Biconomy

Vesting Smart Contract Audit

Prepared by: **Halborn**

Date of Engagement: **November 17th, 2021 - November 23rd, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	11/21/2021	Ataberk Yavuzer
0.2	Document Edits	11/22/2021	Ataberk Yavuzer
0.3	Final Draft	11/22/2021	Ataberk Yavuzer
0.4	Draft Review	11/22/2021	Gabi Urrutia
1.0	Remediation Plan	11/26/2021	Ataberk Yavuzer
1.1	Remediation Plan Review	11/26/2021	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 their **Biconomy** Vesting Contract beginning on November 17th and ending on November 23rd, 2021.

The security assessment was scoped to the Github repository of Biconomy Vesting Contract. An audit of the security risk and implications regarding the changes introduced by the development team at Biconomy before its production release shortly following the assessment's deadline.

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 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 Biconomy team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual review of the code 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
- Manual testing by custom scripts
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (hardhat, Remix IDE, ganache-cli)

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

1. Biconomy Vesting Contracts

   (a) Repository: Biconomy Vesting

   (b) Commit ID: 2c6a5d0c8e982656ba09f80ae17c1229ac8c8afa

   (c) Contracts in scope:

      i. BicoVestingFlat.sol

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	2	1	6	1

LIKELIHOOD

IMPACT

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

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) INVALID CHECK ON CREATECLAIM METHOD LEADS TO UNCLAIM OF TOKENS	High	SOLVED - 11/25/2021
(HAL-02) CLAIMING TOKENS BEFORE UNLOCKTIME LEADS UNCLAIM OF LOCKED TOKENS	High	SOLVED - 11/25/2021
(HAL-03) INTEGER OVERFLOW	Medium	SOLVED - 11/25/2021
(HAL-04) MISSING ROLE-BASED ACCESS CONTROL	Low	ACKNOWLEDGED
(HAL-05) OWNER CAN RENOUNCE OWNERSHIP	Low	ACKNOWLEDGED
(HAL-06) LACK OF ZERO ADDRESS CHECK	Low	SOLVED - 11/25/2021
(HAL-07) POSSIBLE MISUSE OF OWNERSHIP FUNCTIONS	Low	ACKNOWLEDGED
(HAL-08) MISSING ISACTIVE CONTROL ON REVOKE FUNCTION	Low	SOLVED - 11/25/2021
(HAL-09) USE OF BLOCK.TIMESTAMP	Low	ACKNOWLEDGED
(HAL-10) IGNORED RETURN VALUES	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) INVALID CHECK ON CREATECLAIM METHOD LEADS TO UNCLAIM OF TOKENS - HIGH

Description:

The createClaim method on the contract which ables to create claim for beneficiary users is vulnerable to division by zero vulnerability due to invalid check. This method takes several arguments such as _beneficiary, _vestAmount, _unlockAmount, _unlockTime, _startTime and _endTime. Also, this method has several require controls itself.

The following require check makes it possible to enter 0 as the _endTime variable if _startTime variable equals to 0.

Listing 1: BicoVestingFlat.sol

```
require(_endTime >= _startTime, "INVALID_TIME");
```

Setting 0 as both _startTime and _endTime variables will work properly. However,
the claimableAmount method will not work since division by zero occur due to _endTime variable equals to 0. Therefore, it will not be possible to claim tokens for beneficiary. As a result, tokens will be stuck on the contract even claims[beneficiary].isActive equals to true.

Code Location:

Listing 2: BicoVestingFlat.sol (Lines 1156)

```
function createClaim(
        address _beneficiary,
        uint256 _vestAmount,
        uint256 _unlockAmount,
        uint256 _unlockTime,
        uint64 _startTime,
```

```
        uint64 _endTime
    ) public onlyAdmin {
        require(!claims[_beneficiary].isActive, "CLAIM_ACTIVE");
        require(_endTime >= _startTime, "INVALID_TIME");
        require(_beneficiary != address(0), "INVALID_ADDRESS");
        require(_vestAmount > 0, "INVALID_AMOUNT");
        //review
        //should probably use IERC20 interface instead of ERC20 import?
        //need for safe transfer?

        //notice
        //Admin needs to give prior apporve tokens to this contract
        require(
            ERC20(tokenAddress).allowance(msg.sender, address(this)) >=
                (_vestAmount.add(_unlockAmount)),
            "INVALID_ALLOWANCE"
        );
        ERC20(tokenAddress).transferFrom(
            msg.sender,
            address(this),
            _vestAmount
        );
        Claim memory newClaim = Claim({
            isActive: true,
            vestAmount: _vestAmount,
            unlockAmount: _unlockAmount,
            unlockTime: _unlockTime,
            startTime: _startTime,
            endTime: _endTime,
            amountClaimed: 0
        });
        claims[_beneficiary] = newClaim;
        emit ClaimCreated(
            msg.sender,
            _beneficiary,
            _vestAmount,
            _unlockAmount,
            _unlockTime,
            _startTime,
            _endTime
        );
```

```
1194     }
```

Listing 3: BicoVestingFlat.sol (Lines 1255,1256)

```
1254 {
1255         claimPercent = currentTimestamp.sub(_claim.startTime).mul
                (1e18).div(
1256                 _claim.endTime.sub(_claim.startTime)
1257             );
1258         claimAmount = _claim.vestAmount.mul(claimPercent).div(1e18
                ).add(
1259             _claim.unlockAmount
1260         );
1261         unclaimedAmount = claimAmount.sub(_claim.amountClaimed);
1262 }
```

Risk Level:

Likelihood - 4
Impact - 4

Recommendations:

It is recommended to replace the require check above with the following one. Also, it is possible to mitigate this issue by implementing zero check for _endTime variable.

Listing 4: Possible Fix

```
require(_endTime > _startTime, "INVALID_TIME");
```

Listing 5: Possible Fix-2

```
require(_endTime != 0, "INVALID_TIME_FOR_ENDTIME");
```

Remediation Plan:

SOLVED: The Biconomy Team solved this issue by implementing the recommendation above. It has become impossible to set 0 as _endTime variable with this mitigation. As a result, division by zero will not occur in the future.

Commit ID: **f1ad27ca200d00adb1568b9d6a16bc10dda555e3**

3.2 (HAL-02) CLAIMING TOKENS BEFORE UNLOCKTIME LEADS UNCLAIM OF LOCKED TOKENS - HIGH

Description:

The createClaim method on the contract which can to create claim for beneficiary users is vulnerable to unclaimed tokens. If any user tries to claim their rewards before reaching to _unlockTime's timestamp variable, the isActive field changes to false. The contract does not have any method to convert the isActive field to true from false. Therefore, users will not be able to get their unlocked amounts as rewards if they try to claim awards before their _unlockTime.

For example, the contract admin creates a claim with following variables:

Listing 6: Create Claim Example

```
_beneficiary = "0x......",
_vestAmount = "1 Test Token",
_unlockAmount = "200 Test Token",
_unlockTime = current_timestamp + 1 day,
_startTime = 0,
_endTime = current_timestamp + 1 minute
```

If user tries to claim awards after current_timestamp + 1 day, that user will get only 201 Test Tokens. However, if the user tries to claim awards before _unlockTime, user will get only 1 Test Token, the isActive field will be set to false by contract and it will not be possible to change it to true even timestamp reaches to _unlockTime. As a result, 200 Test Tokens will be stuck on the contract.

Code Location:

Listing 7: BicoVestingFlat.sol (Lines 1285)

```
1278 function claim() external whenNotPaused nonReentrant {
1279         address beneficiary = msg.sender;
1280         Claim memory _claim = claims[beneficiary];
1281         require(_claim.isActive, "CLAIM_INACTIVE");
1282         uint256 unclaimedAmount = claimableAmount(beneficiary);
1283         ERC20(tokenAddress).transfer(beneficiary, unclaimedAmount)
             ;
1284         _claim.amountClaimed = _claim.amountClaimed +
             unclaimedAmount;
1285         if (_claim.amountClaimed == _claim.vestAmount) _claim.
             isActive = false;
1286         claims[beneficiary] = _claim;
1287         emit Claimed(beneficiary, unclaimedAmount);
1288     }
```

Risk Level:

Likelihood - 4
Impact - 4

Recommendations:

It is recommended to implement another check to validate the following formula.

Listing 8: Formula

```
_unlockTime < _startTime < _endTime
```

Remediation Plan:

SOLVED: The Biconomy Team solved this issue by implementing the formula above. All time variables will be controlled sequentially.

Commit ID: **f1ad27ca200d00adb1568b9d6a16bc10dda555e3**

3.3 (HAL-03) INTEGER OVERFLOW - MEDIUM

Description:

If you're using an unsigned integer in Solidity, the possible values of your variable ranges from 0 to 2 ^ 256. So, it means that if you are around the max value and increment your variable, it will go back to 0. The same happens if your variable is at 0, and you subtract one, instead of **overflow** it is called **underflow**.

The SafeMath library also protects contracts for possible integer overflows or underflows.

Even this control mechanism exists on the contract, BicoVestingFlat.sol contract is vulnerable to the integer overflow vulnerability due to missing use of add() method.

Code Location:

Listing 9: BicoVestingFlat.sol (Lines 1284)

```
_claim.amountClaimed = _claim.amountClaimed + unclaimedAmount;
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendations:

It is recommended to use SafeMath add() method instead of plus (+) operator.

Listing 10: BicoVestingFlat.sol

```
_claim.amountClaimed = _claim.amountClaimed.add(unclaimedAmount);
```

Remediation Plan:

SOLVED: This issue was removed by replacing the .add() function with plus (+)operator.

Commit ID: **f1ad27ca200d00adb1568b9d6a16bc10dda555e3**

3.4 (HAL-04) MISSING ROLE-BASED ACCESS CONTROL - LOW

Description:

In smart contracts, implementing a correct Access Control policy is an essential step to maintain security and decentralization of permissions on a token. All the features of the smart contract, such as mint/burn tokens and pause contracts, are given by Access Control. For instance, Ownership is the most common form of Access Control. In other words, the owner of a contract (the account that deployed it by default) can do some administrative tasks on it. Nevertheless, other authorization levels are required to follow the principle of least privilege, also known as least authority. Briefly, any process user, or program only can access to the necessary resources or information. Otherwise, the ownership role is useful in a simple system, but more complex projects require the use of more roles by using Role-based access control.

There are multiple important functionalities on BicoVestingFlat.sol contract such as creating claims for beneficiaries, setting new Admin for the contract and pausing/unpausing the contract. It is important to divide these functionalities into multiple roles.

Code Location:

Listing 11: Centralized Functions

```
function pause()
function unpause()
function createClaim(address _beneficiary, uint256 _vestAmount,
    uint256 _unlockAmount, uint256 _unlockTime, uint64 _startTime,
    uint64 _endTime)
function createBatchClaim(address[] memory _beneficiaries,uint256
    [] memory _vestAmounts, uint256[] memory _unlockAmounts,
    uint256[] memory _unlockTimes, uint64[] memory _startTimes,
    uint64[] memory _endTimes)
function setAdmin(address admin, bool enabled)
```

Risk Level:

Likelihood - 3
Impact - 1

Recommendations:

RESOURCE_SETTER, PAUSER roles and onlyResourceSetter, onlyPauser modifiers should be implemented for the following functions to avoid centralization of the contract.

Listing 12: Centralized Functions

```
function pause() onlyPauser
function unpause() onlyPauser
function createClaim(address _beneficiary, uint256 _vestAmount,
    uint256 _unlockAmount, uint256 _unlockTime, uint64 _startTime,
    uint64 _endTime) onlyResourceSetter
function createBatchClaim(address[] memory _beneficiaries,uint256
    [] memory _vestAmounts, uint256[] memory _unlockAmounts,
    uint256[] memory _unlockTimes, uint64[] memory _startTimes,
    uint64[] memory _endTimes) onlyResourceSetter
```

Remediation Plan:

ACKNOWLEDGED: The Biconomy Team acknowledged this issue.

3.5 (HAL-05) OWNER CAN RENOUNCE OWNERSHIP - LOW

Description:

Owner of the contract is usually the account which deploys the contract. As a result, the Owner can perform some privileged functions like transferOwnership(). In BicoVestingFlat.sol smart contract, the renounceOwnership function is used to renounce being Owner. Otherwise, if the ownership was not transferred before, the contract will never have an Owner, which is dangerous.

Code Location:

Listing 13: BicoVestingFlat.sol

```
contract BicoVesting is AccessProtected, Pausable, ReentrancyGuard
    {
. . .
}
```

Risk Level:

Likelihood - 3
Impact - 1

Recommendations:

It is recommended to prevent the current owner from calling the renounceOwnership method before transferring the Ownership to another address. In addition, if a multi-signature wallet is used, calling the renounceOwnership method should be confirmed for two or more users.

Remediation Plan:

ACKNOWLEDGED: The Biconomy Team acknowledged this issue.

3.6 (HAL-06) LACK OF ZERO ADDRESS CHECK - LOW

Description:

The BicoVestingFlat.sol contract have multiple input fields on their both public and private functions. Some of these inputs are required as address variable.

During the test, it has seen some of these inputs are not protected against using the address(0) as the target address. It is not recommended to use zero address as target addresses on the contracts.

Code Location:

Listing 14: BicoVestingFlat.sol (Lines 1077)

```
function setAdmin(address admin, bool enabled) external onlyOwner
    {
        _admins[admin] = enabled;
        emit AdminAccessSet(admin, enabled);
    }
```

Listing 15: BicoVestingFlat.sol (Lines 1144)

```
constructor(address _tokenAddress) {
        tokenAddress = _tokenAddress;
    }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendations:

It is recommended to implement additional address check to detect is current contract getting used as a target address.

Listing 16: BicoVestingFlat.sol

```
function setAdmin(address admin, bool enabled) external onlyOwner
    {
        require(admin != address(0), "Address can not be zero.");
        _admins[admin] = enabled;
        emit AdminAccessSet(admin, enabled);
    }
```

Listing 17: BicoVestingFlat.sol

```
constructor(address _tokenAddress) {
        require(_tokenAddress != address(0), "Address can not be
            zero.");
        tokenAddress = _tokenAddress;
    }
```

Remediation Plan:

SOLVED: This issue was removed by implementing zero address checks to the contract code.

Commit ID: **f1ad27ca200d00adb1568b9d6a16bc10dda555e3**

3.7 (HAL-07) POSSIBLE MISUSE OF OWNERSHIP FUNCTIONS - LOW

Description:

Some ownership functions on the contract come directly from the included libraries. These functions are listed in the **Code Location** section. These functions are thought to have been included mistakenly.

Code Location:

Listing 18: Misused Functions

```
function renounceOwnership()
function transferOwnership(address newOwner)
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendations:

It is recommended to override and disable these functions.

Remediation Plan:

ACKNOWLEDGED: The Biconomy Team acknowledged this issue.

3.8 (HAL-08) MISSING ISACTIVE CONTROL ON REVOKE FUNCTION - LOW

Description:

There is a revoke method that is used to invalidate claims created specifically for beneficiary addresses on the contract. A check over this method has been found to be missing. As a result, the function can be executed again even if the claim is invalidated. This may adversely affect the use of gas.

Code Location:

Listing 19: BicoVestingFlat.sol (Lines 1291)

```
1290 function revoke(address beneficiary) external onlyAdmin {
1291         claims[beneficiary].isActive = false;
1292         emit Revoked(beneficiary);
1293     }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendations:

It is suggested to implement following control to revoke method.

Listing 20: BicoVestingFlat.sol (Lines 1291)

```
1290 function revoke(address beneficiary) external onlyAdmin {
1291         require(claims[beneficiary] != false, "Already invalidated
             .");
1292         claims[beneficiary].isActive = false;
1293         emit Revoked(beneficiary);
1294     }
```

Remediation Plan:

SOLVED: The Biconomy Team solved this issue by adding additional require check to the contract code.

Commit ID: **f1ad27ca200d00adb1568b9d6a16bc10dda555e3**

3.9 (HAL-09) USE OF BLOCK.TIMESTAMP - LOW

Description:

During a manual review, the use of block.timestamp has identified. The contract developers should be aware that this does not mean current time. Miners can influence the value of block.timestamp to perform Maximal Extractable Value (MEV) attacks. The use of block.timestamp creates a risk that miners could perform time manipulation to influence price oracles. Miners can modify the timestamp by up to 900 seconds. It is also known that these block.timestamp values only used on events. However, if a malicious miner exploits any vulnerability on the contract, this miner can confuse incident response teams by manipulating these events.

Code Location:

Listing 21: BicoVestingFlat.sol (Lines 1241,1242,1245,1247,1252,1253,1263,1264)

```
1234 function claimableAmount(address beneficiary)
1235         public
1236         view
1237         returns (uint256)
1238     {
1239         Claim memory _claim = claims[beneficiary];
1240         if (
1241             block.timestamp < _claim.startTime &&
1242             block.timestamp < _claim.unlockTime
1243         ) return 0;
1244         if (_claim.amountClaimed == _claim.vestAmount) return 0;
1245         uint256 currentTimestamp = block.timestamp > _claim.
             endTime
1246             ? _claim.endTime
1247             : block.timestamp;
1248         uint256 claimPercent;
1249         uint256 claimAmount;
1250         uint256 unclaimedAmount;
1251         if (
```

```
                _claim.unlockTime <= block.timestamp &&
                _claim.startTime <= block.timestamp
        ) {
            claimPercent = currentTimestamp.sub(_claim.startTime).
                mul(1e18).div(
                    _claim.endTime.sub(_claim.startTime)
                );
            claimAmount = _claim.vestAmount.mul(claimPercent).div
                (1e18).add(
                _claim.unlockAmount
            );
            unclaimedAmount = claimAmount.sub(_claim.amountClaimed
                );
        } else if (
            _claim.unlockTime > block.timestamp &&
            _claim.startTime <= block.timestamp
        ) {
            claimPercent = currentTimestamp.sub(_claim.startTime).
                mul(1e18).div(
                    _claim.endTime.sub(_claim.startTime)
                );
            claimAmount = _claim.vestAmount.mul(claimPercent).div
                (1e18);
            unclaimedAmount = claimAmount.sub(_claim.amountClaimed
                );
        } else {
            claimAmount = _claim.unlockAmount;
            unclaimedAmount = claimAmount.sub(_claim.amountClaimed
                );
        }
        return unclaimedAmount;
    }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendations:

Use block.number instead of block.timestamp or now to reduce the risk of Maximal Extractable Value (MEV) attacks. Check if the timescale of the project occurs across years, days, and months rather than seconds.

References:

Block Values as a Proxy for Time

Remediation Plan:

ACKNOWLEDGED: The Biconomy Team acknowledged this issue.

3.10 (HAL-10) IGNORED RETURN VALUES - INFORMATIONAL

Description:

The return value of an external call is not stored in a local or state variable. In the BicoVestingFlat.sol contract, the ERC20(tokenAddress).transferFrom() method is called and the return values are ignored on the workflow.

Code Location:

Listing 22: BicoVestingFlat.sol (Lines 1170)

```
1165 require(
1166             ERC20(tokenAddress).allowance(msg.sender, address(this
                    )) >=
1167                 (_vestAmount.add(_unlockAmount)),
1168             "INVALID_ALLOWANCE"
1169         );
1170         ERC20(tokenAddress).transferFrom(
1171             msg.sender,
1172             address(this),
1173             _vestAmount
1174         );
```

Listing 23: BicoVestingFlat.sol (Lines 1283)

```
1278 function claim() external whenNotPaused nonReentrant {
1279         address beneficiary = msg.sender;
1280         Claim memory _claim = claims[beneficiary];
1281         require(_claim.isActive, "CLAIM_INACTIVE");
1282         uint256 unclaimedAmount = claimableAmount(beneficiary);
1283         ERC20(tokenAddress).transfer(beneficiary, unclaimedAmount)
                 ;
1284         _claim.amountClaimed = _claim.amountClaimed +
                 unclaimedAmount;
1285         if (_claim.amountClaimed == _claim.vestAmount) _claim.
                 isActive = false;
```

```
            claims[beneficiary] = _claim;
            emit Claimed(beneficiary, unclaimedAmount);
        }
```

Listing 24: BicoVestingFlat.sol (Lines 1297)

```
1295 function withdrawTokens(address wallet, uint256 amount) external
         onlyOwner nonReentrant {
1296         require(amount > 0, "Nothing to withdraw");
1297         ERC20(tokenAddress).transfer(wallet, amount);
1298     }
```

Risk Level:

Likelihood - 2
Impact - 1

Recommendations:

It is recommended to control returned values for these transfer operations to validate if transfer succeed or not.

Listing 25: Possible Fix

```
(bool success, ) = ERC20(tokenAddress).transfer(wallet, amount);
require(success, "Transfer failed.");
```

Remediation Plan:

ACKNOWLEDGED: Biconomy Team acknowledged this issue.

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. 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, Slither was run on the all-scoped contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Slither Results:

```
BicoVesting.createClaim(address,uint256,uint256,uint256,uint64,uint64) (contracts/vesting/BicoVestingFlat.sol#1147-1194) ignores return value by ERC20(tokenAddress).transferFrom(msg.sender,address(this),_v
estAmount) (contracts/vesting/BicoVestingFlat.sol#1170-1174)
BicoVesting.claim() (contracts/vesting/BicoVestingFlat.sol#1278-1288) ignores return value by ERC20(tokenAddress).transfer(beneficiary,unclaimedAmount) (contracts/vesting/BicoVestingFlat.sol#1283)
BicoVesting.withdrawTokens(address,uint256) (contracts/vesting/BicoVestingFlat.sol#1295-1298) ignores return value by ERC20(tokenAddress).transfer(wallet,amount) (contracts/vesting/BicoVestingFlat.sol#1297
)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

BicoVesting.claimableAmount(address) (contracts/vesting/BicoVestingFlat.sol#1234-1276) performs a multiplication on the result of a division:
        -claimPercent = currentTimestamp.sub(_claim.startTime).mul(1e18).div(_claim.endTime.sub(_claim.startTime)) (contracts/vesting/BicoVestingFlat.sol#1255-1257)
        -claimAmount = _claim.vestAmount.mul(claimPercent).div(1e18).add(_claim.unlockAmount) (contracts/vesting/BicoVestingFlat.sol#1258-1260)
BicoVesting.claimableAmount(address) (contracts/vesting/BicoVestingFlat.sol#1234-1276) performs a multiplication on the result of a division:
        -claimPercent = currentTimestamp.sub(_claim.startTime).mul(1e18).div(_claim.endTime.sub(_claim.startTime)) (contracts/vesting/BicoVestingFlat.sol#1266-1268)
        -claimAmount = _claim.vestAmount.mul(claimPercent).div(1e18) (contracts/vesting/BicoVestingFlat.sol#1269)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

BicoVesting.claim() (contracts/vesting/BicoVestingFlat.sol#1278-1288) uses a dangerous strict equality:
        - _claim.amountClaimed == _claim.vestAmount (contracts/vesting/BicoVestingFlat.sol#1285)
BicoVesting.claimableAmount(address) (contracts/vesting/BicoVestingFlat.sol#1234-1276) uses a dangerous strict equality:
        - _claim.amountClaimed == _claim.vestAmount (contracts/vesting/BicoVestingFlat.sol#1244)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

Reentrancy in BicoVesting.claim() (contracts/vesting/BicoVestingFlat.sol#1278-1288):
        External calls:
        - ERC20(tokenAddress).transfer(beneficiary,unclaimedAmount) (contracts/vesting/BicoVestingFlat.sol#1283)
        State variables written after the call(s):
        - claims[beneficiary] = _claim (contracts/vesting/BicoVestingFlat.sol#1286)
Reentrancy in BicoVesting.createClaim(address,uint256,uint256,uint256,uint64,uint64) (contracts/vesting/BicoVestingFlat.sol#1147-1194):
        External calls:
        - ERC20(tokenAddress).transferFrom(msg.sender,address(this),_vestAmount) (contracts/vesting/BicoVestingFlat.sol#1170-1174)
        State variables written after the call(s):
        - claims[_beneficiary] = newClaim (contracts/vesting/BicoVestingFlat.sol#1184)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

BicoVesting.createBatchClaim(address[],uint256[],uint256[],uint256[],uint64[],uint64[]).i (contracts/vesting/BicoVestingFlat.sol#1213) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

BicoVesting.constructor(address)._tokenAddress (contracts/vesting/BicoVestingFlat.sol#1143) lacks a zero-check on :
                - tokenAddress = _tokenAddress (contracts/vesting/BicoVestingFlat.sol#1144)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
```

AUTOMATED TESTING

4.2 Static Analysis - Possible Findings and Results

According to these test results, **some findings found by Slither were considered as false positives, while some of these findings were real security concerns.** All relevant findings were reviewed by the auditors and relevant findings addressed in the report as security concerns.

Lack of Zero Address Check:

This issue has been declared as **valid** issue since it is possible to set address(0) to tokenAddress variable. This issue has been addressed on the report.

(HAL-06) LACK OF ZERO ADDRESS CHECK

Reentrancy Vulnerability:

This vulnerability has been declared as **False-Positive** since it is not possible to trigger the Reentrancy vulnerability.

Ignored Return Values:

This issue has been declared as a **valid** issue since transfer operations do not return any boolean values. This issue has been addressed on the report.

(HAL-10) IGNORED RETURN VALUES

THANK YOU FOR CHOOSING

HALBORN