ZetaChain - ZetaNode

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **March 21st, 2023 - March 27th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	03/26/2023	Gokberk Gulgun
0.2	Document Updates	03/27/2023	Gokberk Gulgun
0.3	Document Draft Review	03/27/2023	Gabi Urrutia
1.0	Remediation Plan	04/17/2023	Gokberk Gulgun
1.1	Remediation Plan Review	04/17/2023	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

ZetaChain engaged Halborn to conduct a security audit on their smart contracts beginning on March 21st, 2023 and ending on March 27th, 2023. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided 1 week 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 issues that were partially addressed by the ZetaChain 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 contracts' solidity code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing with custom scripts. (Foundry).
- Static Analysis of security for scoped contract, and imported functions manually.
- Testnet deployment (Ganache).

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 Exploitability

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 Impact

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 Severity Coefficient

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N)	1
	Partial (R:P)	0.5
	Full (R:F)	0.25
Scope (s)	Changed (S:C)	1.25
	Unchanged (S:U)	1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

EXECUTIVE OVERVIEW

2.4 SCOPE

1. IN-SCOPE TREE & COMMIT :

BRANCH : pre-audit-review

COMMIT ID : bf5aa35ed2258e9d12a92578faaf5ece991c74e7

The security assessment was scoped to the following smart contracts:

- evm/ERC20Custody.sol.
- zevm/ConnectorZEVM.sol.
- zevm/SystemContract.sol.
- zevm/ZRC20.sol.

2. REMEDIATION COMMIT IDs :

- 51dc57035a7f9feb68aa16f45752d46bb5ce78b0
- 46a1be64690774065539868b23aad710e17e6099
- 3637741fb8b7e036dd81ca763f9fe2f548bcbfec

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	3	1	5	9

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - RECEIVE FUNCTION IS NOT RESTRICTED TO WETH	High (8.4)	SOLVED - 04/07/2023
HAL02 - FEE-ON-TRANSFER DEFLATIONARY TOKENS ARE NOT SUPPORTED	High (8.0)	SOLVED - 04/11/2023
HAL03 - ABSENCE OF SAFETRANSFER/SAFETRANSFERFROM IN TOKEN TRANSFERS	High (8.0)	SOLVED - 04/11/2023
HAL04 - ZRC20 LACKS RESISTANCE TO ERC20 RACE CONDITION ISSUE	Medium (5.9)	SOLVED - 04/11/2023
HAL05 - INSECURE USE OF TX.ORIGIN IN THE SEND FUNCTION	Low (2.3)	RISK ACCEPTED
HAL06 - FEE DEFINITION DOES NOT HAVE UPPER BOUND	Low (2.3)	SOLVED - 04/11/2023
HAL07 - ZETASENT EVENT PARAMETERS ARE NOT VALIDATED ON THE CONTRACT	Low (2.3)	RISK ACCEPTED
HAL08 - POTENTIAL BLOCKING OF WITHDRAW OPERATIONS DUE TO HIGH GAS FEES	Low (2.9)	RISK ACCEPTED
HAL09 - MISSING SLIPPAGE/MIN-RETURN CHECK ONCROSSCHAINCALL FUNCTION	Low (2.9)	RISK ACCEPTED
HAL10 - THE CONTRACTS ARE MISSING NATSPEC	Informational (0.0)	SOLVED - 04/11/2023
HAL11 - FLOATING PRAGMA	Informational (0.0)	SOLVED - 04/11/2023
HAL12 - MISSING EVENTS ON THE UPDATE FUNCTIONS	Informational (0.0)	SOLVED - 04/11/2023
HAL13 - IMMUTABLE VARIABLES	Informational (0.0)	SOLVED - 04/11/2023
HAL14 - EVENT IS MISSING INDEXED FIELDS	Informational (0.0)	SOLVED - 04/11/2023
HAL15 - NO NEED TO INITIALIZE VARIABLES WITH DEFAULT VALUES	Informational (0.0)	SOLVED - 04/11/2023

HAL16 - REVERT STRING SIZE OPTIMIZATION	Informational (0.0)	SOLVED - 04/11/2023
HAL17 - USE CUSTOM ERRORS INSTEAD OF REVERT STRINGS TO SAVE GAS	Informational (0.0)	SOLVED - 04/11/2023
HAL18 - OPTIMIZE UNSIGNED INTEGER COMPARISON	Informational (0.0)	SOLVED - 04/11/2023

FINDINGS & TECH DETAILS

4.1 (HAL-01) HAL01 - RECEIVE FUNCTION IS NOT RESTRICTED TO WETH - HIGH (8.4)

Description:

During the code review, It has been noticed that a potential issue with the implementation of the receive function in the smart contract. The receive function is not restricted to WETH (Wrapped Ether) token transfers, which could lead to unexpected token transfers to your smart contract.

The unrestricted receive function may allow users to send native tokens to the contract. This could result in the undesired accumulation of tokens within the contract, making them permanently locked and inaccessible to the intended recipients.

Code Location:

ConnectorZEVM.sol#L74

Listing 1

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

Proof of Concept:

1. Send native token to ConnectorZEVM contract.
2. Native token sent to ConnectorZEVM contract will be locked.

BVSS:

AO:A/AC:M/AX:L/C:M/I:M/A:M/D:H/Y:M/R:N/S:U (8.4)

Recommendation:

To address this issue, we recommend implementing a check within the receive function to ensure that only WETH token transfers are allowed. This can be achieved by comparing the address of the sender or the transferred token with the known WETH contract address. This modification will prevent accidental or malicious transfers of unsupported tokens to your smart contract, ensuring that only the intended token is accepted.

Remediation Plan:

SOLVED: The ZetaChain team solved the issue by adding the check.

Commit ID : 51dc57035a7f9feb68aa16f45752d46bb5ce78b0

4.2 (HAL-02) HAL02 - FEE-ON-TRANSFER DEFLATIONARY TOKENS ARE NOT SUPPORTED - HIGH (8.0)

Description:

During the audit, it was discovered that the smart contract does not support Fee-On-Transfer or deflationary tokens. These token types implement a fee mechanism, where a percentage of each token transfer is either burned, redistributed to token holders, or allocated to a specific address (e.g., a treasury or liquidity pool). The absence of support for these tokens may limit the smart contract's compatibility with various token projects and ecosystems.

Code Location:

/ERC20Custody.sol#LL126-L152

Listing 2

```
    function deposit(bytes calldata recipient, IERC20 asset, uint256 amount, bytes calldata message) external {
        if (paused) {
            revert IsPaused();
        }
        if (!whitelisted[asset]) {
            revert NotWhitelisted();
        }
        if (address(zeta) != address(0)) {
            zeta.transferFrom(msg.sender, TSSAddress, zetaFee);
        }
        asset.transferFrom(msg.sender, address(this), amount);
        emit Deposited(recipient, asset, amount, message);
    }

    function withdraw(address recipient, IERC20 asset, uint256 amount) external {
        if (paused) {
            revert IsPaused();
        }
        if (msg.sender != TSSAddress) {
            revert InvalidSender();
        }
        if (!whitelisted[asset]) {
            revert NotWhitelisted();
        }
        IERC20(asset).transfer(recipient, amount);
        emit Withdrawn(recipient, asset, amount);
    }
```

Proof of Concept:

1. Assume transfer fee to be 5% and ERC20Custody.sol has 200 token.
2. TSS Address deposit 100 tokens. Now, ERC20Custody.sol has 295 tokens.
3. TSS Address calls the method to withdraw 100 tokens.
4. ERC20Custody.sol ends up having 195 tokens.

BVSS:

AO:A/AC:M/AX:L/C:M/I:M/A:M/D:H/Y:L/R:N/S:U (8.0)

Recommendation:

Consider checking the before and after balance of token transfer on the deposit and withdraw functions.

Remediation Plan:

SOLVED: The ZetaChain team solved the issue by adding the pre-post balance check.

Commit ID : 3637741fb8b7e036dd81ca763f9fe2f548bcbfec

4.3 (HAL-03) HAL03 - ABSENCE OF SAFETRANSFER/SAFETRANSFERFROM IN TOKEN TRANSFERS - HIGH (8.0)

Description:

The given smart contract does not utilize the SafeTransfer or SafeTransferFrom functions for handling token transfers. These functions are part of the OpenZeppelin SafeERC20 library and are designed to provide additional safety checks and error handling when working with ERC20 tokens. Without these safety measures in place, the smart contract may encounter unexpected issues or revert without providing clear error messages during token transfers, which can lead to confusion and difficulty in diagnosing problems.

Code Location:

/ERC20Custody.sol#LL126-L152

Listing 3

```
    function deposit(bytes calldata recipient, IERC20 asset, uint256 amount, bytes calldata message) external {
        if (paused) {
            revert IsPaused();
        }
        if (!whitelisted[asset]) {
            revert NotWhitelisted();
        }
        if (address(zeta) != address(0)) {
            zeta.transferFrom(msg.sender, TSSAddress, zetaFee);
        }
        asset.transferFrom(msg.sender, address(this), amount);
        emit Deposited(recipient, asset, amount, message);
    }

    function withdraw(address recipient, IERC20 asset, uint256 amount) external {
        if (paused) {
            revert IsPaused();
        }
        if (msg.sender != TSSAddress) {
            revert InvalidSender();
        }
        if (!whitelisted[asset]) {
            revert NotWhitelisted();
        }
        IERC20(asset).transfer(recipient, amount);
        emit Withdrawn(recipient, asset, amount);
    }
```

Proof of Concept:

Listing 4

```
contract NoRevertToken {


    // --- Token ---
    function transfer(address dst, uint wad) external returns (bool) {
        return transferFrom(msg.sender, dst, wad);
    }
    function transferFrom(address src, address dst, uint wad) virtual public returns (bool) {
        if (balanceOf[src] < wad) return false;                    // insufficient src bal
        if (balanceOf[dst] >= (type(uint256).max - wad)) return false; // dst bal too high

        if (src != msg.sender && allowance[src][msg.sender] != type(uint).max) {
            if (allowance[src][msg.sender] < wad) return false;          // insufficient allowance
            allowance[src][msg.sender] = allowance[src][msg.sender] - wad;
        }

        balanceOf[src] = balanceOf[src] - wad;
        balanceOf[dst] = balanceOf[dst] + wad;

        emit Transfer(src, dst, wad);
        return true;
    }

}
```

BVSS:

AO:A/AC:M/AX:L/C:M/I:M/A:M/D:H/Y:L/R:N/S:U (8.0)

Recommendation:

To mitigate potential issues and enhance the safety of the smart contract, we recommend incorporating the use of SafeTransfer and SafeTransferFrom functions from the OpenZeppelin SafeERC20 library.

Remediation Plan:

SOLVED: The ZetaChain team solved the issue by using **safeTransfer/safeTransferFrom**.

Commit ID : 46a1be646907740655339868b23aad710e17e6099

4.4 (HAL-04) HAL04 - ZRC20 LACKS RESISTANCE TO ERC20 RACE CONDITION ISSUE - MEDIUM (5.9)

Description:

During the code review, It has been noticed that the ZRC20 implementation is not resistant to the well-known ERC20 race condition issue, also known as the allowance front-running problem. This issue occurs when a user attempts to update their token allowance for a spender while the spender is simultaneously trying to use the existing allowance. If the spender's transaction is confirmed before the user's allowance update, the user might inadvertently grant the spender a higher allowance than intended.

Code Location:

/contracts/zevm/ZRC20.sol

Listing 5

```
    function _approve(address owner, address spender, uint256 amount) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }
```

Proof of Concept:

1. Alice initiates the approve(Bob, 500) function, granting Bob the permission to utilize 500 tokens.
2. Subsequently, Alice reconsiders and calls approve(Bob, 1000), which

amends Bob’s spending allowance to 1000 tokens.

3. Bob observes the transaction and swiftly invokes transferFrom(Alice, X, 500) before its mining completion.
4. If Bob’s transaction is mined before Alice’s, Bob will successfully transfer 500 tokens. Once Alice’s transaction has been mined, Bob can proceed to call transferFrom(Alice, X, 1000).

Bob has transferred 1500 tokens even though this was not Alice’s intention.

BVSS:

AO:A/AC:L/AX:L/C:M/I:M/A:M/D:M/Y:L/R:P/S:C (5.9)

Recommendation:

To mitigate the race condition issue and enhance the security and reliability of the ZRC20 token implementation, we recommend implementing the increaseAllowance and decreaseAllowance functions, instead of only using the approve function for modifying allowances. These functions allow users to update allowance values without first having to reduce them to zero, thus reducing the likelihood of encountering race condition issues.

Remediation Plan:

SOLVED: The ZetaChain team solved the issue by implementing increaseAllowance and decreaseAllowance functions.

Commit ID : 51dc57035a7f9feb68aa16f45752d46bb5ce78b0

4.5 (HAL-05) HAL05 - INSECURE USE OF TX.ORIGIN IN THE SEND FUNCTION - LOW (2.3)

Description:

The given code uses tx.origin as a parameter in the ZetaSent event emission. The tx.origin variable represents the original initiator of the transaction, which might not always be the same as ‘msg.sender’, which represents the direct caller of the function. The tx.origin variable refers to the original sender of a transaction, and its misuse can make the smart contract susceptible to phishing attacks. In a phishing attack, a malicious contract can be created to trick users into interacting with it. When users execute transactions through the malicious contract, the tx.origin variable will still point to the original sender, allowing the attacker to bypass access controls or manipulate the contract state. To mitigate this vulnerability, it is recommended to replace the use of tx.origin with msg.sender, which provides the address of the direct sender of the transaction. This change ensures that only the intended sender can interact with the smart contract, effectively eliminating the risk of phishing attacks.

Code Location:

ConnectorZEVM.sol#L83

Listing 6

```
    function send(ZetaInterfaces.SendInput calldata input) external {
        // transfer wzeta to "fungible" module, which will be burnt by the protocol post processing via hooks.
        require(WZETA(wzeta).transferFrom(msg.sender, address(this), input.zetaValueAndGas) == true, "wzeta.transferFrom fail");
        WZETA(wzeta).withdraw(input.zetaValueAndGas);
        (bool sent,) = FUNGIBLE_MODULE_ADDRESS.call{value: input.zetaValueAndGas}("");
        require(sent, "Failed to send Ether");
        emit ZetaSent(
            tx.origin,
            msg.sender,
            input.destinationChainId,
            input.destinationAddress,
            input.zetaValueAndGas,
            input.destinationGasLimit,
            input.message,
            input.zetaParams
        );
    }
```

Proof of Concept:

Listing 7

```
pragma solidity >=0.7.0 <0.9.0;
interface ConnectorZEVM {
    function send(ZetaInterfaces.SendInput calldata input) external
}

contract AttackerWallet {
    address payable owner;

    constructor() {
        owner = payable(msg.sender);
    }

    receive() external payable {
        ConnectorZEVM(msg.sender).send(input);
    }
}
```

BVSS:

AO:A/AC:L/AX:L/C:M/I:M/A:M/D:N/Y:N/R:F/S:C (2.3)

Recommendation:

To mitigate the risks associated with the use of tx.origin, it is advised to replace tx.origin with msg.sender in the ZetaSent event emission. msg.sender provides a more secure way of identifying the direct caller of the function, reducing the possibility of attacks.

Remediation Plan:

RISK ACCEPTED: The ZetaChain team accepted the risk of this finding.

4.6 (HAL-06) HAL06 - FEE DEFINITION DOES NOT HAVE UPPER BOUND - LOW (2.3)

Description:

The current implementation of the smart contract does not define an upper bound for the fee structure. Without an upper limit in place, there is a possibility that the fees could be set to disproportionately high values, either due to manipulation, incorrect configuration, or a bug in the smart contract. This can lead to a negative user experience, as users may be charged excessive fees for interacting with the contract.

Code Location:

/evm/erc20custody/ERC20Custody.sol#L70

Listing 8

```
function updateZetaFee(uint256 _zetaFee) external {
    if (msg.sender != TSSAddress) {
        revert InvalidSender();
    }
    if (_zetaFee == 0) {
        revert ZeroFee();
    }
    zetaFee = _zetaFee;
}
```

BVSS:

AO:A/AC:L/AX:L/C:M/I:M/A:M/D:N/Y:N/R:F/S:C (2.3)

Recommendation:

To address the issue and enhance the fee structure's security and predictability, we recommend defining an upper bound for the fee rates. This can be done by implementing a maximum fee rate value and a corresponding validation check when setting the fee rate.

Remediation Plan:

SOLVED: The ZetaChain team solved the issue by adding the upper bound.

Commit ID : 51dc57035a7f9feb68aa16f45752d46bb5ce78b0

4.7 (HAL-07) HAL07 - ZETASENT EVENT PARAMETERS ARE NOT VALIDATED ON THE CONTRACT - LOW (2.3)

Description:

During the code review, It has been noticed that ZetaSent events parameters are not validated. For instance, destinationChain id is not checked on the event, that can cause locking of funds on the one side of the chain.

Code Location:

/contracts/zevm/ConnectorZEVM.sol#L85-L90

Listing 9

```
contract ZetaConnectorZEVM is ZetaInterfaces{
    address public wzeta;
    address public constant FUNGIBLE_MODULE_ADDRESS = payable(0x735b14BB79463307AAcBED86DAf3322B1e6226aB);

    event ZetaSent(
        address sourceTxOriginAddress,
        address indexed zetaTxSenderAddress,
        uint256 indexed destinationChainId,
        bytes destinationAddress,
        uint256 zetaValueAndGas,
        uint256 destinationGasLimit,
        bytes message,
        bytes zetaParams
    );

    constructor(address _wzeta) {
        wzeta = _wzeta;
    }

    // the contract will receive ZETA from WETH9.withdraw()
    receive() external payable {}

    function send(ZetaInterfaces.SendInput calldata input) external {
        // transfer wzeta to "fungible" module, which will be burnt by the protocol post processing via hooks.
        require(WZETA(wzeta).transferFrom(msg.sender, address(this), input.zetaValueAndGas) == true, "wzeta.transferFrom fail");
        WZETA(wzeta).withdraw(input.zetaValueAndGas);
        (bool sent,) = FUNGIBLE_MODULE_ADDRESS.call{value: input.zetaValueAndGas}("");
        require(sent, "Failed to send Ether");
        emit ZetaSent(
            tx.origin,
            msg.sender,
            input.destinationChainId,
            input.destinationAddress,
            input.zetaValueAndGas,
            input.destinationGasLimit,
            input.message,
            input.zetaParams
        );
    }
```

BVSS:

AO:A/AC:L/AX:L/C:M/I:M/A:M/D:N/Y:N/R:F/S:C (2.3)

Recommendation:

Consider validating all related parameters on the function.

Remediation Plan:

RISK ACCEPTED: The ZetaChain team accepted the risk of this finding.

4.8 (HAL-08) HAL08 - POTENTIAL BLOCKING OF WITHDRAW OPERATIONS DUE TO HIGH GAS FEES - LOW (2.9)

Description:

In the provided code snippet, the withdraw function requires the user to transfer a gas fee (gasFee) and a protocol flat fee (PROTOCOL_FLAT_FEE) to the FUNGIBLE_MODULE_ADDRESS before proceeding with the withdrawal. During times of network congestion or high gas prices, the total fees (gasFee + PROTOCOL_FLAT_FEE) might become prohibitively expensive for some users. As a result, these users might be unable to withdraw their tokens, effectively blocking them from accessing their funds. The potential for blocked withdraw operations due to high gas fees can result in a negative user experience and may erode trust in the smart contract. Users who are unable to withdraw their tokens might be forced to wait for lower gas prices, which could take an indefinite amount of time, or they might have to pay exorbitant fees to access their funds.

Code Location:

/contracts/zevm/ZRC20.sol#L152

Listing 10

```
    function withdraw(bytes memory to, uint256 amount) external override returns (bool) {
        (address gasZRC20, uint256 gasFee)= withdrawGasFee();
        require(IZRC20(gasZRC20).transferFrom(msg.sender, FUNGIBLE_MODULE_ADDRESS, gasFee+PROTOCOL_FLAT_FEE), "transfer gas fee failed");

        _burn(msg.sender, amount);
        emit Withdrawal(msg.sender, to, amount, gasFee, PROTOCOL_FLAT_FEE);
        return true;
    }
```

BVSS:

AO:A/AC:M/AX:L/C:L/I:L/A:L/D:L/Y:N/R:N/S:U (2.9)

Recommendation:

Consider adjusting the required gas fees based on the current gas price, network conditions, or other factors, ensuring that users are not over-charged during high gas price periods.

Remediation Plan:

RISK ACCEPTED: The ZetaChain team accepted the risk of this finding.

4.9 (HAL-09) HAL09 - MISSING SLIPPAGE/MIN-RETURN CHECK ONCROSSCHAINCALL FUNCTION - LOW (2.9)

Description:

The contracts are missing slippage checks, which can lead to being vulnerable to sandwich attacks. Sandwich attacks are prevalent in the decentralized finance (DeFi) sector. These attacks occur when an adversary observes a trade involving assets X and Y and preemptively purchases asset Y. Following the execution of the victim's trade, the attacker sells the acquired amount of asset Y, capitalizing on the inflated price. The attacker profits by exploiting the knowledge of an impending trade that will increase the asset's price, ultimately causing a financial loss for the protocol.

Lacking appropriate slippage checks, trades may be executed at unfavorable prices, resulting in the acquisition of fewer tokens than the fair market value dictates.

Code Location:

/zevm/SystemContract.sol#LL48-L59

Listing 11

```
    function depositAndCall(
        address zrc20,
        uint256 amount,
        address target,
        bytes calldata message
    ) external {
        if (msg.sender != FUNGIBLE_MODULE_ADDRESS) revert CallerIsNotFungibleModule();
        if (target == FUNGIBLE_MODULE_ADDRESS || target == address(this)) revert InvalidTarget();

        IZRC20(zrc20).deposit(target, amount);
        zContract(target).onCrossChainCall(zrc20, amount, message);
    }
```

BVSS:

AO:A/AC:M/AX:L/C:L/I:L/A:L/D:L/Y:N/R:N/S:U (2.9)

Recommendation:

Without appropriate slippage checks, trades may be executed at unfavorable prices, resulting in the acquisition of fewer tokens than the fair market value dictates. To mitigate this risk, we recommend implementing minimum return amount checks in the following manner:

- Introduce a function parameter to be determined by the transaction initiator, ensuring that the received amount exceeds the specified parameter.

Remediation Plan:

RISK ACCEPTED: The ZetaChain team accepted the risk of this finding.

4.10 (HAL-10) HAL10 - THE CONTRACTS ARE MISSING NATSPEC - INFORMATIONAL (0.0)

Description:

The smart contracts under review are missing NatSpec comments, which are a form of documentation specific to the Ethereum platform. NatSpec is a system for providing human-readable descriptions of contract functions, events, and variables directly within the source code. This documentation can be easily parsed and displayed by development tools, making it easier for developers and users to understand the contract's behavior and purpose. The lack of NatSpec comments can lead to difficulties in understanding, maintaining, and auditing the smart contract code, as well as a subpar user experience when interacting with the contract through user interfaces.

Code Location:

- evm/ERC20Custody.sol
- zevm/ConnectorZEVM.sol
- zevm/SystemContract.sol
- zevm/ZRC20.sol

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:C (0.0)

Recommendation:

To improve the readability, maintainability, and user experience associated with the smart contracts, we recommend adding NatSpec comments to all relevant functions, events, and variables.

Remediation Plan:

SOLVED: The ZetaChain team solved the issue by adding the natspec on the contracts.

Commit ID : 46a1be64690774065539868b23aad710e17e6099

4.11 (HAL-11) HAL11 - FLOATING PRAGMA - INFORMATIONAL (0.0)

Description:

Contracts should be deployed with the same compiler version and flags used during development and testing. Locking the pragma helps to ensure that contracts do not accidentally get deployed using another pragma. For example, an outdated pragma version might introduce bugs that affect the contract system negatively.

Code Location:

ZRC20.sol#L2
- Line 2: `pragma solidity ^0.8.7;`

ERC20Custody.sol
- Line 2: `pragma solidity ^0.8.7;`

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:C (0.0)

Recommendation:

Consider locking the pragma version in the smart contracts. It is not recommended to use a floating pragma in production.

For example: `pragma solidity 0.8.17;`

Remediation Plan:

SOLVED: The ZetaChain team solved the issue by deleting floating pragma.

Commit ID : 51dc57035a7f9feb68aa16f45752d46bb5ce78b0

4.12 (HAL-12) HAL12 - MISSING EVENTS ON THE UPDATE FUNCTIONS - INFORMATIONAL (0.0)

Description:

The smart contract's update functions are not emitting events to notify external applications and users of state changes. Events are an essential tool for tracking and reacting to changes in the smart contract's state, as they provide a record of state transitions and can be used to trigger external actions, such as updating a front-end interface or interacting with other smart contracts. By not emitting events in update functions, the smart contract may make it more difficult for developers to build applications that efficiently track and respond to state changes.

Code Location:

/contracts/zevm/ZRC20.sol#LL160-L173C6

Listing 12

```
    function updateSystemContractAddress(address addr) external {
        require(msg.sender == FUNGIBLE_MODULE_ADDRESS, "permission error");
        SYSTEM_CONTRACT_ADDRESS = addr;
    }

    function updateGasLimit(uint256 gasLimit) external {
        require(msg.sender == FUNGIBLE_MODULE_ADDRESS, "permission error");
        GAS_LIMIT = gasLimit;
    }

    function updateProtocolFlatFee(uint256 protocolFlatFee) external {
        require(msg.sender == FUNGIBLE_MODULE_ADDRESS, "permission error");
        PROTOCOL_FLAT_FEE = protocolFlatFee;
    }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:C (0.0)

Recommendation:

To improve the efficiency and usability of the smart contract, we recommend adding events to all update functions that modify the contract's state. These events should provide a clear record of state changes and include all relevant information necessary for external applications and users to react to the changes.

Remediation Plan:

SOLVED: The ZetaChain team solved the issue by adding events to the functions.

Commit ID : 3637741fb8b7e036dd81ca763f9fe2f548bcbfec

4.13 (HAL-13) HAL13 - IMMUTABLE VARIABLES - INFORMATIONAL (0.0)

Description:

In the ZRC20 contract, There are variables that do not change, so these variables can be marked as immutable to reduce gas cost in the contract.

Code Location:

/contracts/zevm/ZRC20.sol#L14

Listing 13

```
contract ZRC20 is Context, IZRC20, IZRC20Metadata, ZRC20Errors {
    address public constant FUNGIBLE_MODULE_ADDRESS = 0x735b14BB79463307AAcBED86DAf3322B1e6226aB;
    address public SYSTEM_CONTRACT_ADDRESS;
    uint256 public CHAIN_ID;
    CoinType public COIN_TYPE;
    uint256 public GAS_LIMIT;
    uint256 public PROTOCOL_FLAT_FEE = 0;
}
```

/contracts/zevm/SystemContract.sol#L23

Listing 14

```
    address public wZetaContractAddress;
    address public uniswapv2FactoryAddress;
    address public uniswapv2Router02Address;
    address public zetaConnectorZEVMAddress;
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:C (0.0)

Recommendation:

Consider marking variables as immutable.

Remediation Plan:

SOLVED: The ZetaChain team solved the issue by marking the variable as immutable.

Commit ID : 51dc57035a7f9feb68aa16f45752d46bb5ce78b0

4.14 (HAL-14) HAL14 - EVENT IS MISSING INDEXED FIELDS - INFORMATIONAL (0.0)

Description:

Index event fields make the field more quickly accessible to off-chain tools that parse events. However, note that each index field costs extra gas during emission, so it’s not necessarily best to index the maximum allowed per event (three fields).

Code Location:

/evm/erc20custody/ERC20Custody.sol#L41

Listing 15

```
    event Deposited(bytes recipient, IERC20 asset, uint256 amount, bytes message);
    event Withdrawn(address recipient, IERC20 asset, uint256 amount);
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:C (0.0)

Recommendation:

Consider adding an index on the events.

Remediation Plan:

SOLVED: The ZetaChain team solved the issue by adding indexes on the events.

Commit ID : 51dc57035a7f9feb68aa16f45752d46bb5ce78b0

4.15 (HAL-15) HAL15 - NO NEED TO INITIALIZE VARIABLES WITH DEFAULT VALUES - INFORMATIONAL (0.0)

Description:

Initialization to 0 or false is not necessary, as these are the default values in Solidity.

Code Location:

/contracts/zevm/ZRC20.sol#L17

Listing 16

```
uint256 public PROTOCOL_FLAT_FEE = 0;
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:C (0.0)

Recommendation:

Remove the initialization values of 0 or false.

Remediation Plan:

SOLVED: The ZetaChain team solved the issue by removing the initialization values of 0.

Commit ID : 3637741fb8b7e036dd81ca763f9fe2f548bcbfec

4.16 (HAL-16) HAL16 - REVERT STRING SIZE OPTIMIZATION - INFORMATIONAL (0.0)

Description:

Shortening revert strings to fit in 32 bytes will decrease deploy time gas and will decrease runtime gas when the revert condition has been met.

Code Location:

/contracts/zevm/ConnectorZEVM.sol#L95

Listing 17

```
    function setWzetaAddress(address _wzeta) external {
        require(msg.sender == FUNGIBLE_MODULE_ADDRESS, "only fungible module can set wzeta address");
        wzeta = _wzeta;
    }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:C (0.0)

Recommendation:

Shorten the revert strings to fit in 32 bytes. Alternatively, the code could be modified to use custom errors, introduced in Solidity 0.8.4.

Remediation Plan:

SOLVED: The ZetaChain team solved the issue by shortening revert strings to 32 bytes.

Commit ID : 51dc57035a7f9feb68aa16f45752d46bb5ce78b0

4.17 (HAL-17) HAL17 - USE CUSTOM ERRORS INSTEAD OF REVERT STRINGS TO SAVE GAS - INFORMATIONAL (0.0)

Description:

Custom errors are available from solidity version 0.8.4. Custom errors save ~50 gas each time they’re hit by avoiding having to allocate and store the revert string. Not defining the strings also saves deployment gas.

Code Location:

/contracts/zevm/ConnectorZEVM.sol#L95

Listing 18

```
    function setWzetaAddress(address _wzeta) external {
        require(msg.sender == FUNGIBLE_MODULE_ADDRESS, ”only
↳ fungible module can set wzeta address”);
        wzeta = _wzeta;
    }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:C (0.0)

Recommendation:

Consider replacing all revert strings with custom errors.

Remediation Plan:

SOLVED: The ZetaChain team solved the issue by adding custom errors.

Commit ID : 3637741fb8b7e036dd81ca763f9fe2f548bcbfec

4.18 (HAL-18) HAL18 - OPTIMIZE UNSIGNED INTEGER COMPARISON - INFORMATIONAL (0.0)

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:

/contracts/zevm/ZRC20.sol#L144

Listing 19

```
    function withdrawGasFee() public override view returns (address,uint256) {
        address gasZRC20 = ISystem(SYSTEM_CONTRACT_ADDRESS).gasCoinZRC20ByChainId(CHAIN_ID);
        require(gasZRC20 != address(0), "gas coin not set");
        uint256 gasPrice = ISystem(SYSTEM_CONTRACT_ADDRESS).gasPriceByChainId(CHAIN_ID);
        require(gasPrice > 0, "gas price not set");
        uint256 gasFee = gasPrice * GAS_LIMIT + PROTOCOL_FLAT_FEE;
        return (gasZRC20, gasFee);
    }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:C (0.0)

Recommendation:

Change `> 0` comparison with `!= 0`.

Remediation Plan:

SOLVED: The ZetaChain team solved the issue by changing require statements.

Commit ID : 3637741fb8b7e036dd81ca763f9fe2f548bcbfec

AUTOMATED TESTING

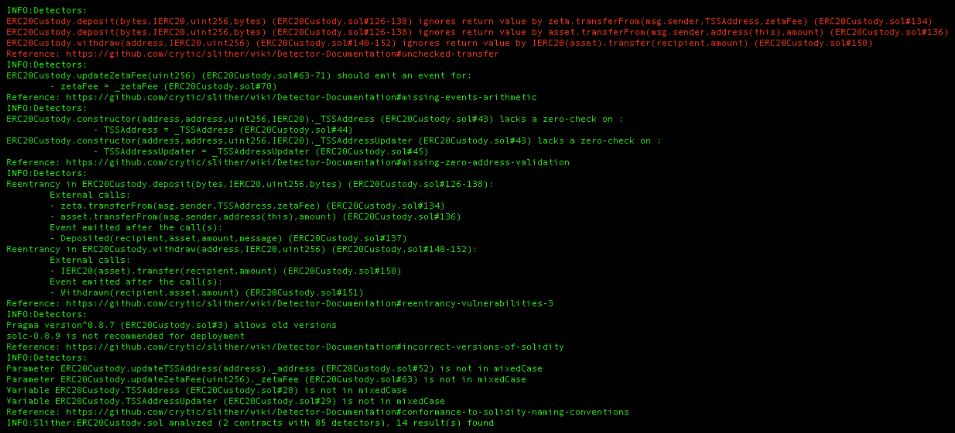
5.1 STATIC ANALYSIS REPORT

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 repository 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.

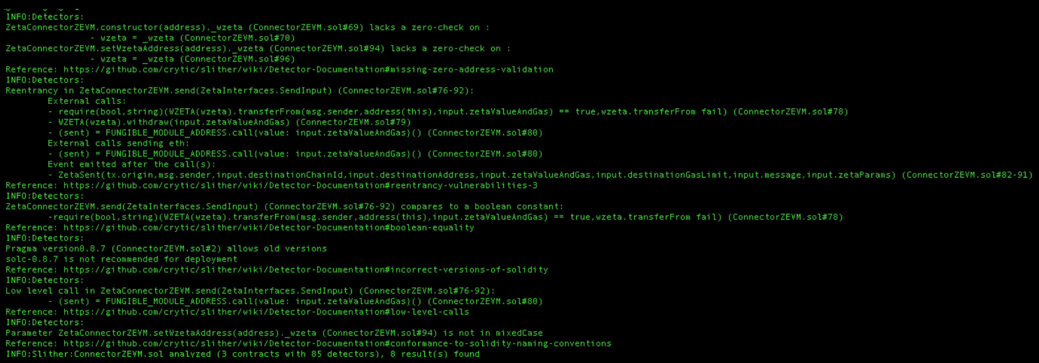
Results:

evm/ERC20Custody.sol



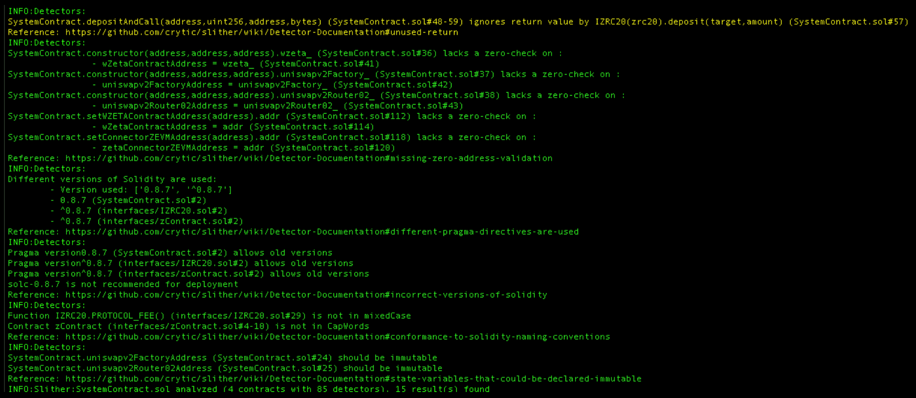
```
INFO:Detectors:
ERC20Custody.deposit(bytes,IERC20,uint256,bytes) (ERC20Custody.sol#126-138) ignores return value by zeta.transferFrom(msg.sender,TSSAddress,zetaFee) (ERC20Custody.sol#134)
ERC20Custody.deposit(bytes,IERC20,uint256,bytes) (ERC20Custody.sol#126-138) ignores return value by asset.transferFrom(msg.sender,address(this),amount) (ERC20Custody.sol#136)
ERC20Custody.withdraw(address,IERC20,uint256) (ERC20Custody.sol#140-152) ignores return value by IERC20(asset).transfer(recipient,amount) (ERC20Custody.sol#150)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
ERC20Custody.updateZetaFee(uint256) (ERC20Custody.sol#63-71) should emit an event for:
        - zetaFee = _zetaFee (ERC20Custody.sol#70)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
ERC20Custody.constructor(address,address,uint256,IERC20)._TSSAddress (ERC20Custody.sol#43) lacks a zero-check on :
                - TSSAddress = _TSSAddress (ERC20Custody.sol#44)
ERC20Custody.constructor(address,address,uint256,IERC20)._TSSAddressUpdater (ERC20Custody.sol#43) lacks a zero-check on :
                - TSSAddressUpdater = _TSSAddressUpdater (ERC20Custody.sol#45)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in ERC20Custody.deposit(bytes,IERC20,uint256,bytes) (ERC20Custody.sol#126-138):
        External calls:
        - zeta.transferFrom(msg.sender,TSSAddress,zetaFee) (ERC20Custody.sol#134)
        - asset.transferFrom(msg.sender,address(this),amount) (ERC20Custody.sol#136)
        Event emitted after the call(s):
        - Deposited(recipient,asset,amount,message) (ERC20Custody.sol#137)
Reentrancy in ERC20Custody.withdraw(address,IERC20,uint256) (ERC20Custody.sol#140-152):
        External calls:
        - IERC20(asset).transfer(recipient,amount) (ERC20Custody.sol#150)
        Event emitted after the call(s):
        - Withdrawn(recipient,asset,amount) (ERC20Custody.sol#151)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Pragma version^0.8.7 (ERC20Custody.sol#3) allows old versions
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter ERC20Custody.updateTSSAddress(address)._address (ERC20Custody.sol#52) is not in mixedCase
Parameter ERC20Custody.updateZetaFee(uint256)._zetaFee (ERC20Custody.sol#63) is not in mixedCase
Variable ERC20Custody.TSSAddress (ERC20Custody.sol#28) is not in mixedCase
Variable ERC20Custody.TSSAddressUpdater (ERC20Custody.sol#29) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Slither:ERC20Custody.sol analyzed (2 contracts with 85 detectors), 14 result(s) found
```

zevm/ConnectorZEVM.sol



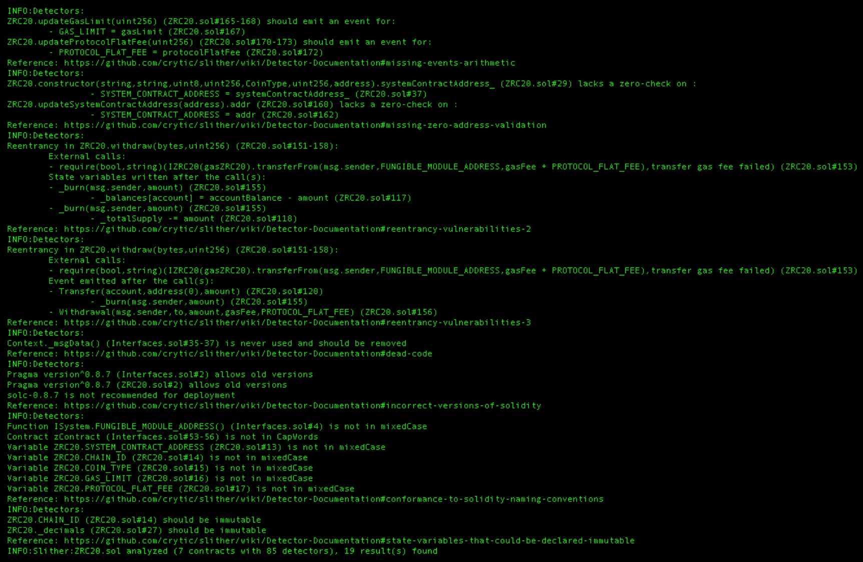
```
INFO:Detectors:
ZetaConnectorZEVM.constructor(address)._wzeta (ConnectorZEVM.sol#69) lacks a zero-check on :
                - wzeta = _wzeta (ConnectorZEVM.sol#70)
ZetaConnectorZEVM.setWzetaAddress(address)._wzeta (ConnectorZEVM.sol#94) lacks a zero-check on :
                - wzeta = _wzeta (ConnectorZEVM.sol#96)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in ZetaConnectorZEVM.send(ZetaInterfaces.SendInput) (ConnectorZEVM.sol#76-92):
        External calls:
        - require(bool,string)(WZETA(wzeta).transferFrom(msg.sender,address(this),input.zetaValueAndGas) == true,wzeta.transferFrom fail) (ConnectorZEVM.sol#78)
        - WZETA(wzeta).withdraw(input.zetaValueAndGas) (ConnectorZEVM.sol#79)
        - (sent) = FUNGIBLE_MODULE_ADDRESS.call{value: input.zetaValueAndGas}() (ConnectorZEVM.sol#80)
        External calls sending eth:
        - (sent) = FUNGIBLE_MODULE_ADDRESS.call{value: input.zetaValueAndGas}() (ConnectorZEVM.sol#80)
        Event emitted after the call(s):
        - ZetaSent(tx.origin,msg.sender,input.destinationChainId,input.destinationAddress,input.zetaValueAndGas,input.destinationGasLimit,input.message,input.zetaParams) (ConnectorZEVM.sol#82-91)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
ZetaConnectorZEVM.send(ZetaInterfaces.SendInput) (ConnectorZEVM.sol#76-92) compares to a boolean constant:
        -require(bool,string)(WZETA(wzeta).transferFrom(msg.sender,address(this),input.zetaValueAndGas) == true,wzeta.transferFrom fail) (ConnectorZEVM.sol#78)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Pragma version0.8.7 (ConnectorZEVM.sol#2) allows old versions
solc-0.8.7 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in ZetaConnectorZEVM.send(ZetaInterfaces.SendInput) (ConnectorZEVM.sol#76-92):
        - (sent) = FUNGIBLE_MODULE_ADDRESS.call{value: input.zetaValueAndGas}() (ConnectorZEVM.sol#80)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter ZetaConnectorZEVM.setWzetaAddress(address)._wzeta (ConnectorZEVM.sol#94) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Slither:ConnectorZEVM.sol analyzed (3 contracts with 85 detectors), 8 result(s) found
```

zevm/SystemContract.sol



```
INFO:Detectors:
SystemContract.depositAndCall(address,uint256,address,bytes) (SystemContract.sol#48-59) ignores return value by IZRC20(zrc20).deposit(target,amount) (SystemContract.sol#57)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
SystemContract.constructor(address,address,address).wzeta_ (SystemContract.sol#36) lacks a zero-check on :
                - wZetaContractAddress = wzeta_ (SystemContract.sol#41)
SystemContract.constructor(address,address,address).uniswapv2Factory_ (SystemContract.sol#37) lacks a zero-check on :
                - uniswapv2FactoryAddress = uniswapv2Factory_ (SystemContract.sol#42)
SystemContract.constructor(address,address,address).uniswapv2Router02_ (SystemContract.sol#38) lacks a zero-check on :
                - uniswapv2Router02Address = uniswapv2Router02_ (SystemContract.sol#43)
SystemContract.setWZETAContractAddress(address).addr (SystemContract.sol#112) lacks a zero-check on :
                - wZetaContractAddress = addr (SystemContract.sol#114)
SystemContract.setConnectorZEVMAddress(address).addr (SystemContract.sol#118) lacks a zero-check on :
                - zetaConnectorZEVMAddress = addr (SystemContract.sol#120)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Different versions of Solidity are used:
        - Version used: ['0.8.7', '^0.8.7']
        - 0.8.7 (SystemContract.sol#2)
        - ^0.8.7 (interfaces/IZRC20.sol#2)
        - ^0.8.7 (interfaces/zContract.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Pragma version0.8.7 (SystemContract.sol#2) allows old versions
Pragma version^0.8.7 (interfaces/IZRC20.sol#2) allows old versions
Pragma version^0.8.7 (interfaces/zContract.sol#2) allows old versions
solc-0.8.7 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Function IZRC20.PROTOCOL_FEE() (interfaces/IZRC20.sol#29) is not in mixedCase
Contract zContract (interfaces/zContract.sol#4-10) is not in CapWords
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
SystemContract.uniswapv2FactoryAddress (SystemContract.sol#24) should be immutable
SystemContract.uniswapv2Router02Address (SystemContract.sol#25) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
INFO:Slither:SystemContract.sol analyzed (4 contracts with 85 detectors), 15 result(s) found
```

zevm/ZRC20.sol



```
INFO:Detectors:
ZRC20.updateGasLimit(uint256) (ZRC20.sol#165-168) should emit an event for:
        - GAS_LIMIT = gasLimit (ZRC20.sol#167)
ZRC20.updateProtocolFlatFee(uint256) (ZRC20.sol#170-173) should emit an event for:
        - PROTOCOL_FLAT_FEE = protocolFlatFee (ZRC20.sol#172)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
ZRC20.constructor(string,string,uint8,uint256,CoinType,uint256,address).systemContractAddress_ (ZRC20.sol#29) lacks a zero-check on :
                - SYSTEM_CONTRACT_ADDRESS = systemContractAddress_ (ZRC20.sol#37)
ZRC20.updateSystemContractAddress(address).addr (ZRC20.sol#160) lacks a zero-check on :
                - SYSTEM_CONTRACT_ADDRESS = addr (ZRC20.sol#162)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in ZRC20.withdraw(bytes,uint256) (ZRC20.sol#151-158):
        External calls:
        - require(bool,string)(IZRC20(gasZRC20).transferFrom(msg.sender,FUNGIBLE_MODULE_ADDRESS,gasFee + PROTOCOL_FLAT_FEE),transfer gas fee failed) (ZRC20.sol#153)
        State variables written after the call(s):
        - _burn(msg.sender,amount) (ZRC20.sol#155)
                - _balances[account] = accountBalance - amount (ZRC20.sol#117)
        - _burn(msg.sender,amount) (ZRC20.sol#155)
                - _totalSupply -= amount (ZRC20.sol#118)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in ZRC20.withdraw(bytes,uint256) (ZRC20.sol#151-158):
        External calls:
        - require(bool,string)(IZRC20(gasZRC20).transferFrom(msg.sender,FUNGIBLE_MODULE_ADDRESS,gasFee + PROTOCOL_FLAT_FEE),transfer gas fee failed) (ZRC20.sol#153)
        Event emitted after the call(s):
        - Transfer(account,address(0),amount) (ZRC20.sol#120)
                - _burn(msg.sender,amount) (ZRC20.sol#155)
        - Withdrawal(msg.sender,to,amount,gasFee,PROTOCOL_FLAT_FEE) (ZRC20.sol#156)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Context._msgData() (Interfaces.sol#35-37) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.7 (Interfaces.sol#2) allows old versions
Pragma version^0.8.7 (ZRC20.sol#2) allows old versions
solc-0.8.7 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Function ISystem.FUNGIBLE_MODULE_ADDRESS() (Interfaces.sol#4) is not in mixedCase
Contract zContract (Interfaces.sol#53-56) is not in CapWords
Variable ZRC20.SYSTEM_CONTRACT_ADDRESS (ZRC20.sol#13) is not in mixedCase
Variable ZRC20.CHAIN_ID (ZRC20.sol#14) is not in mixedCase
Variable ZRC20.COIN_TYPE (ZRC20.sol#15) is not in mixedCase
Variable ZRC20.GAS_LIMIT (ZRC20.sol#16) is not in mixedCase
Variable ZRC20.PROTOCOL_FLAT_FEE (ZRC20.sol#17) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
ZRC20.CHAIN_ID (ZRC20.sol#14) should be immutable
ZRC20._decimals (ZRC20.sol#27) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
INFO:Slither:ZRC20.sol analyzed (7 contracts with 85 detectors), 19 result(s) found
```

- All the reentrancies flagged by Slither were checked individually and are false positives.
- No major issues found by Slither.

5.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues and to identify low-hanging fruits on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the smart contracts and sent the compiled results to the analyzers to locate any vulnerabilities.

MythX results:

- No major issues found by MythX.

THANK YOU FOR CHOOSING

HALBORN