14th March 2019

metadium

SMART CONTRACT AUDIT REPORT

-

version 3.0

DApp Smart Contract Security Audit and General Analysis

HAECHI LABS

Table of Contents

15 Issues (3 Critical, 4 Major, 8 Minor) Found

01. Introduction

This report was written to provide a security audit for the governance-contract smart contract, designed by Metadium. HAECHI LABS conducted the audit focusing on whether Metadium's smart contract is designed and implemented in accordance with publicly released information and whether it has any security vulnerabilities.

The code used for the audit can be found at "METADIUM/governance-contract" Github storage(https://github.com/METADIUM/governance-contract). The last commit used for the audit was "5ab324b29786677aafca21c751249b472fa7b8b0".

02. Summary

The Metadium team implemented a Governance Smart contract with the following features:

- Member management
- Voting
- Distribution of Rewards

HAECHI LABS found one critical issue, another major Issue, and 8 Minor Issues; we also included 2 tips that could help improve the code's usability and efficiency.

Critical issues are security vulnerabilities that MUST be addressed in order to prevent widespread and massive damage. Major issues contain security vulnerabilities or have faulty implementation issues and need to be fixed. Minor issues are some potential risks that require some degree of modification. HAECHI LABS advises addressing all the issues found in this report.

Updated

Severity	Issue	Status
CRITICAL	Weighted value of vote becomes 0 when *lockAmount* is set to 0 by *Gov#init()*.[Wrong Argument]	(Found - v.1.0) (Resolved - v.2.0)
MAJOR	*EnvStorageImp#setGasPriceByBytes()* and *EnvStorageImp#setMaxIdleBlockIntervalByBytes()* change the Staking Max Value.*[Unintended Behavior]*	(Found - v.1.0) (Resolved - v.3.0)
MINOR	Incorrect *DecisionTypes* values might come in from *BallotStorage#createVote()*. [Unintended Behavior]	(Found - v.1.0) (Resolved - v.2.0)
MINOR	Contract should Revert when trying to add a *ZERO encode*. [Wrong Argument]	(Found - v.1.0) (Resolved - v.2.0)

MINOR	Contract should Revert when trying to withdraw *0 Ether*. [Wrong Argument]	(Found - v.1.0) (Resolved - v.2.0)
MINOR	Contract should Revert when trying to ***lock*** *0 Ether*. [Wrong Argument]	(Found - v.1.0) (Resolved - v.3.0)
MINOR	Contract should Revert when trying to ***unlock*** *0 Ether*. [Wrong Argument]	(Found - v.1.0) (Resolved - v.3.0)
MINOR	Contract should Revert when *unlock*ed *ether* is 0. [Wrong Argument]	(Found - v.1.0) (Resolved - v.3.0)
MINOR	In *BallotStorage#_areVariableBallotParamValid()*, parameter *_envVariableName* cannot be 0 in length. [Wrong Requirement]	(Found - v.1.0) (Resolved - v.2.0)
MINOR	Contract should Revert if the *EnvStorage* constructors *_registry* and *_implementation* have the same address. [Unintended Behavior]	(Found - v.1.0) (Resolved - v.2.0)
TIPS	Contract should Revert if a parameter's *registry* is set to *Zero Address*.	(Found - v.1.0) (Resolved - v.2.0)
TIPS	Resetting *Storage variable*	(Found - v.1.0) (Resolved - v.2.0)

19.03.21 [v.2.0] - One Critical issue, 5 Minor issues were modified and one Major issue newly found at the new commit, 1e5e190e519be02d308083c10b65a1f502449dab.

19.04.01 [v.3.0] - One Major issue and three Minor issues were modified at the new commit, eb3fa6f0d4b31c684be24bbccfd5a9a47cd859f3.

03. Contracts subject to audit

- abstract
    - BallotEnums.sol
    - EnvConstants.sol
- interface
    - IBallotStorage.sol
    - IEnvStorage.sol
    - IGov.sol
    - IRegistry.sol
    - IStaking.sol
- proxy
    - OwnedUpgradeabilityProxy.sol
    - Proxy.sol
    - UpgradeabilityProxy.sol
- storage
    - AEnvStorage.sol
    - BallotStorage.sol
    - EnvStorage.sol
    - EnvStorageImp.sol
    - EternalStorage.sol
- Gov.sol
- GovChecker.sol
- GovImp.sol
- Registry.sol
- Staking.sol

04. About HAECHI AUDITS

HAECHI AUDITS is a blockchain-specialized code security auditing service by HAECHI LABS. HAECHI LABS is a leading tech company within the blockchain industry based on its self-developed blockchain technology solutions and R&D capacity.

HAECHI AUDITS' client list includes: major companies like Shinhan Bank, LG, SK Telecom and Kakao's blockchain subsidiary (Ground X); and global cryptocurrency exchange institutes such as Bit-Z, Coinall (OKEx), KuCoin, Liquid, CPDAX, and Huobi Korea. Furthermore, we won the Ethereum Foundation Grant and were selected by Samsung Electronics' startup incubation program (C-lab).

It is HAECHI AUDITS' mission to help clients develop secure smart contracts by providing the most trustworthy security auditing services.

To request audit, please email audit@haechi.io.

Contact : audit@haechi.io
Website : https://haechi.io

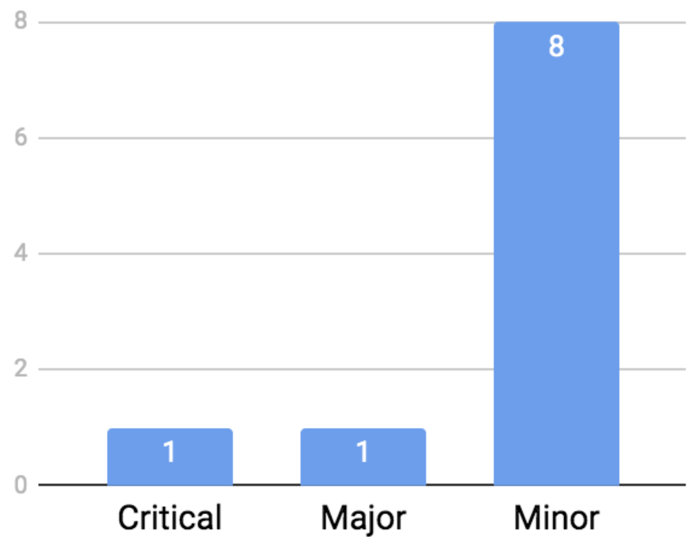
05. Issues Found

The issues found are classified as CRITICAL, MAJOR, MINOR, or TIPS according to their severity.

CRITICAL	Critical issues are security vulnerabilities that MUST be addressed in order to prevent widespread and massive damage.
MAJOR	Major issues contain security vulnerabilities or have faulty implementation issues and need to be fixed.
MINOR	Minor issues are some potential risks that require some degree of modification.
TIPS	Tips issues can make your code more usable and efficient.

"HAECHI AUDITS" recommends Metadium team to resolve all the issues found.

The following explanations of each issue will use a {File name}#{Line number},{Contract name}#{Function/Variable name} format to refer to specific codes. For example, *Sample.sol:20* refers to the 20th line of the Sample.sol file, and *Sample#fallback()* refers to the fallback() function of the Sample contract.



[그림 1] Issue Stats

CRITICAL : Weighted value of vote becomes 0 when *lockAmount* is set to 0 by *Gov#init()*.[Wrong Argument] (Found - v.1.0) (Resolved - v.2.0)

CRITICAL

```
    function init(
        address registry,
        address implementation,
        uint256 lockAmount,
        bytes enode,
        bytes ip,
        uint port
    )
        public onlyOwner
    {
        require(_initialized == false, "Already initialized");

        setRegistry(registry);
        setImplementation(implementation);

        // Lock
        IStaking staking = IStaking(getStakingAddress());
        require(staking.availableBalanceOf(msg.sender) >= lockAmount, "Insufficient
        staking.lock(msg.sender, lockAmount);

        // Add voting member
        memberLength = 1;
        members[memberLength] = msg.sender;
        memberIdx[msg.sender] = memberLength;

        // Add reward member
        rewards[memberLength] = msg.sender;
        rewardIdx[msg.sender] = memberLength;

        // Add node
        nodeLength = 1;
        Node storage node = nodes[nodeLength];
        node.enode = enode;
        node.ip = ip;
        node.port = port;
        nodeIdxFromMember[msg.sender] = nodeLength;
        nodeToMember[nodeLength] = msg.sender;

        _initialized = true;
    }
```

(Gov.sol -

https://github.com/METADIUM/governance-contract/blob/5ab324b29786677aafca21c751249b472fa7b8b0/contracts/Gov.sol#L78)

Problem Statement

lockAmount should be set to a value above zero upon initial registration, but *Gov#init()* does not check for this condition. Also, the value of *Staking#availableBalanceOf()* is seen as greater than or equal to 0 - even when there is no transaction using *Staking#deposit()*, allowing the input to pass this *require* statement.

If *lockAmount* becomes 0, this later causes the value of *Staking#calcVotingWeightWithScaleFactor()* to also become 0 - making votes on Gov suggestions have no weighted value.

Recommendation

- Check if *lockAmount* input is 0.
- If *Staking#availableBalanceOf()* is 0 - in other words, if there is no deposit - the contract should Revert back to its previous state.

Updated

[v.2.0] - A *require* statement has been added to related functions to resolve the issue.

MAJOR : *EnvStorageImp#setGasPriceByBytes()* and *EnvStorageImp#setMaxIdleBlockIntervalByBytes()* change the Staking Max Value. [Unintended Behavior] (Found - v.1.0) (Resolved - v.3.0)

MAJOR

```
    function setGasPriceByBytes(bytes _value) public onlyGov {
        setStakingMax(toUint(_value));
    }

    function setMaxIdleBlockIntervalByBytes(bytes _value) public onlyGov {
        setStakingMax(toUint(_value));
    }
```

(Staking.sol - https://github.com/METADIUM/governance-contract/blob/1e5e190e519be02d308083c10b65a1f502449dab/contracts/storage/EnvStorageImp.sol#L134-L140)

Problem Statement

EnvStorageImp#setGasPriceByBytes() and *EnvStorageImp#setMaxIdleBlockIntervalByBytes()*, instead of changing the intended value, change the Staking Max value.

Recommendation

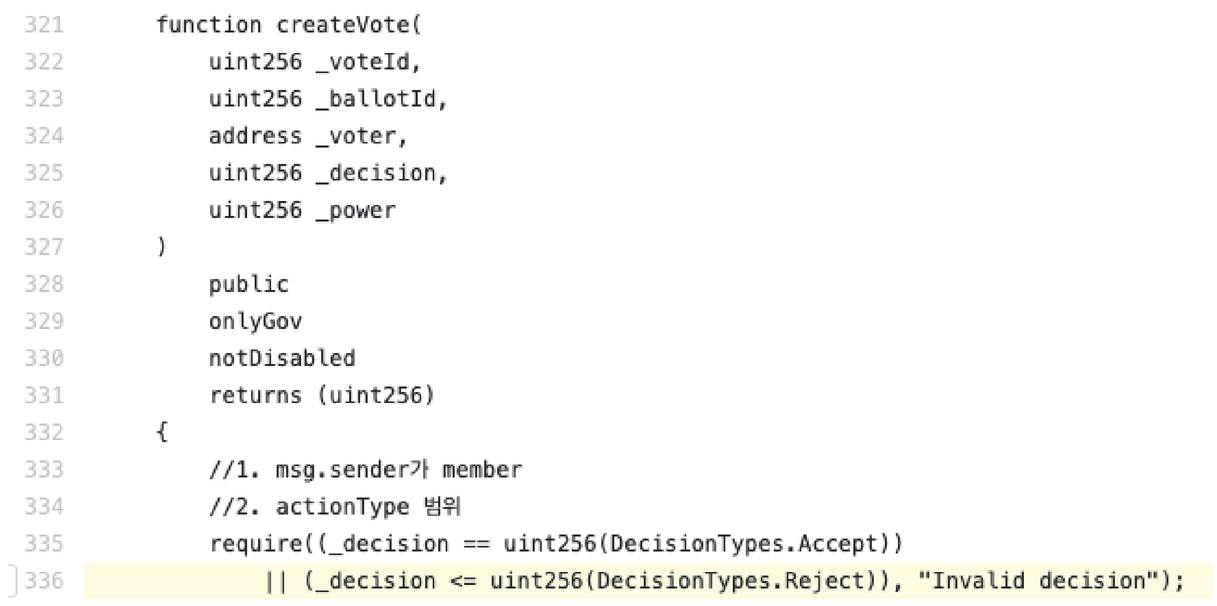
Use each function's *EnvStorageImp#setGasPrice()* and *EnvStorageImp#setMaxIdleBlockInterval()* to change values.

Updated

[v.3.0] - Corresponding statements have been changed to appropriate functions.

MINOR : Incorrect *DecisionTypes* values might come in from *BallotStorage#createVote()*. [Unintended Behavior] (Found - v.1.0) (Resolved - v.2.0)

MINOR



```
321     function createVote(
322         uint256 _voteId,
323         uint256 _ballotId,
324         address _voter,
325         uint256 _decision,
326         uint256 _power
327     )
328         public
329         onlyGov
330         notDisabled
331         returns (uint256)
332     {
333         //1. msg.sender가 member
334         //2. actionType 범위
335         require((_decision == uint256(DecisionTypes.Accept))
336             || (_decision <= uint256(DecisionTypes.Reject)), "Invalid decision");
```

(BallotStorage.sol - https://github.com/METADIUM/governance-contract/blob/5ab324b29786677aafca21c751249b472fa7b8b0/contracts/storage/BallotStorage.sol#L336)

Problem Statement

The *_decision* value, a parameter of *BallotStorage#createVote()*, is only valid when it is either *DecisionTypes.Accept* or *DecisionTypes.Reject*. *DecisionTypes* is an enum value, mapped to *Invalid:0, Accept:1, Reject:2*. Under current implementation, *BallotStorage.sol:336* considers the *DecisionTypes.Invalid* as also valid. Fortunately, within the function, *BallotStorage.sol#_updateBallotForVote()* nullifies such a value, so there is no problem with actual use.

Recommendation

We recommend changing *decision <= uint256(DecisionTypes.Reject)* to *decision == uint256(DecisionTypes.Reject).*

Updated

[v.2.0] - The issue has been resolved by altering the *_decision* type comparison statements.

MINOR : Contract should Revert when trying to add a *ZERO encode*. [Wrong Argument] (Found - v.1.0) (Resolved - v.2.0)

MINOR

```
    function addProposalToAddMember(
        address member,
        bytes enode,
        bytes ip,
        uint port,
        uint256 lockAmount,
        bytes memo
    )
        external
        onlyGovMem
        returns (uint256 ballotIdx)
    {
        require(msg.sender != member, "Cannot add self");
        require(!isMember(member), "Already member");

        ballotIdx = ballotLength.add(1);
        createBallotForMemeber(
            ballotIdx, // ballot id
            uint256(BallotTypes.MemberAdd), // ballot type
            msg.sender, // creator
            address(0), // old member address
            member, // new member address
            enode, // new enode
            ip, // new ip
            port // new port
        );
```

(GovImp.sol -
https://github.com/METADIUM/governance-contract/blob/5ab324b29786677aafca21c751249b472fa7b8b0/contracts
/GovImp.sol#L42)

Problem Statement

GovImp#addProposalToAddMember() functions properly even if its parameter, the *encode* value, is 0, due to *GovImp.sol:42*. This causes a member to be added, but the member cannot be accessed by *encode*.

Recommendation

We recommend adding a logic to ensure *encode* is not 0 using *require(encode != 0)* before running *GovImp#addProposalToAddMember()*.

Updated

[v.2.0] - Related functions have included a *require* statement to resolve the issue.

MINOR : Contract should Revert when trying to withdraw *0 Ether*. [Wrong Argument] (Found - v.1.0) (Resolved - v.2.0)

MINOR

```
    function withdraw(uint256 amount) external nonReentrant {
        require(amount <= availableBalanceOf(msg.sender), "Withdraw amount should be equal or 

        _balance[msg.sender] = _balance[msg.sender].sub(amount);
        msg.sender.transfer(amount);

        emit Unstaked(msg.sender, amount, _balance[msg.sender], availableBalanceOf(msg.sender)
    }
```

(Staking.sol -
https://github.com/METADIUM/governance-contract/blob/5ab324b29786677aafca21c751249b472fa7b8b0/contracts
/Staking.sol#L45)

Problem Statement

Staking#withdraw() functions properly even if *amount* is set to 0. This creates unnecessary gas costs for a completely meaningless operation.

Recommendation

We recommend using *require(amount > 0)* to check that the input value is not 0, and if so, making the transaction a failure.

Updated

[v.2.0] - Related functions have included a *require* statement to resolve the issue.

MINOR : Contract should Revert when trying to *lock* *0 Ether*. [Wrong Argument] (Found - v.1.0) (Resolved - v.3.0)

MINOR

```
    /**
    * @dev Lock fund
    * @param payee The address whose funds will be locked.
    * @param lockAmount The amount of funds will be locked.
    */
    function lock(address payee, uint256 lockAmount) external onlyGov {
        require(_balance[payee] >= lockAmount, "Lock amount should be equal or less than balanc
        require(availableBalanceOf(payee) >= lockAmount, "Insufficient balance that can be lock

        _lockedBalance[payee] = _lockedBalance[payee].add(lockAmount);
        _totalLockedBalance = _totalLockedBalance.add(lockAmount);

        emit Locked(payee, lockAmount, _balance[payee], availableBalanceOf(payee));
    }
```

(Staking.sol - https://github.com/METADIUM/governance-contract/blob/5ab324b29786677aafca21c751249b472fa7b8b0/contracts/Staking.sol#L62)

Problem Statement

In *Staking#lock()*, *lock* functions properly even if *lockAmount* is set to 0. This creates unnecessary gas costs for a completely meaningless operation.

Recommendation

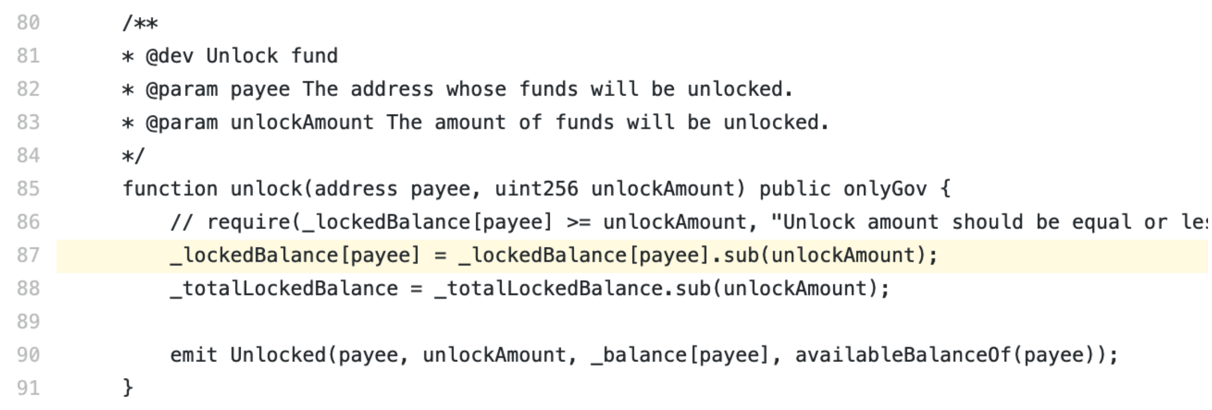
We recommend using *require(lockAmount > 0)* to check that the input value is not 0, and if so, making the transaction a failure.

Updated

[v.3.0] - The issue has been resolved using conditional statements.

MINOR : Contract should Revert when trying to *unlock* *0 Ether*. [Wrong Argument] (Found - v.1.0) (Resolved - v.3.0)

MINOR



```
80      /**
81      * @dev Unlock fund
82      * @param payee The address whose funds will be unlocked.
83      * @param unlockAmount The amount of funds will be unlocked.
84      */
85      function unlock(address payee, uint256 unlockAmount) public onlyGov {
86          // require(_lockedBalance[payee] >= unlockAmount, "Unlock amount should be equal or les
87          _lockedBalance[payee] = _lockedBalance[payee].sub(unlockAmount);
88          _totalLockedBalance = _totalLockedBalance.sub(unlockAmount);
89
90          emit Unlocked(payee, unlockAmount, _balance[payee], availableBalanceOf(payee));
91      }
```

(Staking.sol - https://github.com/METADIUM/governance-contract/blob/5ab324b29786677aafca21c751249b472fa7b8b0/contracts/Staking.sol#L87)

Problem Statement

In *Staking#unlock()*, *unlock* functions properly even if *unlockAmount* is set to 0. This creates unnecessary gas costs for a completely meaningless operation.

Recommendation

We recommend using *require(unlockAmount > 0)* to check that the input value is not 0, and if so, making the transaction a failure.

Updated

[v.3.0] - The issue has been resolved using conditional statements.

MINOR : Contract should Revert when *unlock*ed *ether* is 0. [Wrong Argument] (Found - v.1.0) (Resolved - v.3.0)

MINOR

```
/**
* @dev Transfer locked funds to governance
* @param from The address whose funds will be transfered.
* @param amount The amount of funds will be transfered.
*/
function transferLocked(address from, uint256 amount) external onlyGov {
    unlock(from, amount);
    _balance[from] = _balance[from].sub(amount);
    address rewardPool = getRewardPoolAddress();
    _balance[rewardPool] = _balance[rewardPool].add(amount);
}
```

(Staking.sol - https://github.com/METADIUM/governance-contract/blob/5ab324b29786677aafca21c751249b472fa7b8b0/contracts/Staking.sol#L75)

Problem Statement

In *Staking#transferLocked*, *transferLocked* functions properly even if *amount* is set to 0. This creates unnecessary gas costs for a completely meaningless operation.

Recommendation

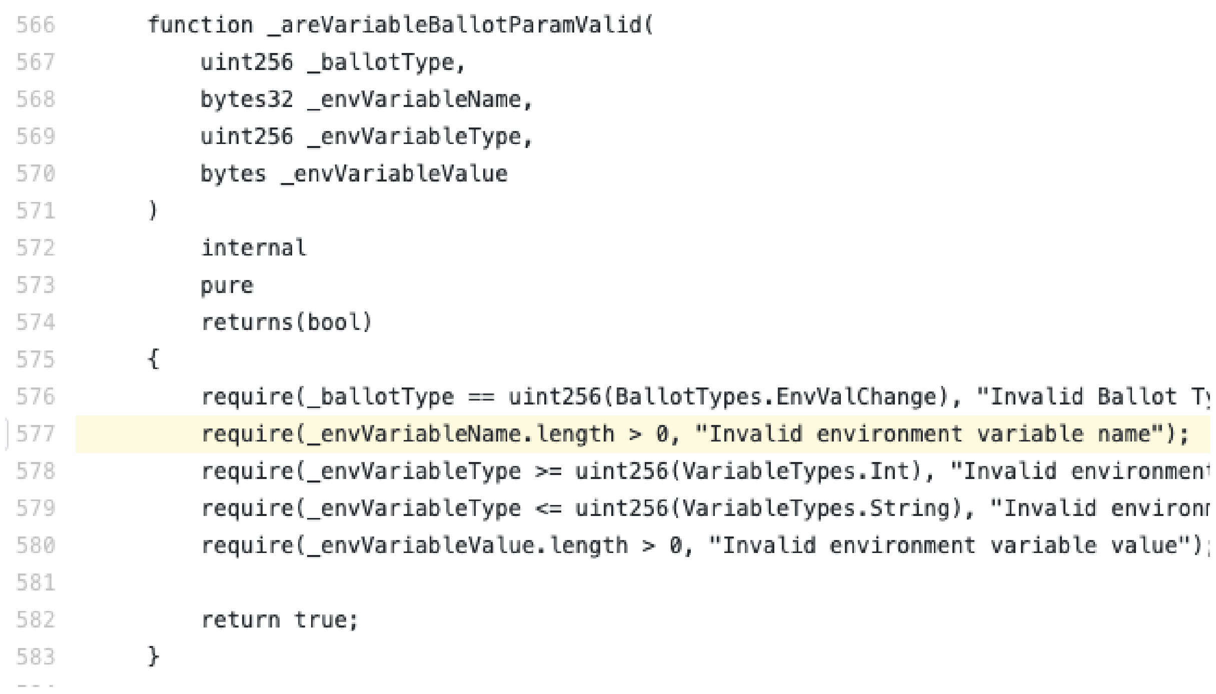
We recommend using *require(amount > 0)* to check that the input value is not 0, and if so, making the transaction a failure.

Updated

[v.3.0] - The issue has been resolved using conditional statements.

MINOR : In *BallotStorage#_areVariableBallotParamValid()*, parameter *_envVariableName* cannot be 0 in length. [Wrong Requirement] (Found - v.1.0) (Resolved - v.2.0)

MINOR



```
566     function _areVariableBallotParamValid(
567         uint256 _ballotType,
568         bytes32 _envVariableName,
569         uint256 _envVariableType,
570         bytes _envVariableValue
571     )
572         internal
573         pure
574         returns(bool)
575     {
576         require(_ballotType == uint256(BallotTypes.EnvValChange), "Invalid Ballot Ty
577         require(_envVariableName.length > 0, "Invalid environment variable name");
578         require(_envVariableType >= uint256(VariableTypes.Int), "Invalid environment
579         require(_envVariableType <= uint256(VariableTypes.String), "Invalid environm
580         require(_envVariableValue.length > 0, "Invalid environment variable value");
581
582         return true;
583     }
```

(BallotStorage.sol - https://github.com/METADIUM/governance-contract/blob/5ab324b29786677aafca21c751249b472fa7b8b0/contracts/storage/BallotStorage.sol#L577)

Problem Statement

The function *BallotStorage#_areVariableBallotParamValid()*'s parameter *_envVariableName* is *bytes32*, and therefore always considered of 32 in length. Therefore, it always passes the *require* statement.

Recommendation

We recommend using *require(_envVariableName > 0)* to ensure the input value is not 0.

Updated

[v.2.0] - Related functions have included a *require* statement to resolve the issue.

MINOR : Contract should Revert if the *EnvStorage* constructors *_registry* and *_implementation* have the same address. [Unintended Behavior] (Found - v.1.0) (Resolved - v.2.0)

MINOR

```
contract EnvStorage is UpgradeabilityProxy, AEnvStorage {

    constructor(address _registry, address _implementation) public {
        setRegistry(_registry);
        setImplementation(_implementation);
    }
}
```

(EnvStorage.sol - https://github.com/METADIUM/governance-contract/blob/5ab324b29786677aafca21c751249b472fa7b8b0/contracts/storage/EnvStorage.sol#L10-L11)

Problem Statement

In *EnvStorage#constructor()*, *EnvStorage* is created even when *_registry* and *_implementation* have the same address; however, *_registry* and *_implementation* cannot have the same address.

Recommendation

Add *require(_registry != _implementation)* to ensure that inputs from *_registry* and *_implementation* are different from each other.

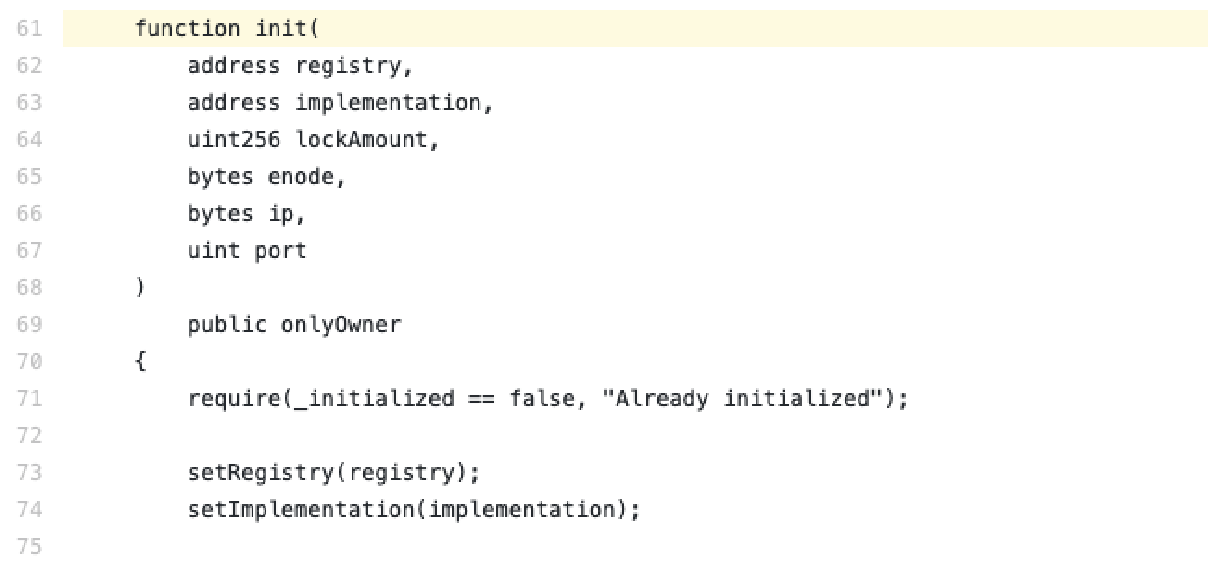
Updated

[v.2.0] - Related functions have included a *require* statement to resolve the issue.

06. Tips

TIPS : Contract should Revert if a parameter’s *registry* is set to *Zero Address*. (Found - v.1.0) (Resolved - v.2.0)

TIPS



```
    function init(
        address registry,
        address implementation,
        uint256 lockAmount,
        bytes enode,
        bytes ip,
        uint port
    )
        public onlyOwner
    {
        require(_initialized == false, "Already initialized");

        setRegistry(registry);
        setImplementation(implementation);

```

(Gov.sol - https://github.com/METADIUM/governance-contract/blob/5ab324b29786677aafca21c751249b472fa7b8b0/contracts/Gov.sol#L61)

Problem Statement

When *registry* is set to *Zero Address*, all functions using *registry* **(all functions using *GovChecker# getContractAddress()*)** become unusable.

The following functions must check if incoming parameters have a *Zero Address*.

- Gov#init()
- GovChecker#setRegistry()
- EnvStorage#constructor()

Recommendation

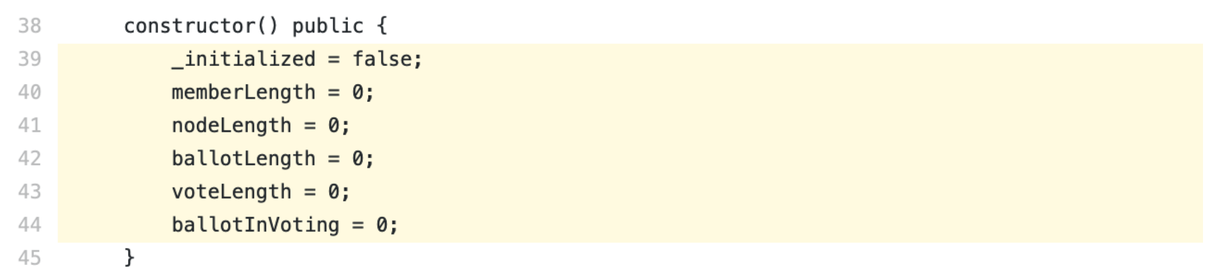
Add a logic to check if a parameter is *Zero Address* before utilizing the parameter.

Updated

[v.2.0] - Related functions have included a *require* statement to resolve the issue.

TIPS : Reset *Storage variable* (Found - v.1.0) (Resolved - v.2.0)

TIPS



```
    constructor() public {
        _initialized = false;
        memberLength = 0;
        nodeLength = 0;
        ballotLength = 0;
        voteLength = 0;
        ballotInVoting = 0;
    }
```

(Gov.sol- https://github.com/METADIUM/governance-contract/blob/5ab324b29786677aafca21c751249b472fa7b8b0/contracts/Gov.sol#L39-L44)

Problem Statement

The initial value of member variables in *Solidity* is 0. If *constructor()* resets these values to 0, it creates unnecessary gas costs.

- Gov#constructor()
- Staking#constructor()

Recommendation

Have *constructor()* receive specific values it wants to reset, or delete the resetting code.

Updated

[v.2.0] - Resetting codes were deleted from related functions.

07. Disclaimer

This report is not an advice on investment, nor does it guarantee adequacy of a business model and/or a bug-free code. This report should be used only to discuss known technical problems. The code may include problems on Ethereum and/or Solidity that are not included in this report. It will be necessary to resolve addressed issues and conduct thorough tests to ensure the safety of the smart contract.