verichains

SECURITY AUDIT OF

META AGE OF EMPIRES SMART CONTRACTS

Public Report

Apr 19, 2022

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Apr 19, 2022. We would like to thank the MAoE for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Meta Age of Empires Smart Contracts. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some vulnerable issues in the contract code. MAoE team has resolved and updated all the recommendations.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Meta Age of Empires Smart Contracts

MAoE is a “PLAY TO EARN” game built on the BSC platform. In Meta Age of Empires, players will embody cyborgs, go on an adventure to uncover the treasures, and learn more about mankind's once-famous civilizations.

Meta Age of Empires features basic yet appealing gameplay that is appropriate for all ages, and the game also demands players to think creatively in order to find a variety of priceless gems. These are the significant benefits that contribute to the game's unique appeal. Being enveloped in the rapid growth of Blockchain technology as well as the NFT coding trend, which is quickly entering the conventional gaming industry with a big number of Crypto believers.

Meta Age of Empires promises to make significant advancements in the project's development as well as the day-by-day completion of the full Ecosystem established by our developers.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the smart contracts of Meta Age of Empires Smart Contracts. It was conducted on the source code provided by the MAoE team.

It was conducted on commit 857616d4f7c760099e6d036850bf56a615bead51 from git repository *https://github.com/Metard-blockchain/MAoE*.

The latest version of the following files were made available in the course of the review:

SHA256 Sum	File
7e17c2f8b71effe2be7dcc68d8c9cf4114215dea25a4ac9d9e72cab9f0f89d83	**BoxNFT.sol**
43d1d7f96d476c7205d367b2fdf59cf4df513c15bdc15029b5d2c8a0729a1ca5	**LuckyNFT.sol**
c778fc354c2f43466a5586fa036b513b32c19fc993269118e4547f2cfa5d5b3e	**NFTCollection.sol**
6ef957bb807cb1afbd693e6f1fd7812b2115f4eb84cb02890bb51ca4a82a9169	**NFTMarketplace.sol**
a4c49a51f8ffce5835c54253b3054dcc815bb94140127de4403b4513e651ca5a	**Token.sol**

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The Meta Age of Empires Smart Contracts was written in Solidity language, with the required version to be ^0.8.0. The source code was written based on OpenZeppelin's library.

There are 5 main contracts in the audit scope. They are MAoE, NFTCollection, BoxNFT, LuckyNFT, and NFTMarketplace.

2.1.1. MAoE contract

This is the ERC20 token contract in the Meta Age of Empires Smart Contracts, which extends Context, ERC20 and Ownable contracts. With Ownable, by default, Contract Owner is the contract deployer, but he can transfer ownership to another address at any time. The contract pre-minted all 250 million tokens to the Contract Owner when deployed.

The contract supports taking fee while transferring tokens (buy, sell and transfer) which can customized by the Contract Owner at any time. The contract also supports blacklisting addresses. The Contract Owner can add/remove addresses to/from blacklist and blacklisted addresses can't transfer their tokens.

The contract also implements bot prevent in bpContract. The bpContract function can be easily disabled anytime by the Contract Owner using the setBPEnabled function. In addition, bpContract is only used in a short time in token public sale IDO then the Contract Owner will disable it forever by the setBPDisableForever function.

Table below lists some properties of the audited MAoE contract (as of the report writing time).

PROPERTY	VALUE
Name	MAoE Token
Symbol	MAoE
Decimals	18
Total Supply	250,000,000 ($x10^{18}$) Note: the number of decimals is 18, so the total representation token will be 250,000,000 or 250 million.

Table 2. MAoE Token properties

2.1.2. NFTCollection contract

This is the ERC721 NFT contract in the Meta Age of Empires Smart Contracts, which extends ERC721URIStorage and Ownable contracts. With Ownable, by default, Contract Owner is the contract deployer, but he can transfer ownership to another address at any time. The Contract Owner can set addresses to be admin which give them permission to call some special functions.

Users can buy boxes using MAoE tokens (limit holding 50 boxes per address). The contract also supports blacklisting addresses. The Contract Owner can add/remove addresses to/from blacklist and blacklisted addresses can't buy boxes. The Contract Owner can also withdraw any amount of any ERC20 tokens in this contract.

Admin can update the number of boxes which an address is holding to any amount. They can also change box price, open box, mint NFT, start/stop event (boxes can only be bought when event is started) and update which token can be used to buy boxes.

2.1.3. BoxNFT contract

this contract is used for opening boxes. BoxNFT extends Ownable and by default, Contract Owner is the contract deployer, but he can transfer ownership to another address at any time. The Contract Owner can set addresses to be admin which give them permission to call some special functions.

Users can open their boxes to obtain random type and rarity (from common to legendary) NFTs. There are only 10 ultra rare and 1 legendary, lower rarities have unlimited amount. The Contract Owner can add/remove addresses to/from blacklist and blacklisted addresses can't open boxes. Admin can start/stop event (boxes can only be opened when event is started) and change the probability ratio of rarities.

2.1.4. LuckyNFT contract

This contract is used for buying lucky NFTs which will be limited to 1000 NFTs (5 ultra rare, 50 super rare, 300 rare and 645 common). LuckyNFT extends Ownable and by default, Contract Owner is the contract deployer, but he can transfer ownership to another address at any time. The Contract Owner can set addresses to be admin which give them permission to call some special functions.

In whitelist period, only whitelisted addresses can buy lucky NFTs and the amount of NFTs that they can buy are set by Admin. After whitelist period, all users can buy lucky NFTs but each user can only buy max 5 lucky NFTs. The Contract Owner can withdraw any amount of any ERC20 tokens in this contract. Admin can start/stop event (lucky NFTs can only be bought when event is started), change lucky NFT price and update which token can be used to buy boxes.

2.1.5. NFTMarketplace contract

This is the marketplace contract in the Meta Age of Empires Smart Contracts, which extends Ownable contract. With Ownable, by default, Contract Owner is contract deployer, but he can transfer ownership to another address at any time. Users can buy/sell MAoE NFTs with a fee of 10% for each transaction.

2.2. Findings

During the audit process, the audit team found some vulnerability in the given version of Meta Age of Empires Smart Contracts.

MAoE fixed the code, according to Verichains's draft report, in commit dc76ce23bfc9b52a1817f9372086c5a232469d04.

2.2.1. LuckyNFT.sol, BoxNFT.sol - Unsafe random function CRITICAL

In random function, block.timestamp in Binance Smart Chain can be predicted, randNonce can be read from the chain. All parameters in random seed function can be predicted or get from chain so attackers have a good chance to control the result of random and get the best rarity when open lucky NFT.

```
function random(uint256 scale) internal returns (uint256) {
    uint256 randomNumber = uint256(
        keccak256(abi.encodePacked(block.timestamp, msg.sender, randNonce…
  ))
    ) % scale;
    randNonce++;
    return randomNumber;
}
```

RECOMMENDATION

If the project is being deployed to the BSC chain, we can use blockhash(block.number - 1) as a random factor. Since the block generation rate in the BSC chain is quite high, this random factor will be safer than the current one. A safer way to generate random numbers is using the Chainlink VRF. More detail can be found here *https://docs.chain.link/docs/chainlink-vrf/*.

UPDATES

- *Apr 19, 2022*: This issue has been acknowledged and fixed by the MAoE team.

2.2.2. BoxNFT.sol, LuckyNFT.sol - Missing contract call blocking CRITICAL

getLuckyNFT and openBox functions don't have any contract blocking mechanism. So, if these functions are called from a smart contract, users can easily revert the whole transaction in case they do not get the best value.

```
function random(uint256 scale) internal returns (uint256) {
    uint256 randomNumber = uint256(
        keccak256(abi.encodePacked(block.timestamp, msg.sender, randNonce…
  ))
    ) % scale;
    randNonce++;
    return randomNumber;
}

function getLuckyNFT() public returns (uint256) {
    address user =msg.sender;
    if (whitelistTime ==1) {
        uint256 res = openLuckyNFTv1(user);
        return res;
    } else {
        uint256 res = openLuckyNFTv2(user);
        return res;
    }
}

function openLuckyNFTv1(address recipient)
    internal
    overMaxNFT
    onlyLuckyWallet(recipient)
    eventGoingOn
    returns (uint256)
{
    uint256 _type = random(5) + 1;
    uint256 _rarity = random(5);
    ...
}

function openLuckyNFTv2(address recipient)
    internal
    overMaxNFT
    userMaxLuckyBox(recipient)
    eventGoingOn
```

```
    returns (uint256)
{
    uint256 _type = random(5) + 1;
    uint256 _rarity = random(5);
    ...
}

function openBox()
    public
    returns(uint256)
{
    address recipient = msg.sender;
    uint256 randomNumber = random(1000);
    uint256 _type = random(5) + 1;
    ...
}
```

RECOMMENDATION

Adding a contract blocking mechanism to any functions calling random, a simple but effective way to do this is to check if msg.sender == tx.origin.

```
modifier onlyNonContractCall() {
    require(msg.sender == tx.origin, "ONLY NON CONTRACT CALL");
    _;
}

function getLuckyNFT() public onlyNonContractCall returns (uint256) {
    address user =msg.sender;
    if (whitelistTime ==1) {
    uint256 res = openLuckyNFTv1(user);
    return res;
    } else {
    uint256 res = openLuckyNFTv2(user);
    return res;
    }
}

function openBox()
    public
    onlyNonContractCall
    returns(uint256)
{
```

```
    address recipient = msg.sender;
    uint256 randomNumber = random(1000);
    uint256 _type = random(5) + 1;
    ...
}
```

UPDATES

- *Apr 19, 2022*: This issue has been acknowledged and fixed by the MAoE team.

2.2.3. BoxNFT.sol - Exploitable random rarity with gas limit CRITICAL

In openBox, the gas for getting legendary NFT is always lower than the gas for getting common NFT so hackers can set gas limit to lower one to make sure they always get legendary NFT or reverted.

```
function openBox()
    public
    returns(uint256)
{
    address recipient = msg.sender;
    uint256 randomNumber = random(1000);
    uint256 _type = random(5) + 1;
    if (randomNumber >= legendaryThreshold){
        uint256 _rarity = 4;
        string memory _tokenURI = nftCollection.getTokenURI(_type, _rarit…
  y);
        uint256 res = nftCollection.mintNFTByBox(recipient, _tokenURI, _t…
  ype, _rarity);
        return res;
    }

    if (randomNumber >=ultraRareThreshold){
        uint256 _rarity = 3;
        string memory _tokenURI = nftCollection.getTokenURI(_type, _rarit…
  y);
        uint256 res = nftCollection.mintNFTByBox(recipient, _tokenURI, _t…
  ype, _rarity);
        return res;
    }

    if (randomNumber >= superRareThreshold){
        uint256 _rarity = 2;
```

```
        string memory _tokenURI = nftCollection.getTokenURI(_type, _rarit…
  y);
        uint256 res = nftCollection.mintNFTByBox(recipient, _tokenURI, _t…
  ype, _rarity);
        return res;
    }

    if (randomNumber >= rareThreshold){
        uint256 _rarity = 1;
        string memory _tokenURI = nftCollection.getTokenURI(_type, _rarit…
  y);
        uint256 res = nftCollection.mintNFTByBox(recipient, _tokenURI, _t…
  ype, _rarity);
        return res;
    }

    if (randomNumber >= commonThreshold){
        uint256 _rarity = 0;
        string memory _tokenURI = nftCollection.getTokenURI(_type, _rarit…
  y);
        uint256 res = nftCollection.mintNFTByBox(recipient, _tokenURI, _t…
  ype, _rarity);
        return res;
    }
}
```

RECOMMENDATION

Always checking common NFT first like the code bellow.

```
function openBox()
    public
    returns(uint256)
{
    address recipient = msg.sender;
    uint256 randomNumber = random(1000);
    uint256 _type = random(5) + 1;
    uint256 _rarity;

    if (randomNumber < rareThreshold){
        _rarity = 0;
    } else if (randomNumber < superRareThreshold){
        _rarity = 1;
```

```
    } else if (randomNumber < ultraRareThreshold){
        _rarity = 2;
    } else if (randomNumber < legendaryThreshold) {
        _rarity = 3;
    } else {
        _rarity = 4;
    }

    string memory _tokenURI = nftCollection.getTokenURI(_type, _rarity);
    uint256 res = nftCollection.mintNFTByBox(recipient, _tokenURI, _type,…
   _rarity);
    return res;
}
```

UPDATES

- *Apr 19, 2022*: This issue has been acknowledged and fixed by the MAoE team.

2.2.4. LuckyNFT.sol - countRarity has not been updated CRITICAL

In openLuckyNFTv1 and openLuckyNFTv2 functions, countRarity[_rarity] is used to compared, but it has not been updated.

```
function openLuckyNFTv1(address recipient)
    internal
    overMaxNFT
    onlyLuckyWallet(recipient)
    eventGoingOn
    returns (uint256)
{
    uint256 _type = random(5) + 1;
    uint256 _rarity = random(5);

    uint256 temp = 0;
    while (countRarity[_rarity] >= maxRarity[_rarity]){
        _rarity = random(5);
        temp ++;
        if (temp > 10 && countRarity[_rarity] >= maxRarity[_rarity]){
            return 0;
        }
    }
    string memory _tokenURI = nftCollection.getTokenURI(_type, _rarity);
    BATK = BAoE(tokenAddress);
```

```
    BATK.transferFrom(recipient, addressReceiver, luckyNFTPrice);
    uint256 res = nftCollection.mintNFT(recipient, _tokenURI, _type, _rar…
  ity);
    luckyWallet[recipient] -= 1;
    userLuckyNFT[recipient] ++;
    totalLuckyNFT ++;

    emit LuckyNFTOpened(recipient, recipient, _tokenURI);
    return res;
}

function openLuckyNFTv2(address recipient)
    internal
    overMaxNFT
    userMaxLuckyBox(recipient)
    eventGoingOn
    returns (uint256)
{
    uint256 _type = random(5) + 1;
    uint256 _rarity = random(5);

    uint256 temp = 0;
    while (countRarity[_rarity] >= maxRarity[_rarity]){
        _rarity = random(5);
        temp ++;
        if (temp > 10 && countRarity[_rarity] >= maxRarity[_rarity]){
            return 0;
        }
    }
    string memory _tokenURI = nftCollection.getTokenURI(_type, _rarity);
    BATK = BAoE(tokenAddress);
    BATK.transferFrom(recipient, addressReceiver, luckyNFTPrice);
    uint256 res = nftCollection.mintNFT(recipient, _tokenURI, _type, _rar…
  ity);
    userLuckyNFT[recipient] ++;
    totalLuckyNFT ++;

    emit LuckyNFTOpened(recipient, recipient, _tokenURI);
    return res;
}
```

RECOMMENDATION

Increasing countRarity[_rarity] after minting NFT.

UPDATES

- *Apr 19, 2022*: This issue has been acknowledged and fixed by the MAoE team.

2.2.5. Token.sol - Wrong fee calculation for admin HIGH

In _feeCalculation function, while checkAntiBot is false, there is a wrong fee calculation for admin when buying from uniswapV2Pair, checkAdmin(sender) should be checkAdmin(recipient) because sender is uniswapV2Pair already.

```
function _feeCalculation(
    address sender,
    address recipient,
    uint256 amount
)
internal
isBlackList(sender)
isBlackList(recipient)
isNotAddressZero(sender)
isNotAddressZero(recipient)
returns (uint256)
{
    ...
    if (recipient == uniswapV2Pair) {
        ...
    } else if (sender == uniswapV2Pair) {
        if (checkAdmin(sender)) { // should be checkAdmin(recipient)
            feeRate = 0;
        } else {
            require(
                amount <=
                (this.balanceOf(uniswapV2Pair) *
                percentAmountWhale) /
                10000,
                "Revert whale transaction"
            );
            feeRate = buyFeeRate;
        }
    } else {
        if (checkAdmin(sender)) { // maybe check for both sender and reci…
  pient
```

```
            feeRate = 0;
        } else {
            feeRate = transferRate;
        }
    }
    ...
}
```

RECOMMENDATION

checkAdmin for the recipient.

UPDATES

- *Apr 19, 2022*: This issue has been acknowledged and fixed by the MAoE team.

2.2.6. LuckyNFT.sol - Missing userMaxLuckyBox check LOW

In openLuckyNFTv1 function, userLuckyNFT is increasing for each lucky box opened, but it is not checked for userMaxLuckyBox(recipient) like in openLuckyNFTv2. Because there is no comments or documents to describe this behaviour, we give it a low severity here.

```
function openLuckyNFTv1(address recipient)
    internal
    overMaxNFT
    onlyLuckyWallet(recipient)
    eventGoingOn
    returns (uint256)
{
    uint256 _type = random(5) + 1;
    uint256 _rarity = random(5);

    uint256 temp = 0;
    while (countRarity[_rarity] >= maxRarity[_rarity]){
        _rarity = random(5);
        temp ++;
        if (temp > 10 && countRarity[_rarity] >= maxRarity[_rarity]){
            return 0;
        }
    }
    string memory _tokenURI = nftCollection.getTokenURI(_type, _rarity);
    BATK = BAoE(tokenAddress);
    BATK.transferFrom(recipient, addressReceiver, luckyNFTPrice);
```

```
    uint256 res = nftCollection.mintNFT(recipient, _tokenURI, _type, _rar…
  ity);
    luckyWallet[recipient] -= 1;
    userLuckyNFT[recipient] ++;
    totalLuckyNFT ++;

    emit LuckyNFTOpened(recipient, recipient, _tokenURI);
    return res;
}

function openLuckyNFTv2(address recipient)
    internal
    overMaxNFT
    userMaxLuckyBox(recipient)
    eventGoingOn
    returns (uint256)
{
    uint256 _type = random(5) + 1;
    uint256 _rarity = random(5);

    uint256 temp = 0;
    while (countRarity[_rarity] >= maxRarity[_rarity]){
        _rarity = random(5);
        temp ++;
        if (temp > 10 && countRarity[_rarity] >= maxRarity[_rarity]){
            return 0;
        }
    }
    string memory _tokenURI = nftCollection.getTokenURI(_type, _rarity);
    BATK = BAoE(tokenAddress);
    BATK.transferFrom(recipient, addressReceiver, luckyNFTPrice);
    uint256 res = nftCollection.mintNFT(recipient, _tokenURI, _type, _rar…
  ity);
    userLuckyNFT[recipient] ++;
    totalLuckyNFT ++;

    emit LuckyNFTOpened(recipient, recipient, _tokenURI);
    return res;
}
```

RECOMMENDATION

Checking userMaxLuckyBox in openLuckyNFTv1 function.

UPDATES

- *Apr 19, 2022*: This issue has been acknowledged by the MAoE team.

2.2.7. NFTCollection.sol, LuckyNFT.sol - Off by one max check LOW

_totalBox and _totalNFT are started from 0 so after checking with modifier (<=), users can have max _userMaxNFT + 1 NFTs and userMaxBox + 1 boxes. The same for lucky NFT.

```
// NFTCollection.sol
uint256 userMaxBox = 50;
uint256 userMaxNFT = 20;

modifier limitUserMaxBox(uint256 _totalBox) {
    require(
        _totalBox <= userMaxBox,
        "You can not buy exceed total box limitation"
    );
    _;
}

modifier limitMaxNFT(uint256 _totalNFT) {
    require(
        _totalNFT <= userMaxNFT,
        "You can not mint exceed total NFT limitation"
    );
    _;
}

// LuckyNFT.sol
uint256 private maxLuckyNFT = 101;
uint256 private totalLuckyNFT = 0;
uint256 private userMaxLuckyNFT = 1;

modifier overMaxNFT() {
    require(totalLuckyNFT <= maxLuckyNFT, "Run out of lucky box");
    _;
}

modifier userMaxLuckyBox(address user) {
    require(userLuckyNFT[user] <= userMaxLuckyNFT, "Run out of lucky box"…
```

```
  );
    _;
}
```

RECOMMENDATION

Replacing <= with <.

UPDATES

- *Apr 19, 2022*: This issue has been acknowledged and fixed by the MAoE team.

2.2.8. NFTCollection.sol - Wrong function name INFORMATIVE

The function name is adminAddBox but it's actually set the number.

```
function adminAddBox(address user, uint256 num)
    public
    limitUserMaxBox(userNumberBox[user])
    onlyAdmin
{
    userNumberBox[user] = num;
    emit BoxAdded(msg.sender, user, userNumberBox[user]);
}
```

RECOMMENDATION

Changing the function name to adminAddBox instead of adminSetBox.

UPDATES

- *Apr 19, 2022*: This issue has been acknowledged and fixed by the MAoE team.

2.2.9. Token.sol - Wrong modifier name INFORMATIVE

Modifier isBlackList required the address not in the blacklist so the name must be isNotInBlacklist.

```
modifier isBlackList(address account) {
    require(!checkBlackList(account), "Revert blacklist");
    _;
}
```

RECOMMENDATION

Changing the name to isNotInBlacklist

```
modifier isNotInBlackList(address account) {
    require(!checkBlackList(account), "Revert blacklist");
    _;
}
```

UPDATES

- *Apr 19, 2022*: This issue has been acknowledged and fixed by the MAoE team.

2.2.10. NFTCollection.sol - Typo INFORMATIVE

There is a typo in the contract. avaiableBox should be availableBox

```
modifier avaiableBox(uint256 _totalBox) {
    require(_totalBox > 0, "You don't have enough box to mint");
    _;
}
```

RECOMMENDATION

Fixing the typo.

UPDATES

- *Apr 19, 2022*: This issue has been acknowledged by the MAoE team.

2.2.11. NFTCollection.sol - Unnecessary assignment INFORMATIVE

There is an unnecessary assignment in the constructor.

```
constructor(address _tokenAddress, uint256 _boxPrice)
    public
    ERC721("BAoE", "NFT")
{
    ...
    adminlist[address(this)] = 1; // unnecessary assignment
    ...
}
```

RECOMMENDATION

Removing the assignment.

UPDATES

- *Apr 19, 2022*: This issue has been acknowledged and fixed by the MAoE team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Apr 19, 2022*	Public Report	Verichains Lab

Table 3. Report versions history