Biconomy - BICO Token

Smart Contract Security Audit

DRAFT

Prepared by: **Halborn**

Date of Engagement: **September 27th, 2021 - October 9th, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	10/06/2021	Ataberk Yavuzer
0.2	Document Edits	10/07/2021	Ataberk Yavuzer
0.3	Document Review	10/08/2021	Gabi Urrutia
1.0	Final Draft	10/11/2021	Ataberk Yavuzer
1.1	Remediation Plan	10/11/2021	Ataberk Yavuzer

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** Token Contracts beginning on September 27th and ending on October 9th, 2021.

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

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

1.2 AUDIT SUMMARY

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

The purpose of this audit is to:

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

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 and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process,and implementation; automated testing techniques help enhance coverage of the bridge code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose
- Smart contract manual code review and walkthrough
- 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 (Brownie, Remix IDE)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident, and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.

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

RISK SCALE - IMPACT

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

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

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

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

1.4 SCOPE

IN-SCOPE:

The security assessment was scoped to the following smart contracts.

Repository URL: https://github.com/bcnmy/biconomy-contracts

- BicoTokenProxy.sol
- BicoTokenImplementation.sol

Commit ID: 4eef2c235705f5f49e9a807c2f47faf89dffa362

OUT-OF-SCOPE:

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

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	2	5	2

LIKELIHOOD

IMPACT

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

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
UNWANTED TOKEN MINTING ON CONTRACT UPGRADE	Medium	SOLVED: 10/10/2021
POSSIBLE FRONT-RUNNING ON INITIALIZATION	Medium	SOLVED: 10/10/2021
LACK OF ADDRESS CONTROL	Low	SOLVED: 10/10/2021
MISSING RE-ENTRANCY GUARD	Low	SOLVED: 10/10/2021
FLOATING PRAGMA	Low	SOLVED: 10/10/2021
PRAGMA VERSION	Low	ACKNOWLEDGED
USE OF BLOCK.TIMESTAMP	Low	ACKNOWLEDGED
UNUSED FUNCTIONS AND VARIABLES	Informational	NOT APPLICABLE
MISUSE OF FUNCTION HOOKS	Informational	SOLVED: 10/11/2021

FINDINGS & TECH DETAILS

3.1 (HAL-01) UNWANTED TOKEN MINTING ON CONTRACT UPGRADE - MEDIUM

Description:

The BICO Token contract has multiple features such as Governed, AccessControl, Pausable and ContextUpgradeable. Also, this contract uses a Proxy Contract to upgrade it's context to implement new features or fixing possible security issues. First of all, Token Contract needs to be deployed. This step is followed by the Proxy Contract initializing the Token Contract. During the initialization process, Token fields are set by the initialize function. Also, 1000000000 BICO tokens will be minted during that process.

Calling the __BicoTokenImplementation_init_unchained function will mint these tokens on every contract upgrade. However, these tokens should be minted on the first initialization only.

Code Location:

Listing 1: contracts/bico-token/bico/BicoTokenImplementation.sol (Lines 580,583)

```
580     function __BicoTokenImplementation_init_unchained(address
            beneficiary) internal initializer {
581         _name = "Biconomy Token";
582         _symbol = "BICO";
583         _mint(msg.sender, 1000000000 * 10 ** decimals());
584         _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
585         _setupRole(PAUSER_ROLE, msg.sender);
586 
587         // EIP-712 domain separator
588         DOMAIN_SEPARATOR = keccak256(
589             abi.encode(
590                 DOMAIN_TYPE_HASH,
591                 keccak256("Biconomy Token"),
592                 keccak256("1"),
593                 address(this),
594                 bytes32(getChainId())
```

```
        )
    );
}
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendations:

It is recommended to implement an additional check to the contract for validating if the contract was already initialized.

Remediation Plan:

SOLVED: Biconomy Team solved this issue by moving the mint function into initialize function. Also, two new variables(_initializedVersion , mintingAllowedAfter) are implemented to control if the contract is initialized for the first time.

Commit ID: 1b38ce8d9b86ff6238e93c883e69538f46472077

3.2 (HAL-02) POSSIBLE FRONT-RUNNING ON INITIALIZATION - MEDIUM

Description:

Token Contract initializer were missing access controls, allowing any user to initialize the contract. By front-running the contract, deployers can initialize the contract. Also, on every initialization, contract mints 1000000000 BICO tokens to the beneficiary address. So, it is possible to any front-runner attacker gets all supply during the initialization by front-running.

In addition, the attacker acclaim the trustedForwarder and Contract Admin roles by exploiting this vulnerability.

Code Location:

Listing 2: contracts/bico-token/bico/BicoTokenImplementation.sol (Lines 580,583,584,585)

```
564 function initialize(address beneficiary, address trustedForwarder)
        public initializer {
565         __BicoTokenImplementation_init_unchained(beneficiary);
566         __ERC2771Context_init(trustedForwarder);
567         __Pausable_init();
568         __AccessControl_init();
569         __Governed_init(msg.sender);
570     }
571
572     function __BicoTokenImplementation_init(address beneficiary,
            address trustedForwarder) internal initializer {
573         __ERC2771Context_init(trustedForwarder);
574         __Pausable_init();
575         __AccessControl_init();
576         __Governed_init(msg.sender);
577         __BicoTokenImplementation_init_unchained(beneficiary);
578     }
579
580     function __BicoTokenImplementation_init_unchained(address
```

```
        beneficiary) internal initializer {
        _name = "Biconomy Token";
        _symbol = "BICO";
        _mint(msg.sender, 1000000000 * 10 ** decimals());
        _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
        _setupRole(PAUSER_ROLE, msg.sender);

        // EIP-712 domain separator
        DOMAIN_SEPARATOR = keccak256(
            abi.encode(
                DOMAIN_TYPE_HASH,
                keccak256("Biconomy Token"),
                keccak256("1"),
                address(this),
                bytes32(getChainId())
            )
        );
    }
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendations:

The BICO Token Contract should be initialized immediately by correct user after deployment. This can be achieved by deployment test script. Use a factory pattern that will deploy and initialize the contracts atomically to prevent front-running of the initialization. Additionally, access control for initialize function should be implemented.

Remediation Plan:

SOLVED: Biconomy Team fixed this issue by directly initializing the contract after deploying the contract in the deployment and upgrade test scripts.

Commit ID: ccdbbe9087a1139946a9b3fc4d9f38b537da241d

3.3 (HAL-03) LACK OF ZERO ADDRESS CONTROL - LOW

Description:

Some functions are missing address validation. Every address should be validated and checked that is different than zero. During the test, it was determined that the address(0) control was not performed on the setTrustedForwarder function.

Code Location:

Listing 3: contracts/bico-token/bico/BicoTokenImplementation.sol (Lines 1034,1035)

```
1034 function setTrustedForwarder(address payable _forwarder) external
         onlyGovernor {
1035         _trustedForwarder = _forwarder;
1036         emit TrustedForwarderChanged(_forwarder, msg.sender);
1037     }
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendations:

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

Remediation Plan:

SOLVED: Biconomy Team has resolved this issue by implementing zero address control into the setTrustedForwarder function.

Commit ID: 5bc8cb7d8815aeab151277c717f58042d002cd52

3.4 (HAL-04) MISSING RE-ENTRANCY GUARD - LOW

Description:

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

Code Location:

Listing 4: Missing Re-Entrancy Guard

```
transfer(address recipient, uint256 amount)
approve(address spender, uint256 amount)
transferFrom(address sender, address recipient, uint256 amount)
increaseAllowance(address spender, uint256 addedValue)
decreaseAllowance(address spender, uint256 subtractedValue)
function approveWithSig(uint8 _v, bytes32 _r, bytes32 _s, uint256
    _deadline, address _sender, uint256 _batchId, address
    _recipient, uint256 _amount)
function transferWithSig(uint8 _v, bytes32 _r, bytes32 _s, uint256
     _deadline, address _sender, uint256 _batchId, address
    _recipient, uint256 _amount)
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendations:

In the BicoTokenImplementation.sol contract, functions above are missing a nonReentrant guard. Use the nonReentrant modifier to avoid introducing future vulnerabilities.

Remediation Plan:

SOLVED: Biconomy Team solved this issue by appending **OpenZeppelin ReentrancyGuard** library to the Token contract. Halborn Team has confirmed that this library implemented properly.

Commit ID: a1974759b906f6f86d7999e66c10302611631eef

3.5 (HAL-05) 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 5

```
libs/BaseRelayRecipient.sol:3:pragma solidity ^0.8.0;
libs/IRelayRecipient.sol:2:pragma solidity ^0.8.0;
test/Greeter.sol:2:pragma solidity ^0.8.0;
BiconomyTokenTransparent.sol:4:pragma solidity ^0.8.2;
bico/BicoTokenProxy.sol:4:pragma solidity ^0.8.0;
bico/ERC20Meta.sol:3:pragma solidity ^0.8.0;
bico/BicoTokenImplementation.sol:2:pragma solidity ^0.8.0;
bico/BicoTokenV2.sol:2:pragma solidity ^0.8.0;
bico/BicoToken.sol:4:pragma solidity ^0.8.0;
bico/BicoToken.sol:9:pragma solidity ^0.8.0;
bico/BicoToken.sol:97:pragma solidity ^0.8.0;
bico/BicoToken.sol:124:pragma solidity ^0.8.0;
bico/BicoToken.sol:335:pragma solidity ^0.8.0;
bico/BicoToken.sol:402:pragma solidity ^0.8.0;
BiconomyTokenUUPS.sol:5:pragma solidity ^0.8.2;
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendations:

Consider locking the pragma version with known bugs for the compiler version. 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: This finding was fixed by the Biconomy Team by locking the pragma version. Since the new commit, the Caret(^) symbol in all pragma versions has been removed.

Commit ID: 5bc8cb7d8815aeab151277c717f58042d002cd52

3.6 (HAL-06) PRAGMA VERSION - LOW

Description:

The project uses one of the latest pragma version (0.8.0) which was released on 16th of December, 2020. The latest pragma version (0.8.9) was released in October 2021. Many pragma versions have been lately released, going from version 0.7.x to the recently released version 0.8.x. in just 6 months.

In the Solitidy Github repository, there is a JSON file where are all bugs finding in the different compiler versions. It should be noted that pragma 0.6.12 and 0.7.6 are widely used by Solidity developers and have been extensively tested in many security audits.

Code Location:

Listing 6

```
libs/BaseRelayRecipient.sol:3:pragma solidity ^0.8.0;
libs/IRelayRecipient.sol:2:pragma solidity ^0.8.0;
test/Greeter.sol:2:pragma solidity ^0.8.0;
BiconomyTokenTransparent.sol:4:pragma solidity ^0.8.2;
bico/BicoTokenProxy.sol:4:pragma solidity ^0.8.0;
bico/ERC20Meta.sol:3:pragma solidity ^0.8.0;
bico/BicoTokenImplementation.sol:2:pragma solidity ^0.8.0;
bico/BicoTokenV2.sol:2:pragma solidity ^0.8.0;
bico/BicoToken.sol:4:pragma solidity ^0.8.0;
bico/BicoToken.sol:9:pragma solidity ^0.8.0;
bico/BicoToken.sol:97:pragma solidity ^0.8.0;
bico/BicoToken.sol:124:pragma solidity ^0.8.0;
bico/BicoToken.sol:335:pragma solidity ^0.8.0;
bico/BicoToken.sol:402:pragma solidity ^0.8.0;
BiconomyTokenUUPS.sol:5:pragma solidity ^0.8.2;
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendations:

If possible, consider using the latest stable pragma version that has been thoroughly tested to prevent potential undiscovered vulnerabilities such as pragma between 0.6.12 - 0.7.6.

References:

- Solidity Releases
- Solidity Bugs By Version

Remediation Plan:

ACKNOWLEDGED: Biconomy Team decided to use pragma 0.8.4.

3.7 (HAL-07) USAGE 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.

Code Location:

Listing 7: contracts/bico-token/bico/BicoTokenImplementation.sol

```
require(_deadline == 0 || block.timestamp <= _deadline, "BICO::
    expired transfer");
```

Risk Level:

Likelihood - 3
Impact - 1

Recommendation:

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. If possible, it is recommended to use Oracles.

Remediation Plan:

ACKNOWLEDGED: Biconomy Team acknowledged this issue and claims that **Usage of block.timestamp** does not possess any security risks.

3.8 (HAL-08) UNUSED FUNCTIONS AND VARIABLES - INFORMATIONAL

Description:

During the test, it was determined that some functions and variables on the contract were not used in any way, although they were defined on the contract. This situation does not pose any risk in terms of security. But it is important for the readability and applicability of the code.

Code Location:

Listing 8: Unused Functions and Variables

```
increaseAllowance(address spender, uint256 addedValue)
decreaseAllowance(address spender, uint256 subtractedValue)
_burn(address account, uint256 amount)
uint 256 batchNonce
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

It is recommended to review these unused functions and variables, and to delete them from the contract if they will continue to be unused.

Remediation Plan:

NOT APPLICABLE: Since all variables and functions addressed on this vulnerability are used on the contract, this vulnerability is not applicable.

- increaseAllowance and decreaseAllowance are public versions for users.
- _batchNonce variable is removed from signature functions.

3.9 (HAL-09) MISUSE OF FUNCTION HOOKS - INFORMATIONAL

Description:

During the audit, it was seen that two hook functions were implemented on the contract. The purpose of using these functions under normal conditions is to detect variables such as gas usage. However, it has been seen that these two functions defined on the contract have no purpose.

Code Location:

Listing 9: contracts/bico-token/bico/BicoTokenImplementation.sol

```
function _beforeTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual {}
```

Listing 10: contracts/bico-token/bico/BicoTokenImplementation.sol

```
function _afterTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual {}
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

It is recommended that these functions be implemented in accordance with their purpose, and if not, they should be deleted from the contract.

Remediation Plan:

SOLVED: Biconomy Team solved this issue by removing unused _beforeTokenTransfer and _afterTokenTransfer hook functions.

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:

```
AccessControlUpgradeable.__gap (contracts/bico-token/bico/BicoTokenImplementation.sol#350) shadows:
        - ERC165Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#35)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing

BicoTokenImplementation.setTrustedForwarder(address)._forwarder (contracts/bico-token/bico/BicoTokenImplementation.sol#1033) lacks a zero-check on :
                - _trustedForwarder = _forwarder (contracts/bico-token/bico/BicoTokenImplementation.sol#1034)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

BicoTokenImplementation.approveWithSig(uint256,uint8,bytes32,bytes32,uint256,address,uint256,address,uint256) (contracts/bico-token/bico/BicoTokenImplementation.sol#938-973) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(_deadline == 0 || block.timestamp <= _deadline,BICO:: expired approval) (contracts/bico-token/bico/BicoTokenImplementation.sol#970)
BicoTokenImplementation.transferWithSig(uint256,uint8,bytes32,bytes32,uint256,address,uint256,address,uint256) (contracts/bico-token/bico/BicoTokenImplementation.sol#987-1023) uses timestamp for comparison
s
        Dangerous comparisons:
        - require(bool,string)(_deadline == 0 || block.timestamp <= _deadline,BICO:: expired transfer) (contracts/bico-token/bico/BicoTokenImplementation.sol#1020)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

ERC2771ContextUpgradeable._msgSender() (contracts/bico-token/bico/BicoTokenImplementation.sol#26-35) uses assembly
        - INLINE ASM (contracts/bico-token/bico/BicoTokenImplementation.sol#29-31)
ECDSA.recover(bytes32,bytes) (contracts/bico-token/bico/BicoTokenImplementation.sol#448-485) uses assembly
        - INLINE ASM (contracts/bico-token/bico/BicoTokenImplementation.sol#462-466)
BicoTokenImplementation.getChainId() (contracts/bico-token/bico/BicoTokenImplementation.sol#1038-1042) uses assembly
        - INLINE ASM (contracts/bico-token/bico/BicoTokenImplementation.sol#1040)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

AccessControlUpgradeable._setRoleAdmin(bytes32,bytes32) (contracts/bico-token/bico/BicoTokenImplementation.sol#331-335) is never used and should be removed
BicoTokenImplementation.__BicoTokenImplementation_init(address) (contracts/bico-token/bico/BicoTokenImplementation.sol#572-578) is never used and should be removed
BicoTokenImplementation._burn(address,uint256) (contracts/bico-token/bico/BicoTokenImplementation.sol#832-847) is never used and should be removed
ECDSA.toEthSignedMessageHash(bytes32) (contracts/bico-token/bico/BicoTokenImplementation.sol#495-499) is never used and should be removed
ERC165Upgradeable.__ERC165_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#23-25) is never used and should be removed
ERC2771ContextUpgradeable._msgData() (contracts/bico-token/bico/BicoTokenImplementation.sol#37-43) is never used and should be removed
StringsUpgradeable.toHexString(uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#39-50) is never used and should be removed
StringsUpgradeable.toString(uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#14-34) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (contracts/bico-token/bico/BicoTokenImplementation.sol#2) 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/IAccessControlUpgradeable.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/utils/StringsUpgradeable.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/introspection/ERC165Upgradeable.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/introspection/IERC165Upgradeable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.7 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Function ERC2771ContextUpgradeable.__ERC2771Context_init(address) (contracts/bico-token/bico/BicoTokenImplementation.sol#14-16) is not in mixedCase
Function ERC2771ContextUpgradeable.__ERC2771Context_init_unchained(address) (contracts/bico-token/bico/BicoTokenImplementation.sol#18-20) is not in mixedCase
Variable ERC2771ContextUpgradeable._trustedForwarder (contracts/bico-token/bico/BicoTokenImplementation.sol#12) is not in mixedCase
Variable ERC2771ContextUpgradeable.__gap (contracts/bico-token/bico/BicoTokenImplementation.sol#44) is not in mixedCase
Function PausableUpgradeable.__Pausable_init() (contracts/bico-token/bico/BicoTokenImplementation.sol#74-76) is not in mixedCase
Function PausableUpgradeable.__Pausable_init_unchained() (contracts/bico-token/bico/BicoTokenImplementation.sol#78-80) is not in mixedCase
Variable PausableUpgradeable.__gap (contracts/bico-token/bico/BicoTokenImplementation.sol#136) is not in mixedCase
Parameter BicoTokenImplementation.setTrustedForwarder(address)._forwarder (contracts/bico-token/bico/BicoTokenImplementation.sol#1033) is not in mixedCase
Variable BicoTokenImplementation.DOMAIN_SEPARATOR (contracts/bico-token/bico/BicoTokenImplementation.sol#530) is not in mixedCase
Function ERC165Upgradeable.__ERC165_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#23-25) is not in mixedCase
Function ERC165Upgradeable.__ERC165_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#27-28) is not in mixedCase
Variable ERC165Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#35) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

BicoTokenImplementation.initialize(address,address) (contracts/bico-token/bico/BicoTokenImplementation.sol#561-568) uses literals with too many digits:
        - _mint(beneficiary,1000000000 * 10 ** decimals()) (contracts/bico-token/bico/BicoTokenImplementation.sol#563)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

GovernedUpgradeable.__gap (contracts/bico-token/bico/BicoTokenImplementation.sol#422) is never used in BicoTokenImplementation (contracts/bico-token/bico/BicoTokenImplementation.sol#504-1045)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

grantRole(bytes32,address) should be declared external:
        - AccessControlUpgradeable.grantRole(bytes32,address) (contracts/bico-token/bico/BicoTokenImplementation.sol#269-271)
revokeRole(bytes32,address) should be declared external:
        - AccessControlUpgradeable.revokeRole(bytes32,address) (contracts/bico-token/bico/BicoTokenImplementation.sol#282-284)
renounceRole(bytes32,address) should be declared external:
        - AccessControlUpgradeable.renounceRole(bytes32,address) (contracts/bico-token/bico/BicoTokenImplementation.sol#300-304)
initialize(address,address) should be declared external:
        - BicoTokenImplementation.initialize(address,address) (contracts/bico-token/bico/BicoTokenImplementation.sol#561-568)
name() should be declared external:
        - BicoTokenImplementation.name() (contracts/bico-token/bico/BicoTokenImplementation.sol#603-605)
symbol() should be declared external:
        - BicoTokenImplementation.symbol() (contracts/bico-token/bico/BicoTokenImplementation.sol#611-613)
totalSupply() should be declared external:
        - BicoTokenImplementation.totalSupply() (contracts/bico-token/bico/BicoTokenImplementation.sol#635-637)
balanceOf(address) should be declared external:
        - BicoTokenImplementation.balanceOf(address) (contracts/bico-token/bico/BicoTokenImplementation.sol#642-644)
transfer(address,uint256) should be declared external:
        - BicoTokenImplementation.transfer(address,uint256) (contracts/bico-token/bico/BicoTokenImplementation.sol#655-658)
allowance(address,address) should be declared external:
        - BicoTokenImplementation.allowance(address,address) (contracts/bico-token/bico/BicoTokenImplementation.sol#663-665)
approve(address,uint256) should be declared external:
        - BicoTokenImplementation.approve(address,uint256) (contracts/bico-token/bico/BicoTokenImplementation.sol#674-677)
transferFrom(address,address,uint256) should be declared external:
        - BicoTokenImplementation.transferFrom(address,address,uint256) (contracts/bico-token/bico/BicoTokenImplementation.sol#709-722)
increaseAllowance(address,uint256) should be declared external:
        - BicoTokenImplementation.increaseAllowance(address,uint256) (contracts/bico-token/bico/BicoTokenImplementation.sol#736-739)
decreaseAllowance(address,uint256) should be declared external:
        - BicoTokenImplementation.decreaseAllowance(address,uint256) (contracts/bico-token/bico/BicoTokenImplementation.sol#755-762)
approveWithSig(uint256,uint8,bytes32,bytes32,uint256,address,uint256,address,uint256) should be declared external:
        - BicoTokenImplementation.approveWithSig(uint256,uint8,bytes32,bytes32,uint256,address,uint256,address,uint256) (contracts/bico-token/bico/BicoTokenImplementation.sol#938-973)
transferWithSig(uint256,uint8,bytes32,bytes32,uint256,address,uint256,address,uint256) should be declared external:
        - BicoTokenImplementation.transferWithSig(uint256,uint8,bytes32,bytes32,uint256,address,uint256,address,uint256) (contracts/bico-token/bico/BicoTokenImplementation.sol#987-1023)
pause() should be declared external:
        - BicoTokenImplementation.pause() (contracts/bico-token/bico/BicoTokenImplementation.sol#1025-1027)
unpause() should be declared external:
        - BicoTokenImplementation.unpause() (contracts/bico-token/bico/BicoTokenImplementation.sol#1029-1031)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

As a result of the tests completed with the Slither tool, some results were obtained and these results were reviewed by Halborn. In line with the reviewed results, it was decided that some vulnerabilities were false-positive and these results were not included in the report. The actual vulnerabilities are already included in the findings on the report.

THANK YOU FOR CHOOSING

HALBORN