// Security Assessment

05.12.2025 - 05.15.2025

LZ Connector

Bonzo Finance

HALBORN

Submit Feedback

LZ Connector - Bonzo Finance

Prepared by: HALBORN Last Updated 05/28/2025

Date of Engagement: May 12th, 2025 - May 15th, 2025

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
6	1	0	0	1	4

1. Introduction

Bonzo Finance engaged Halborn to conduct a security assessment on their smart contracts beginning on May 12th, 2025 and ending on May 17th, 2025. The security assessment was scoped to the smart contracts provided in the Bonzo-Labs/lz-connector-contracts Github repository provided to Halborn . Further details can be found in the Scope section of this report.

TABLE OF CONTENTS

2. Assessment Summary

Halborn was provided 5 (five) days for the engagement, and assigned one full-time security engineer to review the security of the smart contracts in scope. The engineer is a blockchain and smart contract security expert with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the smart contracts.
- Ensure that smart contract functionality operates as intended.

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were partially addressed by the Bonzo Finance team. The main ones were the following:

- Implement appropriate access control mechanisms for the mint and burn functions in MyOFT contract. This can be achieved by using modifiers like onlyOwner or by implementing role-based access control to ensure that only authorized entities can perform these operations.
- Use a local variable to store the loop computation results instead of low-level SSTORE opcode.
- Update Hedera's dependencies and import the latest version of HTS.

3. Test Approach And 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 assessment. 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 assessment:

- 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.
- Static Analysis of security for scoped contract, and imported functions (`slither`).
- Testnet deployment (`Foundry`).

4. 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.

4.1 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:

$$E = \prod m_e$$

4.2 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) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

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

4.3 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.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 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

5. SCOPE

FILES AND REPOSITORY

(a) Repository: lz-connector-contracts

(b) Assessed Commit ID: af7f1f5

(c) Items in scope:

- BaseHTSConnector.sol
- HTSConnector.sol
- MyOFT.sol
- BaseOFTAdapter.sol

Out-of-Scope: Third-party dependencies and economic attacks.

REMEDIATION COMMIT ID:

- 64f2f1b
- d506352

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL HIGH MEDIUM LOW

1 0 0 1

INFORMATIONAL
4

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
MISSING ACCESS CONTROL IN MINT`AND BURN` FUNCTIONS	CRITICAL	SOLVED - 05/22/2025
USAGE OF SSTORE INSIDE LOOPS INCREASES GAS COSTS	LOW	SOLVED - 05/22/2025
INCORRECT NATSPEC DOCUMENTATION	INFORMATIONAL	ACKNOWLEDGED - 05/26/2025
INCORRECT HTS IMPORTS	INFORMATIONAL	ACKNOWLEDGED - 05/26/2025
MISPLACED NONREENTRANT MODIFIER	INFORMATIONAL	ACKNOWLEDGED - 05/26/2025

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
USE EXPLICIT SIZE DECLARATIONS	INFORMATIONAL	ACKNOWLEDGED - 05/26/2025

7. FINDINGS & TECH DETAILS

7.1 MISSING ACCESS CONTROL IN `MINT` AND `BURN` FUNCTIONS

// CRITICAL

Description

The basic implementation of `MyOFT.sol`, as recommended by LayerZero in the official documentation, includes a `_mint` function that should either be called within the constructor or exposed with proper access control mechanisms.

In the current implementation of the provided `MyOFT.sol` contract, both the `mint` and `burn` public functions lack access control measures, such as `onlyOwner` or other role-based or address-based mechanisms. This oversight allows any low-privileged user to mint or burn tokens freely, posing a significant security risk and potentially leading to total drainage of liquidity.

```
    function mint(address _to, uint256 _amount) public {
    function burn(address _from, uint256 _amount) public {
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:C/D:C/Y:N (10.0)

Recommendation

To mitigate this issue, it is recommended to implement appropriate access control mechanisms for the `mint` and `burn` functions. This can be achieved by using modifiers like `onlyOwner` or by implementing role-based access control to ensure that only authorized entities can perform these operations.

Remediation Comment

SOLVED: The **Bonzo team** has solved this issue as recommended.

Remediation Hash

https://github.com/Bonzo-Labs/lz-connector-contracts/commit/64f2f1bc0d7a9232e441e63279823d40fb793929

7.2 USAGE OF SSTORE INSIDE LOOPS INCREASES GAS COSTS

// LOW

Description

The use of the `SSTORE` opcode within loops can result in significant gas inefficiencies and may lead to out-of-gas errors.

This practice is particularly problematic because each `SSTORE` operation consumes a substantial amount of gas, and when executed repeatedly within a loop, it can quickly deplete the gas limit. This can cause transactions to fail, leading to unexpected behavior and potential financial losses.

Found in:

- `BaseOFTAdapter.sol - Line 178`
- `HTSConnector - Line 310`

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:L/I:N/D:N/Y:N (2.5)

Recommendation

It is recommended to use a local variable to store the loop computation results instead of low-level `SSTORE` opcode.

Remediation Comment

SOLVED: The **Bonzo team** has solved this issue as recommended.

Remediation Hash

https://github.com/Bonzo-Labs/lz-connector-contracts/commit/d5063520c2efb773ab4cff01b69887b6e5199507

7.3 INCORRECT NATSPEC DOCUMENTATION

// INFORMATIONAL

Description

The NatSpec documentation on `clearOldTransferRecords` in the `HTSConnector` contract affirms `"Can be called by users to clear their own history or by owner for any user"`, but in fact the function is currently access-controled via `onlyOwner` modifier.

```
    /**
     * @notice Removes oldest transfer records for a user to manage array growth
     * @dev Can be called by users to clear their own history or by owner for any user
     * @param user Address of the user whose transfer history to clear
     * @param count Number of oldest records to remove
     */
    function clearOldTransferRecords(
```

BVSS

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

Recommendation

It is recommended to adjust the NatSpec documentation to reflect the actual function implementation.

Remediation Comment

ACKNOWLEDGED: The **Bonzo team** has acknowledged this finding.

7.4 INCORRECT HTS IMPORTS

// INFORMATIONAL

Description

In the current state of the provided codebase, the contracts within the `hts/` directory are outdated and lack essential functions such as `associateToken()` , `dissociateToken()` and others. These functions are crucial for proper interactions with HTS tokens, including associating the `HTSConnector` to the newly created token and subsequent token operations for end users.

The `HTSConnector` and `BaseHTSConnector` contracts inherit from the parent contracts located in the `hts/` directory. Consequently, the code does not implement the correct version of HTS dependencies but only partial implementations of presumably deprecated code.

BVSS

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

Recommendation

It is recommended to update Hedera's dependencies and import the latest version of HTS.

Remediation Comment

ACKNOWLEDGED: The **Bonzo team** has acknowledged this finding.

7.5 MISPLACED NONREENTRANT MODIFIER

// INFORMATIONAL

Description

It is best practice to place the `nonReentrant` modifier before all other function modifiers, so eventual reentrancy issues in other modifiers are also protected by Reentrancy Guard.

Found in:

- `BaseHTSConnector - Lines 60, 102`
- `HTSConnector - Lines 164, 229, 266, 299`
- `BaseOFTAdapter - Lines 100, 147, 167`

BVSS

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

Recommendation

Adjust function declarations and put the `nonReentrant` modifier before all other modifiers.

Remediation Comment

ACKNOWLEDGED: The **Bonzo team** has acknowledged this finding.

7.6 USE EXPLICIT SIZE DECLARATIONS

// INFORMATIONAL

Description

Throughout the codebase, there are occurrences where integer types are declared inconsistently or implicitly (e.g., `uint` vs. `uint256`, `int` vs. `int256`). Using explicit, fixed-size integer declarations (like `uint256` or `int256`) improves code clarity, avoids ambiguity, and helps prevent overflow/underflow issues when interfacing with external contracts or libraries that expect a specific integer size.

Found in:

- `BaseOFTAdapter.sol - Line 222`
- `HTSConnector - Lines 113, 354`

BVSS

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

Recommendation

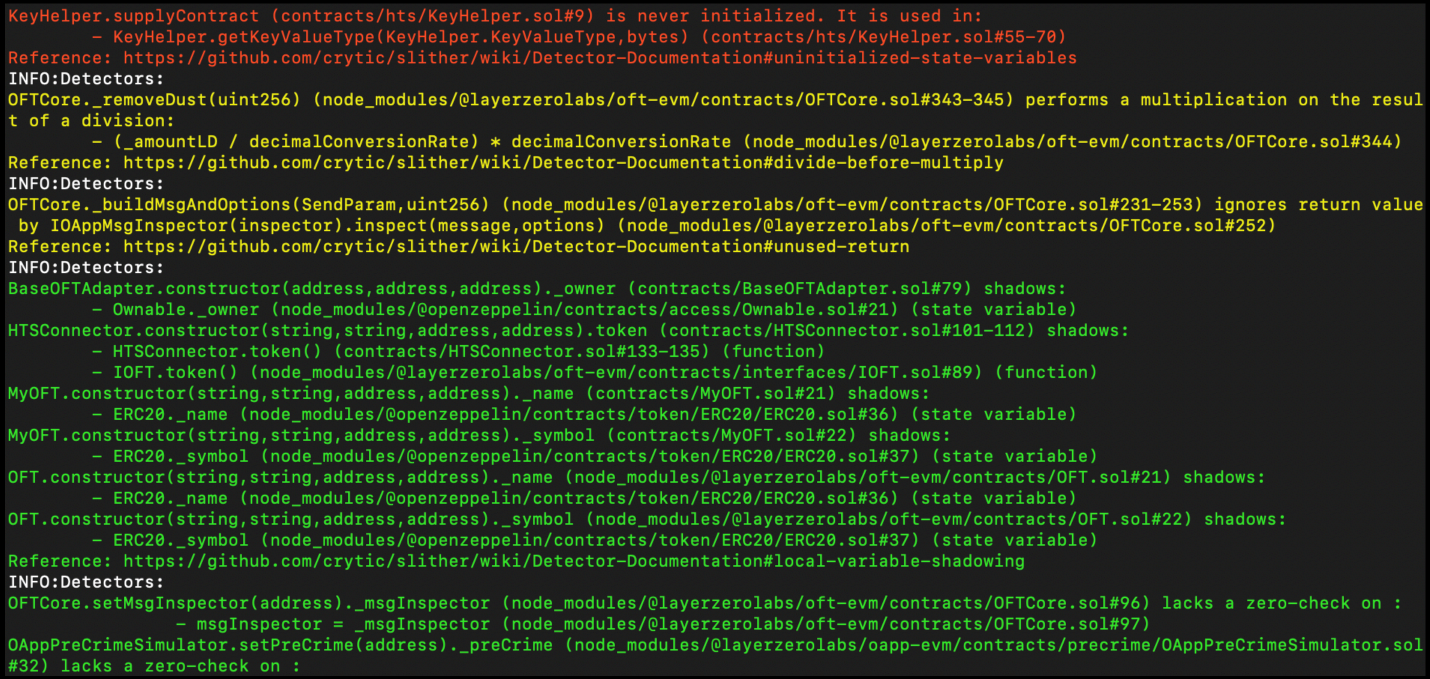
Use explicit types (`uint256`, `int256`, `uint128`, etc.) throughout the code, in order to maintain consistency and enhance readability.

Remediation Comment

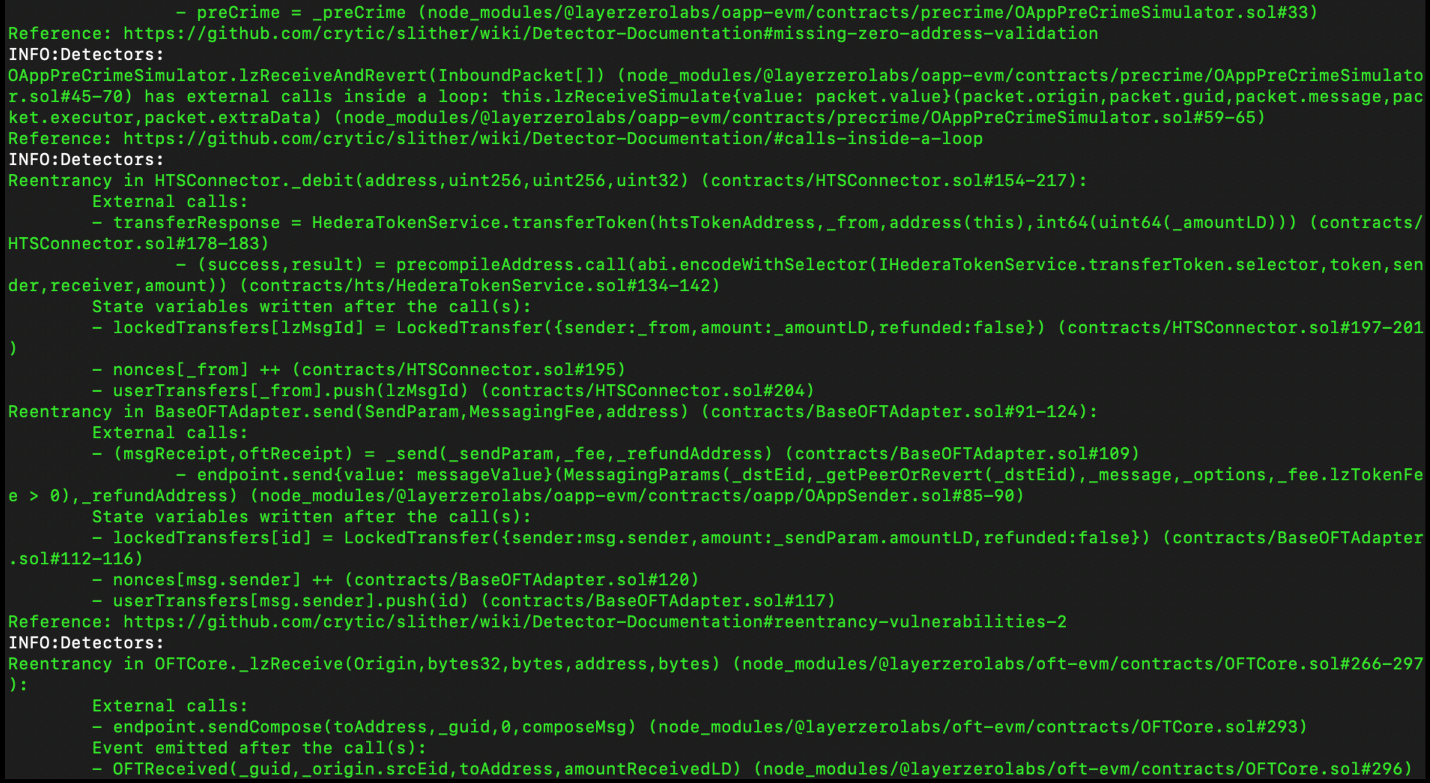
ACKNOWLEDGED: The **Bonzo team** has acknowledged this finding.

8. AUTOMATED TESTING

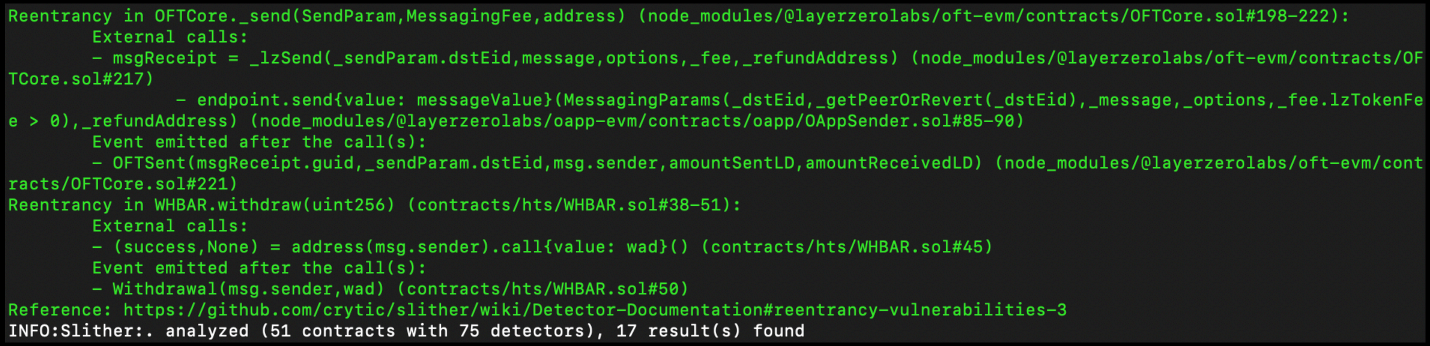
Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their ABIs 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.



```
KeyHelper.supplyContract (contracts/hts/KeyHelper.sol#9) is never initialized. It is used in:
        - KeyHelper.getKeyValueType(KeyHelper.KeyValueType,bytes) (contracts/hts/KeyHelper.sol#55-70)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables
INFO:Detectors:
OFTCore._removeDust(uint256) (node_modules/@layerzerolabs/oft-evm/contracts/OFTCore.sol#343-345) performs a multiplication on the resul
t of a division:
        - (_amountLD / decimalConversionRate) * decimalConversionRate (node_modules/@layerzerolabs/oft-evm/contracts/OFTCore.sol#344)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
OFTCore._buildMsgAndOptions(SendParam,uint256) (node_modules/@layerzerolabs/oft-evm/contracts/OFTCore.sol#231-253) ignores return value
 by IOAppMsgInspector(inspector).inspect(message,options) (node_modules/@layerzerolabs/oft-evm/contracts/OFTCore.sol#252)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
BaseOFTAdapter.constructor(address,address,address)._owner (contracts/BaseOFTAdapter.sol#79) shadows:
        - Ownable._owner (node_modules/@openzeppelin/contracts/access/Ownable.sol#21) (state variable)
HTSConnector.constructor(string,string,address,address).token (contracts/HTSConnector.sol#101-112) shadows:
        - HTSConnector.token() (contracts/HTSConnector.sol#133-135) (function)
        - IOFT.token() (node_modules/@layerzerolabs/oft-evm/contracts/interfaces/IOFT.sol#89) (function)
MyOFT.constructor(string,string,address,address)._name (contracts/MyOFT.sol#21) shadows:
        - ERC20._name (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#36) (state variable)
MyOFT.constructor(string,string,address,address)._symbol (contracts/MyOFT.sol#22) shadows:
        - ERC20._symbol (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#37) (state variable)
OFT.constructor(string,string,address,address)._name (node_modules/@layerzerolabs/oft-evm/contracts/OFT.sol#21) shadows:
        - ERC20._name (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#36) (state variable)
OFT.constructor(string,string,address,address)._symbol (node_modules/@layerzerolabs/oft-evm/contracts/OFT.sol#22) shadows:
        - ERC20._symbol (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#37) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
OFTCore.setMsgInspector(address)._msgInspector (node_modules/@layerzerolabs/oft-evm/contracts/OFTCore.sol#96) lacks a zero-check on :
                - msgInspector = _msgInspector (node_modules/@layerzerolabs/oft-evm/contracts/OFTCore.sol#97)
OAppPreCrimeSimulator.setPreCrime(address)._preCrime (node_modules/@layerzerolabs/oapp-evm/contracts/precrime/OAppPreCrimeSimulator.sol
#32) lacks a zero-check on :
```



```
                - preCrime = _preCrime (node_modules/@layerzerolabs/oapp-evm/contracts/precrime/OAppPreCrimeSimulator.sol#33)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
OAppPreCrimeSimulator.lzReceiveAndRevert(InboundPacket[]) (node_modules/@layerzerolabs/oapp-evm/contracts/precrime/OAppPreCrimeSimulato
r.sol#45-70) has external calls inside a loop: this.lzReceiveSimulate{value: packet.value}(packet.origin,packet.guid,packet.message,pac
ket.executor,packet.extraData) (node_modules/@layerzerolabs/oapp-evm/contracts/precrime/OAppPreCrimeSimulator.sol#59-65)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
Reentrancy in HTSConnector._debit(address,uint256,uint256,uint32) (contracts/HTSConnector.sol#154-217):
        External calls:
        - transferResponse = HederaTokenService.transferToken(htsTokenAddress,_from,address(this),int64(uint64(_amountLD))) (contracts/
HTSConnector.sol#178-183)
                - (success,result) = precompileAddress.call(abi.encodeWithSelector(IHederaTokenService.transferToken.selector,token,sen
der,receiver,amount)) (contracts/hts/HederaTokenService.sol#134-142)
        State variables written after the call(s):
        - lockedTransfers[lzMsgId] = LockedTransfer({sender:_from,amount:_amountLD,refunded:false}) (contracts/HTSConnector.sol#197-201
)
        - nonces[_from] ++ (contracts/HTSConnector.sol#195)
        - userTransfers[_from].push(lzMsgId) (contracts/HTSConnector.sol#204)
Reentrancy in BaseOFTAdapter.send(SendParam,MessagingFee,address) (contracts/BaseOFTAdapter.sol#91-124):
        External calls:
        - (msgReceipt,oftReceipt) = _send(_sendParam,_fee,_refundAddress) (contracts/BaseOFTAdapter.sol#109)
                - endpoint.send{value: messageValue}(MessagingParams(_dstEid,_getPeerOrRevert(_dstEid),_message,_options,_fee.lzTokenFe
e > 0),_refundAddress) (node_modules/@layerzerolabs/oapp-evm/contracts/oapp/OAppSender.sol#85-90)
        State variables written after the call(s):
        - lockedTransfers[id] = LockedTransfer({sender:msg.sender,amount:_sendParam.amountLD,refunded:false}) (contracts/BaseOFTAdapter
.sol#112-116)
        - nonces[msg.sender] ++ (contracts/BaseOFTAdapter.sol#120)
        - userTransfers[msg.sender].push(id) (contracts/BaseOFTAdapter.sol#117)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in OFTCore._lzReceive(Origin,bytes32,bytes,address,bytes) (node_modules/@layerzerolabs/oft-evm/contracts/OFTCore.sol#266-297
):
        External calls:
        - endpoint.sendCompose(toAddress,_guid,0,composeMsg) (node_modules/@layerzerolabs/oft-evm/contracts/OFTCore.sol#293)
        Event emitted after the call(s):
        - OFTReceived(_guid,_origin.srcEid,toAddress,amountReceivedLD) (node_modules/@layerzerolabs/oft-evm/contracts/OFTCore.sol#296)
```



```
Reentrancy in OFTCore._send(SendParam,MessagingFee,address) (node_modules/@layerzerolabs/oft-evm/contracts/OFTCore.sol#198-222):
        External calls:
        - msgReceipt = _lzSend(_sendParam.dstEid,message,options,_fee,_refundAddress) (node_modules/@layerzerolabs/oft-evm/contracts/OF
TCore.sol#217)
                - endpoint.send{value: messageValue}(MessagingParams(_dstEid,_getPeerOrRevert(_dstEid),_message,_options,_fee.lzTokenFe
e > 0),_refundAddress) (node_modules/@layerzerolabs/oapp-evm/contracts/oapp/OAppSender.sol#85-90)
        Event emitted after the call(s):
        - OFTSent(msgReceipt.guid,_sendParam.dstEid,msg.sender,amountSentLD,amountReceivedLD) (node_modules/@layerzerolabs/oft-evm/cont
racts/OFTCore.sol#221)
Reentrancy in WHBAR.withdraw(uint256) (contracts/hts/WHBAR.sol#38-51):
        External calls:
        - (success,None) = address(msg.sender).call{value: wad}() (contracts/hts/WHBAR.sol#45)
        Event emitted after the call(s):
        - Withdrawal(msg.sender,wad) (contracts/hts/WHBAR.sol#50)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Slither:. analyzed (51 contracts with 75 detectors), 17 result(s) found
```

All issues identified by Slither were proved to be false positives or have been added to the issue list in this report.

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.