Monion Staking Contracts

Smart Contract Security Audit

Prepared by ShellBoxes

August 30th, 2022 - September 12th, 2022

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	Monion Global
Version	1.0
Classification	Public

Scope

The Monion Staking Contracts Contract in the Monion Staking Contracts Repository

Repo	Commit Hash
https://github.com/MonionGlobal/stakingcontract	30223e5e03c7c5b1712f5cc5bf766373b629e8a3

Files	MD5 Hash
Admin.sol	46f1a8eab7d1b3b413ee5ac6d521ad05
Monion.sol	f516a74a5f53b01536a697f0a389e1ea
RewardPool.sol	637b8c643cb05ff30003c7efef99b83a
Staking.sol	3c51f5d1f0693487fcc50a37256c47a0

Re-Audit Scope

Repo	Commit Hash
https://github.com/MonionGlobal/stakingcontract	b1ccba238d1fa7557d1957fae6e29aca8fa74887

Files	MD5 Hash
Admin.sol	336b51226c9bf4d93a78068785ed4624
Monion.sol	e38c21fbfd8b9281f7a284221e38d033
RewardPool.sol	af4cbc85dcc4f9756936c03696195571
Staking.sol	44db592953f2398f6ca0999e7820fd23

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1 Introduction

Monion Global engaged ShellBoxes to conduct a security assessment on the Monion Staking Contracts beginning on August 30th, 2022 and ending September 12th, 2022. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About Monion Global

A social NFT marketplace with a multitude of options, faster transactions, unlockable content, social wallet, venues, and exhibitions in the metaverse.

Issuer	Monion Global
Website	`https://www.monion.io/`
Type	Solidity Smart Contract
Audit Method	Whitebox

1.2 Approach & Methodology

ShellBoxes used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

1.2.1 Risk Methodology

Vulnerabilities or bugs identified by ShellBoxes are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.

Impact				
	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low

Likelihood

2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the Monion Staking Contracts implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include , 2 high-severity, 3 medium-severity, 7 low-severity vulnerabilities.

Vulnerabilities	Severity	Status
A.1. The Owner Can Take The Rewards of the Stakers	HIGH	Acknowledged
A.2. Rewards Claiming Can Take Users Balances	HIGH	Fixed
A.3. The Users Can Get Higher APY	MEDIUM	Acknowledged
A.4. APY Is 2% Instead Of 20%	MEDIUM	Fixed
A.5. Possible Desynchronization Between The Pool Balance And The totalSupply	MEDIUM	Fixed
A.7. Missing Address Verification	LOW	Fixed
A.8. Floating Pragma	LOW	Fixed
B.1. Approve Race Condition	LOW	Fixed
B.2. Floating Pragma	LOW	Fixed
C.1. Missing Address Verification	LOW	Fixed
C.2. Floating Pragma	LOW	Fixed
D.1. Missing Transfer Verification	LOW	Fixed

3 Finding Details

A Staking.sol

A.1 The Owner Can Take The Rewards of the Stakers [HIGH]

Description:

The closePool pool function allows the owner to close the pool and withdraw the pool rewards. This represents a significant centralization risk, where the owner have all the control over the rewards of the stakers.

Code:

Listing 1: Staking.sol

```
200 function closePool() external whenPaused onlyOwner {
201     require(!isPoolClosed, "Pool Already Closed");
203     uint amount = rewardPool.poolBalance();
204     isPoolClosed = true;
205     rewardPool.transfer(owner, amount);
207     emit ClosedPool(msg.sender, amount);
208 }
```

Risk Level:

Likelihood – 4

Impact - 5

Recommendation:

It is recommended to remove this functionality as it represents a significant centralization issue.

Status - Acknowledged

The Monion team has acknowledged the issue, stating that the functionality will be used in a situation where rewards have been left in the pool and unclaimed by users after a significantly long time since the contract has closed staking. The team are planning to use a DAO as a caller to this function in order to allow the community to vote on executing the functionality if needed.

A.2 Rewards Claiming Can Take Users Balances [HIGH]

Description:

The contract transfers the reward amount from its own balance of the stakingToken to the user when he or she claims their benefits. Users will have their staked amounts deducted from them if the contract is not funded with the stakingToken, which may prohibit them from withdrawing their money.

Code:

Listing 2: Staking.sol

```
177 function claimRewards()
178     external
179     whenNotPaused
180     updateReward(msg.sender)
181     nonReentrant
182 {
183     // if(balanceOf[msg.sender] <= 0){
184     // revert Staking__NoStakeInPool();
185     // }

187     if (rewards[msg.sender] <= 0) {
188         revert Staking__NoRewardsAvailable();
189     }
190     require(!isPoolClosed, "Too late! Pool has been closed!");
```

```
192     uint amount = rewards[msg.sender];
193     rewards[msg.sender] = 0;
194     stakingToken.transfer(msg.sender, amount);
195
196     emit RewardsClaimed(msg.sender, amount);
197 }
```

Risk Level:

Likelihood – 3
Impact - 5

Recommendation:

Consider including a clause that prevents reward claims from bringing the contract's balance below the totalSupply value, or use the rewardPool contract to distribute rewards. In addition to that, make sure that the contract have enough funds to pay the stakers' rewards.

Status - Fixed

The Monion team has fixed the issue by using the rewardPool contract to distribute rewards.

A.3 The Users Can Get Higher APY [MEDIUM]

Description:

The project states that the APY is set to 20%. However, the users can always add their rewards to the staked amount and therefore getting even more rewards, this will result in a higher APY to the users, which can be estimated using the following formula: ((1 + 0.2/n)^{n} - 1) * 100. n: the number of periods per year when the user restakes his rewards.

Code:

Listing 3: Staking.sol

```
177 function claimRewards()
```

```
178     external
179     whenNotPaused
180     updateReward(msg.sender)
181     nonReentrant
182 {
183     // if(balanceOf[msg.sender] <= 0){
184     // revert Staking__NoStakeInPool();
185     // }

187     if (rewards[msg.sender] <= 0) {
188         revert Staking__NoRewardsAvailable();
189     }
190     require(!isPoolClosed, "Too late! Pool has been closed!");

192     uint amount = rewards[msg.sender];
193     rewards[msg.sender] = 0;
194     stakingToken.transfer(msg.sender, amount);

196     emit RewardsClaimed(msg.sender, amount);
197 }
```

Risk Level:

Likelihood – 4

Impact - 3

Recommendation:

Consider verifying if this behavior is allowed by the business logic or using another token as a reward token for the stakers.

Status - Acknowledged

The Monion team has acknowledged the issue, stating that this is part of the business logic.

A.4 APY Is 2% Instead Of 20% [MEDIUM]

Description:

The project states that the APY is set to 20%. However, for a diff of one year the staked amount gets multiplied by 0,02, therefore, the APY is 2%.

Code:

Listing 4: Staking.sol

```
218 function _calcReward() public view returns (uint256) {
219     uint prevBalance = balanceOf[msg.sender];
220     uint diff = _lastTimeRewardApplicable() -
221         userLastUpdateTime[msg.sender];
222     uint numerator = prevBalance * totalReward * diff;
223     uint denominator = maximumPoolMonions * validityPeriod;
224     return numerator / denominator;
225 }
```

Risk Level:

Likelihood – 2
Impact - 5

Recommendation:

Consider adjusting the numbers to assure the 20% APY, this can be achieved by changing the totalReward to 1000000 * 1e18.

Status - Fixed

The Monion team has fixed the issue by adjusting the values to offer 20% APY as claimed.

A.5 Possible Desynchronization Between The Pool Balance And The totalSupply [MEDIUM]

Description:

The overall balance of the staking contract and the variable totalSupply will become out of sync if a user sends Monion tokens directly to the contract without using the stake function. Tokens belonging to the user may end up being locked in the pool as a result of this.

Code:

Listing 5: Staking.sol

```
49 uint public totalSupply; //Total amount of ERC20 tokens currently staked
       ↪ in the contract.
```

Risk Level:

Likelihood – 2
Impact - 4

Recommendation:

To determine the exact total supply of the contract and avoid locking any funds, consider utilizing the balanceOf function.

Status - Fixed

The Monion team has fixed the issue by using the balanceOf function inside the getStakedBalance function in order to get the balance of the contract.

A.6 Missing Transfer Verification [MEDIUM]

Description:

The ERC20 standard token implementation functions return the transaction status as a boolean. It is a good practice to check for the return status of the function call to ensure that the transaction was executed successfully. It is advised to enclose these function calls with require() to ensure that, when the intended ERC20 function call returns false, the caller transaction also fails.

Code:

Listing 6: Staking.sol

```
141 function _unstake(uint amount)
142     internal
143     whenNotPaused
144     updateReward(msg.sender)
145 {
146     if (balanceOf[msg.sender] - amount < 0) {
147         revert Staking__WithdrawLessThanYourBalance();
148     }
149     balanceOf[msg.sender] -= amount;
150     totalSupply -= amount;

152     stakingToken.transfer(msg.sender, amount);

154     emit Unstaked(
155         msg.sender,
156         amount,
157         balanceOf[msg.sender],
158         rewards[msg.sender],
159         totalSupply
160     );
161 }
```

Listing 7: Staking.sol

```
177 function claimRewards()
178     external
179     whenNotPaused
180     updateReward(msg.sender)
181     nonReentrant
182 {
183     // if(balanceOf[msg.sender] <= 0){
184     // revert Staking__NoStakeInPool();
185     // }

187     if (rewards[msg.sender] <= 0) {
188         revert Staking__NoRewardsAvailable();
189     }
190     require(!isPoolClosed, "Too late! Pool has been closed!");

192     uint amount = rewards[msg.sender];
193     rewards[msg.sender] = 0;
194     stakingToken.transfer(msg.sender, amount);

196     emit RewardsClaimed(msg.sender, amount);
197 }
```

Listing 8: Staking.sol

```
200 function closePool() external whenPaused onlyOwner {
201     require(!isPoolClosed, "Pool Already Closed");

203     uint amount = rewardPool.poolBalance();
204     isPoolClosed = true;
205     rewardPool.transfer(owner, amount);

207     emit ClosedPool(msg.sender, amount);
208 }
```

Risk Level:

Likelihood – 1
Impact - 4

Recommendation:

Use the safeTransfer function from the safeERC20 implementation, or put the transfer call inside an assert or require verifying that it returned true.

Status - Fixed

The Monion team has fixed the issue by using the safeTransfer function from the safeERC20 implementation.

A.7 Missing Address Verification [LOW]

Description:

Certain functions lack a safety check in the address, the address-type arguments should include a zero-address test, otherwise, the contract's functionality may become inaccessible. In the constructor, the contract must ensure that the _stakingToken and the _rewardPool arguments are different from the address(0).

Code:

Listing 9: Staking.sol

```
63 constructor(address _stakingToken, address _rewardPool) ReentrancyGuard
       ↪ () {
64     owner = msg.sender;
65     stakingToken = IERC20(_stakingToken);
66     rewardPool = Distributor(_rewardPool);
67     finishAt = block.timestamp + validityPeriod; //Time after which
           ↪ staking is no longer permitted.
68 }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

We recommend that you make sure the addresses provided in the arguments are different from the address(0).

Status - Fixed

The Monion team has fixed the issue by verifying the addresses provided in the arguments to be different from the address(0).

A.8 Floating Pragma [LOW]

Description:

The contract makes use of the floating-point pragma 0.8. Contracts should be deployed using the same compiler version. Locking the pragma helps ensure that contracts will not unintentionally be deployed using another pragma, which in some cases may be an obsolete version, that may introduce issues to the contract system.

Code:

Listing 10: Staking.sol

```
2 pragma solidity ^0.8;
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Consider locking the pragma version. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status - Fixed

The Monion team has fixed the issue by locking the pragma version to 0.8.7.

B Monion.sol

B.1 Approve Race Condition [LOW]

Description:

The standard ERC20 implementation contains a widely known racing condition in its approve function, wherein a spender can witness the token owner broadcast a transaction altering their approval and quickly sign and broadcast a transaction using transferFrom to move the current approved amount from the owner's balance to the spender. If the spender's transaction is validated before the owner's, the spender will be able to get both approval amounts of both transactions.

Code:

Listing 11: Monion.sol

```
7 contract Monion is ERC20 {
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

We recommend using increaseAllowance and decreaseAllowance functions to modify the approval amount instead of using the approve function to modify it.

Status - Fixed

The Monion team has fixed the issue by implementing the use of increaseAllowance and decreaseAllowance functions to modify the approval amount.

B.2 Floating Pragma [LOW]

Description:

The contract makes use of the floating-point pragma 0.8.4. Contracts should be deployed using the same compiler version. Locking the pragma helps ensure that contracts will not unintentionally be deployed using another pragma, which in some cases may be an obsolete version, that may introduce issues to the contract system.

Code:

Listing 12: Monion.sol

```
2 pragma solidity ^0.8.4;
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Consider locking the pragma version. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status - Fixed

The Monion team has fixed the issue by locking the pragma version to 0.8.7.

C Admin.sol

C.1 Missing Address Verification [LOW]

Description:

Certain functions lack a safety check in the address, the address-type arguments should include a zero-address test, otherwise, the contract's functionality may become inaccessible. In the setStakingAddress, the contract must ensure that the _account argument is different from the address(0).

Code:

Listing 13: Admin.sol

```
17 function setStakingAddress(address _account) external {
18     require(msg.sender == owner, "You cannot call this function");
19     staking = _account;
20 }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

We recommend that you make sure the addresses provided in the arguments are different from the address(0).

Status - Fixed

The Monion team has fixed the issue by verifying the addresses provided in the arguments to be different from the address(0).

C.2 Floating Pragma [LOW]

Description:

The contract makes use of the floating-point pragma 0.8.0. Contracts should be deployed using the same compiler version. Locking the pragma helps ensure that contracts will not unintentionally be deployed using another pragma, which in some cases may be an obsolete version, that may introduce issues to the contract system.

Code:

Listing 14: Admin.sol

```
4 pragma solidity ^0.8.0;
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Consider locking the pragma version. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status - Fixed

The Monion team has fixed the issue by locking the pragma version to 0.8.7.

D RewardPool.sol

D.1 Missing Transfer Verification [LOW]

Description:

The ERC20 standard token implementation functions return the transaction status as a boolean. It is a good practice to check for the return status of the function call to ensure that the transaction was executed successfully. It is the developer's responsibility to enclose these function calls with require() to ensure that, when the intended ERC20 function call returns false, the caller transaction also fails.

Code:

Listing 15: RewardPool.sol

```
29 function transfer(address to, uint256 amount) public nonReentrant() {
30     //confirm address
31     //confirm reward balance
32     require(msg.sender == admin.isStakingAddress(), "You cannot call
           ↪ this function");
33     monion.transfer(to, amount);
34 }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

Use the safeTransfer function from the safeERC20 Implementation, or put the transfer call inside an assert or require verifying that it returned true.

Status - Fixed

The Monion team has fixed the issue by using the safeTransfer function from the safeERC20 implementation.

D.2 Floating Pragma [LOW]

Description:

The contract makes use of the floating-point pragma 0.8.0. Contracts should be deployed using the same compiler version. Locking the pragma helps ensure that contracts will not unintentionally be deployed using another pragma, which in some cases may be an obsolete version, that may introduce issues to the contract system.

Code:

Listing 16: RewardPool.sol

```
2 pragma solidity ^0.8.0;
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Consider locking the pragma version. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status - Fixed

The Monion team has fixed the issue by locking the pragma version to 0.8.7.

4 Best Practices

BP.1 Unnecessary Verifications

Description:

There is no need to check that amount > stakingToken.balanceOf(msg.sender) because the transferFrom will revert if this is the case. The same goes for balanceOf[msg.sender] - amount < 0, the overflow protection in the solidity version 0.8 will revert the transaction.

Code:

Listing 17: Staking.sol

```
107 if (amount > stakingToken.balanceOf(msg.sender)) {
108     revert Staking__StakeExceedsYourBalance();
109 }
```

Listing 18: Staking.sol

```
135 if (balanceOf[msg.sender] - amount < 0) {
136     revert Staking__WithdrawLessThanYourBalance();
137 }
```

BP.2 Remove Dead Code

Description:

There is a commented code in the contract, it is recommended to either utilize this code or removing the commented code.

Code:

Listing 19: Staking.sol

```
183 // if(balanceOf[msg.sender] <= 0){
```

```
184 // revert Staking__NoStakeInPool();
185 // }
```

Listing 20: Staking.sol

```
259 // function _timeChecker() internal view returns(bool){
260
261 // }
```

BP.3 Remove The Hardhat Console In Production

Description:

Remove the hardhat console import before deploying the contract in production.

Code:

Listing 21: Staking.sol

```
8 import "hardhat/console.sol";
```

5 Tests

Results:

```
  Setup of Architecture
    Deployed Processes
      should confirm the staking period of 1 year
      should confirm maximum staked tokens of 500000
      should confirm that the size of the pool is 100000
    Test staking operations
      should confirm that the reward pool is funded (49ms)
      the deployer should fund alice, bob and charlie (145ms)
Alice staked at 2022-09-17 11:20:01
      should allow Alice stake at the start of day 3 (92ms)
Bob staked at 2022-10-09 11:20:03
      should allow Bob stake at the start of day 25 (22 days after Alice
        ↪ ) (85ms)
Charlie staked at 2023-03-13 11:20:05
      should allow Charlie stake at the start of day 180 (178 days after
        ↪  Alice) (93ms)
Bob initiated unstaking at 2023-04-03 11:20:07
Bob unstaked 21000 at 2023-04-04 11:20:08
Bob's balance before withdrawal was 4000, balance after withdrawal is
   ↪ 25000
      should allow Bob to unstake by day 201 (90ms)
Pool balance is: BigNumber { value: "100000" }
Rewards balance due to Bob: BigNumber { value: "2909" }
Bob's balance before claiming reward was 25000, balance after claiming
   ↪ is 27909
      should allow Bob to claim rewards from the pool (81ms)
VM Exception while processing transaction: reverted with custom error '
   ↪ Staking__UnbondingIncomplete(86399)'
```

```
      should enforce unbonding time error for Charlie's unstaking at day
         ↪  201 (63ms)
VM Exception while processing transaction: reverted with custom error '
   ↪ Staking__PoolLimitReached(500000, 500001, 29001)'
      should attempt to stake more than the pool limit (51ms)
     Test Pause and Unpause features
       should allow the admin to pause the contract and limit staking
          ↪ features (38ms)
       should allow the admin to pause the contract and limit unstaking
          ↪  features
       should allow the admin to pause the contract and limit claiming
          ↪ features
       should unpause
     Post Validity test
Current time period is 2023-09-16 11:20:20
VM Exception while processing transaction: reverted with custom error '
   ↪ Staking__PoolExceededValidityPeriod()'
       should not allow for staking (43ms)
       should allow for unstaking of tokens (64ms)
Alice's balance before claiming rewards was 3000
Alice's balance after claiming rewards was 3396
       should allow for claiming of rewards (62ms)
      Tests after owner has closed Pool
VM Exception while processing transaction: reverted with reason string '
   ↪ Pausable: paused'
         should NOT allow any withdrawals of rewards


  20 passing (3s)
```

6 Static Analysis (Slither)

Description:

ShellBoxes expanded the coverage of the specific contract areas using automated testing methodologies. Slither, a Solidity static analysis framework, was one of the tools used. Slither was run on all-scoped contracts in both text and binary formats. This tool can be used to test mathematical relationships between Solidity instances statically and variables that allow for the detection of errors or inconsistent usage of the contracts' APIs throughout the entire codebase.

Results:

```
'npx hardhat compile --force' running
Compiled 11 Solidity files successfully
Distributor.transfer(address,uint256) (contracts/RewardPool.sol#29-34)
    ↪ ignores return value by monion.transfer(to,amount) (contracts/
    ↪ RewardPool.sol#33)
StakingRewards.stake(uint256) (contracts/Staking.sol#93-123) ignores
    ↪ return value by stakingToken.transferFrom(msg.sender,address(this
    ↪ ),amount) (contracts/Staking.sol#111)
StakingRewards._unstake(uint256) (contracts/Staking.sol#130-150) ignores
    ↪  return value by stakingToken.transfer(msg.sender,amount) (
    ↪ contracts/Staking.sol#141)
StakingRewards.claimRewards() (contracts/Staking.sol#177-197) ignores
    ↪ return value by stakingToken.transfer(msg.sender,amount) (
    ↪ contracts/Staking.sol#194)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #unchecked-transfer
Distributor (contracts/RewardPool.sol#13-42) has incorrect ERC20
    ↪ function interface:Distributor.transfer(address,uint256) (
    ↪ contracts/RewardPool.sol#29-34)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #incorrect-erc20-interface
```

```
Reentrancy in StakingRewards.initiateUnstake(uint256) (contracts/Staking
    ↪ .sol#154-174):
 External calls:
 - _unstake(amount) (contracts/Staking.sol#166)
  - stakingToken.transfer(msg.sender,amount) (contracts/Staking.sol#141)
 State variables written after the call(s):
 - unstakingFlagPerUser[msg.sender] = false (contracts/Staking.sol#167)
Reentrancy in StakingRewards.stake(uint256) (contracts/Staking.sol
    ↪ #93-123):
 External calls:
 - stakingToken.transferFrom(msg.sender,address(this),amount) (contracts
     ↪ /Staking.sol#111)
 State variables written after the call(s):
 - balanceOf[msg.sender] += amount (contracts/Staking.sol#113)
 - totalSupply += amount (contracts/Staking.sol#114)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #reentrancy-vulnerabilities-1
StakingRewards._unstake(uint256) (contracts/Staking.sol#130-150)
    ↪ contains a tautology or contradiction:
 - balanceOf[msg.sender] - amount < 0 (contracts/Staking.sol#135)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #tautology-or-contradiction
Admin.setStakingAddress(address)._account (contracts/Admin.sol#17) lacks
    ↪  a zero-check on :
  - staking = _account (contracts/Admin.sol#19)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #missing-zero-address-validation
Reentrancy in StakingRewards._unstake(uint256) (contracts/Staking.sol
    ↪ #130-150):
 External calls:
 - stakingToken.transfer(msg.sender,amount) (contracts/Staking.sol#141)
 Event emitted after the call(s):
 - Unstaked(msg.sender,amount,balanceOf[msg.sender],rewards[msg.sender],
     ↪ totalSupply) (contracts/Staking.sol#143-149)
```

```
Reentrancy in StakingRewards.claimRewards() (contracts/Staking.sol
    ↪ #177-197):
 External calls:
 - stakingToken.transfer(msg.sender,amount) (contracts/Staking.sol#194)
 Event emitted after the call(s):
 - RewardsClaimed(msg.sender,amount) (contracts/Staking.sol#196)
Reentrancy in StakingRewards.closePool() (contracts/Staking.sol#200-208)
    ↪ :
 External calls:
 - rewardPool.transfer(owner,amount) (contracts/Staking.sol#205)
 Event emitted after the call(s):
 - ClosedPool(msg.sender,amount) (contracts/Staking.sol#207)
Reentrancy in StakingRewards.stake(uint256) (contracts/Staking.sol
    ↪ #93-123):
 External calls:
 - stakingToken.transferFrom(msg.sender,address(this),amount) (contracts
     ↪ /Staking.sol#111)
 Event emitted after the call(s):
 - Staked(msg.sender,amount,balanceOf[msg.sender],rewards[msg.sender],
     ↪ totalSupply) (contracts/Staking.sol#116-122)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #reentrancy-vulnerabilities-3
StakingRewards.stake(uint256) (contracts/Staking.sol#93-123) uses
    ↪ timestamp for comparisons
 Dangerous comparisons:
 - block.timestamp > finishAt (contracts/Staking.sol#103)
StakingRewards.initiateUnstake(uint256) (contracts/Staking.sol#154-174)
    ↪ uses timestamp for comparisons
 Dangerous comparisons:
 - block.timestamp > finishAt (contracts/Staking.sol#155)
 - block.timestamp > userToUnstakingTime[msg.sender] (contracts/Staking.
     ↪ sol#165)
StakingRewards._min(uint256,uint256) (contracts/Staking.sol#263-265)
    ↪ uses timestamp for comparisons
```

```
 Dangerous comparisons:
 - x <= y (contracts/Staking.sol#264)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #block-timestamp
console._sendLogPayload(bytes) (node_modules/hardhat/console.sol#7-14)
    ↪ uses assembly
 - INLINE ASM (node_modules/hardhat/console.sol#10-13)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #assembly-usage
Different versions of Solidity is used:
 - Version used: ['>=0.4.22<0.9.0', '^0.8', '^0.8.0', '^0.8.4']
 - ^0.8.0 (node_modules/@openzeppelin/contracts/security/Pausable.sol#4)
 - ^0.8.0 (node_modules/@openzeppelin/contracts/security/ReentrancyGuard
    ↪ .sol#4)
 - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4)
 - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol
    ↪ #4)
 - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/
    ↪ IERC20Metadata.sol#4)
 - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4)
 - ^0.8.0 (contracts/Admin.sol#4)
 - ^0.8.4 (contracts/Monion.sol#2)
 - ^0.8.0 (contracts/RewardPool.sol#4)
 - ^0.8 (contracts/Staking.sol#2)
 - >=0.4.22<0.9.0 (node_modules/hardhat/console.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #different-pragma-directives-are-used
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/security/
    ↪ Pausable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/security/
    ↪ ReentrancyGuard.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/
    ↪ ERC20.sol#4) allows old versions
```

```
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/
   ↪ IERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/
   ↪ extensions/IERC20Metadata.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context
   ↪ .sol#4) allows old versions
Pragma version^0.8.0 (contracts/Admin.sol#4) allows old versions
Pragma version^0.8.0 (contracts/RewardPool.sol#4) allows old versions
Pragma version^0.8 (contracts/Staking.sol#2) is too complex
Pragma version>=0.4.22<0.9.0 (node_modules/hardhat/console.sol#2) is too
   ↪  complex
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #incorrect-versions-of-solidity
Parameter Admin.setStakingAddress(address)._account (contracts/Admin.sol
   ↪ #17) is not in mixedCase
Function StakingRewards._calcReward() (contracts/Staking.sol#218-225) is
   ↪  not in mixedCase
Constant StakingRewards.validityPeriod (contracts/Staking.sol#45) is not
   ↪  in UPPER_CASE_WITH_UNDERSCORES
Constant StakingRewards.maximumPoolMonions (contracts/Staking.sol#46) is
   ↪  not in UPPER_CASE_WITH_UNDERSCORES
Constant StakingRewards.totalReward (contracts/Staking.sol#47) is not in
   ↪  UPPER_CASE_WITH_UNDERSCORES
Contract console (node_modules/hardhat/console.sol#4-1532) is not in
   ↪ CapWords
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #conformance-to-solidity-naming-conventions

StakingRewards.slitherConstructorConstantVariables() (contracts/Staking.
   ↪ sol#23-266) uses literals with too many digits:
 - maximumPoolMonions = 5000000 * 1e18 (contracts/Staking.sol#46)
StakingRewards.slitherConstructorConstantVariables() (contracts/Staking.
   ↪ sol#23-266) uses literals with too many digits:
```

```
 - totalReward = 100000 * 1e18 (contracts/Staking.sol#47)
console.slitherConstructorConstantVariables() (node_modules/hardhat/
    ↪ console.sol#4-1532) uses literals with too many digits:
 - CONSOLE_ADDRESS = address(0x000000000000000000636F6e736F6c652e6c6f67)
     ↪  (node_modules/hardhat/console.sol#5)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #too-many-digits

StakingRewards.contractHasExpired (contracts/Staking.sol#53) should be
    ↪ constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #state-variables-that-could-be-declared-constant
name() should be declared external:
 - ERC20.name() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.
     ↪ sol#62-64)
symbol() should be declared external:
 - ERC20.symbol() (node_modules/@openzeppelin/contracts/token/ERC20/
     ↪ ERC20.sol#70-72)
decimals() should be declared external:
 - ERC20.decimals() (node_modules/@openzeppelin/contracts/token/ERC20/
     ↪ ERC20.sol#87-89)
totalSupply() should be declared external:
 - ERC20.totalSupply() (node_modules/@openzeppelin/contracts/token/ERC20
     ↪ /ERC20.sol#94-96)
balanceOf(address) should be declared external:
 - ERC20.balanceOf(address) (node_modules/@openzeppelin/contracts/token/
     ↪ ERC20/ERC20.sol#101-103)
transfer(address,uint256) should be declared external:
 - ERC20.transfer(address,uint256) (node_modules/@openzeppelin/contracts
     ↪ /token/ERC20/ERC20.sol#113-117)
approve(address,uint256) should be declared external:
 - ERC20.approve(address,uint256) (node_modules/@openzeppelin/contracts/
     ↪ token/ERC20/ERC20.sol#136-140)
transferFrom(address,address,uint256) should be declared external:
```

```
 - ERC20.transferFrom(address,address,uint256) (node_modules/
    ↪ @openzeppelin/contracts/token/ERC20/ERC20.sol#158-167)
increaseAllowance(address,uint256) should be declared external:
 - ERC20.increaseAllowance(address,uint256) (node_modules/@openzeppelin/
    ↪ contracts/token/ERC20/ERC20.sol#181-185)
decreaseAllowance(address,uint256) should be declared external:
 - ERC20.decreaseAllowance(address,uint256) (node_modules/@openzeppelin/
    ↪ contracts/token/ERC20/ERC20.sol#201-210)
isStakingAddress() should be declared external:
 - Admin.isStakingAddress() (contracts/Admin.sol#24-26)
transfer(address,uint256) should be declared external:
 - Distributor.transfer(address,uint256) (contracts/RewardPool.sol
    ↪ #29-34)
poolBalance() should be declared external:
 - Distributor.poolBalance() (contracts/RewardPool.sol#38-40)
getCalcRewardVariables() should be declared external:
 - StakingRewards.getCalcRewardVariables() (contracts/Staking.sol
    ↪ #228-241)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #public-function-that-could-be-declared-external
. analyzed (11 contracts with 77 detectors), 53 result(s) found
```

Conclusion:

Most of the vulnerabilities found by the analysis have already been addressed by the smart contract code review.

7 Conclusion

In this audit, we examined the design and implementation of Monion Staking Contracts contract and discovered several issues of varying severity. Monion Global team addressed 12 issues raised in the initial report and implemented the necessary fixes, while classifying the rest as a risk with low-probability of occurrence. Shellboxes' auditors advised Monion Global Team to maintain a high level of vigilance and to keep those findings in mind in order to avoid any future complications.

For a Contract Audit, contact us at contact@shellboxes.com