HAECHI AUDIT

MBXLAMM

Smart Contract Security Analysis

Published on : Nov 4, 2022

Version v2.1

HAECHI AUDIT

Smart Contract Audit Certificate

MBXLAMM

Security Report Published by HAECHI AUDIT

v1.0 Oct 18, 2022
v1.1 Oct 26, 2022
v2.0 Nov 3, 2022
v2.1 Nov 4, 2022

Auditor : taek lee

Found issues

Severity of Issues	Findings	Resolved	Acknowledged	Comment
Critical	-	-	-	-
High	3	3	-	-
Medium	-	-	-	-
Low	1	1	-	-
Tips	2	2	-	-

TABLE OF CONTENTS

ABOUT US

The most reliable web3 security partner.

HAECHI AUDIT is a flagship service of HAECHI LABS, the leader of the global blockchain industry. We bring together the best Web2 and Web3 experts. Security Researchers with expertise in cryptography, leaders of the global best hacker team, and blockchain/smart contract experts are responsible for securing your Web3 service.

We have secured the most well-known web3 services including 1inch, SushiSwap, Klaytn, Badger DAO, SuperRare, Netmarble, Klaytn and Chainsafe. We have secured $60B crypto assets on over 400 main-nets, Defi protocols, NFT services, P2E, and Bridges.

HAECHI AUDIT is the only blockchain technology company selected for the Samsung Electronics Startup Incubation Program in recognition of our expertise. We have also received technology grants from the Ethereum Foundation and Ethereum Community Fund.

Inquiries: audit@haechi.io

Website: audit.haechi.io

Executive Summary

Purpose of this report

This report was prepared to audit the security of the MBXLAMM contracts developed by the MarbleX team. HAECHI AUDIT conducted the audit focusing on whether the system created by the MarbleX team is soundly implemented and designed as specified in the published materials, in addition to the safety and security of the MBXLAMM.

Codebase Submitted for the Audit

The codes used in this Audit can be found on GitHub (https://github.com/MarblexAudit/AMM).

The last commit of the code used for this Audit is "{c27a22b63b81cf70c73bdebca0f6d0775423e5e9}".

Findings

HAECHI AUDIT found 3 High, 0 Medium and 1 Low severity issues. There are 1 Tips issues explained that would improve the code's usability or efficiency upon modification.

Severity	Issue	Status
High	Swap Fee is always zero	(Resolved - v1.1)
High	BridgeChainBridge is not refreshing the allowance to zero after failed transaction	(Resolved - v1.1)
Low	Desired overflow is blocked	(Resolved - v1.1)
TIPS	MBXLSwapPair.setIncentive() is not used	(Resolved - v1.1)
High	IncentiveDrop.update()/updateAll() function can be used to manipulate the incentive amount	(Resolved - v1.1)

Code Maturity

Criteria	Status	Comment
Level of Documentation	High	Natspec comments exist on the codebase.
Test Coverage	High	Unit Tests are well organized and coverage is sufficient.

Remarks

The project uses a bridge component which is not provided to audit. Since recent hacks are related to bridge operations, do keep in mind that the bridge is not safe even if we audited the bridge contract.

OVERVIEW

Protocol overview

- **IncentiveDrop**

Reward Distribution for MBXLAMM LPs. Distributes token to LPs with given speed.

It can use multiple incentive tokens

- **MBXLAMM**

UniswapV2 style amm with additional feature of incentiveDrop, and supports burnFee, communityFee, swapFee.

And it does not give a fee to LPs. LPs will be rewarded with incentiveDrop

Scope

```
src
├── AddressManager.sol
├── IncentiveDrop.sol
├── IncentiveDropFactory.sol
├── KIP7
│   ├── KIP7.sol
│   ├── KIP7Burnable.sol
│   ├── KIP7Detailed.sol
│   ├── KIP7Mintable.sol
│   ├── KIP7SafeMath.sol
│   └── TestToken.sol
├── MBXSwapFactory.sol
├── MBXSwapKIP7.sol
├── MBXSwapPair.sol
├── MBXSwapRouter.sol
├── bridge
│   ├── BridgeAMMController.sol
│   ├── BridgeChainBridge.sol
│   ├── BridgeChainToken.sol
│   ├── BridgeFactory.sol
│   ├── GameChainBridge.sol
│   ├── GameChainEncoder.sol
│   ├── common
│   │   ├── Bridge.sol
│   │   ├── BridgeCounterPart.sol
│   │   ├── BridgeFee.sol
│   │   ├── BridgeHandledRequests.sol
│   │   ├── BridgeOperator.sol
│   │   ├── BridgeTokens.sol
│   │   ├── BridgeTransfer.sol
│   │   ├── BridgeTransferERC20.sol
│   │   ├── external
│   │   │   ├── ERC20.sol
│   │   │   ├── ERC20Burnable.sol
│   │   │   ├── ERC20Detailed.sol
│   │   │   ├── ERC20Mintable.sol
│   │   │   ├── IERC20.sol
│   │   │   ├── MinterRole.sol
│   │   │   ├── Ownable.sol
│   │   │   ├── Roles.sol
│   │   │   └── SafeMath.sol
│   │   └── servicechain
│   │       └── IERC20BridgeReceiver.sol
│   └── interfaces
│       ├── IBridgeAMMController.sol
│       ├── IBridgeChainBridge.sol
│       ├── IBridgeFactory.sol
│       └── IGameChainBridge.sol
├── interfaces
│   ├── IAddressManager.sol
│   ├── IERC20.sol
│   ├── IIncentiveDrop.sol
│   ├── IIncentiveDropFactory.sol
│   ├── IKIP7.sol
│   ├── IKIPReceiver.sol
│   ├── IMBXSwapFactory.sol
│   ├── IMBXSwapKIP7.sol
│   ├── IMBXSwapPair.sol
│   └── IMBXSwapRouter.sol
├── libraries
│   ├── Address.sol
│   ├── CarefulMath.sol
│   ├── Context.sol
│   ├── ExponentialNoError.sol
│   ├── MBXSwapLibrary.sol
│   ├── Math.sol
│   ├── TransferHelper.sol
│   └── UQ112x112.sol
├── upgrade
│   ├── higherversion
│   │   ├── AddressUpgradeable.sol
│   │   ├── Context.sol
│   │   └── Initializable.sol
│   └── lowerversion
│       ├── AddressUpgradeable.sol
│       ├── Context.sol
│       └── Initializable.sol
│
```

```
└── view
  └── MBXSwapView.sol
```

FINDINGS

1. Swap Fee is always zero

ID: MBXLAMM-01	Severity: High
Type: Logic Error	Difficulty: Low
File: src/MBXSwapPair.sol	

Issue

When calculating the swapFee, MBXSwapPair uses MBXLAmountIn and multiplies fee ratio. But, MBXLAMountIn is always zero when isGameTokenToMBXLSwap is true. It should use MBXLAmountOut + burnFeeOut + communityFeeOut instead.

```
        uint256 gameTokenAmountIn = gameTokenBalance > gameTokenReserve - 
gameTokenAmountOut
            ? gameTokenBalance - (gameTokenReserve - gameTokenAmountOut)
            : 0;
        uint256 MBXLAmountIn = MBXLBalance > MBXLReserve - MBXLAmountOut
            ? MBXLBalance - (MBXLReserve - MBXLAmountOut)
            : 0;
        require(gameTokenAmountIn > 0 || MBXLAmountIn > 0, "E39");
        {
            // avoids stack too deep errors
            uint256 MBXLBalanceAdjusted;
            if (isGameTokenToMBXLSwap) {
                // 0.003 * 1e18
                MBXLBalanceAdjusted = ExponentialNoError.sub_(
                    MBXLBalance,
                    ExponentialNoError.mul_(MBXLAmountIn, ExponentialNoError.Exp({
mantissa: swapFee }))
                );
            } else {
                MBXLBalanceAdjusted = MBXLBalance;
            }

            require(gameTokenBalance * MBXLBalanceAdjusted >= uint256(gameTokenReserve)
* MBXLReserve, "E310");
        }

        _update(gameTokenBalance, MBXLBalance, gameTokenReserve, MBXLReserve);
        emit Swap(msg.sender, gameTokenAmountIn, MBXLAmountIn, gameTokenAmountOut,
MBXLAmountOut, to);
```

[https://github.com/MarblexAudit/AMM/blob/c27a22b63b81cf70c73bdebca0f6d0775423e5e9/src/MBXSwapPair.sol#L363]

Recommendation

1. Short term fix : use MBXLAmountOut + burnFeeOut + communityFeeOut instead of MBXLAmountIn
2. Long term fix : input swapFeeAmountOut as parameter just like communityFeeOut/burnFeeOut for clarity

2. BridgeChainBridge is not refreshing the allowance to zero after failed transaction

ID: MBXLAMM-02	Severity: High
Type: Loss of fund	Difficulty: High
File: src/bridge/BridgeChainBridge.sol	

Issue

When BridgeChainBridge fails to handleERC20, it does not revert the whole transaction, it emits an error event and the overall transaction succeeds. This will lead to allowance being left to bridgeAMMController. Which is risky since there isn't any proof that bridged transactions are not relaying the malicious transactions.

```
        if (_extraData.length > 0) {
            IKIP7(_tokenAddress).approve(bridgeAMMController, _value);

            //
https://ethereum.stackexchange.com/questions/83528/how-can-i-get-the-revert-reason-of-a-call-in-solidity-so-that-i-can-use-it-in-th
            //
https://github.com/Uniswap/v3-periphery/blob/v1.0.0/contracts/base/Multicall.sol
            (bool callSuccess, bytes memory result) =
bridgeAMMController.call(_extraData);

            if (callSuccess) {
                // Controller에 대한 call이 성공하였을 시
                emit BridgeChainResult(_requestTxHash, "S0");
            } else if (result.length < 68) {
                // Controller에 대한 call이 실패하였을 시(에러코드를 반환하지 않은 경우)
                emit BridgeChainResult(_requestTxHash, "F0");
                _requestERC20Transfer(_tokenAddress, address(this), _to, _value, 0, "");
            } else {
                // Controller에 대한 call이 실패하였을 시(AMM 내부에서 revert가 발생하여
에러코드를 반환한 경우)
                assembly {
                    result := add(result, 0x04)
                }
                string memory code = abi.decode(result, (string));

                emit BridgeChainResult(_requestTxHash, code);
                _requestERC20Transfer(_tokenAddress, address(this), _to, _value, 0, "");
            }
        }
```

[https://github.com/MarblexAudit/AMM/blob/c27a22b63b81cf70c73bdebca0f6d0775423e5e9/src/bridge/BridgeChainBridge.sol#L72]

Recommendation

add IKIP7(_tokenAddress).approve(bridgeAMMController, 0) when the call is reverted. Since there can be attack vectors where bridges can relay malicious transactions. But this depends on the bridge relayer implementation.

3. Desired overflow is blocked

ID: MBXLAMM-03	Severity: Low
Type: Bug	Difficulty: High
File: src/MBXSwapPair.sol	

Issue

In MBXSwapPair, _update() function has desired overflow operations but it is blocked because solidity 0.8 blocks overflow. It may lead to frozen funds when block.timestamp is larger than uint32 max value(which is year 2286)

```
    function _update(
        uint256 gameTokenBalance,
        uint256 MBXLBalance,
        uint112 _gameTokenReserve,
        uint112 _MBXLReserve
    ) private {
        require(gameTokenBalance <= type(uint112).max && MBXLBalance <=
type(uint112).max, "E70");
        uint32 blockTimestamp = uint32(block.timestamp % 2**32);
        uint32 timeElapsed = blockTimestamp - blockTimestampLast; // overflow is desired
        if (timeElapsed > 0 && _gameTokenReserve != 0 && _MBXLReserve != 0) {
            // * never overflows, and + overflow is desired
            gameTokenPriceCumulativeLast +=
                uint256(UQ112x112.encode(_MBXLReserve).uqdiv(_gameTokenReserve)) *
                timeElapsed;
            MBXLPriceCumulativeLast +=
uint256(UQ112x112.encode(_gameTokenReserve).uqdiv(_MBXLReserve)) * timeElapsed;
        }
        gameTokenReserve = uint112(gameTokenBalance);
        MBXLReserve = uint112(MBXLBalance);
        blockTimestampLast = blockTimestamp;
        emit Sync(gameTokenReserve, MBXLReserve);
    }
```

[https://github.com/MarblexAudit/AMM/blob/c27a22b63b81cf70c73bdebca0f6d0775423e5e9/src/MBXSwapPair.sol#L243]

Recommendation

use unchecked block for desired overflow operations

4. MBXLSwapPair.setIncentive() is not used

ID: MBXLAMM-04	Severity: Tips
Type: Bug	Difficulty: N/A
File: src/MBXSwapPair.sol	

Issue

MBXLSwapPair.setIncentive() has onlyFactory modifier which means only factory address can call this function but factory does not use this function anywhere in the contract.

```
    function setIncentive(IncentiveDrop.InitialParams memory initialParams) external
onlyFactory {
        if (incentiveDrop() == address(0)) {
            incentiveDropFactory().deploy();
        }

        address incentiveDropAddr = incentiveDrop();
        IIncentiveDrop(incentiveDropAddr).initialize(initialParams);
    }
```

[https://github.com/MarblexAudit/AMM/blob/c27a22b63b81cf70c73bdebca0f6d0775423e5e9/src/MBXSwapPair.sol#L414]

Recommendation

Since this function is not used and same functionality can be achieved with other functions, it is recommended to delete this function

5. IncentiveDrop.update()/updateAll() function can be used to manipulate the incentive amount

ID: MBXLAMM-05	Severity: High
Type: Yield theft	Difficulty: Low
File: src/IncentiveDrop.sol	

Issue

IncentiveDrop.distributeIncentive() uses the user's current balance to distribute the reward. But this balance can be easily manipulated using flashloan.

example :

1. user has 0 token
2. user floashloans 1000000 token
3. user calls IncentiveDrop.update() function and gets reward based on 1000000 token
4. user sells incentive token and earn 30000 token
5. user repays total of 1010000 token and gets 20000 token

this can be amplified if every user does flashloan based incentive update

```
        IncentiveState storage incentiveState_ = incentiveState[index];
        IncentiveUserState storage incentiveUserState_ =
incentiveUserState[index][user];

        ExponentialNoError.Double memory incentiveIndex = ExponentialNoError.Double({
            mantissa: incentiveState_.index
        });
        ExponentialNoError.Double memory incentiveUserIndex =
ExponentialNoError.Double({
            mantissa: incentiveUserState_.index
        });

        incentiveUserState_.index = incentiveState_.index;

        if (incentiveUserIndex.mantissa == 0 && incentiveIndex.mantissa > 0) {
            incentiveUserIndex.mantissa = initialIncentiveIndex;
        }

        ExponentialNoError.Double memory deltaIndex =
ExponentialNoError.sub_(incentiveIndex, incentiveUserIndex);

        uint256 userBalance = pair.balanceOf(user);
        uint256 incentiveDelta = ExponentialNoError.mul_(userBalance, deltaIndex);
        uint256 incentiveAccrued = incentiveUserState_.currentAccrued + incentiveDelta;

        incentiveUserState_.currentAccrued = incentiveAccrued;
        incentiveUserState_.totalAccrued = incentiveUserState_.totalAccrued +
incentiveDelta;

        emit DistributeIncentive(incentiveState_.incentive, index, user, incentiveDelta,
incentiveIndex.mantissa);
```

[https://github.com/MarblexAudit/AMM/blob/c27a22b63b81cf70c73bdebca0f6d0775423e5e9/src/IncentiveDrop.sol#L203]

Recommendation

1. Short term : Add admin modifier for update/updateAll/claimIncentive/claimAll and make admin calls for users.
2. Long term : use SNXRewardPool or MasterCherf style rewardAccumulator

6. Unused function parameter in MBXSwapFactory.removePair

ID: MBXLAMM-06	Severity: TIPS
Type: Unused function parameter	Difficulty: N/A
File: src/MBXSwapFactory.sol	

Issue

MBXSwapFactory.removePair gets “pair address” and “user address” as parameters. But it does not use “user address” in the function.

Recommendation

It is recommended to remove the unused parameter.

Fix

Last Update: 2022.11.04

Fix Comment

Issue 1,2,3,4 was patched in this pull request.

Issue 5 was patched in this commit.

Issue 6 was introduced in this commit.

Issue 6 was patched in this commit

DISCLAIMER

This report does not guarantee investment advice, the suitability of the business models, and codes that are secure without bugs. This report shall only be used to discuss known technical issues. Other than the issues described in this report, undiscovered issues may exist such as defects on the main network. In order to write secure smart contracts, correction of discovered problems and sufficient testing thereof are required.

Appendix. A

Severity Level

CRITICAL	Must be addressed as a vulnerability that has the potential to seize or freeze substantial sums of money.
HIGH	Has to be fixed since it has the potential to deny users compensation or momentarily freeze assets.
MEDIUM	Vulnerabilities that could halt services, such as DoS and Out-of-Gas, need to be addressed.
LOW	Issues that do not comply with standards or return incorrect values
TIPS	Tips that makes the code more usable or efficient when modified

Difficulty Level

	Low	**Medium**	**High**
Privilege	anyone	Miner/Block Proposer	Admin/Owner
Capital needed	Small or none	Gas fee or volatile as price change	More than exploited amount
Probability	100%	Depend on environment	Hard as mining difficulty

Vulnerability Category

Arithmetic	▪ Integer under/overflow vulnerability ▪ floating point and rounding accuracy
Access & Privilege Control	▪ Manager functions for emergency handle ▪ Crucial function and data access ▪ Count of calling important task, contract state change, intentional task delay
Denial of Service	▪ Unexpected revert handling ▪ Gas limit excess due to unpredictable implementation
Miner Manipulation	▪ Dependency on the block number or timestamp. ▪ Frontrunning
Reentrancy	▪Proper use of Check-Effect-Interact pattern. ▪Prevention of state change after external call ▪ Error handling and logging.
Low-level Call	▪ Code injection using delegatecall ▪ Inappropriate use of assembly code
Off-standard	▪ Deviate from standards that can be an obstacle of interoperability.
Input Validation	▪ Lack of validation on inputs.
Logic Error/Bug	▪ Unintended execution leads to error.
Documentation	▪Coherency between the documented spec and implementation
Visibility	▪ Variable and function visibility setting
Incorrect Interface	▪ Contract interface is properly implemented on code.

End of Document