



Biconomy – LiquidityPoolManager

Smart Contract Security Audit

Prepared by: Halborn

Date of Engagement: July 6th – 16th, 2021

Visit: Halborn.com

DOCUMENT REVISION HISTORY	4
CONTACTS	4
1 EXECUTIVE OVERVIEW	5
1.1 INTRODUCTION	6
1.2 AUDIT SUMMARY	6
1.3 TEST APPROACH & METHODOLOGY	6
RISK METHODOLOGY	7
1.4 SCOPE	9
2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW	10
3 FINDINGS & TECH DETAILS	11
3.1 (HAL-01) RENOUNCING PAUSER ROLE WHEN CONTRACT IS PAUSED - HIGH	13
Description	13
Risk Level	13
Proof Of Concept	14
Recommendation	16
Remediation Plan	16
3.2 (HAL-02) OWNER CAN RENOUNCE OWNERSHIP - LOW	17
Description	17
Risk Level	17
Code Location	17
Recommendation	17
Remediation Plan	18
3.3 (HAL-03) LACK OF ZERO ADDRESS CHECK ON CONSTRUCTOR - LOW	19
Description	19

	Risk Level	19
	Code Location	19
	Recommendation	20
	Remediation Plan	20
3.4	(HAL-04) TAUTOLOGY EXPRESSIONS - LOW	21
	Description	21
	Risk Level	21
	Code Location	21
	Recommendation	21
	Remediation Plan	22
3.5	(HAL-05) EXPERIMENTAL FEATURES ENABLED - INFORMATIONAL	23
	Description	23
	Risk Level	23
	Code Location	23
	Recommendation	23
	Remediation Plan	24
3.6	(HAL-06) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL 25	
	Description	25
	Code Location	26
	Recommendation	26
	Remediation Plan	26
3.7	(HAL-07) MISSING EVENTS EMITTING - INFORMATIONAL	27
	Description	27
	Risk Level	27
	Code Location	27

	Recommendation	28
	Remediation Plan	28
4	AUTOMATED TESTING	29
4.1	STATIC ANALYSIS REPORT	30
	Description	30
	Results	30
4.2	AUTOMATED SECURITY SCAN	31
	Description	31
	Results	32

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	07/12/2021	Gabi Urrutia
0.9	Document Edits	07/15/2021	Ataberk Yavuzer
1.0	Final Review	07/16/2021	Gabi Urrutia
1.1	Remediation Plan	07/28/2021	Ataberk Yavuzer
1.1	Remediation Plan Review	07/30/2021	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
Ataberk Yavuzer	Halborn	Ataberk.Yavuzer@halborn.com



EXECUTIVE OVERVIEW



1.1 INTRODUCTION

Biconomy engaged Halborn to conduct a security assessment on a smart contract beginning on July 6th, 2021 and ending July 16th, 2021. The security assessment was scoped to the smart contract provided in the Github repository [Biconomy Repository](#) Halborn conducted this audit to measure security risk and identify any new vulnerabilities introduced during the final stages of development before the production release. The security assessment was scoped to the smart contract [LiquidityPoolManager.sol](#).

1.2 AUDIT SUMMARY

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

The purpose of this audit to achieve the following:

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

Though this security audit's outcome is satisfactory, only the most essential aspects were tested and verified to achieve objectives and deliverables set in the scope due to time and resource constraints. It is essential to note the use of the best practices for secure smart-contract development.

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 the smart contract audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts 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.
- 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.
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions.(Slither)
- Testnet deployment (RemixIDE)

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. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was 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.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
----------	------	--------	-----	---------------

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 smart contracts:

- `LiquidityPoolManager.sol`

Commit ID: `15c76b914caa7430aee61fd11c789ef19de205b9`

OUT-OF-SCOPE:

Other smart contracts in the repository, external libraries and economics attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	0	3	3

LIKELIHOOD

IMPACT

			(HAL-01)	
	(HAL-02) (HAL-03)			
(HAL-05)	(HAL-04)			
(HAL-06) (HAL-07)				

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - RENOUNCING PAUSER ROLE WHEN CONTRACT IS PAUSED	High	SOLVED: 07/27/2021
HAL02 - OWNER CAN RENOUNCE OWNERSHIP	Low	SOLVED: 07/28/2021
HAL03 - LACK OF ZERO ADDRESS CHECK ON CONSTRUCTOR	Low	SOLVED: 07/27/2021
HAL04 - TAUTOLOGY EXPRESSIONS	Low	SOLVED: 07/27/2021
HAL05 - EXPERIMENTAL FEATURES ENABLED	Informational	SOLVED: 07/27/2021
HAL06 - POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	SOLVED: 07/27/2021
HAL07 - MISSING EVENTS EMITTING	Informational	SOLVED: 07/27/2021



FINDINGS & TECH DETAILS



3.1 (HAL-01) RENOUNCING PAUSER ROLE WHEN CONTRACT IS PAUSED - HIGH

Description:

There are multiple roles on the `LiquidityPoolManager` contract such as `Owner`, `Pauser` and `TrustedForwarder`. According to the contract functions, it is possible to set Owner and Pauser addresses to zero by using `renounceOwnership` and `renouncePauser` functions. The `Pauser` role can pause the contract. If the private key of this Pauser is compromised, an attacker can pause the contract and renounce the pauser role, then the attacker can render the contract useless forever.

Risk Level:

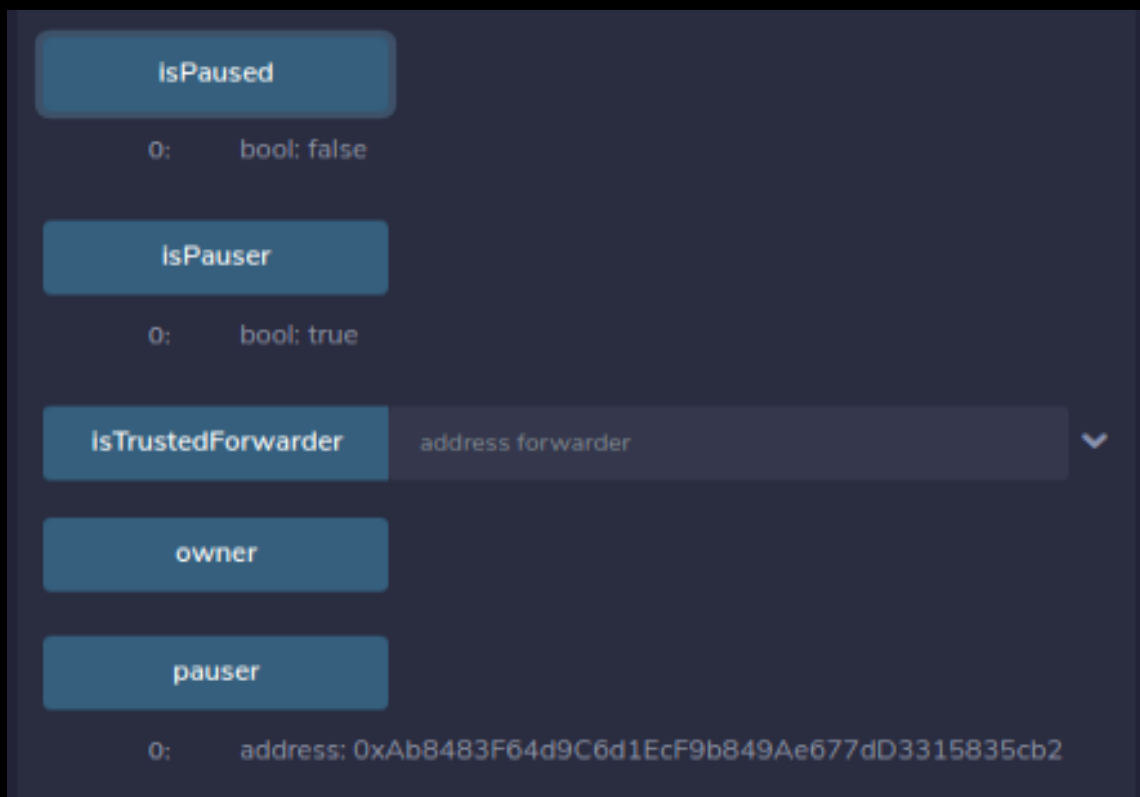
Likelihood - 4

Impact - 4

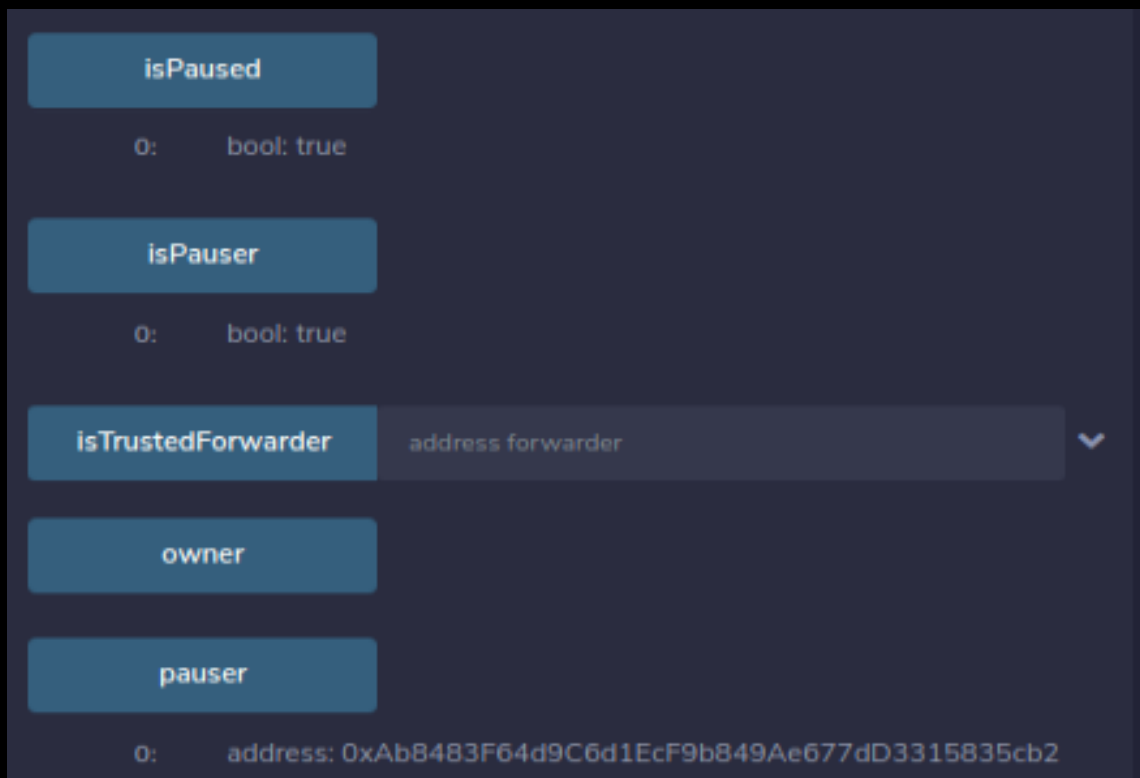
Proof Of Concept:

This attack has three stages which are described below:

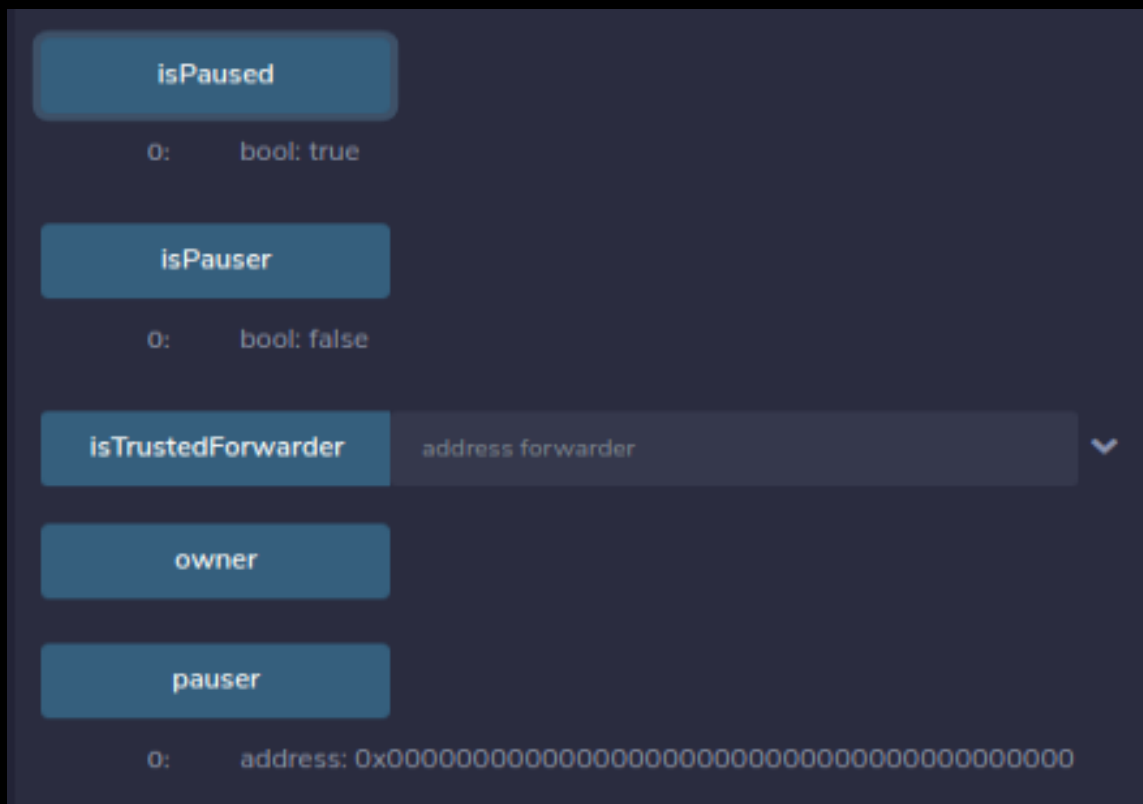
1. In the first screenshot, everything looks proper. Three functions are called during the step one: `isPaused()`, `isPauser()` and `pauser()` public functions in order.



2. After the first step, the `pause()` function has to be called by any user which has `Pauser` role. It can be seen that `isPaused()` function returned `true` as a boolean variable on the following screenshot.



3. Finally, the `renouncePauser()` function needs to be called on the contract that is in the pause state. After this action, the Pauser address will be set to `address(0)` and the contract will never be able to return from the paused state.



Recommendation:

It is recommended to add the `whenNotPaused` modifier to the `renouncePauser` function. In addition, the pauser role should not be the `address(0)` for not disrupting the flow of the contract.

Remediation Plan:

SOLVED: Biconomy Team solved this issue by adding `whenNotPaused` modifier to the `renouncePauser` function.

3.2 (HAL-02) OWNER CAN RENOUNCE OWNERSHIP - LOW

Description:

The **Owner** of the contract is usually the account that deploys the contract. As a result, the **Owner** is able to perform some privileged functions like `setBaseGas()` and `setExecutorManager()`. In the `LiquidityPoolManager.sol` smart contract, the `renounceOwnership` function is used to renounce the **Owner** permission. Renouncing ownership before transferring would result in the contract having no **Owner**, eliminating the ability to call privileged functions.

Risk Level:

Likelihood - 2

Impact - 3

Code Location:

Listing 1: `LiquidityPoolManager.sol` Ownable (Lines 15)

```
15 contract LiquidityPoolManager is ReentrancyGuard, Ownable,  
    BaseRelayRecipient, Pausable {  
16
```

Recommendation:

It's recommended that the Owner is not able to call `renounceOwnership` without transferring the Ownership to other address before. In addition, if a multi-signature wallet is used, calling `renounceOwnership` function should be confirmed for two or more users. As an other solution, Renounce Ownership functionality can be disabled with the following line.

Listing 2: Disable RenounceOwnership (Lines 1)

```
1 function renounceOwnership () public override onlyOwner {  
2     revert ("can 't renounceOwnership here "); // not possible  
    with this smart contract  
3 }
```

Remediation Plan:

SOLVED: Biconomy Team added a new statement for `renounceOwnership` and `renouncePauser` functions to solve the issue.

3.3 (HAL-03) LACK OF ZERO ADDRESS CHECK ON CONSTRUCTOR - LOW

Description:

The `LiquidityPoolManager` contract includes different type of roles. For example, there is `Owner` role for setting the `adminFee` value. For another example, the `Pauser` role pauses the contract if something wrong with the transactions or contract logic. It is important to provide these roles to valid addresses. These roles should be driven by people. There are too many address checks in the `LiquidityPoolManager` contract to keep these roles safe. For example, it is not possible to set `Owner`, `Pauser` and `TrustedForwarder` addresses to `address(0)` after initialization of the contract. However, it is possible to set `Pauser` address to `address(0)` because of the lack of address control on constructor.

Risk Level:

Likelihood - 2

Impact - 3

Code Location:

Listing 3: LiquidityPoolManager.sol (Lines 68,69,70,71)

```

67     constructor(address _executorManagerAddress, address owner,
        address pauser, address _trustedForwarder, uint256
        _adminFee) public Ownable(owner) Pausable(pauser) {
68         require(_executorManagerAddress != address(0), "
            ExecutorManager Contract Address cannot be 0");
69         require(owner != address(0), "Owner Address cannot be 0");
70         require(_trustedForwarder != address(0), "TrustedForwarder
            Contract Address cannot be 0");
71         require(_adminFee != 0, "AdminFee cannot be 0");
72         executorManager = ExecutorManager(_executorManagerAddress)
            ;
73         trustedForwarder = _trustedForwarder;
74         adminFee = _adminFee;

```

```
75         baseGas = 21000;  
76     }
```

Recommendation:

Implementing zero address check on constructor strongly recommended by Halborn team.

Remediation Plan:

SOLVED: Biconomy Team added address control check to the constructor.

3.4 (HAL-04) TAUTOLOGY EXPRESSIONS - LOW

Description:

In contract `LiquidityPoolManager.sol`, tautology expressions have been detected. Such expressions are of no use since they always evaluate true/false regardless of the context they are used in.

Risk Level:

Likelihood - 2

Impact - 2

Code Location:

Listing 4: `LiquidityPoolManager.sol` (Lines 154)

```
153 function addTokenLiquidity( address tokenAddress, uint256 amount )  
    public tokenChecks(tokenAddress) whenNotPaused {  
154     require(amount > 0, "amount should be greater then 0");
```

Listing 5: `LiquidityPoolManager.sol` (Lines 273)

```
272 function withdrawErc20(address tokenAddress) public onlyOwner  
    whenNotPaused {  
273     uint256 profitEarned = (IERC20(tokenAddress).balanceOf(  
        address(this))).sub(tokensInfo[tokenAddress].liquidity)  
        ;  
274     require(profitEarned > 0, "Profit earned is 0");  
275     address payable sender = _msgSender();
```

Recommendation:

Correct the expressions. Since `amount` and `profitEarned` variables are declared as type `uint256`, they are always greater or equal to 0.

Remediation Plan:

SOLVED: Biconomy Team fixed expressions on modifiers related to the current issue.

3.5 (HAL-05) EXPERIMENTAL FEATURES ENABLED - INFORMATIONAL

Description:

ABIEncoderV2 is enabled and the use of experimental features could be dangerous on live deployments. The experimental ABI encoder does not handle non-integer values shorter than 32 bytes properly. This applies to `bytesNN` types, `bool`, `enum` and other types when they are part of an array or a struct and encoded directly from storage. This means these storage references have to be used directly inside `abi.encode(...)` as arguments in external function calls or in event data without prior assignment to a local variable. The types **bytesNN** and **bool** will result in corrupted data while `enum` might lead to an invalid revert.

Risk Level:

Likelihood - 1

Impact - 2

Code Location:

Listing 6: LiquidityPoolManager.sol (Lines 4)

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity 0.7.6;
4 pragma experimental ABIEncoderV2;
```

Recommendation:

When possible, do not use experimental features in the final live deployment. Validate and check that all the conditions above are true for integers and arrays (i.e. all using `uint256`).

Remediation Plan:

SOLVED: Biconomy Team removed `experimental` keyword on pragma section.

3.6 (HAL-06) POSSIBLE MISUSE OF PUBLIC FUNCTIONS – INFORMATIONAL

Description:

In public functions, array arguments are immediately copied to memory, while external functions can read directly from `calldata`. Reading `calldata` is cheaper than memory allocation. Public functions need to write the arguments to memory because public functions may be called internally. Internal calls are passed internally by pointers to memory. Thus, the function expects its arguments being located in memory when the compiler generates the code for an internal function.

Also, methods do not necessarily have to be public if they are only called within the contract-in such case they should be marked `internal`. In addition to that if a function not used in the contract second time, `external` keyword should be appended to the function for a better gas optimization.

Code Location:

Misused Functions Without External Keyword:

Listing 7: LiquidityPoolManager.sol

```
1 changeAdminFee()
2 setExecutorManager()
3 setTrustedForwarder()
4 setTokenTransferOverhead()
5 addSupportedToken()
6 removeSupportedToken()
7 updateTokenCap()
8 addNativeLiquidity()
9 removeNativeLiquidity()
10 addTokenLiquidity()
11 removeTokenLiquidity()
12 depositNative()
13 sendFundsToUser()
14 withdrawErc20()
15 withdrawNative()
```

Recommendation:

Consider as much as possible declaring external variables instead of public variables. As for best practice, you should use external if you expect that the function will only be called externally and use public if you need to call the function internally. To sum up, all can access to public functions, external functions only can be accessed externally and internal functions can only be called within the contract.

Remediation Plan:

SOLVED: Biconomy Team replaced external to public in several functions.

3.7 (HAL-07) MISSING EVENTS EMITTING - INFORMATIONAL

Description:

It has been observed that critical functionality is missing emitting event for `setTrustedForwarder` and `changeAdminFee` functions. These functions should

Risk Level:

Likelihood - 1

Impact - 1

Code Location:

Listing 8: LiquidityPoolManager.sol

```
82     function changeAdminFee(uint256 newAdminFee) public onlyOwner
      whenNotPaused {
83         require(newAdminFee != 0, "Admin Fee cannot be 0");
84         adminFee = newAdminFee;
85     }
```

Listing 9: LiquidityPoolManager.sol

```
104 function setTrustedForwarder( address forwarderAddress ) public
     onlyOwner {
105     require(forwarderAddress != address(0), "Forwarder Address
        cannot be 0");
106     trustedForwarder = forwarderAddress;
107 }
```

Recommendation:

Consider emitting an event when calling `setTrustedForwarder` and `changeAdminFee` functions.

Listing 10

```
1 event changeAdminFee(uint256 newAdminFee);
2 event setTrustedForwarder(address forwarderAddress);
```

Remediation Plan:

SOLVED: `Biconomy Team` solved this issue by defining new events on the contract to notify users such as `AdminFeeChanged` and `TrustedForwarderChanged`.



AUTOMATED TESTING



4.1 STATIC ANALYSIS REPORT

Description:

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

Results:

LiquidityPoolManager.sol

```
INFO:Detectors:
ExecutorManager.constructor(address).owner (contracts/6/ExecutorManager.sol#23) shadows:
  - Ownable.owner() (contracts/6/libs/Ownable.sol#40-42) (function)
LiquidityPoolManager.constructor(address,address,address,address,uint256).owner (contracts/6/insta-swaps/LiquidityPoolManager.sol#67) shadows:
  - Ownable.owner() (contracts/6/libs/Ownable.sol#40-42) (function)
LiquidityPoolManager.constructor(address,address,address,address,uint256).pauser (contracts/6/insta-swaps/LiquidityPoolManager.sol#67) shadows:
  - Pausable.pauser() (contracts/6/libs/Pausable.sol#82-84) (function)
Ownable.constructor(address).owner (contracts/6/libs/Ownable.sol#22) shadows:
  - Ownable.owner() (contracts/6/libs/Ownable.sol#40-42) (function)
Pausable.constructor(address).pauser (contracts/6/libs/Pausable.sol#39) shadows:
  - Pausable.pauser() (contracts/6/libs/Pausable.sol#82-84) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
LiquidityPoolManager.setTrustedForwarder(address) (contracts/6/insta-swaps/LiquidityPoolManager.sol#104-107) should emit an event for:
  - trustedForwarder = forwarderAddress (contracts/6/insta-swaps/LiquidityPoolManager.sol#106)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control
INFO:Detectors:
LiquidityPoolManager.changeAdminFee(uint256) (contracts/6/insta-swaps/LiquidityPoolManager.sol#82-85) should emit an event for:
  - adminFee = newAdminFee (contracts/6/insta-swaps/LiquidityPoolManager.sol#84)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
Pausable.constructor(address).pauser (contracts/6/libs/Pausable.sol#39) lacks a zero-check on :
  - pauser = pauser (contracts/6/libs/Pausable.sol#40)
LiquidityPoolManager.withdrawNative().sender (contracts/6/insta-swaps/LiquidityPoolManager.sol#285) lacks a zero-check on :
  - (success) = sender.call(value: profitEarned)() (contracts/6/insta-swaps/LiquidityPoolManager.sol#286)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
```

All relevant findings were founded in the manual code review.

4.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruit 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 testers machine and sent the compiled results to the analyzers to locate any vulnerabilities. In addition, security detections are only in scope.

Results:

LiquidityPoolManager.sol

Report for insta-swaps/LiquidityPoolManager.sol
<https://dashboard.mythx.io/#/console/analyses/75c0192e-4ff5-4773-aca5-a5e28dad87c>

Line	SWC Title	Severity	Short Description
78	(SWC-000) Unknown	Medium	Function could be marked as external.
82	(SWC-000) Unknown	Medium	Function could be marked as external.
95	(SWC-000) Unknown	Medium	Function could be marked as external.
99	(SWC-000) Unknown	Medium	Function could be marked as external.
104	(SWC-000) Unknown	Medium	Function could be marked as external.
109	(SWC-000) Unknown	Medium	Function could be marked as external.
113	(SWC-000) Unknown	Medium	Function could be marked as external.
121	(SWC-000) Unknown	Medium	Function could be marked as external.
125	(SWC-000) Unknown	Medium	Function could be marked as external.
131	(SWC-000) Unknown	Medium	Function could be marked as external.
140	(SWC-000) Unknown	Medium	Function could be marked as external.
153	(SWC-000) Unknown	Medium	Function could be marked as external.
163	(SWC-000) Unknown	Medium	Function could be marked as external.
176	(SWC-000) Unknown	Medium	Function could be marked as external.
223	(SWC-000) Unknown	Medium	Function could be marked as external.
231	(SWC-000) Unknown	Medium	Function could be marked as external.
272	(SWC-000) Unknown	Medium	Function could be marked as external.
282	(SWC-000) Unknown	Medium	Function could be marked as external.

All relevant findings were founded in the manual code review.



THANK YOU FOR CHOOSING

// HALBORN

