// HALBORN

Chiliz - Bridge Updates

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **November 30th, 2022 - December 12th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	12/4/2022	Gokberk Gulgun
0.2	Document Updates	12/12/2022	Gokberk Gulgun
0.3	Draft Review	12/12/2022	Gabi Urrutia
1.0	Remediation Plan	01/13/2023	Gokberk Gulgun
1.1	Remediation Plan Review	01/15/2023	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Gokberk Gulgun	Halborn	Gokberk.Gulgun@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Chiliz engaged Halborn to conduct a security audit on their bridge smart contract updates beginning on November 30th, 2022 and ending on December 12th, 2022. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided two weeks for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn identified some security risks that were successfully addressed by the Chiliz team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the audit:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions. (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Scanning of solidity files for vulnerabilities, security hot-spots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment. (Brownie, Remix IDE)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.

3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

1. IN-SCOPE TREE & COMMIT

- 094856fd48bb4627842c72e914df09fe9689ad81

IN-SCOPE SMART CONTRACTS (INCLUDE ONLY UPDATES) :

- Bridge.sol.
- NativeHandler.sol.

2. REMEDIATION COMMITS:

LATEST COMMIT ID : 3660ae57c26131770796467d4c123b2274cf6159

- 0ad5bd40bfb1c321e2ff1030f75e202f9762cce6
- add9f81660e4ab58778f4706e00fc4ac4234a153
- 72176b53657bc49d31bfd5f32ef73376505a5935
- 8a0ffc0bea678f15ec4bcf318740b4170ee3ba95
- 49e3c75605632664540b946df8bda7892fc7e884
- 369bb8ba9926ad704e8aebd2692f03d9f343021e
- d715e18b47b17b63d3b0fdeb51dade1b18e535ee
- 0976828419dbaa9feabfa017a1032c4203abf6da
- 8d2c1cfb973eae4935ce167c8ae304ddcb9f843b
- a2814edaea8eaf7c6df0a176846645834b9a8e3e
- bf171d4028b5af946c091baa77a413b3927a44bc
- 3660ae57c26131770796467d4c123b2274cf6159
- 568ac8b609f9945bbbafa454afe394fdd8270458
- 6830bf7e74f4a2bb7675c768f24a8a2f35889df5
- f574625d79d4701d2a89acfd9a5b139c352e9a91
- add9f81660e4ab58778f4706e00fc4ac4234a153

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
4	3	2	3	7

LIKELIHOOD

IMPACT

		(HAL-05) (HAL-06) (HAL-07)		(HAL-01) (HAL-02) (HAL-03) (HAL-04)
	(HAL-08) (HAL-09)			
(HAL-11) (HAL-12)				
	(HAL-10)			
(HAL-13) (HAL-14) (HAL-15) (HAL-17) (HAL-18) (HAL-19)	(HAL-16)			

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - MISSING COMPARISON BETWEEN MSG VALUE AND AMOUNT LEADS TO DRAINING OF THE FUNDS	Critical	SOLVED - 01/13/2023
HAL-02 - INTEGER UNDERFLOW IN THE DEPOSIT FUNCTION	Critical	SOLVED - 01/13/2023
HAL-03 - LACK OF WHITELISTING ON THE CHAIN IDS	Critical	SOLVED - 01/13/2023
HAL-04 - TOKENS CAN BE STUCKED IF THE SAME CHAIN-ID USED IN THE BRIDGE	Critical	SOLVED - 01/13/2023
HAL-05 - LACK OF QUORUM DEFINITION ON THE RELAYERS	High	SOLVED - 01/13/2023
HAL-06 - HANDLER SHOULD ACCEPT PAYMENTS THROUGH ONLY BRIDGE CONTRACT	High	SOLVED - 01/13/2023
HAL-07 - LACK OF REFUND MECHANISM FOR OVERPAYMENT	High	SOLVED - 01/13/2023
HAL-08 - IMPROPER UPPER BOUND ON THE FEE DEFINITION	Medium	SOLVED - 01/13/2023
HAL-09 - USE CALL INSTEAD OF TRANSFER	Medium	SOLVED - 01/13/2023
HAL-10 - MISSING REENTRANCY GUARD	Low	SOLVED - 01/13/2023
HAL-11 - NATSPEC IS NOT COMPATIBLE WITH THE IMPLEMENTATION	Low	SOLVED - 01/13/2023
HAL-12 - MISSING RECEIVER ADDRESS CHECK	Low	SOLVED - 01/13/2023
HAL-13 - REDUNDANT FUNCTION PARAMETER IN THE NATIVEHANDLER	Informational	SOLVED - 01/13/2023
HAL-14 - REDUNDANT ARRAY DEFINITION IN THE CONSTRUCTOR	Informational	SOLVED - 01/13/2023
HAL-15 - DELETE - ABI CODER V2 FOR GAS OPTIMIZATION	Informational	SOLVED - 01/13/2023
HAL-16 - UPGRADE PRAGMA TO AT LEAST 0.8.4	Informational	SOLVED - 01/13/2023

HAL-17 - OPTIMIZE UNSIGNED INTEGER COMPARISON	Informational	SOLVED - 01/13/2023
HAL-18 - CHANGE FUNCTION VISIBILITY FROM PUBLIC TO EXTERNAL	Informational	SOLVED - 01/13/2023
HAL-19 - REDUNDANT SAFEMATH FUNCTIONS USAGE IN CONTRACT	Informational	SOLVED - 01/13/2023

FINDINGS & TECH DETAILS

3.1 (HAL-01) MISSING COMPARISON BETWEEN MSG VALUE AND AMOUNT LEADS TO DRAINING OF THE FUNDS - CRITICAL

Description:

A smart contract that is missing a **msg.value == amount** check may be vulnerable to various types of attacks or errors. This check is typically used to verify that the amount of Ether (or other value token) being sent to the contract is equal to the expected amount.

For example, if a user attempts to transfer 100 tokens but includes a **msg.value** of 200, the contract will not compare the two values and instead process the transfer as if the user were attempting to transfer 200 tokens. This will result in the user being able to withdraw 100 tokens without having the necessary balance, effectively draining 100 tokens from the contract.

However, this contract is missing a **msg.value == amount** check. This means that if a user calls the transfer function and sends more or less Ether than the specified amount, the contract will not detect this and will still attempt to transfer the specified amount of Ether. This could potentially lead to a variety of problems, such as:

- Overpayment: If a user sends more Ether than the specified amount, the excess Ether will be transferred to the recipient, but will not be accounted for by the contract. This means that the user may have overpaid for the transfer, and will not be able to recover the excess Ether.
- Underpayment: If a user sends less Ether than the specified amount, the contract will still attempt to transfer the full amount of Ether. This means that the transfer will fail, and the user will not be able to recover the Ether they sent.
- Invalid state: If a user sends a different amount of Ether than the specified amount, the contract will be in an invalid state, as the

amount of Ether transferred will not match the amount specified in the transfer function. This could potentially cause the contract to malfunction or become unresponsive.

Overall, a smart contract that is missing a **msg.value == amount** check may be vulnerable to various types of errors or attacks. It is important to include this check in order to ensure that the contract operates correctly and securely.

Code Location:

Bridge.sol#L308

Listing 1

```
    function deposit(uint8 destinationChainID, bytes32 resourceID, bytes calldata data) external payable whenNotPaused {
        require(msg.value >= _fee, "invalid value");

        address handler = _resourceIDToHandlerAddress[resourceID];
        require(handler != address(0), "resourceID not mapped to handler");

        uint64 depositNonce = ++_depositCounts[destinationChainID];
        _depositRecords[depositNonce][destinationChainID] = data;

        IDepositExecute depositHandler = IDepositExecute(handler);
        if (msg.value > 0) {
            uint256 valueToSend = msg.value - _fee;
            if (valueToSend > 0) {
                payable(handler).transfer(valueToSend);
            }
        }
        depositHandler.deposit(resourceID, destinationChainID, depositNonce, msg.sender, data);

        emit Deposit(destinationChainID, resourceID, depositNonce);
    }

```

Proof of concept:

```
test/ExploitTester.t.sol:42:9: Warning: Failure condition of 'send' ignored. Consider using 'transfer' instead.
        msg.sender.send(5000 ether);
        ^------------------------^


src/handlers/NativeHandler.sol:101:9: Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
        bytes   calldata data
        ^-------------------^


src/handlers/NativeHandler.sol:138:50: Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
    function executeProposal(bytes32 resourceID, bytes calldata data) external override onlyBridge {
                                                 ^----------------^


src/handlers/NativeHandler.sol:175:23: Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
    function withdraw(address tokenAddress, address recipient, uint amount) external override onlyBridge {
                      ^-----------------^


src/Bridge.sol:89:5: Warning: Function state mutability can be restricted to view
    function _onlyAdminOrRelayer() private {
    ^ (Relevant source part starts here and spans across multiple lines).


src/Bridge.sol:94:5: Warning: Function state mutability can be restricted to view
    function _onlyAdmin() private {
    ^ (Relevant source part starts here and spans across multiple lines).


src/Bridge.sol:98:5: Warning: Function state mutability can be restricted to view
    function _onlyRelayers() private {
    ^ (Relevant source part starts here and spans across multiple lines).


src/handlers/HandlerHelpers.sol:30:5: Warning: Function state mutability can be restricted to view
    function _onlyBridge() private {
    ^ (Relevant source part starts here and spans across multiple lines).


Running 2 tests for test/ExploitTester.t.sol:ExploitTester
[PASS] testExploit() (gas: 156301)
Traces:
  [156301] ExploitTester::testExploit()
    ├─ [144061] Bridge::deposit(value: 500)(2, 0xfdae1ba7c826abdc4c99903c8056f82a1a04a615c94c895c32d24a82e8ecf7cd, 0x)
    │   ├─ [55] NativeHandler::fallback(value: 400)()
    │   │   └─ ← ()
    │   ├─ [102006] NativeHandler::deposit(0xfdae1ba7c826abdc4c99903c8056f82a1a04a615c94c895c32d24a82e8ecf7cd, 2, 1, ExploitTester: [0xb4c79da80f259C7Aee6E5b2Aa729021864227e84], 0x)
    │   │   ├─ [410] Bridge::_fee()
    │   │   │   └─ ← 100
    │   │   ├─ emit Deposit(amount: 100000000000000000, depositNonce: 1)
    │   │   └─ ← ()
    │   ├─ emit Deposit(destinationChainID: 2, resourceID: 0xfdae1ba7c826abdc4c99903c8056f82a1a04a615c94c895c32d24a82e8ecf7cd, depositNonce: 1)
    │   └─ ← ()
    └─ ← ()
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

Based on our analysis, the following actions are recommended:

- Add a comparison between the msg.value and the amount in the deposit function.
- Test the updated deposit function to ensure that it is functioning properly and preventing the draining of funds.

Implementing these recommendations will improve the security and stability of the system by preventing the accidental or malicious drain of contract funds.

Remediation Plan:

SOLVED: The Chiliz team solved the issue by adding a validation on the Handler and Bridge contracts.

Commit ID: 0ad5bd40bfb1c321e2ff1030f75e202f9762cce6

3.2 (HAL-02) INTEGER UNDERFLOW IN THE DEPOSIT FUNCTION - CRITICAL

Description:

In the Solidity programming language, an integer underflow occurs when an integer variable is decremented below its minimum possible value. In Solidity, all integer variables have a fixed size and a fixed range of possible values. For example, a uint8 variable can hold a value between 0 and 255 (inclusive), and a int256 variable can hold a value between -2^255 and 2^255-1 (inclusive).

The deposit function in the contract contains an integer underflow issue when calculating the amountMinusFee variable. This can happen when the value of the “amount” variable is less than the value of the bridgeFee variable. If an integer underflow occurs in the deposit function, the result will be a very large positive number. This can cause problems in the contract, as the large positive value of amountMinusFee will be treated as a valid amount. For example, it could be recorded in the depositRecords mapping and used to update the balance of the depositor.

Code Location:

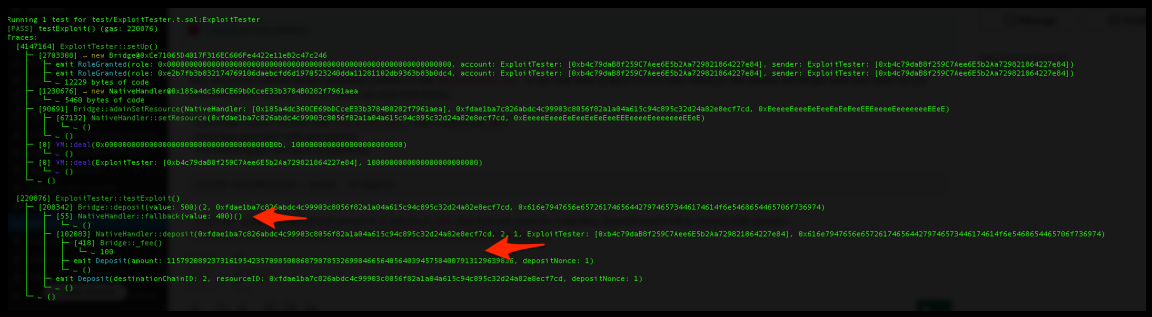
NativeHandler.sol#L91

Listing 2

```
    function deposit(
        bytes32 resourceID,
        uint8   destinationChainID,
        uint64  depositNonce,
        address depositer,
        bytes   calldata data
    ) external override onlyBridge {
        bytes   memory recipientAddress;
        uint256        amount;
        uint256        lenRecipientAddress;

```

```
12          assembly {
13              amount := calldataload(0xC4)
14              recipientAddress := mload(0x40)
15              lenRecipientAddress := calldataload(0xE4)
16              mstore(0x40, add(0x20, add(recipientAddress,
↳ lenRecipientAddress)))
17
18              calldatacopy(
19                  recipientAddress, // copy to
↳ destinationRecipientAddress
20                  0xE4, // copy from calldata @ 0x104
21                  sub(calldatasize(), 0xE) // copy size (
↳ calldatasize - 0x104)
22              )
23          }
24
25          address tokenAddress = _resourceIDToTokenContractAddress[
↳ resourceID];
26          require(_contractWhitelist[tokenAddress], "provided
↳ tokenAddress is not whitelisted");
27
28          uint256 bridgeFee = IBridge(_bridgeAddress)._fee();
29          uint256 amountMinusFee = amount - bridgeFee;
30          require(amountMinusFee > 0, "Invalid amount");
31
32          _depositRecords[destinationChainID][depositNonce] =
↳ DepositRecord(
33              tokenAddress,
34              uint8(lenRecipientAddress),
35              destinationChainID,
36              resourceID,
37              recipientAddress,
38              depositer,
39              amountMinusFee
40          );
41      }
```

Proof of concept:



Risk Level:

Likelihood - 5

Impact - 5

Recommendation:

To address the issue of integer underflow leading to the draining of funds in the Solidity contract, we recommend the following actions:

- Review the code of the contract to identify all instances where integer underflow can occur.
- Implement appropriate safeguards to prevent integer underflow in each of these instances. This may involve using the SafeMath library, using the “require” statement to check for underflow, or using other techniques.
- Test the updated contract to ensure that it is functioning properly, and that integer underflow is no longer possible.
- Monitor the contract for any further issues related to integer underflow and address them as necessary.

Implementing these recommendations will help to prevent the accidental or malicious draining of funds from the contract due to integer underflow, improving the security and stability of the system.

Remediation Plan:

SOLVED: The Chiliz team solved the issue by upgrading to **pragma 0.8.17** in contracts.

Commit ID: add9f81660e4ab58778f4706e00fc4ac4234a153

3.3 (HAL-03) LACK OF WHITELISTING ON THE CHAIN IDS - CRITICAL

Description:

If chain IDs are not whitelisted in a smart contract, it may be vulnerable to various types of attacks or errors. A whitelist is a list of approved or allowed values, and in the context of a smart contract, it is used to specify which chain IDs are allowed to interact with the contract.

This could potentially lead to a variety of problems, such as:

- Loss of tokens: If the contract attempts to transfer tokens to an unauthorized chain ID, the tokens may be lost or stolen, and the user will not be able to recover them.
- Invalid state: If the contract attempts to transfer tokens to an unauthorized chain ID, the contract's internal state may become inconsistent or invalid. For example, if the contract maintains a record of all token transfers, it may record an invalid transfer to an unauthorized chain ID, which could cause the contract to malfunction or become unresponsive.
- Denial of service: If an attacker can call the transfer function with an unauthorized chain ID, they may be able to prevent legitimate users from transferring tokens to certain chain IDs. This could potentially cause a denial of service, as users would not be able to transfer tokens to the affected chain IDs.

Overall, a smart contract that does not have a whitelist of allowed chain IDs may be vulnerable to various types of attacks or errors. It is important to include a whitelist of allowed chain IDs in order to ensure that the contract operates correctly and securely.

Code Location:

NativeHandler.sol#L91

Listing 3

```
    function deposit(
        bytes32 resourceID,
        uint8   destinationChainID,
        uint64  depositNonce,
        address depositer,
        bytes   calldata data
    ) external override onlyBridge {
        bytes   memory recipientAddress;
        uint256        amount;
        uint256        lenRecipientAddress;

        assembly {
            amount := calldataload(0xC4)
            recipientAddress := mload(0x40)
            lenRecipientAddress := calldataload(0xE4)
            mstore(0x40, add(0x20, add(recipientAddress,
↳ lenRecipientAddress)))

            calldatacopy(
                recipientAddress, // copy to
↳ destinationRecipientAddress
                0xE4, // copy from calldata @ 0x104
                sub(calldatasize(), 0xE) // copy size (
↳ calldatasize - 0x104)
            )
        }

        address tokenAddress = _resourceIDToTokenContractAddress[
↳ resourceID];
        require(_contractWhitelist[tokenAddress], "provided
↳ tokenAddress is not whitelisted");

        uint256 bridgeFee = IBridge(_bridgeAddress)._fee();
        uint256 amountMinusFee = amount - bridgeFee;
        require(amountMinusFee > 0, "Invalid amount");

        _depositRecords[destinationChainID][depositNonce] =
↳ DepositRecord(
            tokenAddress,
            uint8(lenRecipientAddress),
            destinationChainID,
            resourceID,
            recipientAddress,
```

```
38                depositer,
39                amountMinusFee
40            );
41      }
```

Proof of concept:

```
src/handlers/NativeHandler.sol:101:9: Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
        bytes   calldata data
        ^-----------------^


src/handlers/NativeHandler.sol:138:50: Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
    function executeProposal(bytes32 resourceID, bytes calldata data) external override onlyBridge {
                                                 ^---------------^


src/handlers/NativeHandler.sol:175:23: Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
    function withdraw(address tokenAddress, address recipient, uint amount) external override onlyBridge {
                      ^-----------------^


src/Bridge.sol:89:5: Warning: Function state mutability can be restricted to view
    function _onlyAdminOrRelayer() private {
    ^ (Relevant source part starts here and spans across multiple lines).


src/Bridge.sol:94:5: Warning: Function state mutability can be restricted to view
    function _onlyAdmin() private {
    ^ (Relevant source part starts here and spans across multiple lines).


src/Bridge.sol:98:5: Warning: Function state mutability can be restricted to view
    function _onlyRelayers() private {
    ^ (Relevant source part starts here and spans across multiple lines).


src/handlers/HandlerHelpers.sol:30:5: Warning: Function state mutability can be restricted to view
    function _onlyBridge() private {
    ^ (Relevant source part starts here and spans across multiple lines).


Running 2 tests for test/ExploitTester.t.sol:ExploitTester
[PASS] testExploit() (gas: 156301)
Traces:
  [156301] ExploitTester::testExploit()
    ├─ [144061] Bridge::deposit{value: 500}(111, 0xfdae1ba7c826abdc4c99903c8056f82a1a04a615c94c895c32d24a82e8ecf7cd, 0x)
    │   ├─ [55] NativeHandler::fallback{value: 400}()
    │   │   └─ ← ()
    │   ├─ [102006] NativeHandler::deposit(0xfdae1ba7c826abdc4c99903c8056f82a1a04a615c94c895c32d24a82e8ecf7cd, 111, 1, ExploitTester: [0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84], 0x)
    │   │   ├─ [418] Bridge::_fee()
    │   │   │   └─ ← 100
    │   │   ├─ emit Deposit(amount: 100000000000000000, depositNonce: 1)
    │   │   └─ ← ()
    │   ├─ emit Deposit(destinationChainID: 111, resourceID: 0xfdae1ba7c826abdc4c99903c8056f82a1a04a615c94c895c32d24a82e8ecf7cd, depositNonce: 1)
    │   └─ ← ()
    └─ ← ()
```

Proof of concept code:

Listing 4

```
contract Tester is DSTest {

      address public user1 = address(0xB0B);
      Bridge internal bridge;
      NativeHandler internal nativeHandler;
      CheatCodes cheats = CheatCodes(0
↳ x7109709ECfa91a80626fF3989D68f67F5b1DD12D);
    function setUp() public {
          address [] memory path = new address[](2);
          path[0] = address(this);
          path[1] = address(this);
          bridge = new Bridge(1,path,5,100,100);

          bytes32 [] memory pathByteGG = new bytes32[](2);
```

```
14          pathByteGG[0] = keccak256(abi.encodePacked("User1"));
15          pathByteGG[1] = keccak256(abi.encodePacked("User2"));
16
17          nativeHandler = new NativeHandler(address(bridge),
↳ pathByteGG,path,path);
18
19          bytes32 tokenResource = keccak256(abi.encodePacked("NATIVE
↳ "));
20          bridge.adminSetResource(address(nativeHandler),
↳ tokenResource,0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE);
21
22          cheats.deal(user1,1_000_000 ether);
23          cheats.deal(address(this), 1_000_000 ether);
24      }
25
26      function testExploit() public {
27          bytes memory test = "";
28          bytes32 tokenResource = keccak256(abi.encodePacked("NATIVE
↳ "));
29          bridge.deposit{value: 5e2}(anyChainId as uint8 Type,
↳ tokenResource, test);
30      }
31  }
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

One potential recommendation for addressing the issue of lack of whitelisting on the chain IDs is to implement a whitelisting mechanism in the contract. This could involve adding a mapping or array that stores the approved chain IDs, and checking this whitelist before allowing any transactions to be processed on the corresponding chain.

For example, the contract could contain a `chainIDWhitelist` mapping, where each key is a chain ID and the corresponding value is a boolean indicating whether the chain ID is approved. The `deposit` function could then be

updated to check the chainIDWhitelist before processing any transactions.

Remediation Plan:

SOLVED: The Chiliz team solved the issue by adding a whitelist on chain ids.

Commit ID: 72176b53657bc49d31bfd5f32ef73376505a5935

3.4 (HAL-04) TOKENS CAN BE STUCKED IF THE SAME CHAIN-ID USED IN THE BRIDGE - CRITICAL

Description:

In the current implementation of the token bridge, the chain-id is used to identify the source and destination chains when transferring tokens. If the same chain-id is used on both sides of the bridge, the system is unable to determine which chain the tokens are coming from and which chain they are going to, and the tokens become stuck.

Code Location:

NativeHandler.sol#L91

Listing 5

```
    function deposit(
        bytes32 resourceID,
        uint8   destinationChainID,
        uint64  depositNonce,
        address depositer,
        bytes   calldata data
    ) external override onlyBridge {
        bytes   memory recipientAddress;
        uint256        amount;
        uint256        lenRecipientAddress;

        assembly {
            amount := calldataload(0xC4)
            recipientAddress := mload(0x40)
            lenRecipientAddress := calldataload(0xE4)
            mstore(0x40, add(0x20, add(recipientAddress,
↳ lenRecipientAddress)))

            calldatacopy(
                recipientAddress, // copy to
↳ destinationRecipientAddress
```

```
                0xE4, // copy from calldata @ 0x104
                sub(calldatasize(), 0xE) // copy size (
↳ calldatasize - 0x104)
            )
        }
        address tokenAddress = _resourceIDToTokenContractAddress[
↳ resourceID];
        require(_contractWhitelist[tokenAddress], "provided
↳ tokenAddress is not whitelisted");

        uint256 bridgeFee = IBridge(_bridgeAddress)._fee();
        uint256 amountMinusFee = amount - bridgeFee;
        require(amountMinusFee > 0, "Invalid amount");

        _depositRecords[destinationChainID][depositNonce] =
↳ DepositRecord(
            tokenAddress,
            uint8(lenRecipientAddress),
            destinationChainID,
            resourceID,
            recipientAddress,
            depositer,
            amountMinusFee
        );
    }
```

Proof of concept:

Listing 6

```
contract Tester is DSTest {
     address public user1 = address(0xB0B);
     Bridge internal bridge;
     NativeHandler internal nativeHandler;
     CheatCodes cheats = CheatCodes(0
↳ x7109709ECfa91a80626fF3989D68f67F5b1DD12D);
   function setUp() public {
        address [] memory path = new address[](2);
        path[0] = address(this);
        path[1] = address(this);
        bridge = new Bridge(1,path,5,100,100);

```

```
12          bytes32 [] memory pathByteGG = new bytes32[](2);
13          pathByteGG[0] = keccak256(abi.encodePacked("User1"));
14          pathByteGG[1] = keccak256(abi.encodePacked("User2"));
15
16          nativeHandler = new NativeHandler(address(bridge),
↳ pathByteGG,path,path);
17
18          bytes32 tokenResource = keccak256(abi.encodePacked("NATIVE
↳ "));
19          bridge.adminSetResource(address(nativeHandler),
↳ tokenResource,0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE);
20
21          cheats.deal(user1,1_000_000 ether);
22          cheats.deal(address(this), 1_000_000 ether);
23      }
24
25      function testExploit() public {
26          bytes memory test = "";
27          bytes32 tokenResource = keccak256(abi.encodePacked("NATIVE
↳ "));
28          ## Chain operates with chain ID as 1.
29          bridge.deposit{value: 5e2}(1, tokenResource, test);
30      }
31  }
```

Proof of concept Steps:

- Set up a token bridge with the same chain-id on both sides of the bridge.
- Attempt to transfer tokens across the bridge.
- Observe that the tokens become stuck and are not transferred to the destination chain.

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

Based on our analysis, it is recommended the following actions:

- Update the implementation of the token bridge to prevent the use of the same chain-id on both sides of the bridge.
- Implement a validation step that checks the chain-id of the source and destination chains and ensures that they are different.
- Test the updated token bridge to ensure that it is functioning properly and preventing tokens from becoming stuck.

Remediation Plan:

SOLVED: The Chiliz team solved the issue by adding the chain id validation.

Commit ID: 72176b53657bc49d31bfd5f32ef73376505a5935

3.5 (HAL-05) LACK OF QUORUM DEFINITION ON THE RELAYERS - HIGH

Description:

The lack of a quorum definition on the relayer threshold in this system could potentially result in low participation and lack of consensus on important decisions. This is because without a minimum number of participants required to reach consensus, it is possible for some relayers to make decisions that do not reflect the views of the majority of relayers. This can hinder the effectiveness and efficiency of the system, and can lead to conflicts and disputes among relayers. It is important for the system to define a quorum for the relayer threshold to ensure that sufficient participation is achieved and that decisions are made with the support and consensus of the majority of relayers. This can help to promote collaboration and consensus among relayers and improve the functioning and reliability of the system.

Code Location:

Bridge.sol#L201

Listing 7

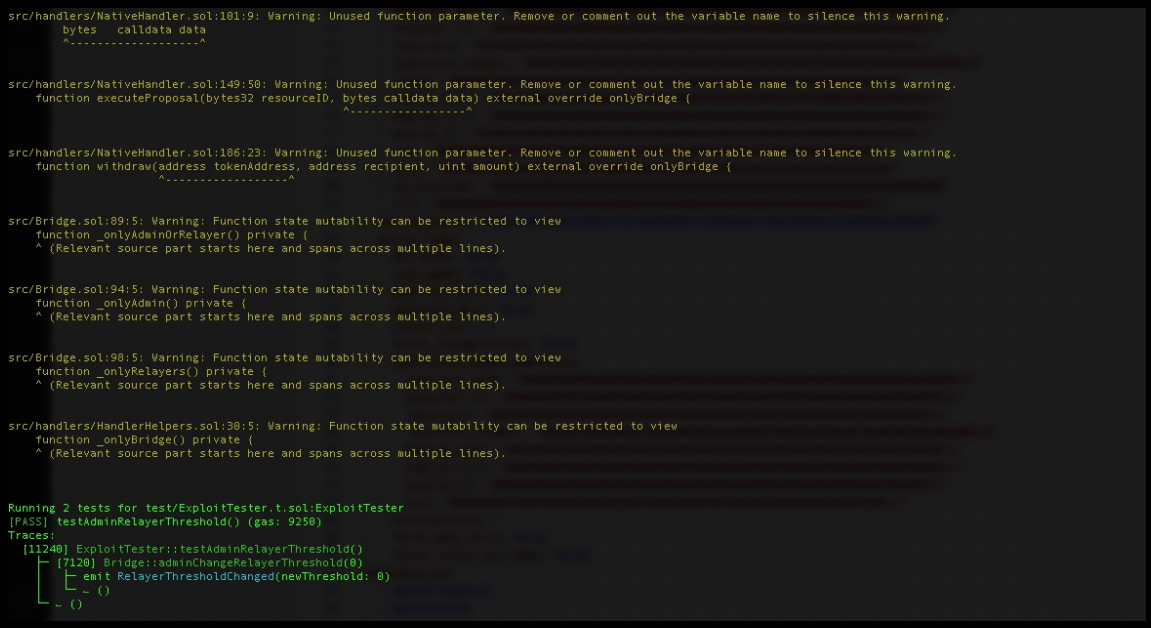
```
    /**
        @notice Modifies the number of votes required for a
↳ proposal to be considered passed.
        @notice Only callable by an address that currently has the
↳  admin role.
        @param newThreshold Value {_relayerThreshold} will be
↳ changed to.
        @notice Emits {RelayerThresholdChanged} event.
     */
    function adminChangeRelayerThreshold(uint newThreshold)
↳ external onlyAdmin {
        _relayerThreshold = newThreshold;
        emit RelayerThresholdChanged(newThreshold);
    }

```

Proof of concept code:

Listing 8

```
contract Tester is DSTest {

      address public user1 = address(0xB0B);
      Bridge internal bridge;
      NativeHandler internal nativeHandler;
      CheatCodes cheats = CheatCodes(0
↳ x7109709ECfa91a80626fF3989D68f67F5b1DD12D);
    function setUp() public {
          address [] memory path = new address[](2);
          path[0] = address(this);
          path[1] = address(this);
          bridge = new Bridge(1,path,5,100,100);

          bytes32 [] memory pathByteGG = new bytes32[](2);
          pathByteGG[0] = keccak256(abi.encodePacked("User1"));
          pathByteGG[1] = keccak256(abi.encodePacked("User2"));

          nativeHandler = new NativeHandler(address(bridge),
↳ pathByteGG,path,path);

          bytes32 tokenResource = keccak256(abi.encodePacked("NATIVE
↳ "));
          bridge.adminSetResource(address(nativeHandler),
↳ tokenResource,0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE);

          cheats.deal(user1,1_000_000 ether);
          cheats.deal(address(this), 1_000_000 ether);
    }


    function testAdminRelayerThreshold() public {
          bridge.adminChangeRelayerThreshold(0);
    }
}
```

Proof of concept:



Risk Level:

Likelihood - 3
Impact - 5

Recommendation:

It is recommended that the system define a quorum for the relayer threshold to ensure that sufficient participation is achieved and that decisions are made with the support and consensus of the majority of relayers. This can be done by setting a specific minimum number of relayers required to reach consensus on key decisions, or by using a formula or algorithm to dynamically calculate the quorum based on factors such as the total number of active relayers or the value of the decision being made. By defining a quorum for the relayer threshold, the system can help to promote collaboration and consensus among relayers and improve the functioning and reliability of the system.

Remediation Plan:

SOLVED: The Chiliz team solved the issue by adding the quorum relayer threshold definition.

Commit ID: 8a0ffc0bea678f15ec4bcf318740b4170ee3ba95

3.6 (HAL-06) HANDLER SHOULD ACCEPT PAYMENTS THROUGH ONLY BRIDGE CONTRACT - HIGH

Description:

The ability of the handler to accept payments through any user using the receive() function in Solidity in this system could potentially result in loss of funds. To prevent sending funds through the receive() and fallback() functions in Solidity, you can implement checks and safeguards in your contract to ensure that only payments from specific approved contracts are accepted. This can be done by specifying a list of approved contracts that are allowed to send funds to your contract, and by implementing appropriate checks to verify that the sender of the payment is one of these approved contracts. For example, you can use the msg.sender variable in Solidity to check the sender of the payment, and only accept payments from the approved contracts. You can also use the require() function to enforce these checks and prevent payments from unauthorized contracts from being accepted. It is important to carefully review and test your contract to ensure that it properly implements these checks and safeguards.

Code Location:

NativeHandler.sol#L187

Listing 9

```
1     receive() external payable { }
```

Proof of concept:

```
Compiler run successful
test/ExploitTester.t.sol:42:9: Warning: Failure condition of 'send' ignored. Consider using 'transfer' instead.
        msg.sender.send(5000 ether);
        ^-------------------------^


src/handlers/NativeHandler.sol:101:9: Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
        bytes   calldata data
        ^-------------------^


src/handlers/NativeHandler.sol:149:50: Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
    function executeProposal(bytes32 resourceID, bytes calldata data) external override onlyBridge {
                                                 ^-----------------^


src/handlers/NativeHandler.sol:186:23: Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
    function withdraw(address tokenAddress, address recipient, uint amount) external override onlyBridge {
                      ^------------------^


src/Bridge.sol:89:5: Warning: Function state mutability can be restricted to view
    function _onlyAdminOrRelayer() private {
    ^ (Relevant source part starts here and spans across multiple lines).


src/Bridge.sol:94:5: Warning: Function state mutability can be restricted to view
    function _onlyAdmin() private {
    ^ (Relevant source part starts here and spans across multiple lines).


src/Bridge.sol:98:5: Warning: Function state mutability can be restricted to view
    function _onlyRelayers() private {
    ^ (Relevant source part starts here and spans across multiple lines).


src/handlers/HandlerHelpers.sol:30:5: Warning: Function state mutability can be restricted to view
    function _onlyBridge() private {
    ^ (Relevant source part starts here and spans across multiple lines).


Running 1 test for test/ExploitTester.t.sol:ExploitTester
[PASS] testHandlerDepositDirectly() (gas: 7021)
Traces:
  [7021] ExploitTester::testHandlerDepositDirectly()
    ├─ [0] 0x00a329c0648769A73afAc7F9381E08FB43dBEA72::fallback{value: 5000000000000000000000}()
    │   └─ ← ()
    └─ ← ()
```

Proof of concept:

Listing 10

```
contract Tester is DSTest {

      address public user1 = address(0xB0B);
      Bridge internal bridge;
      NativeHandler internal nativeHandler;
      CheatCodes cheats = CheatCodes(0
↳ x7109709ECfa91a80626fF3989D68f67F5b1DD12D);
   function setUp() public {
         address [] memory path = new address[](2);
         path[0] = address(this);
         path[1] = address(this);
         bridge = new Bridge(1,path,5,100,100);

         bytes32 [] memory pathByteGG = new bytes32[](2);
         pathByteGG[0] = keccak256(abi.encodePacked("User1"));
         pathByteGG[1] = keccak256(abi.encodePacked("User2"));

```

```
17          nativeHandler = new NativeHandler(address(bridge),
↳ pathByteGG,path,path);
18
19          bytes32 tokenResource = keccak256(abi.encodePacked("NATIVE
↳ "));
20          bridge.adminSetResource(address(nativeHandler),
↳ tokenResource,0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE);
21
22          cheats.deal(user1,1_000_000 ether);
23          cheats.deal(address(this), 1_000_000 ether);
24      }
25
26      function testHandlerDepositDirectly() public {
27          address(nativeHandler).transfer(5000 ether);
28      }
29  }
```

Risk Level:

Likelihood - 3
Impact - 5

Recommendation:

It is recommended that the system implement checks and safeguards to restrict the handler to only accept payments through bridge contract using the receive() function in Solidity. This can be done by specifying a list of approved contracts that are allowed to send funds to the handler, and by implementing appropriate checks to verify that the sender of the payment is one of these approved contracts. For example, the contract can use the msg.sender variable in Solidity to check the sender of the payment, and only accept payments from the approved contracts.

Remediation Plan:

SOLVED: The Chiliz team solved the issue by adding a necessary check in the handler.

Commit ID: 49e3c75605632664540b946df8bda7892fc7e884

3.7 (HAL-07) LACK OF REFUND MECHANISM FOR OVERPAYMENT - HIGH

Description:

ERC20Handler contract handles ERC20 deposits and deposit executions. In the given code, the deposit function allows users to deposit native tokens, but it does not include any mechanism for refunding the user if they accidentally send more tokens than required when interacting with ERC20Handler. If a user accidentally sends more native tokens than required, they will lose their excess funds without any way to recover them. This can lead to user dissatisfaction and a loss of trust in the contract. Additionally, the contract may be at risk of attack from users who deliberately attempt to overpay in order to steal the excess funds.

Code Location:

Bridge.sol#L308

Listing 11

```
    function deposit(uint8 destinationChainID, bytes32 resourceID,
↳  bytes calldata data) external payable whenNotPaused {
        require(msg.value >= _fee, "invalid value");

        address handler = _resourceIDToHandlerAddress[resourceID];
        require(handler != address(0), "resourceID not mapped to
↳ handler");

        uint64 depositNonce = ++_depositCounts[destinationChainID
↳ ];
        _depositRecords[depositNonce][destinationChainID] = data;

        IDepositExecute depositHandler = IDepositExecute(handler);
        if (msg.value > 0) {
            uint256 valueToSend = msg.value - _fee;
            if (valueToSend > 0) {
                payable(handler).transfer(valueToSend);
            }
```

```
16          }
17          depositHandler.deposit(resourceID, destinationChainID,
↳ depositNonce, msg.sender, data);
18
19          emit Deposit(destinationChainID, resourceID, depositNonce)
↳ ;
20      }
21
```

Proof of concept:

Listing 12

```
1     function testRefund() public {
2           bytes memory test = "";
3           bytes32 tokenResource = keccak256(abi.encodePacked("ERC20"
↳ ));
4           bridge.deposit{value: 5e2}(111, tokenResource, test);
5     }
```

Risk Level:

Likelihood - 3
Impact - 5

Recommendation:

To address this issue, the deposit function should include a mechanism for refunding the user if they accidentally send more native tokens than required. This can be implemented by checking the amount of tokens received in the deposit function and refunding the excess amount to the user using the transfer function. This will ensure that users are protected from losing their funds due to overpayment, and will help to maintain trust in the contract.

Remediation Plan:

SOLVED: The Chiliz team solved the issue by adding a refund mechanism in the related handler.

Commit ID: 369bb8ba9926ad704e8aebd2692f03d9f343021e

3.8 (HAL-08) IMPROPER UPPER BOUND ON THE FEE DEFINITION - MEDIUM

Description:

The improper definition of an upper bound on fees in this system could potentially result in excessive fees and harm the user experience. This is because the upper bound is not adequately constrained, and it is possible for fees to be set at levels that are much higher than expected or reasonable. This can create a financial burden for users and discourage the use of the system. In addition, excessive fees can also lead to centralization and concentration of power in the hands of those who can afford to pay high fees, which can create additional security and integrity risks for the system. It is important for the system to properly define the upper bound on fees to ensure that fees are not set at excessive levels and to protect the user experience and the security of the system.

Code Location:

Bridge.sol#L298

Listing 13

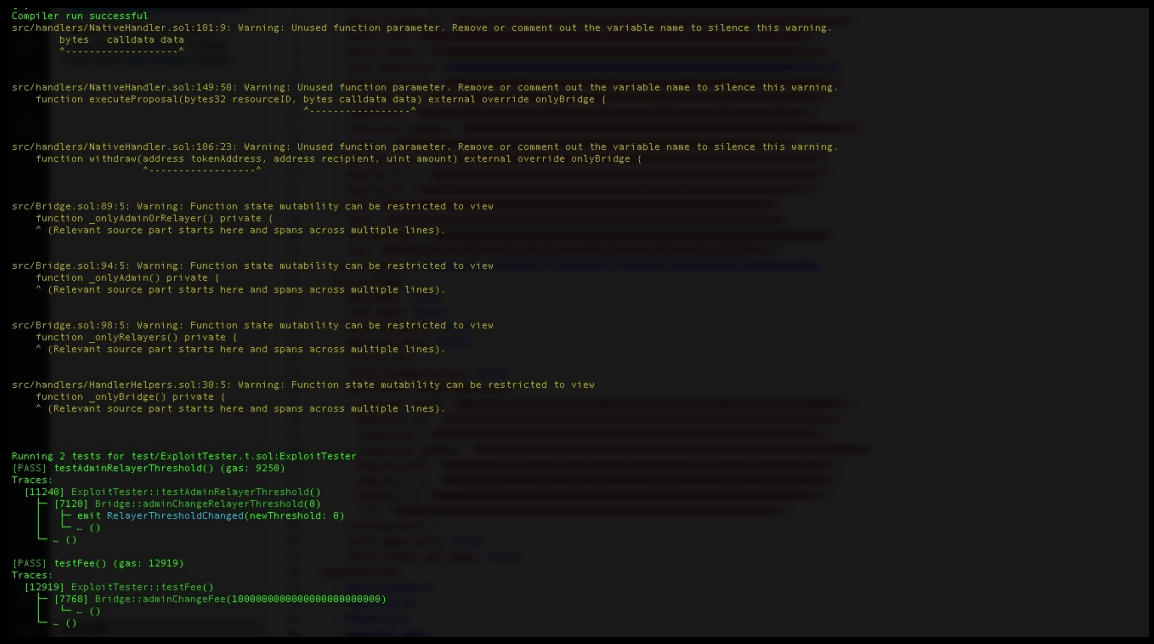
```
    /**
        @notice Changes deposit fee.
        @notice Only callable by admin.
        @param newFee Value {_fee} will be updated to.
     */
    function adminChangeFee(uint newFee) external onlyAdmin {
        require(_fee != newFee, "Current fee is equal to new fee")
↳ ;
        _fee = newFee;
    }
```

Proof of concept code:

Listing 14

```
contract Tester is DSTest {

      address public user1 = address(0xB0B);
      Bridge internal bridge;
      NativeHandler internal nativeHandler;
      CheatCodes cheats = CheatCodes(0
↳ x7109709ECfa91a80626fF3989D68f67F5b1DD12D);
    function setUp() public {
          address [] memory path = new address[](2);
          path[0] = address(this);
          path[1] = address(this);
          bridge = new Bridge(1,path,5,100,100);

          bytes32 [] memory pathByteGG = new bytes32[](2);
          pathByteGG[0] = keccak256(abi.encodePacked("User1"));
          pathByteGG[1] = keccak256(abi.encodePacked("User2"));

          nativeHandler = new NativeHandler(address(bridge),
↳ pathByteGG,path,path);

          bytes32 tokenResource = keccak256(abi.encodePacked("NATIVE
↳ "));
          bridge.adminSetResource(address(nativeHandler),
↳ tokenResource,0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE);

          cheats.deal(user1,1_000_000 ether);
          cheats.deal(address(this), 1_000_000 ether);
    }

    function testFee() public {
          bridge.adminChangeFee(100000000000000000000000);
    }

}
```

Proof of concept:



```
Compiler run successful
src/handlers/NativeHandler.sol:101:9: Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
        bytes   calldata data
        ^-------------------^


src/handlers/NativeHandler.sol:149:50: Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
    function executeProposal(bytes32 resourceID, bytes calldata data) external override onlyBridge {
                                                 ^----------------^


src/handlers/NativeHandler.sol:186:23: Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
    function withdraw(address tokenAddress, address recipient, uint amount) external override onlyBridge {
                      ^-----------------^


src/Bridge.sol:89:5: Warning: Function state mutability can be restricted to view
    function _onlyAdminOrRelayer() private {
    ^ (Relevant source part starts here and spans across multiple lines).


src/Bridge.sol:94:5: Warning: Function state mutability can be restricted to view
    function _onlyAdmin() private {
    ^ (Relevant source part starts here and spans across multiple lines).


src/Bridge.sol:98:5: Warning: Function state mutability can be restricted to view
    function _onlyRelayers() private {
    ^ (Relevant source part starts here and spans across multiple lines).


src/handlers/HandlerHelpers.sol:30:5: Warning: Function state mutability can be restricted to view
    function _onlyBridge() private {
    ^ (Relevant source part starts here and spans across multiple lines).



Running 2 tests for test/ExploitTester.t.sol:ExploitTester
[PASS] testAdminRelayerThreshold() (gas: 9250)
Traces:
  [11240] ExploitTester::testAdminRelayerThreshold()
    ├─ [7120] Bridge::adminChangeRelayerThreshold(0)
    │   ├─ emit RelayerThresholdChanged(newThreshold: 0)
    │   └─ ()
    └─ ()

[PASS] testFee() (gas: 12919)
Traces:
  [12919] ExploitTester::testFee()
    ├─ [7768] Bridge::adminChangeFee(100000000000000000000000)
    │   └─ ()
    └─ ()
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

It is recommended that the system properly define the upper bound on fees to ensure that fees are not set at excessive levels and to protect the user experience. This can be done by setting a specific maximum limit on fees, or by using a formula or algorithm to dynamically calculate the maximum fee based on factors such as the current market conditions or the value of the transaction. By properly defining the upper bound on fees, the system can help to prevent excessive fees and improve the user experience.

Remediation Plan:

SOLVED: The Chiliz team solved the issue by adding an upper limit to the fee.

Commit ID: d715e18b47b17b63d3b0fdeb51dade1b18e535ee

3.9 (HAL-09) USE CALL INSTEAD OF TRANSFER - MEDIUM

Description:

The transfer function is not recommended for sending native token due to its, 2300 gas unit limit. Instead, call can be used to circumvent the gas limit.

Code Location:

Bridge.sol#L308

Listing 15

```
    function deposit(uint8 destinationChainID, bytes32 resourceID, bytes calldata data) external payable whenNotPaused {
        require(msg.value >= _fee, "invalid value");

        address handler = _resourceIDToHandlerAddress[resourceID];
        require(handler != address(0), "resourceID not mapped to handler");

        uint64 depositNonce = ++_depositCounts[destinationChainID];
        _depositRecords[depositNonce][destinationChainID] = data;

        IDepositExecute depositHandler = IDepositExecute(handler);
        if (msg.value > 0) {
            uint256 valueToSend = msg.value - _fee;
            if (valueToSend > 0) {
                payable(handler).transfer(valueToSend);
            }
        }
        depositHandler.deposit(resourceID, destinationChainID, depositNonce, msg.sender, data);

        emit Deposit(destinationChainID, resourceID, depositNonce);
    }
```

```
21
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

Use call instead of transfer for sending native token.

Remediation Plan:

SOLVED: The Chiliz team solved the issue by changing the call transfer function.

Commit ID: 0976828419dbaa9feabfa017a1032c4203abf6da

3.10 (HAL-10) MISSING REENTRANCY GUARD - LOW

Description:

To protect against cross-function re-entrancy attacks, it may be necessary to use a mutex. By using this lock, an attacker can no longer exploit the withdrawal function with a recursive call. OpenZeppelin has its own mutex implementation called ReentrancyGuard which provides a modifier to any function called nonReentrant that guards the function with a mutex against re-entrancy attacks.

Code Location:

Bridge.sol#L308

Listing 16

```
    function deposit(uint8 destinationChainID, bytes32 resourceID,
↳  bytes calldata data) external payable whenNotPaused {
        require(msg.value >= _fee, "invalid value");

        address handler = _resourceIDToHandlerAddress[resourceID];
        require(handler != address(0), "resourceID not mapped to
↳ handler");

        uint64 depositNonce = ++_depositCounts[destinationChainID
↳ ];
        _depositRecords[depositNonce][destinationChainID] = data;

        IDepositExecute depositHandler = IDepositExecute(handler);
        if (msg.value > 0) {
            uint256 valueToSend = msg.value - _fee;
            if (valueToSend > 0) {
                payable(handler).transfer(valueToSend);
            }
        }
        depositHandler.deposit(resourceID, destinationChainID,
↳ depositNonce, msg.sender, data);

```

```
19          emit Deposit(destinationChainID, resourceID, depositNonce)
↳ ;
20      }
21
```

NativeHandler.sol#L91

Listing 17

```
1       function deposit(
2           bytes32 resourceID,
3           uint8   destinationChainID,
4           uint64  depositNonce,
5           address depositer,
6           bytes   calldata data
7       ) external override onlyBridge {
8           bytes   memory recipientAddress;
9           uint256        amount;
10          uint256        lenRecipientAddress;
11
12          assembly {
13              amount := calldataload(0xC4)
14              recipientAddress := mload(0x40)
15              lenRecipientAddress := calldataload(0xE4)
16              mstore(0x40, add(0x20, add(recipientAddress,
↳ lenRecipientAddress)))
17
18              calldatacopy(
19                  recipientAddress, // copy to
↳ destinationRecipientAddress
20                  0xE4, // copy from calldata @ 0x104
21                  sub(calldatasize(), 0xE) // copy size (
↳ calldatasize - 0x104)
22              )
23          }
24
25          address tokenAddress = _resourceIDToTokenContractAddress[
↳ resourceID];
26          require(_contractWhitelist[tokenAddress], "provided
↳ tokenAddress is not whitelisted");
27
28          uint256 bridgeFee = IBridge(_bridgeAddress)._fee();
29          uint256 amountMinusFee = amount - bridgeFee;
```

```
30          require(amountMinusFee > 0, "Invalid amount");
31
32          _depositRecords[destinationChainID][depositNonce] =
↳ DepositRecord(
33              tokenAddress,
34              uint8(lenRecipientAddress),
35              destinationChainID,
36              resourceID,
37              recipientAddress,
38              depositer,
39              amountMinusFee
40          );
41      }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Functions in the code location section are missing nonReentrant modifiers. It is recommended to add the OpenZeppelin ReentrancyGuard library to the project and use the nonReentrant modifier to avoid introducing future re-entrancy vulnerabilities.

Remediation Plan:

SOLVED: The Chiliz team solved the issue by adding a re-entrancy guard on the related functions.

Commit ID: 8d2c1cfb973eae4935ce167c8ae304ddcb9f843b

3.11 (HAL-11) NATSPEC IS NOT COMPATIBLE WITH THE IMPLEMENTATION - LOW

Description:

In the contract, **NatSpec** is incompatible with the implementation, which could lead to confusion and misunderstandings among users and developers. This is because **NatSpec** is a standard for documenting and annotating smart contracts, and it is designed to provide clear and concise information about the functions and behavior of a contract. However, if NatSpec is incompatible with the implementation of the contract, this information may not accurately reflect the actual behavior of the contract, and users and developers may not be able to fully understand and use the contract. It is important for the system to ensure that **NatSpec** is compatible with the implementation of the contract, to provide clear and accurate information to users and developers and to improve the usability and reliability of the contract.

Code Location:

NativeHandler.sol#L89

Listing 18

```
1     /*
2         marked true in {_burnList}, deposited tokens will be
↳ burned, if not, they will be locked.
3      */
4
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended that the system ensure that **NatSpec** is compatible with the implementation of the contract. This can be done by carefully reviewing and updating the NatSpec documentation for the contract to ensure that it accurately reflects the functions and behavior of the contract. This can help to provide clear and concise information to users and developers, and can improve the usability and reliability of the contract. It is also recommended to regularly review and update the **NatSpec** documentation as the implementation of the contract changes, to ensure that the **NatSpec** documentation remains accurate and up-to-date. By ensuring the compatibility of NatSpec with the implementation of the contract, the system can help to improve the user experience and trust in the contract.

Remediation Plan:

SOLVED: The Chiliz team solved the issue by updating the natspec in the functions.

Commit ID: a2814edaea8eaf7c6df0a176846645834b9a8e3e

3.12 (HAL-12) MISSING RECEIVER ADDRESS CHECK - LOW

Description:

In this system, the Solidity code does not include a check to prevent sending transactions to the zero address. The zero address is a special address that is reserved in Ethereum, and it represents the absence of an address. Sending a transaction to the zero address is equivalent to sending it to nowhere, and it will be lost and not executed. However, if the Solidity code does not check for the zero address, it is possible for a malicious actor to exploit this and send transactions to the zero address, potentially resulting in the loss of funds. It is important for the system to include a zero address check in the Solidity code, to prevent transactions from being sent to the zero address and to protect against potential loss of funds.

Code Location:

NativeHandler.sol#L104

Listing 19

```
    function deposit(
        bytes32 resourceID,
        uint8   destinationChainID,
        uint64  depositNonce,
        address depositer,
        bytes   calldata data
    ) external override onlyBridge {
        bytes   memory recipientAddress;
        uint256        amount;
        uint256        lenRecipientAddress;

        assembly {
            amount := calldataload(0xC4)
            recipientAddress := mload(0x40)
            lenRecipientAddress := calldataload(0xE4)
```

```
16              mstore(0x40, add(0x20, add(recipientAddress,
↳ lenRecipientAddress)))
17
18              calldatacopy(
19                  recipientAddress, // copy to
↳ destinationRecipientAddress
20                  0xE4, // copy from calldata @ 0x104
21                  sub(calldatasize(), 0xE) // copy size (
↳ calldatasize - 0x104)
22              )
23          }
24
25          address tokenAddress = _resourceIDToTokenContractAddress[
↳ resourceID];
26          require(_contractWhitelist[tokenAddress], "provided
↳ tokenAddress is not whitelisted");
27
28          uint256 bridgeFee = IBridge(_bridgeAddress)._fee();
29          uint256 amountMinusFee = amount - bridgeFee;
30          require(amountMinusFee > 0, "Invalid amount");
31
32          _depositRecords[destinationChainID][depositNonce] =
↳ DepositRecord(
33              tokenAddress,
34              uint8(lenRecipientAddress),
35              destinationChainID,
36              resourceID,
37              recipientAddress,
38              depositer,
39              amountMinusFee
40          );
41      }
42
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended that the system include a zero address check in the Solidity code, to prevent transactions from being sent to the zero address. This can be done by implementing a check that verifies that the recipient address of a transaction is not the zero address. For example, the Solidity code can use the require() function to check if the recipient address is equal to the zero address, and if it is, the transaction can be rejected and an error message can be returned. By implementing this check, the system can help to prevent transactions from being sent to the zero address and to protect against potential loss of funds. It is important to carefully review and test the Solidity code to ensure that the zero address check is implemented correctly and is effective at preventing transactions from being sent to the zero address.

Remediation Plan:

SOLVED: The Chiliz team solved the issue by adding receiver address check.

Commit ID: bf171d4028b5af946c091baa77a413b3927a44bc

3.13 (HAL-13) REDUNDANT FUNCTION PARAMETER IN THE NATIVEHANDLER - INFORMATIONAL

Description:

In the bridge, the **NativeHandler** contract includes a **tokenAddress** function parameter that is not used by the function itself. This redundant parameter could potentially cause confusion and misunderstandings, as it may not be clear to users and developers why the parameter is included or what it is used for. In addition, the presence of this redundant parameter could also increase the complexity of the code and make it more difficult to maintain and update. It is important for the system to avoid using redundant function parameters, to improve the clarity and usability of the code and to reduce the complexity and maintenance costs of the system.

Code Location:

NativeHandler.sol#L180

Listing 20

```
    /**
        @notice Used to manually release native tokens from the handler.
        @param tokenAddress Address of token contract to release.
        @param recipient Address to release tokens to.
        @param amount The amount of native tokens to release.
     */
    function withdraw(address tokenAddress, address recipient, uint amount) external override onlyBridge {
        payable(recipient).transfer(amount);
    }

```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended that the system remove the redundant function parameter from the NativeHandler contract. This can be done by carefully reviewing the function and identifying any unused parameters, and then removing those parameters from the function signature. By removing the redundant parameter, the system can help to improve the clarity and usability of the code, and it can also help to reduce the complexity and maintenance costs of the system. It is important to carefully review and test the code to ensure that the removal of the redundant parameter does not cause any unexpected behavior or errors.

Remediation Plan:

SOLVED: The Chiliz team solved the issue by changing the logic in the related function.

Commit ID: 3660ae57c26131770796467d4c123b2274cf6159

3.14 (HAL-14) REDUNDANT ARRAY DEFINITION IN THE CONSTRUCTOR - INFORMATIONAL

Description:

The presence of a redundant array definition in the constructor could potentially lead to confusion and misunderstanding, and could also lead to unnecessary code complexity and maintenance costs.

Code Location:

NativeHandler.sol#L45

Listing 21

```
    constructor(
        address          bridgeAddress,
        bytes32[] memory initialResourceIDs,
        address[] memory initialContractAddresses,
        address[] memory burnableContractAddresses
    ) public {
        require(initialResourceIDs.length ==
↳ initialContractAddresses.length,
            "initialResourceIDs and initialContractAddresses len
↳ mismatch");

        _bridgeAddress = bridgeAddress;

        for (uint256 i = 0; i < initialResourceIDs.length; i++) {
            _setResource(initialResourceIDs[i],
↳ initialContractAddresses[i]);
        }

        for (uint256 i = 0; i < burnableContractAddresses.length;
↳ i++) {
            _setBurnable(burnableContractAddresses[i]);
        }
    }

```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended that the system remove the redundant array definition from the constructor. This can be done by carefully reviewing the constructor and identifying any unused array definitions, and then removing those definitions from the constructor. By removing the redundant array definition, the system can help to improve the clarity and usability of the code, and it can also help to reduce the complexity and maintenance costs of the system. It is important to carefully review and test the code to ensure that the removal of the redundant array definition does not cause any unexpected behavior or errors.

Remediation Plan:

SOLVED: The Chiliz team solved the issue by deleting the redundant parameter in the constructor.

Commit ID: 568ac8b609f9945bbbafa454afe394fdd8270458

3.15 (HAL-15) DELETE - ABI CODER V2 FOR GAS OPTIMIZATION - INFORMATIONAL

Description:

From **Pragma 0.8.0**, **ABI coder v2** is activated by default. The pragma abicoder v2 can be deleted from the repository. That will provide gas optimization.

Code Location:

Bridge.sol#L2

Listing 22

```
1 pragma solidity 0.6.4;
2 pragma experimental ABIEncoderV2;
3
4 import "@openzeppelin/contracts/access/AccessControl.sol";
5 import "./utils/Pausable.sol";
6 import "./utils/SafeMath.sol";
7 import "./interfaces/IDepositExecute.sol";
8 import "./interfaces/IBridge.sol";
9 import "./interfaces/IERCHandler.sol";
10 import "./interfaces/IGenericHandler.sol";
11
12 /**
13     @title Facilitates deposits, creation and votiing of deposit
↳ proposals, and deposit executions.
14     @author ChainSafe Systems.
15  */
16 contract Bridge is Pausable, AccessControl, SafeMath
```

NativeHandler.sol#L2

Listing 23

```
pragma solidity 0.6.4;
pragma experimental ABIEncoderV2;

import "../interfaces/IDepositExecute.sol";
import "../interfaces/IBridge.sol";
import "./HandlerHelpers.sol";
import "../ERC20Safe.sol";
import "@openzeppelin/contracts/presets/ERC20PresetMinterPauser.sol";

contract NativeHandler is IDepositExecute, HandlerHelpers, ERC20Safe {}
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider updating pragma to **0.8.17** and ABI coder v2 is activated by default. It is recommended to delete redundant codes. Solidity v0.8.0 Breaking Changes

Remediation Plan:

SOLVED: The Chiliz team solved the issue by updating pragma.

Commit ID: 6830bf7e74f4a2bb7675c768f24a8a2f35889df5

3.16 (HAL-16) UPGRADE PRAGMA TO AT LEAST 0.8.4 - INFORMATIONAL

Description:

Using newer compiler versions and the optimizer gives gas optimizations and additional safety checks for free.

The advantages of versions `=0.8.*=` over `=<0.8.0=` are:

- Safemath by default from 0.8.0 (can be more gas efficient than some library based safemath).
- Low level inliner from 0.8.2, leads to cheaper runtime gas. Especially relevant when the contract has small functions. For example, OpenZeppelin libraries typically have a lot of small helper functions and if they are not inlined, they cost an additional 20 to 40 gas because of 2 extra jump instructions and additional stack operations needed for function calls.
- Optimizer improvements in packed structs: Before 0.8.3, storing packed structs, in some cases, used an additional storage read operation. After EIP-2929, if the slot was already cold, this means unnecessary stack operations and extra deploy time costs. However, if the slot was already warm, this means additional cost of 100 gas alongside the same unnecessary stack operations and extra deploy time costs.
- Custom errors from 0.8.4, leads to cheaper deploy time cost and run time cost. Note: the run time cost is only relevant when the revert condition is met. In short, replace revert strings by custom errors.

Risk Level:

Likelihood - 2
Impact - 1

Recommendation:

Consider updating the pragma to minimum 0.8.4 version.

Remediation Plan:

SOLVED: The Chiliz team solved the issue by updating pragma.

Commit ID: 6830bf7e74f4a2bb7675c768f24a8a2f35889df5

3.17 (HAL-17) OPTIMIZE UNSIGNED INTEGER COMPARISON - INFORMATIONAL

Description:

The check != 0 costs less gas compared to > 0 for unsigned integers in require statements with the optimizer enabled. While it may seem that > 0 is cheaper than !=0, this is only true without the optimizer enabled and outside a require statement. If the optimizer is enabled at 10k, and it is in a require statement, that would be more gas efficient.

Code Location:

Bridge.sol#L302-L304

Listing 24

```
        IDepositExecute depositHandler = IDepositExecute(handler);
        if (msg.value > 0) {
            uint256 valueToSend = msg.value - _fee;
            if (valueToSend > 0) {
                payable(handler).transfer(valueToSend);
            }
        }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Change > 0 comparison with != 0.

Remediation Plan:

SOLVED: The Chiliz team solved the issue by deleting the related code section in the contract.

Commit ID: 6830bf7e74f4a2bb7675c768f24a8a2f35889df5

3.18 (HAL-18) CHANGE FUNCTION VISIBILITY FROM PUBLIC TO EXTERNAL - INFORMATIONAL

Description:

There is a function declared as public that is never called internally within the contract. It is best practice to mark such functions as external instead, as this saves gas (especially in the case where the function takes arguments, as external functions can read arguments directly from calldata instead of having to allocate memory).

Code Location:

Bridge.sol#L384

Listing 25

```
1     function cancelProposal(uint8 chainID, uint64 depositNonce,
↳ bytes32 dataHash) public onlyAdminOrRelayer
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

All the public functions in the contract are not called internally, so access can be changed to external to reduce gas.

Remediation Plan:

SOLVED: The Chiliz team solved the issue by changing the visibility of the function.

Commit ID: f574625d79d4701d2a89acfd9a5b139c352e9a91

3.19 (HAL-19) REDUNDANT SAFEMATH FUNCTIONS USAGE IN CONTRACT - INFORMATIONAL

Description:

In a Solidity contract, the developer has included multiple instances of the SafeMath library functions, even though only one instance is needed. Including redundant instances of the SafeMath library can lead to an increase in the size of the contract, which can negatively impact the contract's performance and gas usage.

Code Location:

Bridge.sol#L151

Listing 26

```
    function add(uint256 a, uint256 b) internal pure returns (
↳ uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers,
↳ reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (
↳ uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
```

```
19      }
20
21      /**
22       * @dev Returns the subtraction of two unsigned integers,
↳ reverting with custom message on
23       * overflow (when the result is negative).
24       *
25       * Counterpart to Solidity's `-` operator.
26       *
27       * Requirements:
28       * - Subtraction cannot overflow.
29       */
30      function sub(uint256 a, uint256 b, string memory errorMessage)
↳  internal pure returns (uint256) {
31          require(b <= a, errorMessage);
32          uint256 c = a - b;
33
34          return c;
35      }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

To avoid the impact of redundant SafeMath usage, the developer should remove any unnecessary instances of the SafeMath library from the contract. This can help to reduce the size of the contract and improve its performance and gas efficiency. Additionally, the developer should ensure that the SafeMath functions are used consistently throughout the contract to prevent any potential vulnerabilities related to arithmetic operations.

Remediation Plan:

SOLVED: The Chiliz team solved the issue by upgrading **pragma 0.8.17** in contracts. Also, the SafeMath libraries are deleted from the contracts.

Commit ID: add9f81660e4ab58778f4706e00fc4ac4234a153

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repositories and was able to compile them correctly into their ABIs and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Slither results:

```
Bridge.adminChangeFee(uint256) (contracts/Bridge.sol#262-265) should emit an event for:
        - _fee = newFee (contracts/Bridge.sol#264)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

Bridge.transferFunds(address[],uint256[]) (contracts/Bridge.sol#431-435) has external calls inside a loop: addrs[i].transfer(amounts[i]) (contracts/Bridge.sol#433)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

Reentrancy in Bridge.deposit(uint8,bytes32,bytes) (contracts/Bridge.sol#292-311):
        External calls:
        - depositHandler.deposit(resourceID,destinationChainID,depositNonce,msg.sender,data) (contracts/Bridge.sol#308)
        External calls sending eth:
        - address(handler).transfer(valueToSend) (contracts/Bridge.sol#305)
        Event emitted after the call(s):
        - Deposit(destinationChainID,resourceID,depositNonce) (contracts/Bridge.sol#310)
Reentrancy in Bridge.executeProposal(uint8,uint64,bytes,bytes32) (contracts/Bridge.sol#407-423):
        External calls:
        - depositHandler.executeProposal(proposal._resourceID,data) (contracts/Bridge.sol#420)
        Event emitted after the call(s):
        - ProposalEvent(chainID,depositNonce,proposal._status,proposal._resourceID,proposal._dataHash) (contracts/Bridge.sol#422)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Address.isContract(address) (node_modules/@openzeppelin/contracts/utils/Address.sol#24-33) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#31)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity is used:
        - Version used: ['0.6.4', '^0.6.0', '^0.6.2']
        - 0.6.4 (contracts/Bridge.sol#1)
        - ABIEncoderV2 (contracts/Bridge.sol#2)
        - 0.6.4 (contracts/interfaces/IBridge.sol#1)
        - 0.6.4 (contracts/interfaces/IDepositExecute.sol#1)
        - 0.6.4 (contracts/interfaces/IERCHandler.sol#1)
        - 0.6.4 (contracts/interfaces/IGenericHandler.sol#1)
        - ^0.6.0 (contracts/utils/Pausable.sol#3)
        - ^0.6.0 (contracts/utils/SafeMath.sol#3)
        - ^0.6.0 (node_modules/@openzeppelin/contracts/GSN/Context.sol#1)
        - ^0.6.0 (node_modules/@openzeppelin/contracts/access/AccessControl.sol#1)
        - ^0.6.2 (node_modules/@openzeppelin/contracts/utils/Address.sol#1)
        - ^0.6.0 (node_modules/@openzeppelin/contracts/utils/EnumerableSet.sol#1)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Bridge.constructor(uint8,address[],uint256,uint256,uint256) (contracts/Bridge.sol#109-123) has costly operations inside a loop:
        - _totalRelayers ++ (contracts/Bridge.sol#120)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop



Low level call in Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#51-57):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts/utils/Address.sol#55)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Variable Bridge._chainID (contracts/Bridge.sol#18) is not in mixedCase
Variable Bridge._relayerThreshold (contracts/Bridge.sol#19) is not in mixedCase
Variable Bridge._totalRelayers (contracts/Bridge.sol#20) is not in mixedCase
Variable Bridge._totalProposals (contracts/Bridge.sol#21) is not in mixedCase
Variable Bridge._fee (contracts/Bridge.sol#22) is not in mixedCase
Variable Bridge._expiry (contracts/Bridge.sol#23) is not in mixedCase
Variable Bridge._depositCounts (contracts/Bridge.sol#39) is not in mixedCase
Variable Bridge._resourceIDToHandlerAddress (contracts/Bridge.sol#41) is not in mixedCase
Variable Bridge._depositRecords (contracts/Bridge.sol#43) is not in mixedCase
Variable Bridge._proposals (contracts/Bridge.sol#45) is not in mixedCase
Variable Bridge._hasVotedOnProposal (contracts/Bridge.sol#47) is not in mixedCase
Function IBridge._chainID() (contracts/interfaces/IBridge.sol#12) is not in mixedCase
Function IBridge._fee() (contracts/interfaces/IBridge.sol#18) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (node_modules/@openzeppelin/contracts/GSN/Context.sol#23)" inContext (node_modules/@openzeppelin/contracts/GSN/Context.sol#13-26)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

Reentrancy in Bridge.deposit(uint8,bytes32,bytes) (contracts/Bridge.sol#292-311):
        External calls:
        - address(handler).transfer(valueToSend) (contracts/Bridge.sol#305)
        Event emitted after the call(s):
        - Deposit(destinationChainID,resourceID,depositNonce) (contracts/Bridge.sol#310)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4

cancelProposal(uint8,uint64,bytes32) should be declared external:
        - Bridge.cancelProposal(uint8,uint64,bytes32) (contracts/Bridge.sol#384-394)
paused() should be declared external:
        - Pausable.paused() (contracts/utils/Pausable.sol#37-39)
getRoleMemberCount(bytes32) should be declared external:
        - AccessControl.getRoleMemberCount(bytes32) (node_modules/@openzeppelin/contracts/access/AccessControl.sol#83-85)
getRoleMember(bytes32,uint256) should be declared external:
        - AccessControl.getRoleMember(bytes32,uint256) (node_modules/@openzeppelin/contracts/access/AccessControl.sol#99-101)
getRoleAdmin(bytes32) should be declared external:
        - AccessControl.getRoleAdmin(bytes32) (node_modules/@openzeppelin/contracts/access/AccessControl.sol#109-111)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

```
ERC20PresetMinterPauser.constructor(string,string).name (node_modules/@openzeppelin/contracts/presets/ERC20PresetMinterPauser.sol#33) shadows:
        - ERC20.name() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#64-66) (function)
ERC20PresetMinterPauser.constructor(string,string).symbol (node_modules/@openzeppelin/contracts/presets/ERC20PresetMinterPauser.sol#33) shadows:
        - ERC20.symbol() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#72-74) (function)
ERC20.constructor(string,string).name (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#55) shadows:
        - ERC20.name() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#64-66) (function)
ERC20.constructor(string,string).symbol (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#55) shadows:
        - ERC20.symbol() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#72-74) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

NativeHandler.constructor(address,bytes32[],address[],address[]).bridgeAddress (contracts/handlers/NativeHandler.sol#42) lacks a zero-check on :
        - _bridgeAddress = bridgeAddress (contracts/handlers/NativeHandler.sol#50)
NativeHandler.withdraw(address,address,uint256).recipient (contracts/handlers/NativeHandler.sol#180) lacks a zero-check on :
        - address(recipient).transfer(amount) (contracts/handlers/NativeHandler.sol#181)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in NativeHandler.deposit(bytes32,uint8,uint64,address,bytes) (contracts/handlers/NativeHandler.sol#91-131):
        External calls:
        - bridgeFee = IBridge(_bridgeAddress)._fee() (contracts/handlers/NativeHandler.sol#110)
        State variables written after the call(s):
        - _depositRecords[destinationChainID][depositNonce] = DepositRecord(tokenAddress,uint8(lenRecipientAddress),destinationChainID,resourceID,recipientAddress,depositer,amountMinusFee) (contracts/handlers/NativeHan
dler.sol#122-130)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

NativeHandler.deposit(bytes32,uint8,uint64,address,bytes) (contracts/handlers/NativeHandler.sol#91-131) uses assembly
        - INLINE ASM (contracts/handlers/NativeHandler.sol#102-113)
NativeHandler.executeProposal(bytes32,bytes) (contracts/handlers/NativeHandler.sol#143-172) uses assembly
        - INLINE ASM (contracts/handlers/NativeHandler.sol#147-160)
        - INLINE ASM (contracts/handlers/NativeHandler.sol#165-167)
Address.isContract(address) (node_modules/@openzeppelin/contracts/utils/Address.sol#24-33) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#31)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity is used:
        - Version used: ['0.6.4', '^0.6.0', '^0.6.2']
        - 0.6.4 (contracts/ERC20Safe.sol#1)
        - 0.6.4 (contracts/handlers/HandlerHelpers.sol#1)
        - 0.6.4 (contracts/handlers/NativeHandler.sol#1)
        - ABIEncoderV2 (contracts/handlers/NativeHandler.sol#2)
        - 0.6.4 (contracts/interfaces/IBridge.sol#1)
        - 0.6.4 (contracts/interfaces/IDepositExecute.sol#1)
        - 0.6.4 (contracts/interfaces/IERCHandler.sol#1)
        - ^0.6.0 (node_modules/@openzeppelin/contracts/GSN/Context.sol#1)
        - ^0.6.0 (node_modules/@openzeppelin/contracts/access/AccessControl.sol#1)
        - ^0.6.0 (node_modules/@openzeppelin/contracts/math/SafeMath.sol#1)
        - ^0.6.0 (node_modules/@openzeppelin/contracts/presets/ERC20PresetMinterPauser.sol#1)
        - ^0.6.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#1)
        - ^0.6.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20Burnable.sol#1)
        - ^0.6.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20Pausable.sol#1)
        - ^0.6.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#1)
        - ^0.6.2 (node_modules/@openzeppelin/contracts/utils/Address.sol#1)
        - ^0.6.0 (node_modules/@openzeppelin/contracts/utils/EnumerableSet.sol#1)



Pragma version0.6.4 (contracts/ERC20Safe.sol#1) allows old versions
Pragma version0.6.4 (contracts/handlers/HandlerHelpers.sol#1) allows old versions
Pragma version0.6.4 (contracts/handlers/NativeHandler.sol#1) allows old versions
Pragma version0.6.4 (contracts/interfaces/IBridge.sol#1) allows old versions
Pragma version0.6.4 (contracts/interfaces/IDepositExecute.sol#1) allows old versions
Pragma version0.6.4 (contracts/interfaces/IERCHandler.sol#1) allows old versions
Pragma version^0.6.0 (node_modules/@openzeppelin/contracts/GSN/Context.sol#1) allows old versions
Pragma version^0.6.0 (node_modules/@openzeppelin/contracts/access/AccessControl.sol#1) allows old versions
Pragma version^0.6.0 (node_modules/@openzeppelin/contracts/math/SafeMath.sol#1) allows old versions
Pragma version^0.6.0 (node_modules/@openzeppelin/contracts/presets/ERC20PresetMinterPauser.sol#1) allows old versions
Pragma version^0.6.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#1) allows old versions
Pragma version^0.6.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20Burnable.sol#1) allows old versions
Pragma version^0.6.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20Pausable.sol#1) allows old versions
Pragma version^0.6.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#1) allows old versions
Pragma version^0.6.2 (node_modules/@openzeppelin/contracts/utils/Address.sol#1) allows old versions
Pragma version^0.6.0 (node_modules/@openzeppelin/contracts/utils/EnumerableSet.sol#1) allows old versions
Pragma version^0.6.0 (node_modules/@openzeppelin/contracts/utils/Pausable.sol#1) allows old versions
solc-0.6.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in ERC20Safe._safeCall(IERC20,bytes) (contracts/ERC20Safe.sol#100-108):
        - (success,returndata) = address(token).call(data) (contracts/ERC20Safe.sol#101)
Low level call in Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#51-57):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts/utils/Address.sol#55)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Variable HandlerHelpers._bridgeAddress (contracts/handlers/HandlerHelpers.sol#11) is not in mixedCase
Variable HandlerHelpers._resourceIDToTokenContractAddress (contracts/handlers/HandlerHelpers.sol#14) is not in mixedCase
Variable HandlerHelpers._tokenContractAddressToResourceID (contracts/handlers/HandlerHelpers.sol#17) is not in mixedCase
Variable HandlerHelpers._contractWhitelist (contracts/handlers/HandlerHelpers.sol#20) is not in mixedCase
Variable HandlerHelpers._burnList (contracts/handlers/HandlerHelpers.sol#23) is not in mixedCase
Variable NativeHandler._depositRecords (contracts/handlers/NativeHandler.sol#27) is not in mixedCase
Function IBridge._chainID() (contracts/interfaces/IBridge.sol#12) is not in mixedCase
Function IBridge._fee() (contracts/interfaces/IBridge.sol#18) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (node_modules/@openzeppelin/contracts/GSN/Context.sol#23)" inContext (node_modules/@openzeppelin/contracts/GSN/Context.sol#13-26)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

fundERC20(address,address,uint256) should be declared external:
        - ERC20Safe.fundERC20(address,address,uint256) (contracts/ERC20Safe.sol#22-25)
getRoleMemberCount(bytes32) should be declared external:
        - AccessControl.getRoleMemberCount(bytes32) (node_modules/@openzeppelin/contracts/access/AccessControl.sol#83-85)
getRoleMember(bytes32,uint256) should be declared external:
        - AccessControl.getRoleMember(bytes32,uint256) (node_modules/@openzeppelin/contracts/access/AccessControl.sol#99-101)
getRoleAdmin(bytes32) should be declared external:
        - AccessControl.getRoleAdmin(bytes32) (node_modules/@openzeppelin/contracts/access/AccessControl.sol#109-111)
grantRole(bytes32,address) should be declared external:
        - AccessControl.grantRole(bytes32,address) (node_modules/@openzeppelin/contracts/access/AccessControl.sol#123-127)
revokeRole(bytes32,address) should be declared external:
        - AccessControl.revokeRole(bytes32,address) (node_modules/@openzeppelin/contracts/access/AccessControl.sol#138-142)
renounceRole(bytes32,address) should be declared external:
        - AccessControl.renounceRole(bytes32,address) (node_modules/@openzeppelin/contracts/access/AccessControl.sol#158-162)
```

- No major issues were found by Slither.

THANK YOU FOR CHOOSING

// HALBORN