childRecipient: recipient,
      amount: amount
    })
  );
  ...
}
```

Recommendation

Process this special case if we plan to support MKR token.

Status

This issue has been resolved by the team with commit 398fdf2.

5. Missing gas refund mechanism

Severity: Medium	Category: Business Logic
Target: - contracts/parentLayer/ParentBridgeCoreFacet.sol	

Description

When users bridge tokens from L1 to L2, they are required to pay L2 gas fees. These fees are calculated based on the user-specified `gasLimit` and a configurable `l2BaseFee`, which can be updated by the contract owner to reflect current conditions on the Reddio network. To ensure successful execution, users may choose to overestimate the required gas, resulting in an overpayment. However, the current design lacks a mechanism to refund any excess gas fees after execution, leading to potential inefficiencies and unnecessary costs for users.

contracts/parentLayer/ParentBridgeCoreFacet.sol: L133-L149

```
function sendDownwardMessage(
  uint32 payloadType,
  bytes calldata payload,
  uint256 ethAmount,
  uint256 gasLimit,
  uint256 value
) external onlySelf whenNotPaused {
  // compute and deduct the messaging fee to fee vault.
  uint256 fee = estimateCrossMessageFee(gasLimit);
  require(value >= fee + ethAmount, "Insufficient msg.value");
  appendCrossDomainMessage(payloadType, payload, gasLimit);
  emit DownwardMessage(payloadType, payload);
}
function estimateCrossMessageFee(
  uint256 _gasLimit
) public view returns (uint256) {
  return
    IChildGasPriceOracleFacet(address(this)).estimateCrossDomainMessageFee(
      _gasLimit
    );
}
```

Recommendation

It is recommended to add a mechanism for refunding excess cross-chain fees.

Status

This issue has been resolved by the team with commit 398fdf2.

6. Revisit bridge fee logic

Severity: Medium	Category: Business Logic
Target: - contracts/parentLayer/ParentBridgeCoreFacet.sol	

Description

When users bridge tokens from L1 to L2, they are required to pay L2 gas fees. These fees are calculated using the user-provided `gasLimit` and a configurable `l2BaseFee`. On L1, the bridge fee is charged in native Ether, while on L2, the actual gas cost is denominated in RED tokens. Due to potential price volatility between Ether and RED, this mismatch can result in users being charged significantly more or less than the intended amount, leading to unpredictable fee outcomes.

contracts/parentLayer/ParentBridgeCoreFacet.sol: L133-L149

```
function sendDownwardMessage(
  uint32 payloadType,
  bytes calldata payload,
  uint256 ethAmount,
  uint256 gasLimit,
  uint256 value
) external onlySelf whenNotPaused {
  // compute and deduct the messaging fee to fee vault.
  uint256 fee = estimateCrossMessageFee(gasLimit);
  require(value >= fee + ethAmount, "Insufficient msg.value");
  appendCrossDomainMessage(payloadType, payload, gasLimit);
  emit DownwardMessage(payloadType, payload);
}
function estimateCrossMessageFee(
  uint256 _gasLimit
) public view returns (uint256) {
  return
    IChildGasPriceOracleFacet(address(this)).estimateCrossDomainMessageFee(
      _gasLimit
    );
}
```

Recommendation

Consider charging a fee in terms of RED.

Status

This issue has been resolved by the team with commit 398fdf2.

7. Centralization risk

Severity: Medium	Category: Business Logic

Target:
- access/AccessControlFacet.sol
- childLayer/DownwardMessageDispatcherFacet.sol
- parentLayer/ParentBridgeCoreFacet.sol

Description

In the `AccessControlFacet`,`ChildGasPriceOracle` contract, there exists a privileged role called `owner`. The `owner` has the authority to execute some key functions such as `setRoleAdmin`, `pauseBridge` and `setSystemAddress`.

If the `owner`'s private key is compromised, an attacker could trigger these functions to update the role admin or system address.

contracts/access/AccessControlFacet.sol: L57-L59

```
function setRoleAdmin(bytes32 role, bytes32 adminRole) external onlyOwner {
  LibAccessControl._setRoleAdmin(role, adminRole);
}
```

contracts/childLayer/DownwardMessageDispatcherFacet.sol: L57-L62

```
function setSystemAddress(address newSystemAddress) external onlyOwner {
  require(newSystemAddress != address(0), "Invalid address");
  DownwardMessageDispatcherStorage storage ds = _dispatcherStorage();
  emit SystemAddressUpdated(ds.systemAddress, newSystemAddress);
  ds.systemAddress = newSystemAddress;
}
```

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

This issue has been acknowledged by the team.

8. Fee-on-transfer token does not work in evm bridge

Severity: Low	Category: Business Logic
Target: - contracts/parentLayer/tokenMessages/ParentTokenMessageTransmitterFacet.sol	

Description

When bridging ERC20 tokens, the protocol transfers a specified `amount` from the user on L1 and mints an equivalent amount on L2. However, if the token is a *Fee-On-Transfer* ERC20, the actual amount received by the contract on L1 may be less than the input `amount` due to transfer fees. This discrepancy results in over-minting on L2, potentially leading to imbalances and security risks.

contracts/parentLayer/tokenMessages/ParentTokenMessageTransmitterFacet.sol: L158-L196

```
function depositERC20Token(
  address tokenAddress,
  address recipient,
  uint256 amount,
  uint256 gasLimit
) external payable { // payable for gas fee.
  IERC20Token erc20Token = IERC20Token(tokenAddress);
  erc20Token.safeTransferFrom(msg.sender, address(this), amount);
  ...

  IParentBridgeCore bridgeCore = IParentBridgeCore(address(this));
  bytes memory payload = abi.encode(
    ParentERC20TokenLocked({
      tokenAddress: tokenAddress,
      tokenName: erc20Token.name(),
      tokenSymbol: erc20Token.symbol(),
      decimals: erc20Token.decimals(),
      parentSender: msg.sender,
      childRecipient: recipient,
      amount: amount
    })
  ...
);
```

Recommendation

Check the actual received balance if we want to support the FeeOnTransfer token.

Status

This issue has been resolved by the team with commit 398fdf2.

2.3 Informational Findings

9. Incorrect check in _registerERC1155Token

Severity: Informational	Category: Business Logic

Target:
- contracts/childLayer/tokenMessages/ChildTokenMessageProcessorFacet.sol

Description

The `_registerERC1155Token` function checks whether the token has already registered a bridgedContract. However, this check incorrectly uses the incorrect check `getBridgedERC721Token`.

contracts/childLayer/tokenMessages/ChildTokenMessageProcessorFacet.sol: L163-L165

```
function _registerERC1155Token(
  address tokenAddress
) internal returns (address) {
  require(
    LibToken.getBridgedERC721Token(tokenAddress) == address(0),
    "Token already registered!"
  );
  ...
```

Recommendation

Change `getBridgedERC721Token` to `getBridgedERC1155Token`.

Status

This issue has been resolved by the team with commit 398fdf2.

10. redundant code

Severity: Informational	Category: Redundancy

Target:
- contracts/childLayer/DownwardMessageDispatcherFacet.sol
- contracts/timelock/TimelockController.sol

Description

contracts/childLayer/DownwardMessageDispatcherFacet.sol: L40

```
address constant SYSTEM_ADDRESS = 0x0000000000000000000000000000000000000000;

modifier onlySystem() {
  //require(msg.sender == SYSTEM_ADDRESS, "Only system can call this function");
  DownwardMessageDispatcherStorage storage ds = _dispatcherStorage();
  require(
    msg.sender == ds.systemAddress,
    "Only system can call this function"
  );

  _;
}
```

The `SYSTEM_ADDRESS` constant is no longer used in the current system, and the stored system address is used directly instead.

contracts/timelock/TimelockController.sol: L247-L257

```
function _cancel(
  bytes32 operationId
) internal returns (bytes32) {
  // Check operation status
  Operation storage operation = _timelockStorage().operations[operationId];
  if (operation.status == OperationState.Unset) {
    revert TimelockOperationNotFound(operationId);
  }
  if (operation.status != OperationState.Pending) {
    revert("TimelockController: operation cannot be cancelled");
  }
  ...
```

In the `_cancel` function, checking if the state is equal to Pending is sufficient. The previous line of code that checks if the state is equal to Unset is redundant.

Recommendation

It is recommended to optimize redundant code to reduce gas costs.

Status

This issue has been resolved by the team with commit 398fdf2.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 935b352:

File	SHA-1 hash
TimelockFacet.sol	51a2f77d39c0414a8f7a037c186870f7335169b7
TimelockController.sol	e7be841f14b16355c2da0ee02536148fb01a88ee
AccessControlFacet.sol	1df17889ccd572c1ef562ede7c6fe5e84b678563
DiamondInit.sol	2874923a608329c39c730d80afb8262a1cc6cd6d
ParentTokenMessageProcessorFacet.sol	77d164919d78ecb1583dd7a8edbb3f55107a132d
ParentTokenMessageTransmitterFacet.sol	2e5948bd7007d79417479fa4c68a3f74de62277f
LibParentLayerTokenStorage.sol	8dd0646ad17ad8979edf693f6a2ac19676cad4d2
ChildGasPriceOracle.sol	aada2cee480d22a319bdc9150a5221ac77839532
ParentBridgeCoreFacet.sol	ff0e8cfc195efe9ab41693a77fde9c84cad8058c
UpwardMessageDispatcherFacet.sol	8d88f8303a6b27102c9d621eab3af2cbdbd28e6b
ParentStateVerifierFacet.sol	57ff826e330c0ddaaf0a67a64e0f1be2babf0467
LibAccessControl.sol	8ec6d002f3f48aba5e29640564088a7205bcc82d
LibDiamond.sol	81a85e2e3a545636ca6cc5c74036210eb5123816
PatriciaMerkleTrieVerifier.sol	ae506bfd4668216ca70a9e29e2f184e617a28e01
LibAddress.sol	2c222b30339eb1c34c93683ff59428367ea53199
AddressAliasHelper.sol	d2a3504dac7256f2f2cdbe50e4cb05eeb5308c27
WithdrawTrieVerifier.sol	b3dd03096bda520f103c90c2420416a887582f8e
ReentrancyGuard.sol	0422cc49fc149fc8bd380130a6af992b0e6cdb2f
Pausable.sol	63afd62d733c969167a3a8939722e10ca76d8a8d
Ownable.sol	8a4b749c008e852339280b9fac1fab1eef4a9703
CommonStructs.sol	3713ad753373b3a70c938fb4cf7fae686d19051f
Constants.sol	00cf13800192bc7c857d24f63db2cfea25b0e5f5
SelfCallable.sol	279deebf6141e56713952d85874bb217ed3c46c1

AccessControl.sol	d5bf3d627689851137633eec1197b35b57311a99
TimelockStorage.sol	479c18dce8636ecb3c4a7330d06efec5bf3fe499
AccessControlStorage.sol	f0e4b1a9efe2ea25364924d449b1b2774dca1d33
ReentrancyGuardStorage.sol	546223f91e43443ccce55316894e65a55027718c
DiamondCutFacet.sol	2f7af996b402bf0ae190be45dd9d4b714f5c59a3
OwnershipFacet.sol	e23b90131ac85c3b457dd5cfc8bdc7f630054c43
ERC721TokenFactoryFacet.sol	6650e7a98872af7ee4b8001c1e301979454232d6
ERC20TokenFactoryFacet.sol	56836cd2cfb226dc0503f9d550068b975a63f4d3
DiamondLoupeFacet.sol	2fef7c4b29dcbbb8ef13b000fce6a6afecd4bb6d
ERC1155TokenFactoryFacet.sol	e401becf89e4d56276b2f0f8ab4c7d3def0d9a6d
Diamond.sol	c61f2b6181df2ce8d99da1c65c5873f70c84931d
LibChildLayerTokenStorage.sol	e46c1e4aa29ba70dc73a7284a8e4efda360d0d98
ChildTokenMessageTransmitterFacet.sol	52ebf2ce81e5d52506e8084bcd5b14656e110f6c
ChildTokenMessageProcessorFacet.sol	ea0dc8e6f178922e79371a0c65df1574edbf7ebc
DownwardMessageDispatcherFacet.sol	bc276571d0b215e972fd6dfb2646214af0635574
ChildBridgeCoreFacet.sol	30ac97d9ae60ef11cd69d107ee0f5f3242f00e43
ERC721Token.sol	d920b812daef248f0659c8cf99449b0cb2cb3b39
IERC721Token.sol	982b69e5492767dbc602d028023f37a07627fb30
ERC20Token.sol	c8130e332f86fb28c7d09a52dab2081c82f20923
IERC1155Token.sol	c26a5d565c6fa1f1a80730c7700aef022f76dcee
ERC1155Token.sol	39aa0e430ac1c982b67fd14e123e5303c5095e90
IERC20Token.sol	1b9fb9cf6d446496131f2a1fc0a46e1c7d365c70
IChildBridgeCore.sol	8b7e3196deabf04e3fc240d366bccc70d71d6fa9
IERC1155TokenFactory.sol	bdf2cd3e87499ed277080ff54f42a3e01c76ec29
IParentStateVerifier.sol	cd3271c6c8d089b929db4b6c5575865c21c7b84c
IDiamondCut.sol	451846f920e7e5c2266dc114fbb05d1cba0eb16c
IERC721TokenFactory.sol	9724da541493371b3f798ad2b98d382d907e532e
IChildGasPriceOracle.sol	093d91a0d6549988a61c45a0b095323d860f7be8
IParentBridgeCore.sol	6d04a2d015dc418e955d2da49c69fe0084769c9f

IDownwardMessageDispatcher.sol	cea72e100654572d983d9aada29d4e87563f4afc
IERC20TokenFactory.sol	3daadca237f2cfabe7aeca9d0146fa01c284280a
IChildTokenMessageTransmitter.sol	e18dcafbd892131e758b04650347ce4f29b6d333
IDiamondLoupe.sol	d0e75c276baa7dfddd6d9b9ee8d20e29ac4d8ba4
IParentTokenMessageTransmitter.sol	e097c07d513697bd6a054362626e5701aa02b644
ITokenFactory.sol	1a0243da10ad5871794155750cc52f759d53719d
IUpwardMessageDispatcher.sol	18da9732a5f30bdf0beb9b008b8e4dca8fb34a5b
IERC165.sol	422feac48ef86b9ecec570c6a0f7359eeccee15d
IParentTokenMessageProcessor.sol	49618e9ae94682bb2a2b2b8a657803c6954b9a1e
IERC173.sol	02be54e69b3cb5e93a136d82e44b78958a7b78dc
IChildTokenMessageProcessor.sol	2bed2fd98cecd5efb4c0e9117e4c9ab51dfd0f5b