Feb 18, 2024

ZAN

Security Assessment ConstantStaking

—

Professional Service

Table of Contents

1. Overview

1.1. Executive Summary

ConstantStaking is a staking protocol for ERC20 tokens. This report has been prepared for ConstantStaking project to discover issues and vulnerabilities in the source code of this project as well as any contract dependencies that were not part of an officially recognized library.

Conducted by Static Analysis, Formal Verification and Manual Review, **we did not find any security vulnerabilities and only 7 informational issues were identified**.

The project team acknowledged the 7 informational issues and decided to keep no change.

1.2. Project Summary

Project Name	ConstantStaking
Platform	Polygon
Language	Solidity
Codebase	Final Audit: • https://github.com/dexteam09/contrast/tree/5cf24c02f55113b6e8009323881375f874b0ff35

1.3. Assessment Summary

Delivery Date	Feb 18, 2024
Audit Methodology	Static Analysis, Formal Verification, Manual Review

1.4. Assessment Scope

ID	File	File Hash
1	/ConstantStaking.sol	5487c4fa0bd22a54233dee5a9607af01

2. Checklist

2.1. Code Security

Reentrancy	DelegateCall	Integer Overflow
Input Validation	Unchecked this.call	Frozen Money
Arbitrary External Call	Unchecked Owner Transfer	Do-while Continue
Right-To-Left-Override Character	Unauthenticated Storage Access	Risk For Weak Randomness
TxOrigin	Missing Checks for Return Values	Diamond Inheritance
ThisBalance	VarType Deduction	Array Length Manipulation
Uninitialized Variable	Shadow Variable	Divide Before Multiply
Affected by Compiler Bug		

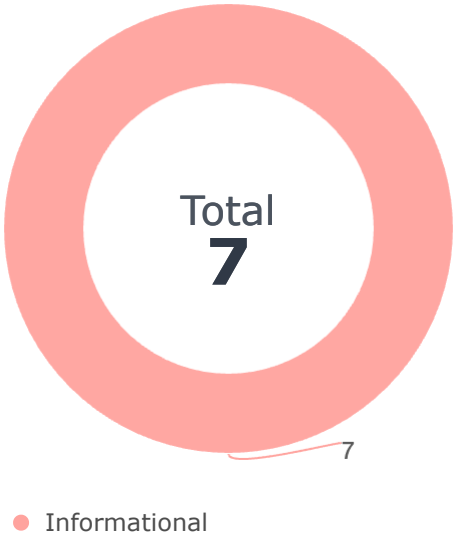
2.2. Optimization Suggestion

Compiler Version	Improper State Variable Modification
Function Visibility	Deprecated Function
Externally Controlled Variables	Code Style
Constant Specific	Event Specific
Return Value Unspecified	Inexistent Error Message
State Variable Defined Without Storage Location	Import Issue
Compare With Timestamp/Block Number/Blockhash	Constructor in Base Contract Not Implemented
Delete Struct Containing the Mapping Type	Usage of '=+'
Paths in the Modifier Not End with "_" or Revert	Non-payable Public Functions Use msg.value
Lack of SafeMath	Compiler Error/Warning
Tautology Issue	Loop Depends on Array Length
Redundant/Duplicated/Dead Code	Code Complexity/Code Inefficiency
Undeclared Resource	Optimizable Return Statement
Unused Resource	

2.3. Business Security

The Code Implementation is Consistent With Comments, Project White Papers and Other Materials
Permission Check
Address Check

3. Findings



Code Security

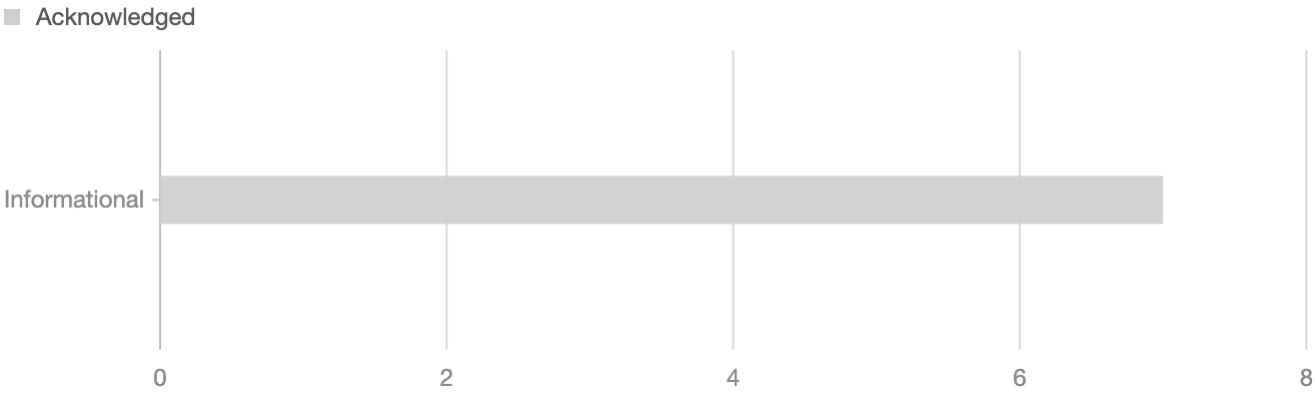
no vulnerabilities.

Optimization Suggestion

Informational: 7

Business Security

no vulnerabilities.



ID	Title	Category	Severity	Status
I-01	Function Visibility Can Be External	Optimization Suggestion	Informational	Acknowledged
I-02	Floating Pragma	Optimization Suggestion	Informational	Acknowledged
I-03	Use CustomError Instead of String	Optimization Suggestion	Informational	Acknowledged
I-04	No Check of Address Params with Zero Address	Optimization Suggestion	Informational	Acknowledged
I-05	Use ++i/--i Instead of i++/i--	Optimization Suggestion	Informational	Acknowledged
I-06	Use storage Instead of memory for Struct or Array	Optimization Suggestion	Informational	Acknowledged
I-07	Unused Interface	Optimization Suggestion	Informational	Acknowledged

I-01: Function Visibility Can Be External

Informational: Optimization Suggestion

File Location: /ConstantStaking.sol:244,250

Description

Functions that are not called should be declared as external.

/ConstantStaking.sol

```
244   function rewards(address sender) public view returns (uint256, uint256) {
245       Claim memory claimInfo = claims[sender];
246       uint256 reward = calculateRewards(sender);
```

/ConstantStaking.sol

```
250   function stakes(address sender) public view returns (uint256) {
251       Claim memory claimInfo = claims[sender];
252       uint256 all = calculateMTO(sender) + claimInfo.amount;
```

Recommendation

Functions that are not called in the contract should be declared as external.

Alleviation

Acknowledged. The ConstantStaking team decided to keep no change.

I-02: Floating Pragma

Informational: Optimization Suggestion

File Location: /ConstantStaking.sol:2

Description

Contracts should be deployed with fixed compiler version which has been tested thoroughly or make sure to lock the contract compiler version in the project configuration. Locked compiler version ensures that contracts will not be compiled by untested compiler version.

/ConstantStaking.sol

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.0;
3
4  import '@openzeppelin/contracts/token/ERC20/IERC20.sol';
```

Recommendation

Use a fixed compiler version, and consider whether the bugs in the selected compiler version (https://github.com/ethereum/solidity/releases) will affect the contract.

Alleviation

Acknowledged. The ConstantStaking team decided to keep no change.

I-03: Use CustomError Instead of String

Informational: Optimization Suggestion

File Location: /ConstantStaking.sol:21,35,106,125,126,127,184,197,212,215,218,229,230,231,232

Description

When using `require` or `revert`, CustomError is more gas efficient than string description, as the error message described using CustomError is only compiled into four bytes. Especially when string exceeds 32 bytes, more gas will be consumed. Generally, around 250-270 gas can be saved for one CustomError replacement when compiler optimization is turned off, 60-80 gas can be saved even if compiler optimization is turned on.

/ConstantStaking.sol

```
19      (bool success, bytes memory data) =
20                          token.call(abi.encodeWithSelector(IERC20.
                            transferFrom.selector, from, to, value));
21      require(success && (data.length == 0 || abi.decode(data, (bool))),
        'STF');
22  }
23
```

/ConstantStaking.sol

```
33  ) internal {
34      (bool success, bytes memory data) = token.call(abi.encodeWithSelector
        (IERC20.transfer.selector, to, value));
35      require(success && (data.length == 0 || abi.decode(data, (bool))),
        'ST');
36  }
37
```

/ConstantStaking.sol

```
104     */
105 function _checkOwner() internal view virtual {
106     require(owner() == _msgSender(), "Ownable: caller is not the owner");
107 }
108
```

/ConstantStaking.sol

```
123     */
124 function transferOwnership(address newOwner) public virtual onlyOwner {
125     require(
126         newOwner != address(0),
127         "Ownable: new owner is the zero address"
```

/ConstantStaking.sol

```
183  function setAPY(uint256 _apy) external onlyOwner {
184      require(_apy <= MAX_APY, "ConstantStaking: out of range");
185      apy = _apy;
186  }
```

/ConstantStaking.sol

```
196  function setClaimInterval(uint256 _ts) external onlyOwner {
197      require(_ts <= CLAIM_INTERVAL_MAX, "Claim interval cannot exceed 365
     days");
198      claim_interval = _ts;
199  }
```

/ConstantStaking.sol

```
210  function applyClaim() external {
211      Claim storage claimInfo = claims[msg.sender];
212      require(claimInfo.amount == 0, "ConstantStaking: have already
     applied");
213
214      uint256 allMTO = calculateMTO(msg.sender);
```

/ConstantStaking.sol

```
214  uint256 allMTO = calculateMTO(msg.sender);
215  require(allMTO > 0, "ConstantStaking: no staking");
216  // MTO.safeTransfer(msg.sender, allMTO);
217  uint256 allRewards = calculateRewards(msg.sender);
```

/ConstantStaking.sol

```
216  // MTO.safeTransfer(msg.sender, allMTO);
217  uint256 allRewards = calculateRewards(msg.sender);
218  require(allRewards > 0, "ConstantStaking: no rewards");
219
220  claimInfo.amount = allMTO;
```

/ConstantStaking.sol

```
227  function claim() external {
228      Claim memory claimInfo = claims[msg.sender];
229      require(claimInfo.amount > 0, "ConstantStaking: no apply");
230      require(
231          block.timestamp >= claimInfo.startTs,
232          "ConstantStaking: claim too early"
```

Recommendation

Use CustomError instead of string for `require` or `revert` description.

Alleviation

Acknowledged. The ConstantStaking team decided to keep no change.

I-04: No Check of Address Params with Zero Address

Informational: Optimization Suggestion

File Location: /ConstantStaking.sol:177,188,192

Description

The input parameter of the address type in the function does not use the zero address for verification.

/ConstantStaking.sol

```
177   constructor(address _mto, address _gmto) {
178       MTO = _mto;
179       GMTO = _gmto;
```

/ConstantStaking.sol

```
188   function setMTO(address _mto) external onlyOwner {
189       MTO = _mto;
190   }
```

/ConstantStaking.sol

```
192   function setGMTO(address _gmto) external onlyOwner {
193       GMTO = _gmto;
194   }
```

Recommendation

It is recommended to perform zero address verification on the input parameters of the address type.

Alleviation

Acknowledged. The ConstantStaking team decided to keep no change.

I-05: Use ++i/--i Instead of i++/i--

Informational: Optimization Suggestion

File Location: /ConstantStaking.sol:259,269

Description

Compared with `i++`, `++i` can save about 5 gas per use. Compared with `i--`, `--i` can save about 3 gas per use in for loop.

/ConstantStaking.sol

```
257   Order[] memory orders = stakings[sender];
258   uint256 all = 0;
259   for (uint i = 0; i < orders.length; i++) {
260       Order memory order = orders[i];
261       all += order.amount;
```

/ConstantStaking.sol

```
267   Order[] memory orders = stakings[sender];
268   uint256 all = 0;
269   for (uint i = 0; i < orders.length; i++) {
270       Order memory order = orders[i];
271       uint256 ts = block.timestamp - order.createdTs;
```

Recommendation

It is recommended to use `++i`/`--i` instead of `i++`/`i--` in for loop.

Alleviation

Acknowledged. The ConstantStaking team decided to keep no change.

I-06: Use storage Instead of memory for Struct or Array

Informational: Optimization Suggestion

File Location: /ConstantStaking.sol:245,251,257,267

Description

When reading a state variable of data type struct or array, declaring a storage variable to receive the struct or array is more gas-efficient than declaring a memory variable. This is because assigning data in storage to a memory variable would cause all fields or elements of the struct or array to be read from storage, even if not all fields or elements are used. Additionally, reading fields from the new memory variable would incur additional `mload` operations, thereby increasing gas consumption. For structs, declaring a storage variable to receive the state variable can save approximately 3000 gas. For an array with a length of 1000, declaring a storage variable to receive the state variable can save approximately 2 million gas.

/ConstantStaking.sol

```
244  function rewards(address sender) public view returns (uint256, uint256) {
245      Claim memory claimInfo = claims[sender];
246      uint256 reward = calculateRewards(sender);
247      return (claimInfo.reward, reward);
```

/ConstantStaking.sol

```
250  function stakes(address sender) public view returns (uint256) {
251      Claim memory claimInfo = claims[sender];
252      uint256 all = calculateMTO(sender) + claimInfo.amount;
253      return all;
```

/ConstantStaking.sol

```
256  function calculateMTO(address sender) internal view returns (uint256) {
257      Order[] memory orders = stakings[sender];
258      uint256 all = 0;
259      for (uint i = 0; i < orders.length; i++) {
```

/ConstantStaking.sol

```
266  function calculateRewards(address sender) internal view returns
     (uint256) {
267      Order[] memory orders = stakings[sender];
268      uint256 all = 0;
269      for (uint i = 0; i < orders.length; i++) {
```

Recommendation

It is recommended to declare a storage variable instead of a memory variable when reading a state variable of data type struct or array.

Alleviation

Acknowledged. The ConstantStaking team decided to keep no change.

I-07: Unused Interface

Informational: Optimization Suggestion

File Location: /ConstantStaking.sol:143

Description

The interface `IGMTO` is not used in contract `ConstantStaking`.

/ConstantStaking.sol

```
143  interface IGMTO {
144      function mint(address to, uint256 amount) external;
145  }
```

Recommendation

It is recommended to remove the interface `IGMTO` for code convention.

Alleviation

Acknowledged. The ConstantStaking team decided to keep no change.

4. Disclaimer

No description, statement, recommendation or conclusion in this report shall be construed as endorsement, affirmation or confirmation of the project. The security assessment is limited to the scope of work as stipulated in the Statement of Work.

This report is prepared in response to source code, and based on the attacks and vulnerabilities in the source code that already existed or occurred before the date of this report, excluding any new attacks or vulnerabilities that exist or occur after the date of this report. The security assessment are solely based on the documents and materials provided by the customer, and the customer represents and warrants documents and materials are true, accurate and complete.

CONSULTANT DOES NOT MAKE AND HEREBY DISCLAIMS ANY REPRESENTATIONS OR WARRANTIES OF ANY KIND, WHETHER EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, REGARDING THE SERVICES, DELIVERABLES, OR ANY OTHER MATTER PERTAINING TO THIS REPORT.

CONSULTANT SHALL NOT BE RESPONSIBLE FOR AND HEREBY DISCLAIMS MERCHANTABILITY, FITNESS FOR PURPOSE, TITLE, NON-INFRINGEMENT OR NON-APPROPRIATION OF INTELLECTUAL PROPERTY RIGHTS OF A THIRD PARTY, SATISFACTORY QUALITY, ACCURACY, QUALITY, COMPLETENESS, TIMELINESS, RESPONSIVENESS, OR PRODUCTIVITY OF THE SERVICES OR DELIVERABLES.

CONSULTANT EXCLUDES ANY WARRANTY THAT THE SERVICES AND DELIVERABLES WILL BE UNINTERRUPTED, ERROR FREE, FREE OF SECURITY DEFECTS OR HARMFUL COMPONENTS, REVEAL ALL SECURITY VULNERABILITIES, OR THAT ANY DATA WILL NOT BE LOST OR CORRUPTED.

CONSULTANT SHALL NOT BE RESPONSIBLE FOR (A) ANY REPRESENTATIONS MADE BY ANY PERSON REGARDING THE SUFFICIENCY OR SUITABILITY OF SERVICES AND DELIVERABLES IN ANY ACTUAL APPLICATION, OR (B) WHETHER ANY SUCH USE WOULD VIOLATE OR INFRINGE THE APPLICABLE LAWS, OR (C) REVIEWING THE CUSTOMER MATERIALS FOR ACCURACY.

5. Appendix

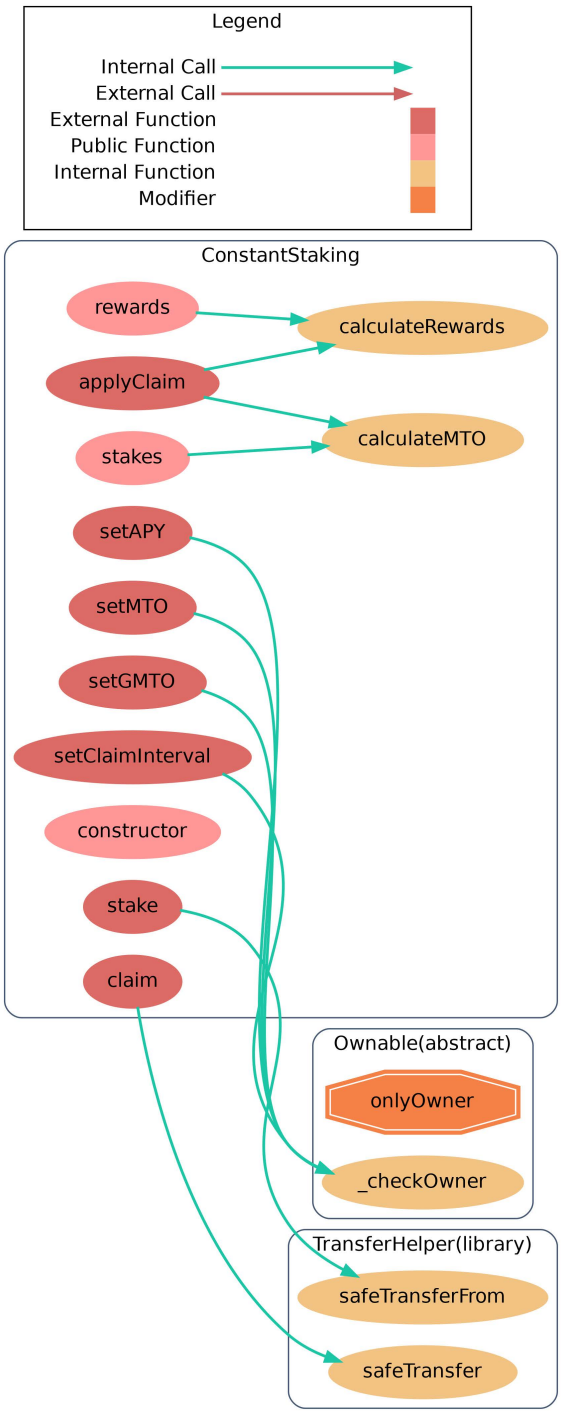
5.1 Visibility

Contract	FuncName	Visibility	Mutability	Modifiers
ConstantStaking	_CTOR_	public	Y	
ConstantStaking	setAPY	external	Y	onlyOwner
ConstantStaking	setMTO	external	Y	onlyOwner
ConstantStaking	setGMTO	external	Y	onlyOwner
ConstantStaking	setClaimInterval	external	Y	onlyOwner
ConstantStaking	stake	external	Y	
ConstantStaking	applyClaim	external	Y	
ConstantStaking	claim	external	Y	
ConstantStaking	rewards	public	N	
ConstantStaking	stakes	public	N	
ConstantStaking	calculateMTO	internal	N	
ConstantStaking	calculateRewards	internal	N	

5. Appendix

5.2 Call Graph

ConstantStaking



5. Appendix

5.3 Inheritance Graph

ConstantStaking