Aura Finance - Sidechain

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **May 9th, 2023 - June 6th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	06/02/2023	Luis Buendia
0.2	Document Updates	06/06/2023	Luis Buendia
0.3	Draft Review	06/06/2023	Gokberk Gulgun
0.4	Draft Review	06/06/2023	Gabi Urrutia
1.0	Remediation Plan	06/12/2023	Luis Buendia
1.1	Remediation Plan Review	06/14/2023	Gokberk Gulgun
1.2	Remediation Plan Review	06/14/2023	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
Luis Buendia	Halborn	Luis.Buendia@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Aura Finance engaged Halborn to conduct a security audit on their smart contracts beginning on May 9th, 2023 and ending on June 6th, 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 four 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 mostly addressed by the Aura Finance team.

1.3 SCOPE

IN-SCOPE:

The security assessment was scoped to the following Aura Repository :

- AuraBalOFT.sol
- AuraBalProxyOFT.sol
- AuraOFT.sol
- AuraProxyOFT.sol
- Create2Factory.sol
- CrossChainConfig.sol
- CrossChainConfigMessages.sol
- L1Coordinator.sol
- L2Coordinator.sol
- PausableOFT.sol
- PausableProxyOFT.sol
- PauseGuardian.sol
- BridgeDelegateReceiver.sol
- BridgeDelegateSender.sol
- GnosisBridgeSender.sol
- SimpleBridgeDelegateSender.sol

Aura Smart Contracts Commit ID: 3bfc8cb9ae76cbc7a8ba08b8717cefba0d17c82e

Also, the next contracts from convex are included Convex Smart Contracts

- BoosterLite.sol
- PoolManagerLite.sol
- BaseRewardPool4626.sol
- VoterProxyLite.sol

Convex Smart Contracts Commit ID: 3cd1ce3657bae8abb975b9dd06f28247c22880d3

REMEDIATION COMMITS:

- Commit IDs:
  - 5f716ad8d0cf997e951d8d7d58dd7a2568d2658e
  - 079274b5875ea20cefb32860556d1d61970a6c81
  - 8e17d3dfab9272b84cdcd5cbe5d35f9356fd51b6
  - 9198edb43afdc782d5ad5b28565a4e81234624bb
  - b5baaa08f12078d8936ff0bfcf159eb901150e14

1.4 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 (Anvil).

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

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	3	10

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
SETTING RECEIVER TO MSG.SENDER CAN LOCK FUNDS PERMANENTLY	Medium (6.3)	SOLVED - 06/09/2023
DO NOT DECREASE INTERNAL TOTAL SUPPLY IF RESCUED TOKEN IS NOT INNER TOKEN	Low (2.3)	SOLVED - 06/09/2023
STAKEALL FUNCTION CAN STAKE WRONG AMOUNT	Low (2.3)	SOLVED - 06/09/2023
CONTROL toAddress SIZE	Low (2.3)	RISK ACCEPTED
MEASURE BALANCE FOR REWARD DISTRUBUTION	Informational (0.0)	ACKNOWLEDGED
EVENT IS NOT EMITTED CORRECTLY ON HARVEST FUNCTION	Informational (0.0)	SOLVED - 06/09/2023
FUNCTIONS SHOULD BE PAUSABLE	Informational (0.0)	SOLVED - 06/09/2023
CHANGE STRINGS FOR CUSTOM ERRORS TO SAVE GAS	Informational (0.0)	ACKNOWLEDGED
LACK OF REENTRANCY PROTECTION	Informational (0.0)	SOLVED - 06/09/2023
INCONSISTENT NAMING CONVENTION	Informational (0.0)	ACKNOWLEDGED
LACK OF UPGRADABILITY PATTERN	Informational (0.0)	ACKNOWLEDGED
CENTRALIZATION RISK	Informational (0.0)	ACKNOWLEDGED
EXTERNAL CALL ON LOOP	Informational (0.0)	ACKNOWLEDGED
LACK OF TWO STEP OWNERSHIP TRANSFER	Informational (0.0)	ACKNOWLEDGED

FINDINGS & TECH DETAILS

4.1 (HAL-01) SETTING RECEIVER TO MSG.SENDER CAN LOCK FUNDS PERMANENTLY - MEDIUM (6.3)

Description:

The function lock from the contract AuraOFT.sol uses the msg.sender as payload parameter to send the address through LayerZero to indicate the contract of the main chain the address that can withdraw the specified amount of tokens from the Locker.sol contract.

The addresses on a side chain do not need to correspond in all cases with the address on the other chain. This situation is specific for smart contracts. So, if this function is used through an SC, the tokens may get locked in the 'Locker' without the chance of recovering it.

Code Location:

Listing 1: AuraOFT.sol (Line 63)

```
59 function lock(uint256 _cvxAmount) external payable {
60     require(_cvxAmount > 0, "!amount");
61     _debitFrom(msg.sender, canonicalChainId, bytes(""), _cvxAmount
↳ );
62
63     bytes memory payload = CCM.encodeLock(msg.sender, _cvxAmount);
64
65     CrossChainConfig.Config memory config = configs[
↳ canonicalChainId][AuraOFT.lock.selector];
66
67     _lzSend(
68         canonicalChainId, ////////// Parent chain ID
69         payload, /////////////////// Payload
70         payable(msg.sender), /////// Refund address
71         config.zroPaymentAddress, // ZRO payment address
72         config.adapterParams, ////// Adapter params
73         msg.value ///////////////// Native fee
74     );
```

```
    emit Locked(msg.sender, _cvxAmount);
}
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:N/A:N/D:H/Y:N/R:N/S:C (6.3)

Recommendation:

Consider implementing a receiver address as a parameter for the lock function.

Remediation Plan:

SOLVED: The Aura Finance team solved the issue by adding a receiver address as a parameter on the following commit ID:

- 5f716ad8d0cf997e951d8d7d58dd7a2568d2658e

4.2 (HAL-02) DO NOT DECREASE INTERNAL TOTAL SUPPLY IF RESCUED TOKEN IS NOT INNER TOKEN - LOW (2.3)

Description:

The function rescue from the AuraBalProxyOFT.sol contract is implemented to allow the owner of the contract to give back the tokens to a user in case of an accidental transfer to the contract. This function allows transferring **any** token in the contract to the specified address.

However, the function subtracts the indicated amount as parameter to the internalTotalSupply. So, if the token is not the inner token, this may cause the contract to stop working in the long term.

Code Location:

Listing 2: AuraBalProxyOFT.sol (Line 328)

```
318 function rescue(
319     address _token,
320     address _to,
321     uint256 _amount
322 ) external override {
323     require(msg.sender == sudo, "!sudo");
324
325     // Adjust the internalTotalSupply. This means we have to
    ↳ harvest and process
326     // any rewards if we want to rescue the entire
    ↳ underlyingBalance of the bridge
327     // otherwise this will underflow
328     internalTotalSupply -= _amount;
329
330     if (_token == address(innerToken)) {
331         _withdraw(_amount);
332     }
333     IERC20(_token).safeTransfer(_to, _amount);
334 }
```

BVSS:

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

Recommendation:

Consider placing the subtraction inside the if statement, to ensure the correct handling of the internalTotalSupply. Moreover, it can also be considered a third case, where the inner token is stacked on the contract but is not inside the vault, so it will not be required to do the subtraction, Although it is the inner token.

The next code snippets illustrate the first suggested approach.

Listing 3: AuraBalProxyOFT.sol (Line 329)

```
318 function rescue(
319     address _token,
320     address _to,
321     uint256 _amount
322 ) external override {
323     require(msg.sender == sudo, "!sudo");
324
325     // Adjust the internalTotalSupply. This means we have to
    ↳ harvest and process
326     // any rewards if we want to rescue the entire
    ↳ underlyingBalance of the bridge
327     // otherwise this will underflo
328     if (_token == address(innerToken)) {
329         internalTotalSupply -= _amount;
330         _withdraw(_amount);
331     }
332
333     IERC20(_token).safeTransfer(_to, _amount);
334 }
```

Also, as suggested, there can exist other mitigation that also evaluates if the innerToken is inside the vault.

Listing 4: AuraBalProxyOFT.sol (Lines 322,329)

```
318 function rescue(
319     address _token,
320     address _to,
321     uint256 _amount
322     bool insideVault;
323 ) external override {
324     require(msg.sender == sudo, "!sudo");
325
326     // Adjust the internalTotalSupply. This means we have to
  ↳ harvest and process
327     // any rewards if we want to rescue the entire
  ↳ underlyingBalance of the bridge
328     // otherwise this will underflo
329     if (_token == address(innerToken) && insideVault) {
330         internalTotalSupply -= _amount;
331         _withdraw(_amount);
332     }
333
334     IERC20(_token).safeTransfer(_to, _amount);
335 }
```

Remediation Plan:

SOLVED: The Aura Finance team solved the issue by moving the storage update inside the if statement on the following commit ID:

- 5f716ad8d0cf997e951d8d7d58dd7a2568d2658e

4.3 (HAL-03) STAKEALL FUNCTION CAN STAKE WRONG AMOUNT - LOW (2.3)

Description:

The _stakeAll function from the AuraBalProxyOFT.solcontract stakes all the balance of the inner token in the vault. However, this can amount can be different from the actual amount transferred, measured on the trace of the execution by the inherited _debitFrom function from the ProxyOFT.sol contract.

This can create accounting problems on the internalTotalSupply state variable and, moreover, affect the whole bridging system.

Code Location:

Listing 5: AuraBalProxyOFT.sol

```
function _stakeAll() internal {
    uint256 amount = innerToken.balanceOf(address(this));
    IGenericVault(vault).deposit(amount, address(this));
}
```

Listing 6: AuraBalProxyOFT.sol (Lines 172,174)

```
166 function _debitFrom(
167     address _from,
168     uint16 _srcChainId,
169     bytes memory _toAddress,
170     uint256 _amount
171 ) internal override returns (uint256) {
172     uint256 amount = super._debitFrom(_from, _srcChainId,
      ↳ _toAddress, _amount);
173     internalTotalSupply += amount;
174     _stakeAll();
175     return amount;
176 }
```

Listing 7: ProxyOFT.sol (Line 36)

```
27 function _debitFrom(
28     address _from,
29     uint16,
30     bytes memory,
31     uint256 _amount
32 ) internal virtual override returns (uint256) {
33     require(_from == _msgSender(), "ProxyOFT: owner is not send
   ↳ caller");
34     uint256 before = innerToken.balanceOf(address(this));
35     innerToken.safeTransferFrom(_from, address(this), _amount);
36     return innerToken.balanceOf(address(this)) - before;
37 }
```

Listing 8: OFCore.sol (Line 83)

```
72 function _send(
73     address _from,
74     uint16 _dstChainId,
75     bytes memory _toAddress,
76     uint256 _amount,
77     address payable _refundAddress,
78     address _zroPaymentAddress,
79     bytes memory _adapterParams
80 ) internal virtual {
81     _checkAdapterParams(_dstChainId, PT_SEND, _adapterParams,
   ↳ NO_EXTRA_GAS);
82
83     uint256 amount = _debitFrom(_from, _dstChainId, _toAddress,
   ↳ _amount);
84
85     bytes memory lzPayload = abi.encode(PT_SEND, _toAddress,
   ↳ amount);
86     _lzSend(_dstChainId, lzPayload, _refundAddress,
   ↳ _zroPaymentAddress, _adapterParams, msg.value);
87
88     emit SendToChain(_dstChainId, _from, _toAddress, amount);
89 }
```

BVSS:

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

Recommendation:

Consider using the amount returned from the inherited _debitFrom function to deposit in the vault the exact number of tokens transferred in the transaction.

Remediation Plan:

SOLVED: The Aura Finance team fixed the issue by adding a return amount to stakeAll function on the following commit ID:

- b5baaa08f12078d8936ff0bfcf159eb901150e14

4.4 (HAL-04) CONTROL toAddress SIZE - LOW (2.3)

Description:

The functions sendFrom implemented on the contracts PausableProxyOFT and PausableOFT allows the user to set an arbitrary value to the byte stream parameter _toAddress. If this value is too long, the message may not be received correctly on the destiny chain.

Although LayerZero RelayerV2 contract has a maximum size that avoids breaking the communication, it is the responsibility of the protocol to ensure a maximum toAddress size.

Code Location:

Listing 9: PausableProxyOFT.sol (Lines 130,137)

```
127 function sendFrom(
128     address _from,
129     uint16 _dstChainId,
130     bytes calldata _toAddress,
131     uint256 _amount,
132     address payable _refundAddress,
133     address _zroPaymentAddress,
134     bytes calldata _adapterParams
135 ) public payable override whenNotPaused {
136     super.sendFrom(_from, _dstChainId, _toAddress, _amount,
    ↳ _refundAddress, _zroPaymentAddress, _adapterParams);
137 }
```

Listing 10: PausableOFT.sol (Lines 25,31)

```
22 function sendFrom(
23     address _from,
24     uint16 _dstChainId,
25     bytes calldata _toAddress,
26     uint256 _amount,
27     address payable _refundAddress,
```

```
    address _zroPaymentAddress,
    bytes calldata _adapterParams
) public payable override whenNotPaused {
    super.sendFrom(_from, _dstChainId, _toAddress, _amount,
    ↳ _refundAddress, _zroPaymentAddress, _adapterParams);
}
```

BVSS:

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

Recommendation:

Consider implementing a require statement that limits the maximum size of the toAddress.

- Reference

Remediation Plan:

RISK ACCEPTED: The Aura Finance team has started communications with the LayerZero team to ensure if it is really necessary to implement a control for the parameter.

4.5 (HAL-05) MEASURE BALANCE FOR REWARD DISTRUBUTION - INFORMATIONAL (0.0)

Description:

The function _processHarvestableTokens on the AuraBalProxyOFT.sol contract iterates over the extra reward tokens and calls the getReward function from the vault. This function transfers all the rewards to the AuraBalProxyOFT contract. Then this contract uses its balance of this token to measure the obtained rewards.

However, the contract may contain tokens sent accidentally to this contract and count them as rewards. This can generate a cascade effect of bad accounting on the overall system functionality.

Code Location:

Listing 11: AuraBalProxyOFT.sol (Lines 391,392)

```
function _processHarvestableTokens() internal returns (HarvestToken[] memory harvestTokens) {
    // Set up an array to contain all the tokens that need to be harvested
    // this will be all the extra rewards tokens and auraBAL
    uint256 extraRewardsLength = IGenericVault(vault).extraRewardsLength();
    harvestTokens = new HarvestToken[](extraRewardsLength + 1);

    // Add auraBAL as the first reward token to be harvested
    //
    // To calculate rewards we need to know the delta between auraBAL on the sidechains
    // and the auraBAL available on this bridge contract.
    //
    // - internalTotalSupply: auraBAL supply transferred to L2s
    // - underlyingBalance:   auraBAL balance of the bridge in the vault
```

```
    // - totalClaimable:      auraBAL that is claimable since the last harvest
    uint256 underlyingBalance = IGenericVault(vault).balanceOfUnderlying(address(this));
    uint256 rewards = underlyingBalance - internalTotalSupply - totalClaimable[address(innerToken)];
    harvestTokens[0] = HarvestToken(address(innerToken), rewards);

    // Loop through the extra reward token on the vault and add them to the
    // harvestTokens array for processing
    for (uint256 i = 0; i < extraRewardsLength; i++) {
        address extraRewards = IGenericVault(vault).extraRewards(i);
        address rewardToken = IVirtualRewards(extraRewards).rewardToken();
        IVirtualRewards(extraRewards).getReward();
        uint256 balance = IERC20(rewardToken).balanceOf(address(this));
        // Part of the balance is sat in the contract waiting to be claimable.
        // Subtract that from the current balance to get the newly harvested rewards
        uint256 rewardAmount = balance.sub(totalClaimable[rewardToken]);
        harvestTokens[i + 1] = HarvestToken(rewardToken, rewardAmount);
    }
}
```

BVSS:

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

Recommendation:

It is recommended to check the balance before and after calling the getReward function to obtain the exact amount of tokens obtained as rewards.

Remediation Plan:

ACKNOWLEDGED: The Aura Finance team stated that if AURA tokens are sent accidentally to the contract they will be distributed as rewards and that the rescue function is not designed for avoiding that.

4.6 (HAL-06) EVENT IS NOT EMITTED CORRECTLY ON HARVEST FUNCTION - INFORMATIONAL (0.0)

Description:

The harvest function of the AuraBalProxyOFT.sol contract emits an event at the end of the execution. However, this event does not reflect any of the handled data, except for an input parameter of the function.
This function receives an array of unsigned integers and an unsigned integer as input parameter. The array should contain the values of the total amounts of auraBal tokens on each sidechain. On the other hand, the other parameter should be the sum of all individual amounts from the array. The function harvests the rewards from the vaults and stores on state variables the amount corresponding to each chain according to the percentage that each chain has.

Code Location:

Listing 12: AuraBalProxyOFT.sol (Line 245)

```
function harvest(uint256[] memory _totalUnderlying, uint256 _totalUnderlyingSum) external {
    require(authorizedHarvesters[msg.sender], "!harvester");

    uint256 srcChainIdsLen = harvestSrcChainIds.length;
    require(srcChainIdsLen == _totalUnderlying.length, "!parity");

    HarvestToken[] memory harvestTokens = _processHarvestableTokens();

    // For each chain we are sending rewards to loop through the harvestable
    // tokens and add the proportional rewards to the claimable mapping
    //
    // Keep track of the sum of the totalUnderlying to verify the user input
```

```
    // _totalUnderlyingSum is correct
    uint256 harvestTokenslen = harvestTokens.length;
    for (uint256 j = 0; j < harvestTokenslen; j++) {
        HarvestToken memory harvestToken = harvestTokens[j];
        uint256 totalHarvested = 0;
        uint256 accUnderlying = 0;

        for (uint256 i = 0; i < srcChainIdsLen; i++) {
            uint256 totalUnderlying = _totalUnderlying[i];
            uint256 amount = harvestToken.rewards.mul(totalUnderlying).div(_totalUnderlyingSum);

            totalHarvested += amount;
            accUnderlying += totalUnderlying;

            claimable[harvestToken.token][harvestSrcChainIds[i]] += amount;
        }

        totalClaimable[harvestToken.token] += totalHarvested;
        require(accUnderlying == _totalUnderlyingSum, "!sum");
    }
    emit Harvest(msg.sender, _totalUnderlyingSum);
}
```

BVSS:

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

Recommendation:

Consider adding useful information to the emitted event.

Remediation Plan:

SOLVED: The Aura Finance team fixed the issue by adding the event and require on the last loop, avoiding emitting it when the array is empty and only emitting it once on commit ID:

- 079274b5875ea20cefb32860556d1d61970a6c81

4.7 (HAL-07) FUNCTIONS SHOULD BE PAUSABLE - INFORMATIONAL (0.0)

Description:

The contracts AuraBalProxyOFT.sol and AuraOFT.sol inherited from PauseGuardian.sol contract. However, there are some functions that are not pausable and should be considered to implement the whenNotPaused modifier. These functions are:

- processClaimable from AuraBalProxyOFT.sol.
- lock from AuraOFT.sol.

On the other hand, the coordinators (L1Coordinator and L2Coordinator) contracts do not have any mechanism to be paused. However, as these contracts are mainly interacting with other protocol components, they could not require any pausable functionality.

BVSS:

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

Recommendation:

Consider reviewing the pausable security model and implementing it if required on other functions.

Remediation Plan:

SOLVED: The Aura Finance team fixed the issue by adding pausable modifiers to the functions on commit IDS:

- 8e17d3dfab9272b84cdcd5cbe5d35f9356fd51b6
- 9198edb43afdc782d5ad5b28565a4e81234624bb

4.8 (HAL-08) CHANGE STRINGS FOR CUSTOM ERRORS 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 are hit by avoiding having to allocate and store the revert string. Not defining strings also saves deployment gas.

BVSS:

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

Recommendation:

Consider replacing all revert strings with custom errors.

Remediation Plan:

ACKNOWLEDGED: The Aura Finance team acknowledge the issue.

4.9 (HAL-09) LACK OF REENTRANCY PROTECTION - INFORMATIONAL (0.0)

Description:

The current contracts do not implement re-entrancy protection. Although it's true that it has not been found any exploitable vector, the contracts sending messages through LayerZero, give back the execution to the caller on several functions.

The smart contracts follow strictly the Checks Effects Interactions pattern; however, it is still recommended to enforce the non-re-entrant modifier to all sensible functions.

BVSS:

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

Recommendation:

Consider adding the non-re-entrant modifier to all sensible functions.

Remediation Plan:

SOLVED: The Aura Finance team solved the issue by adding reentrance guards on commit ID:
- 5f716ad8d0cf997e951d8d7d58dd7a2568d2658e

4.10 (HAL-10) INCONSISTENT NAMING CONVENTION - INFORMATIONAL (0.0)

Description:

The naming convention across the different files on the repository, including tests and deployments also, mixes the variable naming between Aura and Convex.

This makes it easier to create bugs due to confusions on the development side. Moreover, it makes the code harder to understand for the auditors and other parties.

BVSS:

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

Recommendation:

Design a style guide and a consistent naming convention across the whole project.

Remediation Plan:

ACKNOWLEDGED: The Aura Finance team acknowledged the risk of the issue.

4.11 (HAL-11) LACK OF UPGRADABILITY PATTERN - INFORMATIONAL (0.0)

Description:

The current version of the project does not allow contracts to be upgraded. This can be useful either to fix potential unwanted behaviors and also to add new functionalities in future releases of the protocol.

BVSS:

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

Recommendation:

Consider adding proxy contracts that allow contracts to be upgradable.

Remediation Plan:

ACKNOWLEDGED: The Aura Finance team acknowledged the risk of the issue.

4.12 (HAL-12) CENTRALIZATION RISK - INFORMATIONAL (0.0)

Description:

The current protocol relies on several multisig wallets, as well as some EOAs that can perform maintenance tasks over the smart contracts. The usage of multisigs allows avoiding single points of failure, which increases the security permission model.

However, it is important to notice that in functions such as harvest from the AuraBalProxyOFT.sol contract, if parameters are introduced incorrectly, users' yields can be affected. On the other hand, all the bests efforts from the Aura Finance team are set to make the security permission model work as always expected in favor of the users.

BVSS:

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

Recommendation:

Consider a future plan that may allow the protocol to work as autonomous as possible. Nonetheless, this is a design decision and the issue does not represent any threat by itself.

Remediation Plan:

ACKNOWLEDGED: The Aura Finance team acknowledged the risk of the issue.

4.13 (HAL-13) EXTERNAL CALL ON LOOP - INFORMATIONAL (0.0)

Description:

External calls inside a loop increase gas usage or might lead to a denial-of-service attack. The function _processHarvestableTokens iterates through the extraRewardsLength that corresponds to the number of tokens set as extra rewards on the vault.

It is important to remark that, as the Aura team explained, there should not be more than one token as an extra reward.

Code Location:

Listing 13: AuraBalProxyOFT.sol

```
for (uint256 i = 0; i < extraRewardsLength; i++) {
    address extraRewards = IGenericVault(vault).extraRewards(i);
    address rewardToken = IVirtualRewards(extraRewards).
↳ rewardToken();
    IVirtualRewards(extraRewards).getReward();
    uint256 balance = IERC20(rewardToken).balanceOf(address(this))
↳ ;
    // Part of the balance is sat in the contract waiting to be
↳ claimable.
    // Subtract that from the current balance to get the newly
↳ harvested rewards
    uint256 rewardAmount = balance.sub(totalClaimable[rewardToken
↳ ]);
    harvestTokens[i + 1] = HarvestToken(rewardToken, rewardAmount)
↳ ;
}
```

BVSS:

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

Recommendation:

It is recommended to set the max length to which a for loop can iterate. If possible, use pull over push strategy for external calls.

Remediation Plan:

ACKNOWLEDGED: The Aura Finance team acknowledged the risk of the issue.

4.14 (HAL-14) LACK OF TWO STEP OWNERSHIP TRANSFER - INFORMATIONAL (0.0)

Description:

The current ownership transfer process for all the contracts inheriting from Ownable involves the current owner calling the transferOwnership() function:

Listing 14: Ownable.sol

```
function transferOwnership(address newOwner) public virtual
↳ onlyOwner {
    require(newOwner != address(0), "Ownable: new owner is the
↳ zero address");
    _setOwner(newOwner);
}
```

If the nominated EOA account is not a valid account, it is entirely possible that the owner may accidentally transfer ownership to an uncontrolled account, losing the access to all functions with the onlyOwner modifier.

BVSS:

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

Recommendation:

It is recommended to implement a two-step process where the owner nominates an account and the nominated account needs to call an acceptOwnership() function for the transfer of the ownership to fully succeed. This ensures the nominated EOA account is a valid and active account.

Remediation Plan:

ACKNOWLEDGED: The Aura Finance team acknowledged the risk of the issue.

RECOMMENDATIONS OVERVIEW

RECOMMENDATIONS OVERVIEW

1. Allow users to set a receiver address when locking funds from side chain to main chain.
2. Do not reduce the internal total supply state variable if it is not needed on the rescue function.
3. Measure the balance delta when harvesting rewards to store the appropriate values on the state variables.
4. Do not stake entire balance and use the returned value calculated during the function execution.
5. Limit the maximum _toAddress size as suggested from LayerZero team.
6. Consider more valuable information to emit on the harvest event.
7. Consider adding the pausable modifier to other sensitive functions.
8. Consider adding reentrancy protection.
9. Consider establishing a consistent naming convention across the whole project.

AUTOMATED TESTING

6.1 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:

AuraBalProxyOFT.sol

Report for [redacted]\aura-contracts\contracts\sidechain\AuraBalProxyOFT.sol
[redacted]

Line	SWC Title	Severity	Short Description
44	(SWC-110) Assert Violation	Unknown	Public state variable with array type causing reacheable exception by default.
142	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
143	(SWC-110) Assert Violation	Unknown	Out of bounds array access
152	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
173	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+=" discovered
191	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-=" discovered
227	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
228	(SWC-110) Assert Violation	Unknown	Out of bounds array access
232	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
233	(SWC-110) Assert Violation	Unknown	Out of bounds array access
236	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+=" discovered
237	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+=" discovered
239	(SWC-110) Assert Violation	Unknown	Out of bounds array access
239	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+=" discovered
242	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+=" discovered
263	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-=" discovered
269	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+=" discovered
328	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-=" discovered
372	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
383	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
384	(SWC-110) Assert Violation	Unknown	Out of bounds array access
388	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
396	(SWC-110) Assert Violation	Unknown	Out of bounds array access
396	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered

L1Coordinator.sol

Report for [redacted]aura-contracts\contracts\sidechain\L1Coordinator.sol
[redacted]

Line	SWC Title	Severity	Short Description
177	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+=" discovered

PausableProxyOFT.sol

Report for [redacted]aura-contracts\contracts\sidechain\PausableProxyOFT.sol
[redacted]

Line	SWC Title	Severity	Short Description
136	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+=" discovered

ProxyOFT.sol

Report for [redacted]aura-contracts\contracts\layerzero\token\oft\extension\ProxyOFT.sol
[redacted]

Line	SWC Title	Severity	Short Description
19	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
36	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
46	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered

- No major issues found by MythX.

THANK YOU FOR CHOOSING

// HALBORN