verichains

SECURITY AUDIT OF

MAR3 SMART CONTRACT

Public Report

Dec 26, 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 *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 was prepared by Verichains Lab on Dec 26, 2023. We would like to thank the Mar3 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 Mar3 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 2 vulnerability issues in the contract code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Mar3 Smart Contract

Mar3 stands out as a leading AI-powered SocialFi dApp, offering a unique platform for individuals to connect with others and earn rewards within the Web3 ecosystem.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of Mar3 Smart Contract.

There are two contracts within our audit scope: the `Mar3` and `Mar3Staking` contracts.

We audit the snapshot taken on Dec 26, 2023, which includes the last version of the `Mar3` contract deployed on the Binance Smart Chain Mainnet at the address `0x9D44C04ef10Cbd4ba321E51A54F1354d0799fEEF`. The details of the deployed smart contract are listed in Table 1.

FIELD	VALUE
Contract Name	Mar3
Contract Address	0x9D44C04ef10Cbd4ba321E51A54F1354d0799fEEF
Compiler Version	v0.8.9+commit.e5eed63a
Optimization Enabled	Yes with 200 runs
Explorer	*https://bscscan.com/address/0x9D44C04ef10Cbd4ba321E51A54F1354d0799fEEF*

Table 1. The deployed smart contract details

We audit the snapshot taken on Dec 26, 2023, which includes the last version of the `Mar3Staking` contract deployed on the Binance Smart Chain Testnet at the address `0x207C693B577a10F8F4A6CEF525801F189DAC95dC`.

The details of the deployed smart contract are listed in Table 2.

FIELD	VALUE
Contract Name	Mar3Staking
Contract Address	0x207C693B577a10F8F4A6CEF525801F189DAC95dC
Compiler Version	v0.8.9+commit.e5eed63a
Optimization Enabled	Yes with 200 runs
Explorer	https://testnet.bscscan.com/address/0x207C693B577a10F8F4A6CEF525801F189DAC95dC

Table 2. The deployed smart contract details

1.3. Audit methodology

Our security audit process includes four steps:

- Mechanism Design is reviewed to look for any potential problems.
- Source codes are scanned/tested for commonly known and more specific vulnerabilities using public and our in-house security analysis tool.
- Manual audit of the codes for security issues. The source code is manually analyzed to look for any potential problems.
- Set up a testing environment to debug/analyze found issues and verifies our attack PoCs.

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 functioning; creates a critical risk to the application; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the application with high impact; needs to be fixed with high priority.

SEVERITY LEVEL	DESCRIPTION
MEDIUM	A vulnerability that could affect the desired outcome of executing the application 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 3. Severity levels

1.4. Disclaimer

Mar3 acknowledges that the security services provided by Verichains, are conducted to the best of their professional abilities but cannot guarantee 100% coverage of all security vulnerabilities. Mar3 understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, Mar3 agrees that Verichains shall not be held responsible or liable, and shall not be charged for any hacking incidents that occur due to security vulnerabilities not identified during the audit process.

1.5. Acceptance Minute

This final report served by Verichains to the Mar3 will be considered an Acceptance Minute. Within 7 days, if no any further responses or reports is received from the Mar3, the final report will be considered fully accepted by the Mar3 without the signature.

2. AUDIT RESULT

2.1. Overview

The Mar3 Smart Contract was written in `Solidity` language, with the required version to be `^0.8.9`. The source code was written based on OpenZeppelin's library.

2.1.1. Mar3 Token Contract

The `Mar3` token contract is an ERC20 token contract that only inherits `ERC20` contract from OpenZeppelin.

The smart contract is `ERC20` implementation that has some properties (as of the report writing time):

PROPERTY	VALUE
Name	MAR3 TOKEN
Symbol	MAR3
Decimals	18
Total Supply	2,000,000,000 (x10^{18}) Note: the number of decimals is 18, so the total representation token will be 2,000,000,000 or 2 billion.

Table 4. The Mar3 Smart Contract properties

For the ERC20 token, the security audit team has the list of centralization issues below:

Checklist	Status	Passed
Upgradeable	No	Yes
Fee modifiable	No	Yes
Mintable	No	Yes
Burnable	No	Yes
Pausable	No	Yes
Trading cooldown	No	Yes

Checklist	Status	Passed
Has blacklist	No	Yes
Has whitelist	No	Yes

Table 5. The decentralization checklist

2.1.2. Mar3Staking contract

This is the staking contract in the Mar3 Smart Contract which extends `Ownable` and `Pausable`. With `Ownable`, by default, the contract owner is the contract deployer, but he can transfer ownership to another address at any time. The token owner can pause/unpause contract using `Pausable` contract, user can only stake tokens when contract is not paused. In addition, the contract owner has the ability to withdraw any tokens and update the list of `adminList` members, who can add/edit, pause/unpause pools.

Users will stake their tokens using the `stake()` and `stakeLp()` functions. With the `stakeLp()` function, users will need the signature of the `signer` and only need to transfer their NFT to the contract. After completing the lock, they can withdraw using the `unstake()` and `unstakeLp()` functions.

2.2. Findings

During the audit process, the audit team found 2 vulnerabilities in the given version of Mar3 Smart Contract.

Severity	Name	Status
LOW	The constant `ONE_DAY_IN_SECONDS` is set to the wrong value	NEW
LOW	Missing check `expiredAt` variable	NEW
INFORMATIVE	Should check the `signer` variable	NEW
INFORMATIVE	Redundancy in comment	NEW
INFORMATIVE	Missing necessary events	NEW

2.2.1. The constant `ONE_DAY_IN_SECONDS` is set to the wrong value LOW

Positions:

- `Mar3Staking`#L55

Description:

The constant `ONE_DAY_IN_SECONDS` is set to a value of 10 for testing purposes, as indicated in the comment. This causes the waiting time for user unstaking to be very low.

RECOMMENDATION

It should be reset to the value at line 53, and lines 54-55 should be removed.

```
// uint256 public constant ONE_DAY_IN_SECONDS = 86400;
// Testing only
uint256 public constant ONE_DAY_IN_SECONDS = 10;
```

2.2.2. Missing check `expiredAt` variable LOW

Positions:

- `Mar3Staking`#stakeLp()

Description:

The variable `expiredAt` is used to limit the time of use for the signature. However, if the value of this variable is large, the signature may be reused after users `unstakeLp()`.

RECOMMENDATION

Should limit the `expiredAt` variable within a specific time range.

2.2.3. Should check the `signer` variable INFORMATIVE

Positions:

- `Mar3Staking`#L58
- `Mar3Staking`#L119

Description:

- Because the variable `signer` will only be changed once in the constructor function, it should be declared immutable.
- The constructor function should check whether the parameter `signer_` passed in is different from the `address(0)`. The `signer`'s address will be returned as `address(0)` in case the signature fails, thus, the `stakeLp()` function will bypass the check step `recoverSigner(...) == signer` with the signer being `address(0)`.

RECOMMENDATION

```
address public immutable signer;

constructor(address signer_) {
    require(signer_ != address(0), "Wrong signer_");
    signer = signer_;

    poolIdCount.increment();
}
```

2.2.4. Redundancy in comment INFORMATIVE

Positions:

- Mar3Staking#L129

Description:

The comment is used for redundant code, so it should be removed.

2.2.5. Missing necessary events INFORMATIVE

Positions:

- Mar3Staking#createPool()
- Mar3Staking#updatePool()
- Mar3Staking#pausePool()
- Mar3Staking#unpausePool()
- Mar3Staking#setAdmins()
- Mar3Staking#emergencyTokenWithdraw()

Description:

Events are important in Solidity because they provide an efficient way to notify external applications of changes that occur on the blockchain. They allow contracts to emit messages to the blockchain network that can be detected and processed by other contracts or external services...

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Dec 26, 2023*	Public Report	Verichains Lab

Table 6. Report versions history