Competitive Security Assessment

IoTeX_DAO_Management

Mar 27th, 2024

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

• Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.

• Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.

• Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.

• Verify the code base is compliant with the most up-to-date industry standards and security best practices.

• Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

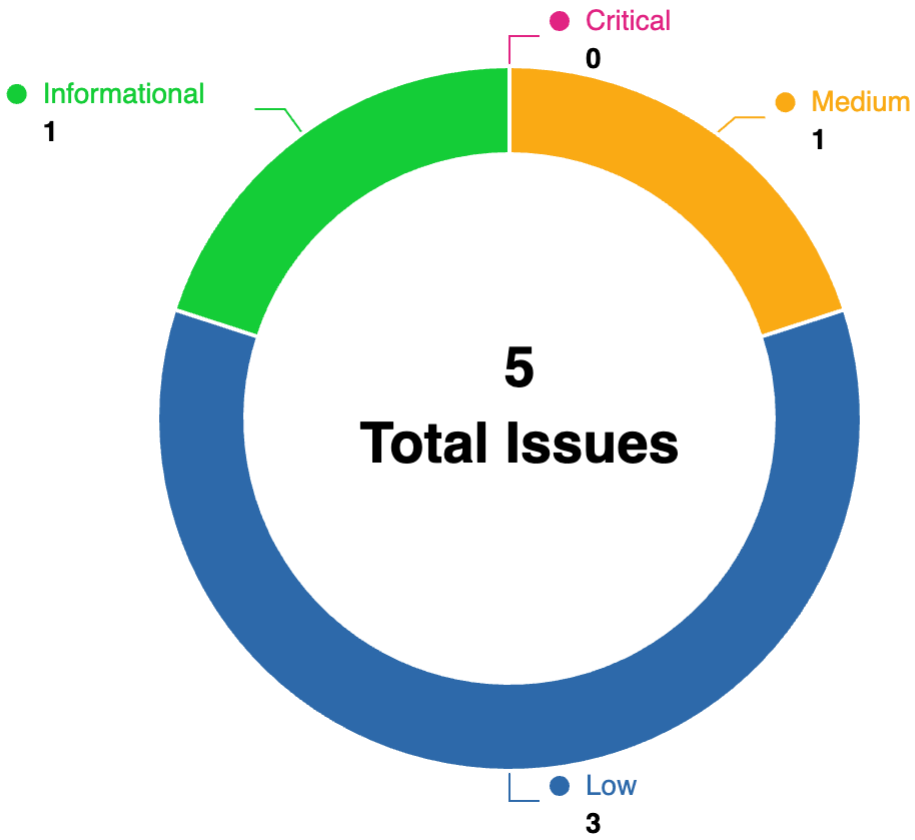
Overview

Project Name	IoTeX_DAO_Management
Language	Solidity
Codebase	• https://github.com/iotexproject/marshall-dao/ • audit version - 0e62a240235414c67e115dd3f05f367d7073c1e6 • final version - c807ea2bfbfcd0b2748afa0bd94fdce0369d7a99
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Audit Scope

File	SHA256 Hash
./contracts/Vault.sol	53f087ffb7ea24b16ebb6c06b3c816e2e3a04906a259 37758af38ea144fe3980
./contracts/RewardsDistributor.sol	79d32e456bebc32615e1d5a01e5fec1399a29fb5da45 094044e5721b8981a6a7
./contracts/Voter.sol	7bee5e79a1257e5a63860c43c38530aca4ab32e1e97 2ddce12a931ef51752ae6
./contracts/VotingEscrow.sol	ed22b8f2b69e37a2d5dc9580473037908aa402820a3 e5085e4061f15745631eb
./contracts/gauges/Gauge.sol	8a8909c240e913198cafba2c062649658fc267fd581a 70e81cb63851ac287bf7
./contracts/rewards/BribeVotingReward.sol	b8889be367923364510ed478f950017f966e3053c48 0e9fdb8879567acf461cc

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
IOT-1	Inproper calculation mechanism for vault upadates.	Logical	Medium	Fixed	0xffchain
IOT-2	Vault.donate(): `_token` is controlled by user, could be malicious	Logical	Low	Acknowledged	ret2basic
IOT-3	Use call not transfer	Language Specific	Low	Fixed	n16h7m4r3, 0xffchain
IOT-4	Lack of notification for important state changes.	Logical	Low	Fixed	0xffchain
IOT-5	Use `calldata` instead of `memory` for function parameters	Gas Optimization	Informational	Mitigated	n16h7m4r3

IOT-1:Inproper calculation mechanism for vault upadates.

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	0xffchain

Code Reference

- code/contracts/Vault.sol#L60-L81

```
60: function updatePeriod() external returns (uint256 _period) {
61:     _period = activePeriod;
62:     if (block.timestamp >= _period + WEEK) {
63:       epochCount++;
64:       _period = (block.timestamp / WEEK) * WEEK;
65:       activePeriod = _period;
66:       uint256 _emission = weekly;
67:
68:       uint256 _balanceOf = address(this).balance;
69:       if (_balanceOf < _emission) {
70:         revert InsufficientFund();
71:       }
72:       uint256 _veEmission = (_emission * veRate) / 10000;
73:
74:       payable(address(rewardsDistributor)).transfer(_veEmission);
75:       rewardsDistributor.checkpointToken(); // checkpoint token balance that was just minted in r
ewards distributor
76:
77:       voter.notifyRewardAmount{value: _emission - _veEmission}();
78:
79:       emit Emission(msg.sender, _emission);
80:     }
81:   }
```

Description

0xffchain: The calculation of the vault weekly emission is dependent on that a person calls the function once every-week unfailingly, and if for any reason the function is not called, the reward holders forfeit their reward for the week. This is not meant to be the case, as a decentralized application, it is meant to decrease the trust dependence of each actor. Suppose for a reason the function is. not called, the emision is meant to cover for the current week and weeks past.

Recommendation

0xffchain: emission should cover for all periods past from the last emission period till the current, the calculation should be like so:

```
    CurrentPeriod = (block.timestamp / WEEK) * WEEK;
    elapsed =(currentPeriod - _period) / week.
    _emission = emission * elapsed
```

Client Response

0xffchain: Fixed - Fixed in 6033864

IOT-2:Vault.donate(): `_token` is controlled by user, could be malicious

Category	Severity	Client Response	Contributor
Logical	Low	Acknowledged	ret2basic

Code Reference

- code/contracts/Vault.sol#L123-L126

```
123: function donate(address _token, uint256 _amount) external {
124:     IERC20(_token).safeTransferFrom(msg.sender, address(this), _amount);
125:     emit Donation(msg.sender, _token, _amount);
126:   }
```

Description

ret2basic: There is no restriction on the first parameter **`address _token`** for the function Vault.donate(), therefore user can deploy a malicious contract and use that address here:

```
function donate(address _token, uint256 _amount) external {
    IERC20(_token).safeTransferFrom(msg.sender, address(this), _amount);
    emit Donation(msg.sender, _token, _amount);
}
```

The event **`Donation`** will still be emitted but the amount doesn't reflect the actual amount that the protocol receives. If there is any third party app monitoring that event, it could be tricked to think the user donated a lot of tokens but in fact user donated nothing.

Recommendation

ret2basic: Use a whitelist to restrict **`_token`**.

Client Response

ret2basic: Acknowledged - Does not care malicious token, because this is only for known tokens.

IOT-3:Use call not transfer

Category	Severity	Client Response	Contributor
Language Specific	Low	Fixed	n16h7m4r3, 0xffchain

Code Reference

- code/contracts/gauges/Gauge.sol#L79

```
79: payable(_account).transfer(reward);
```

- code/contracts/RewardsDistributor.sol#L148
- code/contracts/RewardsDistributor.sol#L178

```
148: payable(_owner).transfer(amount);
```

```
178: payable(_owner).transfer(amount);
```

- code/contracts/Vault.sol#L101-L110

```
101: function withdraw(address _token, address payable _recipcient, uint256 _amount) external {
102:     if (msg.sender != governor) revert NotGovernor();
103:
104:     if (_token == address(0)) {
105:       _recipcient.transfer(_amount);
106:     } else {
107:       IERC20(_token).safeTransfer(_recipcient, _amount);
108:     }
109:     emit Withdraw(msg.sender, _token, _recipcient, _amount);
110:   }
```

Description

n16h7m4r3: The `transfer()` function only allows the recipient to use 2300 gas. If the recipient uses more than that, transfers will fail. In the future gas costs might change increasing the likelihood of that happening.
Keep in mind that `call()` introduces the risk of reentrancy. But, as long as the router follows the checks effects interactions pattern it should be fine. It's not supposed to hold any tokens anyway.
0xffchain: When sending ETH, use call() instead of transfer(). The transfer() function only allows the recipient to use 2300 gas and sload opcode already cost 800 gas. If the recipient needs more than that, transfers will fail. In the future gas costs might change increasing the likelihood of that happening. If this happens it means the user can not withdraw its claim causing a possible DOS for the user for that day and thus loosing out on its claim. And if the receiving account is a proxy contract, it might not recieve it correctly.

Recommendation

n16h7m4r3: Replace transfer() calls with call(). Keep in mind to check whether the call was successful by validating the return value:

```
(bool success, ) = payable(_account).call{value: reward}("");
require(success, "Transfer failed.")
```

0xffchain: Change the method used to call and not transfer and add a reentrancy guard if need be, but keeping in mind that it is an admin function, meaning that admin already has full access so there is no need for any special protection as the admin already has the needed powers.

Client Response

n16h7m4r3: Fixed - fixed in c807ea2
0xffchain: Fixed - fixed in c807ea2

IOT-4:Lack of notification for important state changes.

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	0xffchain

Code Reference

- code/contracts/Vault.sol#L53-L57

```
53: function setGovernor(address _governor) public {
54:     if (msg.sender != governor) revert NotGovernor();
55:     if (_governor == address(0)) revert ZeroAddress();
56:     governor = _governor;
57:   }
```

Description

0xffchain: An event notification should be sent out for all important state change in the system, this is more important as the state changed here is the change of the most important actor in the system, which is the governor, and keep in mind that the governor has the most privilege in the codebase. such change of a powerful user should not be done silently, it should send a notification so all interested/vested party are aware of the change.

Recommendation

0xffchain: Send a notification for all important state change.

Client Response

0xffchain: Fixed - Fixed in 6033864

IOT-5:Use `calldata` instead of `memory` for function parameters

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Mitigated	n16h7m4r3

Code Reference

- code/contracts/rewards/BribeVotingReward.sol#L12

```
12: address[] memory _rewards
```

- code/contracts/Voter.sol#L190

```
190: function _vote(uint256 _tokenId, uint256 _weight, address[] memory _poolVote, uint256[] memory
_weights) internal {
```

- code/contracts/VotingEscrow.sol#L76

```
76: function initialize(address[] memory _tokens) public initializer {
```

Description

n16h7m4r3: By default, Solidity loads function arguments passed to a function into memory. This means that when you pass an argument to a function, it is automatically loaded into memory, even if you don't intend to modify it. Loading unnecessary data into memory can result in unnecessary gas costs and reduced efficiency.
Using **`calldata`** instead of **`memory`** can help mitigate this issue. When you mark a data type as **`calldata`**, Solidity avoids automatically loading the data into memory. This optimization is particularly useful for function arguments that do not need to be changed within the function.

Recommendation

n16h7m4r3: Using the **`calldata`** keyword instead of **`memory`** for function arguments that do not need to be changed within the function, you can avoid unnecessary loading of data into memory and optimize gas usage.

Client Response

n16h7m4r3: Mitigated - 1. code/contracts/rewards/BribeVotingReward.sol#L12 can't change to **`calldata`**, because it's constructor parameter
2. code/contracts/VotingEscrow.sol#L76 changed, 1029fd9
3. code/contracts/Voter.sol#L190 can't change to **`calldata`**, because it's internal method parameter

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3's prior written consent in each instance.

This report is not an "endorsement" or "disapproval" of any particular project or team. This report is not an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3's position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.