Competitive Security Assessment

MagmaStablecoin

Dec 27th, 2023

Secure3

secure3.io

Summary 3

Overview 4

Audit Scope 5

Code Assessment Findings 7

MSC-1:Missing storage gap in upgradable contract `OwnableUpgradeable` 9

MSC-2:Events without `emit` prefix is deprecated and will result in a compilation error 13

MSC-3:Use `disableInitializers` to prevent any future reinitialization 17

MSC-4:Use OpenZeppelin's `ECDSA` instead of built-in `ecrecover` 21

MSC-5:Ownership change should use two-step process 23

MSC-6:Redundant code 26

MSC-7:Missing event 27

MSC-8:Incorrect token name 29

MSC-9:Missing Zero Address Check in contracts/Timelock.sol 30

Disclaimer 31

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 Detail

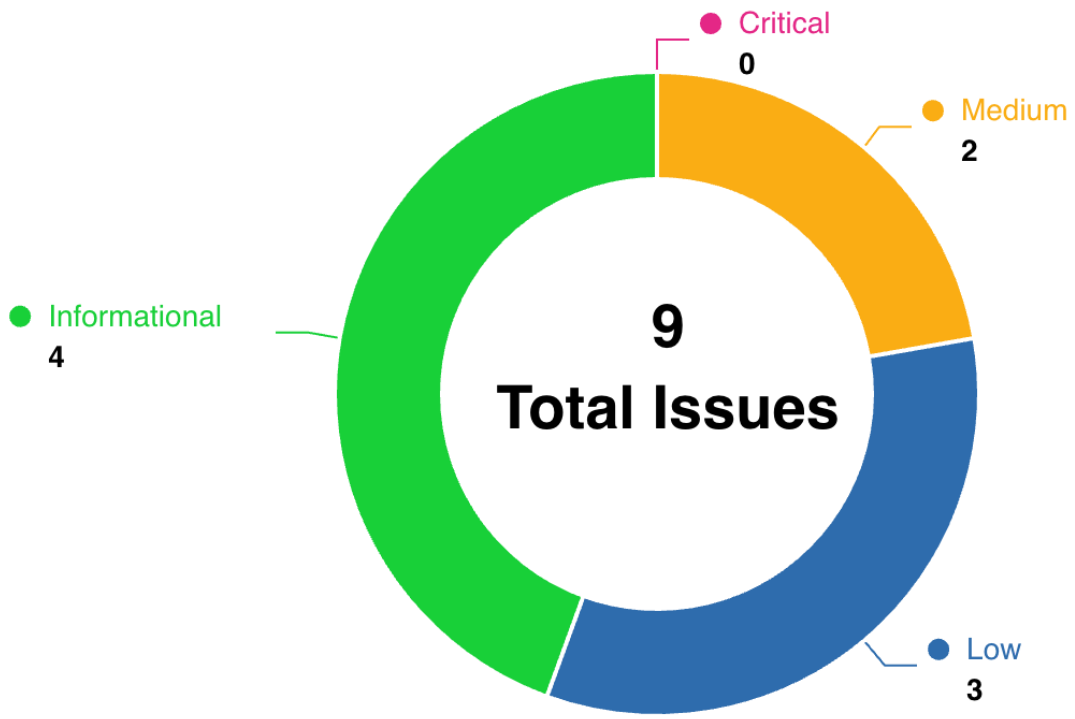
Project Name	MagmaStablecoin
Platform & Language	Solidity
Codebase	• https://github.com/magma-fi/WEN-Contracts • audit commit - 37e92613994f2cfc90d1390c373f9a95c132bf8c • final commit - 670a50b1d2201a4cb13f98a9c149221bb59c46aa
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Audit Scope

File	SHA256 Hash
contracts/TroveManager.sol	0c01cce19234e1808401790f46e192928b0ef594f634fb26ebc60d6f36e919fa
contracts/StabilityPool.sol	1bd35bd37700322f1ea6b557e4e025da005d0a25d4246fb28602f6113cecb4bc
contracts/BorrowerOperations.sol	77d65377e740cd0e6323294c235d22d4bc031191182315d63b0d9b3c725c716a
contracts/LUSDToken.sol	072398487fd1189af438f7de655c86dbb15842bfdb3aa4187cdf9c97bf74dd44
contracts/SortedTroves.sol	5768e793434cd36d25ce4861083b317581d389a0cda7d91b32f1931971941120
contracts/HintHelpers.sol	46a75f509804d57232b830eb646ac911850bb4bda75995ffe8aeca83aac20dff
contracts/Interfaces/ITroveManager.sol	0716317cab913c534d6358083b6288b8cea50e4f730c6b481e2aea21241532a9
contracts/ActivePool.sol	3fa2d8abc1875daf8e635e5f2dd5b3058ffa8266c64952367bcc47641de6ac83
contracts/CollSurplusPool.sol	d5e54677bf87432407151b55973be4d8fa78e18c3a1247f2b269add0a9fd5e03
contracts/Timelock.sol	9b90ca69055d571172d86260e2e79f7468a883c6cf35a98dc7be1a0dd06d70a8
contracts/DefaultPool.sol	6a56c5b8e3d1ef2451e133519598b9d8b4ff0b44a6d0cd022795d968cbfb9bb3
contracts/Dependencies/LiquityBase.sol	a9a995f470e74b2c16061311c98a5a1ae35643deae29639b10c54b92e7da51b1
contracts/Interfaces/IBorrowerOperations.sol	911f9309f667ea90b28f8082bf0d5f3b484fc060fd47fc5099f48a5cc2110ee7
contracts/Dependencies/OwnableUpgradeable.sol	fe6f69bb935934e3d07fb18fe08eb6b0e3c6eb55adf62b56af96d45f9066349f
contracts/Dependencies/CheckContract.sol	72ff6d26f2ff986a722bda9c6de65df235d654339ac926926bd24da9515f3b1b

contracts/Dependencies/MagmaUpgradeableProxy.sol	**a51ca12cc51a2eedd3f181033f48c9b6d24ceaf86eaff302024d85db2b03c7f2**
contracts/Dependencies/MagmaProxyAdmin.sol	**7607cfc091176e4fc32ea670729b96f892cdd906ac21c660893c30ec2fdef9ec**

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
MSC-1	Missing storage gap in upgradable contract `OwnableUpgradeable`	Write to Arbitrary Storage Location	Medium	Fixed	biakia, Yaodao
MSC-2	Events without `emit` prefix is deprecated and will result in a compilation error	Code Style	Medium	Fixed	biakia, yekong, Yaodao

MSC-3	Use `disableInitializers` to prevent any future reinitialization	Code Style	Low	Fixed	biakia, slowfrog, 0xxm, Xi_Zi
MSC-4	Use OpenZeppelin's `ECDSA` instead of built-in `ecrecover`	Signature Forgery or Replay	Low	Acknowledged	biakia
MSC-5	Ownership change should use two-step process	Privilege Related	Low	Fixed	biakia
MSC-6	Redundant code	Code Style	Informational	Fixed	Hupixiong3, Xi_Zi
MSC-7	Missing event	Code Style	Informational	Fixed	Xi_Zi, yekong
MSC-8	Incorrect token name	Code Style	Informational	Fixed	Yaodao
MSC-9	Missing Zero Address Check in contracts/Timelock.sol	Code Style	Informational	Fixed	Xi_Zi

MSC-1:Missing storage gap in upgradable contract `OwnableUpgradeable`

Category	Severity	Client Response	Contributor
Write to Arbitrary Storage Location	**Medium**	**Fixed**	**biakia, Yaodao**

Code Reference

- code/contracts/BorrowerOperations.sol#L16
- code/contracts/TroveManager.sol#L16
- code/contracts/Dependencies/OwnableUpgradeable.sol#L19-L76
- code/contracts/ActivePool.sol#L19
- code/contracts/StabilityPool.sol#L149

```
16:contract BorrowerOperations is LiquityBase, OwnableUpgradeable, CheckContract, Initializable, IBo
rrowerOperations {

16:contract TroveManager is LiquityBase, OwnableUpgradeable, CheckContract, ITroveManager, Initializ
able {

19:contract OwnableUpgradeable {
20:     address private _owner;
21:
22:     event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
23:
24:     function __Ownable_init() internal {
25:         address msgSender = msg.sender;
26:         _owner = msgSender;
27:         emit OwnershipTransferred(address(0), msgSender);
28:     }
29:
30:     /**
31:      * @dev Returns the address of the current owner.
32:      */
33:     function owner() public view returns (address) {
34:         return _owner;
35:     }
36:
37:     /**
38:      * @dev Throws if called by any account other than the owner.
39:      */
40:     modifier onlyOwner() {
41:         require(isOwner(), "Ownable: caller is not the owner");
42:         _;
43:     }
44:
45:     /**
46:      * @dev Returns true if the caller is the current owner.
47:      */
48:     function isOwner() public view returns (bool) {
49:         return msg.sender == _owner;
50:     }
51:
52:     /**
53:      * @dev Leaves the contract without owner. It will not be possible to call
```

```
54:     * `onlyOwner` functions anymore.
55:     *
56:     * NOTE: Renouncing ownership will leave the contract without an owner,
57:     * thereby removing any functionality that is only available to the owner.
58:     *
59:     * NOTE: This function is not safe, as it doesn’t check owner is calling it.
60:     * Make sure you check it before calling it.
61:     */
62:    function _renounceOwnership() internal {
63:        emit OwnershipTransferred(_owner, address(0));
64:        _owner = address(0);
65:    }
66:
67:    /**
68:     * @dev Transfers ownership of the contract to a new account (`newOwner`).
69:     * Can only be called by the current owner.
70:     */
71:    function transferOwnership(address newOwner) public virtual onlyOwner {
72:        require(newOwner != address(0), "Ownable: new owner is the zero address");
73:        emit OwnershipTransferred(_owner, newOwner);
74:        _owner = newOwner;
75:    }
76:}

19:contract ActivePool is OwnableUpgradeable, CheckContract, IActivePool, Initializable {

149:contract StabilityPool is LiquityBase, OwnableUpgradeable, CheckContract, IStabilityPool, Initia
lizable {
```

Description

biakia : The contract `OwnableUpgradeable` is an upgradeable contract. The best practice for an upgradeable contract is to use storage gap to allow future versions of that contract to use up those slots without affecting the storage layout of child contracts. Please refer to the guide in openzeppelin's doc:https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#storage-gaps

Yaodao : These contracts extend the contract `OwnableUpgradeable`, which is an upgradeable contract. It is recommended to use storage gaps as a convention for reserving storage slots in a base contract to allow future versions of that contract to use up those slots without affecting the storage layout of child contracts.

Reference link: https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#storage-gaps

Recommendation

biakia : Consider adding a storage gap in `OwnableUpgradeable`:

```
contract OwnableUpgradeable {
    address private _owner;
     uint256[49] __gap;
     ...
     ...
}
```

Yaodao : Recommend using storage gaps in the contract.

Client Response

Fixed

MSC-2:Events without `emit` prefix is deprecated and will result in a compilation error

Category	Severity	Client Response	Contributor
Code Style	Medium	Fixed	biakia, yekong, Yaodao

Code Reference

- code/contracts/ActivePool.sol#L95-L105
- code/contracts/ActivePool.sol#L98
- code/contracts/ActivePool.sol#L104

```
95:function increaseLUSDDebt(uint _amount) external override {
96:        _requireCallerIsBOorTroveM();
97:        LUSDDebt  = LUSDDebt.add(_amount);
98:        ActivePoolLUSDDebtUpdated(LUSDDebt);
99:    }
100:
101:    function decreaseLUSDDebt(uint _amount) external override {
102:        _requireCallerIsBOorTroveMorSP();
103:        LUSDDebt = LUSDDebt.sub(_amount);
104:        ActivePoolLUSDDebtUpdated(LUSDDebt);
105:    }

98:ActivePoolLUSDDebtUpdated(LUSDDebt);

104:ActivePoolLUSDDebtUpdated(LUSDDebt);
```

Description

biakia : In contract `ActivePool`, the following code will result in a compilation error due to the missing `emit` keyword:

```
function increaseLUSDDebt(uint _amount) external override {
        _requireCallerIsBOorTroveM();
        LUSDDebt  = LUSDDebt.add(_amount);
        ActivePoolLUSDDebtUpdated(LUSDDebt);
    }

    function decreaseLUSDDebt(uint _amount) external override {
        _requireCallerIsBOorTroveMorSP();
        LUSDDebt = LUSDDebt.sub(_amount);
        ActivePoolLUSDDebtUpdated(LUSDDebt);
    }
```

You can try this case in remix:

```
pragma solidity 0.6.11;

abstract contract EventTest{

    event ActivePoolLUSDDebtUpdated(uint _LUSDDebt);

    function testEvent () public {
        ActivePoolLUSDDebtUpdated(100);
    }
}
```

The error shows here:

```
contracts/EventTest.sol:8:9: TypeError: Event invocations have to be prefixed by "emit".
        ActivePoolLUSDDebtUpdated(100);
        ^-----------------------------^
```

yekong : The coding style is inconsistent in handling events. In `ActivePool.increaseLUSDDebt()` and `ActivePool.decreaseLUSDDebt()` functions, they handling events without `emit`. In other contracts, `emit` is used to record events.

```
    function increaseLUSDDebt(uint _amount) external override {
        _requireCallerIsBOorTroveM();
        LUSDDebt  = LUSDDebt.add(_amount);
        ActivePoolLUSDDebtUpdated(LUSDDebt);
    }

    function decreaseLUSDDebt(uint _amount) external override {
        _requireCallerIsBOorTroveMorSP();
        LUSDDebt = LUSDDebt.sub(_amount);
        ActivePoolLUSDDebtUpdated(LUSDDebt);
    }
```

Yaodao : The contract uses pragma solidity 0.6.11 as the compiled version. The keyword `emit` can't be omitted when declaring an event for the solidity 0.6.11.

As a result, the following codes will cause compilation errors.

```
    function increaseLUSDDebt(uint _amount) external override {
        _requireCallerIsBOorTroveM();
        LUSDDebt  = LUSDDebt.add(_amount);
        ActivePoolLUSDDebtUpdated(LUSDDebt);
    }

    function decreaseLUSDDebt(uint _amount) external override {
        _requireCallerIsBOorTroveMorSP();
        LUSDDebt = LUSDDebt.sub(_amount);
        ActivePoolLUSDDebtUpdated(LUSDDebt);
    }
```

Recommendation

biakia : Consider using `emit`:

```
  function increaseLUSDDebt(uint _amount) external override {
        _requireCallerIsBOorTroveM();
        LUSDDebt  = LUSDDebt.add(_amount);
        emit ActivePoolLUSDDebtUpdated(LUSDDebt);
    }

    function decreaseLUSDDebt(uint _amount) external override {
        _requireCallerIsBOorTroveMorSP();
        LUSDDebt = LUSDDebt.sub(_amount);
        emit ActivePoolLUSDDebtUpdated(LUSDDebt);
    }
```

yekong : It is suggested to add the `emit` in front of the `ActivePoolLUSDDebtUpdated()`.

```
    function increaseLUSDDebt(uint _amount) external override {
        _requireCallerIsBOorTroveM();
        LUSDDebt  = LUSDDebt.add(_amount);
        emit ActivePoolLUSDDebtUpdated(LUSDDebt);
    }

    function decreaseLUSDDebt(uint _amount) external override {
        _requireCallerIsBOorTroveMorSP();
        LUSDDebt = LUSDDebt.sub(_amount);
        emit ActivePoolLUSDDebtUpdated(LUSDDebt);
    }
```

Yaodao : Recommend adding the keyword `emit` for the event.

Client Response

Fixed

MSC-3:Use `disableInitializers` to prevent any future reinitialization

Category	Severity	Client Response	Contributor
Code Style	Low	Fixed	biakia, slowfrog, 0xxm, Xi_Zi

Code Reference

- code/contracts/BorrowerOperations.sol#L16
- code/contracts/TroveManager.sol#L16
- code/contracts/ActivePool.sol#L19
- code/contracts/ActivePool.sol#L38-L40
- code/contracts/ActivePool.sol#L38
- code/contracts/BorrowerOperations.sol#L95-L97
- code/contracts/BorrowerOperations.sol#L95
- code/contracts/StabilityPool.sol#L149
- code/contracts/TroveManager.sol#L224-L226
- code/contracts/TroveManager.sol#L224
- code/contracts/StabilityPool.sol#L271-L274
- code/contracts/StabilityPool.sol#L271

```
16:contract BorrowerOperations is LiquityBase, OwnableUpgradeable, CheckContract, Initializable, IBo
rrowerOperations {

16:contract TroveManager is LiquityBase, OwnableUpgradeable, CheckContract, ITroveManager, Initializ
able {

19:contract ActivePool is OwnableUpgradeable, CheckContract, IActivePool, Initializable {

38:function initialize() initializer external {
39:         __Ownable_init();
40:     }

38:function initialize() initializer external {

38:function initialize() initializer external {
39:         __Ownable_init();
40:     }

95:function initialize() initializer external {
96:         __Ownable_init();
97:     }

95:function initialize() initializer external {

149:contract StabilityPool is LiquityBase, OwnableUpgradeable, CheckContract, IStabilityPool, Initia
lizable {

224:function initialize() initializer external {
225:         __Ownable_init();
226:     }

224:function initialize() initializer external {

271:function initialize() initializer external {
272:         __Ownable_init();
273:         P = DECIMAL_PRECISION;
274:     }

271:function initialize() initializer external {
```

Description

biakia : The `ActivePool`, `StabilityPool`, `TroveManager` and `BorrowerOperations` are upgradeable contracts and can be initialized by any address. This is not a security problem in the sense that it impacts the system directly, as the attacker will not be able to cause any contract to self-destruct or modify any value in the proxy contract. However, taking ownership of implementation contracts can open other attack vectors, like social engineer or phishing attack. See docs: https://docs.openzeppelin.com/contracts/4.x/api/proxy#Initializable-_disableInitializers--

slowfrog : several upgradeable contracts implement `initialize` functions like this:

```
function initialize() initializer external {
        __Ownable_init();
        P = DECIMAL_PRECISION;
    }
```

this would set the first `msg.sender`, which would be the owner of `proxyAdmin`, as the owner of these upgradeable contracts. However as long as the implementation contract was deployed, before the proxy admin initializes the contract by `delegateCall` the implementation contract through proxy contract, anybody could call the `initialize` function of the implementation contract and set himself as the owner of the implementation contract, which might cause unexpected behavior.

0xxm : In the function `initialize`, some critical state variables are often set, but the `initialize` function has no access control. A malicious attacker could front-run the initialize transaction to gain ownership of the contract or set critical configuration variables. This can be repeated as a Denial Of Service (DOS) type of attack, effectively preventing contract deployment, leading to unrecoverable gas expenses.

Xi_Zi : The initialization method of the contract is not protected and anyone can call the initialization function of the logical contract.

```
    function initialize() initializer external {
        __Ownable_init();
    }
 function __Ownable_init() internal {
        address msgSender = msg.sender;
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }
```

OpenZeppelin advises against leaving logical contracts in an uninitialized state. refer : https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#initializing_the_implementation_contract

Recommendation

biakia : Consider using `disableInitializers` in these contracts:

```
constructor() {
        _disableInitializers();
}
```

slowfrog : call `_disableInitializers` in constructor to avoid the implementation contract being initialized by unexpected caller

```
constructor() {
    _disableInitializers();
}
```

0xxm : Add authentication to `initialize` function, or always bundle contract initialize and contract deployment into a single transaction.

Xi_Zi : Follow openzeplin's advice

```
constructor() {
    _disableInitializers();
}
function initialize() initializer external {
        __Ownable_init();
    }
```

Client Response

Fixed

MSC-4:Use OpenZeppelin's `ECDSA` instead of built-in `ecrecover`

Category	Severity	Client Response	Contributor
Signature Forgery or Replay	Low	Acknowledged	biakia

Code Reference

- code/contracts/LUSDToken.sol#L171-L192

```
171:function permit
172:    (
173:        address owner,
174:        address spender,
175:        uint amount,
176:        uint deadline,
177:        uint8 v,
178:        bytes32 r,
179:        bytes32 s
180:    )
181:        external
182:        override
183:    {
184:        require(deadline >= now, 'LUSD: expired deadline');
185:        bytes32 digest = keccak256(abi.encodePacked('\x19\x01',
186:                         domainSeparator(), keccak256(abi.encode(
187:                         _PERMIT_TYPEHASH, owner, spender, amount,
188:                         _nonces[owner]++, deadline))));
189:        address recoveredAddress = ecrecover(digest, v, r, s);
190:        require(recoveredAddress == owner, 'LUSD: invalid signature');
191:        _approve(owner, spender, amount);
192:    }
```

Description

biakia : The built-in function `ecrecover` is susceptible to signature malleability, which would lead to replay attacks.
References: https://swcregistry.io/docs/SWC-117/
https://swcregistry.io/docs/SWC-121/

Recommendation

biakia : Consider using OpenZeppelin's `ECDSA` library.

Client Response

Acknowledged

MSC-5:Ownership change should use two-step process

Category	Severity	Client Response	Contributor
Privilege Related	**Low**	**Fixed**	**biakia**

Code Reference

- code/contracts/Dependencies/OwnableUpgradeable.sol#L19-L76

```
19:contract OwnableUpgradeable {
20:    address private _owner;
21:
22:    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
23:
24:    function __Ownable_init() internal {
25:        address msgSender = msg.sender;
26:        _owner = msgSender;
27:        emit OwnershipTransferred(address(0), msgSender);
28:    }
29:
30:    /**
31:     * @dev Returns the address of the current owner.
32:     */
33:    function owner() public view returns (address) {
34:        return _owner;
35:    }
36:
37:    /**
38:     * @dev Throws if called by any account other than the owner.
39:     */
40:    modifier onlyOwner() {
41:        require(isOwner(), "Ownable: caller is not the owner");
42:        _;
43:    }
44:
45:    /**
46:     * @dev Returns true if the caller is the current owner.
47:     */
48:    function isOwner() public view returns (bool) {
49:        return msg.sender == _owner;
50:    }
51:
52:    /**
53:     * @dev Leaves the contract without owner. It will not be possible to call
54:     * `onlyOwner` functions anymore.
55:     *
56:     * NOTE: Renouncing ownership will leave the contract without an owner,
57:     * thereby removing any functionality that is only available to the owner.
58:     *
59:     * NOTE: This function is not safe, as it doesn’t check owner is calling it.
```

```
60:     * Make sure you check it before calling it.
61:     */
62:    function _renounceOwnership() internal {
63:        emit OwnershipTransferred(_owner, address(0));
64:        _owner = address(0);
65:    }
66:
67:    /**
68:     * @dev Transfers ownership of the contract to a new account (`newOwner`).
69:     * Can only be called by the current owner.
70:     */
71:    function transferOwnership(address newOwner) public virtual onlyOwner {
72:        require(newOwner != address(0), "Ownable: new owner is the zero address");
73:        emit OwnershipTransferred(_owner, newOwner);
74:        _owner = newOwner;
75:    }
76:}
```

Description

biakia : The contract `OwnableUpgradeable` does not implement a two-step process for transferring ownership. So ownership of the contract can be easily lost when making a mistake when transferring ownership.

Recommendation

biakia : Consider using `Ownable2StepUpgradeable` (https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/access/Ownable2StepUpgradeable.sol) instead.

Client Response

Fixed

MSC-6:Redundant code

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	Hupixiong3, Xi_Zi

Code Reference

- code/contracts/Interfaces/ITroveManager.sol#L24
- code/contracts/StabilityPool.sol#L246

```
24:event LQTYTokenAddressChanged(address _lqtyTokenAddress);

246:event DefaultPoolAddressChanged(address _newDefaultPoolAddress);
```

Description

Hupixiong3 : There is an unused event LQTYTokenAddressChanged in the ITroveManager contract, and the unused code should be removed for code specification and maintainability.
Xi_Zi : The presence of events that are declared but never used in the codebase.

```
contract StabilityPool is LiquityBase, OwnableUpgradeable, CheckContract, IStabilityPool, Initializa
ble {
    event DefaultPoolAddressChanged(address _newDefaultPoolAddress);
    }
}
```

Recommendation

Hupixiong3 : Delete the unused LQTYTokenAddressChanged event.
Xi_Zi : Remove unused events or emit them in the right place

Client Response

Fixed

MSC-7:Missing event

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	Xi_Zi, yekong

Code Reference

- code/contracts/DefaultPool.sol#L104-L108
- code/contracts/CollSurplusPool.sol#L118-L121

```
104:receive() external payable {
105:        _requireCallerIsActivePool();
106:        ETH = ETH.add(msg.value);
107:        emit DefaultPoolETHBalanceUpdated(ETH);
108:    }

118:receive() external payable {
119:        _requireCallerIsActivePool();
120:        ETH = ETH.add(msg.value);
121:    }
```

Description

Xi_Zi : There is a record of the amount of ETH in the contract, and calling claimColl triggers an event to record EtherSent, which records the amount of eth taken

```
    function claimColl(address _account) external override {
       . . .

       ETH = ETH.sub(claimableColl);
       emit EtherSent(_account, claimableColl);

      . . .
   }

However, when the eth is stored in the pool, the related event is missing.
```

```
    function _requireCallerIsActivePool() internal view {
        require(
            msg.sender == activePoolAddress,
            "CollSurplusPool: Caller is not Active Pool");
    }

    // --- Fallback function ---

    receive() external payable {
        _requireCallerIsActivePool();
        ETH = ETH.add(msg.value);
    }
```

yekong : The `DefaultPool.receive()` function has emit, but `CollSurplusPool.receive()` has none.

```
    receive() external payable {
        _requireCallerIsActivePool();
        ETH = ETH.add(msg.value);
        emit DefaultPoolETHBalanceUpdated(ETH);
    }
```

Recommendation

Xi_Zi : It is recommended to record the trigger events for both taking out of the pool and storing into ETH

yekong : Suggest to add event of `DefaultPool.receive()` function.

```
event CollSurplusPoolETHBalanceUpdated(uint _ETH);

receive() external payable {
    _requireCallerIsActivePool();
    ETH = ETH.add(msg.value);
    emit CollSurplusPoolETHBalanceUpdated(ETH);
}
```

Client Response

Fixed

MSC-8:Incorrect token name

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	Yaodao

Code Reference

- code/contracts/Dependencies/LiquityBase.sol#L28
- code/contracts/LUSDToken.sol#L31-L32

```
28:uint constant public LUSD_GAS_COMPENSATION = 1e18;

31:string constant internal _NAME = "WEN";
32:    string constant internal _SYMBOL = "WEN";
```

Description

Yaodao : The following codes named the token to be `WEN` instead of LUSD. However, the `LUSD` is still used to name all the related variables and error messages.

For example, the variable `LUSD_GAS_COMPENSATION` in the `LiquityBase`.

Recommendation

Yaodao : Recommend updating associated names.

Client Response

Fixed

MSC-9:Missing Zero Address Check in contracts/Timelock.sol

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	Xi_Zi

Code Reference

- code/contracts/Timelock.sol#L28-L34

```
28:constructor(address admin_, uint delay_) public {
29:        require(delay_ >= MINIMUM_DELAY, "Timelock::constructor: Delay must exceed minimum dela
y.");
30:        require(delay_ <= MAXIMUM_DELAY, "Timelock::setDelay: Delay must not exceed maximum dela
y.");
31:
32:        admin = admin_;
33:        delay = delay_;
34:    }
```

Description

Xi_Zi : This Function is lack of zero address check in important operation, which may cause some unexpected result.

```
    constructor(address admin_, uint delay_) public {
        require(delay_ >= MINIMUM_DELAY, "Timelock::constructor: Delay must exceed minimum delay.");
        require(delay_ <= MAXIMUM_DELAY, "Timelock::setDelay: Delay must not exceed maximum dela
y.");

        admin = admin_;
        delay = delay_;
    }
```

Recommendation

Xi_Zi : advised to use msg.sender as admin or perform zero address detection on admin_. Otherwise, permission may be lost

Client Response

Fixed

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.