



verichains

SECURITY AUDIT OF

MILADY MEME COIN SMART

CONTRACT



Public Report

May 17, 2023

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 xRP20 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 was prepared by Verichains Lab on May 17, 2023. We would like to thank the Milady Meme Coin 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 Milady Meme Coin Smart Contract. 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 3 vulnerable issues in the contract code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY	5
1.1. About Milady Meme Coin Smart Contract	5
1.2. Audit scope	5
1.3. Audit methodology	6
1.4. Disclaimer	7
2. AUDIT RESULT	8
2.1. Overview	8
2.1.1. BridgePool.sol	8
2.1.2. Ladys.sol	8
2.1.3. MemBridge.sol	8
2.2. Findings	8
2.2.1. Ladys.sol - Centralize in the mint, burn and blacklist functions LOW	9
2.2.2. MemBridge.sol - signatureThreshold must always be greater than 0 LOW	9
2.2.3. MemBridge.sol - The signer must always different from address 0 LOW	10
2.2.4. BridgePool.sol - The contract still approves the old token when switching to a new token	
INFORMATIVE	12
3. VERSION HISTORY	13

1. MANAGEMENT SUMMARY

1.1. About Milady Meme Coin Smart Contract

LADYS may not be the meme coin Miladys want, but LADYS is the meme coin Miladys need in these times of unbridled memetic power.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the Milady Meme Coin Smart Contract. The audited contracts are the Milady Meme Coin Smart Contract that deployed on Ethereum Mainnet. The details of the deployed smart contract are listed in the tables below.

FIELD	VALUE
Contract Name	LadysToken
Contract Address	0x12970e6868f88f6557b76120662c1b3e50a646bf
Compiler Version	v0.8.19+commit.7dd6d404
Optimization Enabled	Yes with 200 runs
Explorer	https://etherscan.io/token/0x12970e6868f88f6557b76120662c1b3e50a646bf

Table 1. The LadysToken smart contract details

FIELD	VALUE
Contract Name	BridgePool
Contract Address	0x507771e32a1921837fa31170b3ba615c19bd0666
Compiler Version	v0.8.19+commit.7dd6d404
Optimization Enabled	Yes with 200 runs
Explorer	https://etherscan.io/address/0x507771e32a1921837fa31170b3ba615c19bd0666

Table 2. The BridgePool smart contract details

FIELD	VALUE
Contract Name	MemBridge
Contract Address	0x6e42386aee73322ac9b7d5b5467088e0d14ef7bc
Compiler Version	v0.8.19+commit.7dd6d404
Optimization Enabled	Yes with 200 runs
Explorer	https://etherscan.io/address/0x6e42386aee73322ac9b7d5b5467088e0d14ef7bc

Table 3. The MemBridge smart contract details

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 were 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 4. 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 Milady Meme Coin Smart Contract was written in `Solidity` language, with the required version to be `^0.8.0`. The source code was written based on OpenZeppelin's library.

2.1.1. BridgePool.sol

The `BridgePool` contract extends `Ownable` contract, by default, the contract Owner is the contract deployer, but he can transfer ownership to another address at any time. At deployment, the contract transferred ownership to another address. The contract owner can set a new token and approve a maximum amount of `uint256` units for the `MemBridge` contract.

2.1.2. Ladys.sol

The `LadysToken` contract extends `ERC20` and `AccessControl` contracts. `AccessControl` allows the contract to implement role-based access control mechanisms. There are 3 roles: `DEFAULT_ADMIN_ROLE`, `MINTER_ROLE` and `BURNER_ROLE`.

- Users with `DEFAULT_ADMIN_ROLE` role can blacklist any address and also change values such as: `uniswapV2Pair`, `maxHoldingAmount`, `minHoldingAmount`, and `limited`. With the `limited` variable, if it is enabled, when a user buys tokens such that the amount in their wallet after the purchase must be greater than `minHoldingAmount` and less than `maxHoldingAmount`. The blacklisted addresses cannot perform any action relating to transferring tokens.
- Users with the `MINTER_ROLE` role can mint any amount of tokens to any address.
- Users with the `BURNER_ROLE` role can burn any amount of tokens belonging to any address.

2.1.3. MemBridge.sol

The `MemBridge` extends `AccessControl` contract. `AccessControl` allows the contract to implement role-based access control mechanisms. There are 2 roles: `DEFAULT_ADMIN_ROLE` and `PAUSE_ROLE`. Users with `DEFAULT_ADMIN_ROLE` can change the values of `chainIdEther` and `chainIDSupport`, add/remove signers, set signature threshold, and change the token address.

Users with the `PAUSE_ROLE` role can enable/disable the bridge and claim functionality. Users can only bridge or claim tokens when the `PAUSE_ROLE` allows it.

2.2. Findings

During the audit process, the audit team found 3 vulnerabilities in the given version of Milady Meme Coin Smart Contract.

2.2.1. Ladys.sol - Centralize in the mint, burn and blacklist functions **LOW**

- The person with `MINTER_ROLE`/`BURNER_ROLE` can mint/burn tokens of any user.
- The person with `DEFAULT_ADMIN_ROLE` can blacklist any address, and those addresses cannot perform any action relating to token transfer.

```
function blacklist(address _address, bool _isBlacklisting) external
onlyRole(DEFAULT_ADMIN_ROLE) {
    blacklists[_address] = _isBlacklisting;
}
function mint(address _to, uint256 _amount) external onlyRole(MINTER_ROLE) {
    _mint(_to, _amount);
}
function burn(address _from, uint256 _amount) external onlyRole(BURNER_ROLE) {
    _burn(_from, _amount);
}
```

UPDATES

- *May 17, 2023:* This issue has been acknowledged by the Milady Meme Coin team.

2.2.2. MemBridge.sol - `signatureThreshold` must always be greater than 0 **LOW**

Users with `DEFAULT_ADMIN_ROLE` can change the value of `signatureThreshold` using the `setThreshold()` function, but there is no requirement that the value must always be greater than 0, which could allow an attacker to pass an empty `_proofs` parameter to make `verifySignature()` always return true.

```
function verifySignature(
    uint256 _fromChainID,
    string memory _txHash,
    uint256 _amount,
    Proof[] memory _proofs
) internal returns (bool) {
    uint256 _countSignature;
    address[] memory _signatories = new address[](_proofs.length);
    for (uint256 i = 0; i < _proofs.length; i++) {
        Proof memory _proof = _proofs[i];
        bytes32 _hashSignature = keccak256(
            abi.encode(
                getChainID(),
                _fromChainID,
                tx.origin,
                address(this),
                _txHash,
                _amount
            )
        );
    }
};
```

```

        address _signatory = ecrecover(_hashSignature, _proof.v, _proof.r, _proof.s);
        require(
            !checkDuplicateSignatory(_signatories, _signatory),
            "Duplicated Signatory"
        );
        if (!signers.contains(_signatory)) {
            return false;
        }
        _signatories[i] = _signatory;
        _countSignature ++;
    }
    require(
        isUsedTxHash[_fromChainID][_txHash] == 0,
        "The transaction hash has already been used"
    );
    isUsedTxHash[_fromChainID][_txHash] = 1;
    return signatureThreshold <= _countSignature;
}

function setThreshold(uint256 _signatureThreshold) external onlyRole(DEFAULT_ADMIN_ROLE) {
    signatureThreshold = _signatureThreshold;
}

```

RECOMMENDATION

Add a require statement that `_signatureThreshold` is always greater than 0 when making changes.

```

function setThreshold(uint256 _signatureThreshold) external onlyRole(DEFAULT_ADMIN_ROLE) {
    require(_signatureThreshold > 0, "Should greater than 0");
    signatureThreshold = _signatureThreshold;
}

```

UPDATES

- May 17, 2023: This issue has been acknowledged by the Milady Meme Coin team.

2.2.3. MemBridge.sol - The `signer` must always different from address 0 **LOW**

`ecrecover` function always returns address 0 for invalid signatures, so it is recommended to require that the signer address is always different from address 0 when adding a signer.

```

function verifySignature(
    uint256 _fromChainID,
    string memory _txHash,
    uint256 _amount,
    Proof[] memory _proofs
) internal returns (bool) {
    uint256 _countSignature;
    address[] memory _signatories = new address[](_proofs.length);

```

```

for (uint256 i = 0; i < _proofs.length; i++) {
    Proof memory _proof = _proofs[i];
    bytes32 _hashSignature = keccak256(
        abi.encode(
            getChainID(),
            _fromChainID,
            tx.origin,
            address(this),
            _txHash,
            _amount
        )
    );
    address _signatory = ecrecover(_hashSignature, _proof.v, _proof.r, _proof.s);
    require(
        !checkDuplicateSignatory(_signatories, _signatory),
        "Duplicated Signatory"
    );
    if (!signers.contains(_signatory)) {
        return false;
    }
    _signatories[i] = _signatory;
    _countSignature++;
}
require(
    isUsedTxHash[_fromChainID][_txHash] == 0,
    "The transaction hash has already been used"
);
isUsedTxHash[_fromChainID][_txHash] = 1;
return signatureThreshold <= _countSignature;
}

function setSigners(address[] memory _signers) external onlyRole(DEFAULT_ADMIN_ROLE) {
    for (uint256 i = 0; i < signers.length(); i++) {
        signers.remove(signers.at(i));
    }
    for (uint256 i = 0; i < _signers.length; i++) {
        signers.add(_signers[i]);
    }
}
}

```

RECOMMENDATION

Add a requirement that all elements in the `_signers` array must always be different from address 0 when making changes.

```

function setSigners(address[] memory _signers) external onlyRole(DEFAULT_ADMIN_ROLE) {
    for (uint256 i = 0; i < signers.length(); i++) {
        signers.remove(signers.at(i));
    }
    for (uint256 i = 0; i < _signers.length; i++) {
        require(_signers[i] != address(0), "Invalid signer address");
    }
}

```



```
        signers.add(_signers[i]);  
    }  
}
```

UPDATES

- *May 17, 2023:* This issue has been acknowledged by the Milady Meme Coin team.

2.2.4. BridgePool.sol - The contract still approves the old token when switching to a new token **INFORMATIVE**

The owner can change to a new token, but does not set the allowance of the old token to 0 for the `MemBridge` contract.

```
function authorizeBridge(address _memBridgeContract) external onlyOwner {  
    token.approve(_memBridgeContract, type(uint256).max);  
}  
function setToken(address _tokenAddress) external onlyOwner {  
    token = IERC20(_tokenAddress);  
}
```

RECOMMENDATION

It is recommended to set the allowance of the old token to 0 for the `MemBridge` contract before switching to the new token.

UPDATES

- *May 17, 2023:* This issue has been acknowledged by the Milady Meme Coin team.

Report for Milady Meme Coin

Security Audit – Milady Meme Coin Smart Contract

Version: 1.1 - Public Report

Date: May 17, 2023



3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	<i>May 13,2023</i>	Private Report	Verichains Lab
1.1	<i>May 17,2023</i>	Public Report	Verichains Lab

Table 5. Report versions history