verichains

SECURITY AUDIT OF

MIRAKLE

Public Report

Jul 31, 2023

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Fuse Network	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications. Fuse is an enterprise-grade, use-case agnostic, reliable, and secure decentralized EVM-compatible public blockchain.
FUSE	A cryptocurrency whose blockchain is generated by the Fuse Network platform. FUSE is used for payment of transactions and computing services in the Fuse 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 Jul 31, 2023. We would like to thank the Mirakle 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 Mirakle. 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.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Mirakle

Mirakle is a decentralized spot and perpetual exchange that supports low swap fees and zero price impact trades.

Trading is supported by a unique multi-asset pool that earns liquidity providers fees from market making, swap fees and leverage trading.

Dynamic pricing is supported by SupraOracles.

Mirakle is a decentralized exchange allowing trading without the need for a username or password. The platform uses an aggregate price feed which reduces the risk of liquidations from temporary wicks.

Currently, Mirakle is on testnet Fuse with a faucet for trading.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the Mirakle.It was conducted on commit `e76c869aae902503a21e457fc74e5086ef56cb55` from git repository *https://github.com/mirakle-io/mirakle-contract*.

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 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 Mirakle was written in `Solidity` language, with the required versions being `0.6.12`, `0.8.7`, and `0.8.12`. The source code was written based on OpenZeppelin's library.

The Mirakle is supported the following features:

- Swapping tokens with low swap fees and zero price impact trades.
- Staking tokens to earn rewards.
- Opening a position with leverage.
- Increasing, decrease a position with leverage.
- Closing a position.
- Liquidating a position.
- Depositing and withdrawing a position.
- Buying and selling tokens QLP and Qiji

2.2. Findings

During the audit process, the audit team found some vulnerabilities in the given version of Mirakle.

Mirakle team fixed the code, according to Verichains's draft report in commit `1f02ced7e7c27d8104fd77e93306fc9d06c0c124`.

#	Issue	Severity	Status
1	`VerifyPersonalSign` does not verify `chainId` and contract address which can be reused in other chains	CRITICAL	Fixed
2	Redundant code	INFORMATIVE	Fixed

2.2.1. `VerifyPersonalSign` does not verify `chainId` and contract address which can be reused in other chains - CRITICAL

Affected files:

- contracts/tokens/BridgeToken.sol

The `permit` function performs the change allowance of a target for a spender when the caller has a valid signature.

But the function `verifyPersonalSign` does not use `chainId` and current contract address to verify if the signature is valid. The attackers can reuse the signature to replay attacks in other domains (other contracts or chains).

Please refer to the EIP-2612 specification: *https://eips.ethereum.org/EIPS/eip-2612#specification*

For example. We have a scenario:

- `Target` is a User A whose tokens
- `Spender` is a User B
- User A signs an approval transaction in the Ethereum chain to permit User B to transfer tokens.
- In a chain Arbitrum, User B may call this method using User A's signature to approve tokens again
- So user B steals tokens of user A in the Arbitrum chain

The `transferAndPermit` function performs transfer tokens from `target` address to `to` address when the caller has a valid signature.

Same as above, we have a scenario:

- Target is a User A whose tokens
- `to` is a User B
- In a chain Ethereum, User A sign a transaction to transfer tokens to User B
- In a chain Arbitrum, User B may call this method using User A's signature to transfer tokens again.
- So user B steals tokens of user A in the Arbitrum chain

```
function permit(
        address target,
        address spender,
        uint256 value,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external override {
        require(block.timestamp <= deadline, "WERC10: Expired permit");

        bytes32 hashStruct = keccak256(
            abi.encode(
                PERMIT_TYPEHASH,
                target,
                spender,
                value,
                nonces[target]++,
                deadline
```

```
            )
        );

        require(
            verifyEIP712(target, hashStruct, v, r, s) ||
                verifyPersonalSign(target, hashStruct, v, r, s)
        );

        // _approve(owner, spender, value);
        allowance[target][spender] = value;
        emit Approval(target, spender, value);
    }

function transferWithPermit(
        address target,
        address to,
        uint256 value,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external returns (bool) {
        require(block.timestamp <= deadline, "WERC10: Expired permit");

        bytes32 hashStruct = keccak256(
            abi.encode(
                TRANSFER_TYPEHASH,
                target,
                to,
                value,
                nonces[target]++,
                deadline
            )
        );

        require(
            verifyEIP712(target, hashStruct, v, r, s) ||
                verifyPersonalSign(target, hashStruct, v, r, s)
        );

        require(to != address(0) || to != address(this));

        uint256 balance = balanceOf[target];
        require(balance >= value, "WERC10: transfer amount exceeds balance");

        balanceOf[target] = balance - value;
        balanceOf[to] += value;
        emit Transfer(target, to, value);
```

```
        return true;
    }

...

function prefixed(bytes32 hash) internal pure returns (bytes32) {
        return
            keccak256(
                abi.encodePacked("\x19Ethereum Signed Message:\n32", hash)
            );
    }

function verifyPersonalSign(
        address target,
        bytes32 hashStruct,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) internal pure returns (bool) {
        bytes32 hash = prefixed(hashStruct);
        address signer = ecrecover(hash, v, r, s);
        return (signer != address(0) && signer == target);
    }
```

RECOMMENDATION

Add `chainId` and `address(this)` to `verifyPersonalSign` function like in the `verifyEIP712` function.

UPDATES

- *Jul 21, 2023*: This issue has been acknowledged by the Mirakle team. They only use for testing on testnet environment.
- *Jul 31, 2023*: This issue has been fixed by the Mirakle team in commit `1f02ced7e7c27d8104fd77e93306fc9d06c0c124`.

2.2.2. Redundant code - INFORMATIVE

Affected files:

- contracts/oracle/FastPriceFeed.sol

Function `setPricesWithBitsAndExecuteIncrease` only executes increase positions so decrease positions parts is not necessary here. The same for `setPricesWithBitsAndExecuteDecrease`.

```
function setPricesWithBitsAndExecuteIncrease(
    uint256 _priceBits,
    uint256 _timestamp,
    uint256 _endIndexForIncreasePositions,
```

```
    uint256 _endIndexForDecreasePositions,
    uint256 _maxIncreasePositions,
    uint256 _maxDecreasePositions
) external onlyUpdater {
    _setPricesWithBits(_priceBits, _timestamp);

    IPositionRouter _positionRouter = IPositionRouter(positionRouter);
    uint256 maxEndIndexForIncrease = _positionRouter
        .increasePositionRequestKeysStart()
        .add(_maxIncreasePositions);
    uint256 maxEndIndexForDecrease = _positionRouter
        .decreasePositionRequestKeysStart()
        .add(_maxDecreasePositions);

    if (_endIndexForIncreasePositions > maxEndIndexForIncrease) {
        _endIndexForIncreasePositions = maxEndIndexForIncrease;
    }

    if (_endIndexForDecreasePositions > maxEndIndexForDecrease) {
        _endIndexForDecreasePositions = maxEndIndexForDecrease;
    }

    _positionRouter.executeIncreasePositions(
        _endIndexForIncreasePositions,
        payable(msg.sender)
    );
}

function setPricesWithBitsAndExecuteDecrease(
    uint256 _priceBits,
    uint256 _timestamp,
    uint256 _endIndexForIncreasePositions,
    uint256 _endIndexForDecreasePositions,
    uint256 _maxIncreasePositions,
    uint256 _maxDecreasePositions
) external onlyUpdater {
    _setPricesWithBits(_priceBits, _timestamp);

    IPositionRouter _positionRouter = IPositionRouter(positionRouter);
    uint256 maxEndIndexForIncrease = _positionRouter
        .increasePositionRequestKeysStart()
        .add(_maxIncreasePositions);
    uint256 maxEndIndexForDecrease = _positionRouter
        .decreasePositionRequestKeysStart()
        .add(_maxDecreasePositions);

    if (_endIndexForIncreasePositions > maxEndIndexForIncrease) {
        _endIndexForIncreasePositions = maxEndIndexForIncrease;
    }
```

```
    if (_endIndexForDecreasePositions > maxEndIndexForDecrease) {
        _endIndexForDecreasePositions = maxEndIndexForDecrease;
    }

    _positionRouter.executeDecreasePositions(
        _endIndexForDecreasePositions,
        payable(msg.sender)
    );
}
```

RECOMMENDATION

Remove redundant code.

```
function setPricesWithBitsAndExecuteIncrease(
    uint256 _priceBits,
    uint256 _timestamp,
    uint256 _endIndexForIncreasePositions,
    uint256 _maxIncreasePositions
) external onlyUpdater {
    _setPricesWithBits(_priceBits, _timestamp);

    IPositionRouter _positionRouter = IPositionRouter(positionRouter);

    uint256 maxEndIndexForIncrease = _positionRouter
        .increasePositionRequestKeysStart()
        .add(_maxIncreasePositions);

    if (_endIndexForIncreasePositions > maxEndIndexForIncrease) {
        _endIndexForIncreasePositions = maxEndIndexForIncrease;
    }

    _positionRouter.executeIncreasePositions(
        _endIndexForIncreasePositions,
        payable(msg.sender)
    );
}

function setPricesWithBitsAndExecuteDecrease(
    uint256 _priceBits,
    uint256 _timestamp,
    uint256 _endIndexForDecreasePositions,
    uint256 _maxDecreasePositions
) external onlyUpdater {
    _setPricesWithBits(_priceBits, _timestamp);

    IPositionRouter _positionRouter = IPositionRouter(positionRouter);
```

```
    uint256 maxEndIndexForDecrease = _positionRouter
        .decreasePositionRequestKeysStart()
        .add(_maxDecreasePositions);

    if (_endIndexForDecreasePositions > maxEndIndexForDecrease) {
        _endIndexForDecreasePositions = maxEndIndexForDecrease;
    }

    _positionRouter.executeDecreasePositions(
        _endIndexForDecreasePositions,
        payable(msg.sender)
    );
}
```

UPDATES

- *Jul 21, 2023*: This issue has been acknowledged and removed unused code by the Mirakle team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Jul 21, 2023*	Public Report	Verichains Lab
1.1	*Jul 31, 2023*	Public Report	Verichains Lab

Table 2. Report versions history