

Advanced Manual **Smart Contract Audit**

September 23, 2022

Audit requested by



UniWswap MasterChef

No contract address

Table of Contents

1. Audit Summary

- 1.1 Audit scope
- 1.2 Tokenomics
- 1.3 Source Code

2. Disclaimer

3. Global Overview

- 3.1 Informational issues
- 3.2 Low-risk issues
- 3.3 Medium-risk issues
- 3.4 High-risk issues

4. Vulnerabilities Findings

5. Contract Privileges

- 5.1 Maximum Fee Limit Check
- 5.2 Contract Pausability Check
- 5.3 Max Transaction Amount Check
- 5.4 Exclude From Fees Check
- 5.5 Ability to Mint Check
- 5.6 Ability to Blacklist Check
- 5.7 Owner Privileges Check

6. Notes

- 6.1 Notes by Coinsult
- 6.2 Notes by UniWswap MasterChef

7. Contract Snapshot

8. Website Review

9. Certificate of Proof

Audit Summary

Audit Scope

Project Name	UniWswap MasterChef
Website	https://uniwswap.com/
Blockchain	Ethereum PoW
Smart Contract Language	Solidity
Contract Address	
Audit Method	Static Analysis, Manual Review
Date of Audit	23 September 2022

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

Tokenomics

Not available

Source Code

Coinsult was commissioned by UniWswap MasterChef to perform an audit based on the following code:

<https://github.com/uniwswap/uniw-contracts/blob/main/masterchef/MasterChef.sol>

Disclaimer

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

Coinsult is not responsible if a project turns out to be a scam, rug-pull or honeypot. We only provide a detailed analysis for your own research.

Coinsult is not responsible for any financial losses. Nothing in this contract audit is financial advice, please do your own research.

The information provided in this audit is for informational purposes only and should not be considered investment advice. Coinsult does not endorse, recommend, support or suggest to invest in any project.


Coinsult can not be held responsible for when a project turns out to be a rug-pull, honeypot or scam.

Global Overview

Manual Code Review

In this audit report we will highlight the following issues:

Vulnerability Level	Total	Pending	Acknowledged	Resolved
Informational	0	0	0	0
Low-Risk	5	5	0	0
Medium-Risk	0	0	0	0
High-Risk	0	0	0	0

 **Low-Risk:** Could be fixed, will not bring problems.

Contract variables unknown to Coinsult

```
IToken _token,  
IBoosterNFT _boosterNFT,  
uint256 _tokenPerBlock,  
uint256 _startBlock,  
address _devAddress
```

Recommendation

These constants will be entered during deployment, during the audit we are unsure which variables will be entered.

● **Low-Risk:** Could be fixed, will not bring problems.

Too many digits

Literals with many digits are difficult to read and review.

```
fee = _amount.mul(pool.depositFee).div(10000);
```

Recommendation

Use: Ether suffix, Time suffix, or The scientific notation

Exploit scenario

```
contract MyContract{
    uint 1_ether = 1000000000000000000;
}
```

While `1_ether` looks like 1 ether, it is 10 ether. As a result, it's likely to be used incorrectly.

● **Low-Risk:** Could be fixed, will not bring problems.

Unchecked transfer

The return value of an external transfer/transferFrom call is not checked.

```
// Safe TOKEN transfer function, just in case if rounding error causes pool to not have enough Token.
function safeTokenTransfer(address _to, uint256 _amount, uint256 _pid) internal {
    uint256 boost = 0;
    token.transfer(_to, _amount);
    boost = getBoost(_to, _pid).mul(_amount).div(10000);
    payReferralCommission(msg.sender, _amount);
    if (boost > 0) token.mint(_to, boost);
}
```

Recommendation

Use SafeERC20, or ensure that the transfer/transferFrom return value is checked.

Exploit scenario

```
contract Token {
    function transferFrom(address _from, address _to, uint256 _value) public returns (bool success);
}
contract MyBank{
    mapping(address => uint) balances;
    Token token;
    function deposit(uint amount) public{
        token.transferFrom(msg.sender, address(this), amount);
        balances[msg.sender] += amount;
    }
}
```

Several tokens do not revert in case of failure and return false. If one of these tokens is used in MyBank, deposit will not revert if the transfer fails, and an attacker can call deposit for free..

● **Low-Risk:** Could be fixed, will not bring problems.

Divide before multiply

Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.

```
uint256 tokenReward = multiplier.mul(tokenPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
accTokenPerShare = accTokenPerShare.add(tokenReward.mul(1e18).div(lpSupply));
```

Recommendation

Consider ordering multiplication before division.

Exploit scenario

```
contract A {
    function f(uint n) public {
        coins = (oldSupply / n) * interest;
    }
}
```

If n is greater than $oldSupply$, $coins$ will be zero. For example, with $oldSupply = 5$; $n = 10$, $interest = 2$, $coins$ will be zero. If $(oldSupply * interest / n)$ was used, $coins$ would have been 1. In general, it's usually a good idea to re-arrange arithmetic to perform multiplication before division, unless the limit of a smaller type makes this dangerous.

● **Low-Risk:** Could be fixed, will not bring problems.

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function setBoosterNFT(address _boosterNFT) public onlyOwner {
    require(_boosterNFT != address(0x0), "_boosterNFT address should be valid address");

    boosterNFT = IBoosterNFT(_boosterNFT);
}
```

Recommendation

Emit an event for critical parameter changes.

Exploit scenario

```
contract C {

    modifier onlyAdmin {
        if (msg.sender != owner) throw;
        _;
    }

    function updateOwner(address newOwner) onlyAdmin external {
        owner = newOwner;
    }
}
```

updateOwner() has no event, so it is difficult to track off-chain changes in the buy price.

Other Owner Privileges Check

Coinsult lists all important contract methods which the owner can interact with.

✔ No other important owner privileges to mention.

Notes

Notes by UniWswap MasterChef

No notes provided by the team.

Notes by Coinconsult

 No notes provided by Coinconsult

Contract Snapshot

This is how the constructor of the contract looked at the time of auditing the smart contract.

```
contract MasterChef is Ownable, ReentrancyGuard, Whitelist {
    using SafeMath for uint256;
    using SafeBEP20 for IBEP20;

    // Bonus multiplier for early lp makers.
    uint256 public constant BONUS_MULTIPLIER = 1;

    // Info of each user.
    struct UserInfo {
        uint256 amount;           // How many LP tokens the user has provided.
        uint256 rewardDebt;       // Reward debt. See explanation below.
    }

    // Info of each pool.
    struct PoolInfo {
        IBEP20 lpToken;           // Address of LP token contract.
        uint256 allocPoint;       // How many allocation points assigned to this pool. TOKEN to distribute
        uint256 lastRewardBlock;  // Last block number that TOKEN distribution occurs.
        uint256 accTokenPerShare; // Accumulated TOKEN per share, times 1e18. See below.
        uint256 depositFee;       // Pool Deposit fee
        bool isPrivatePool;       // private pool only be accessible by particular address
    }

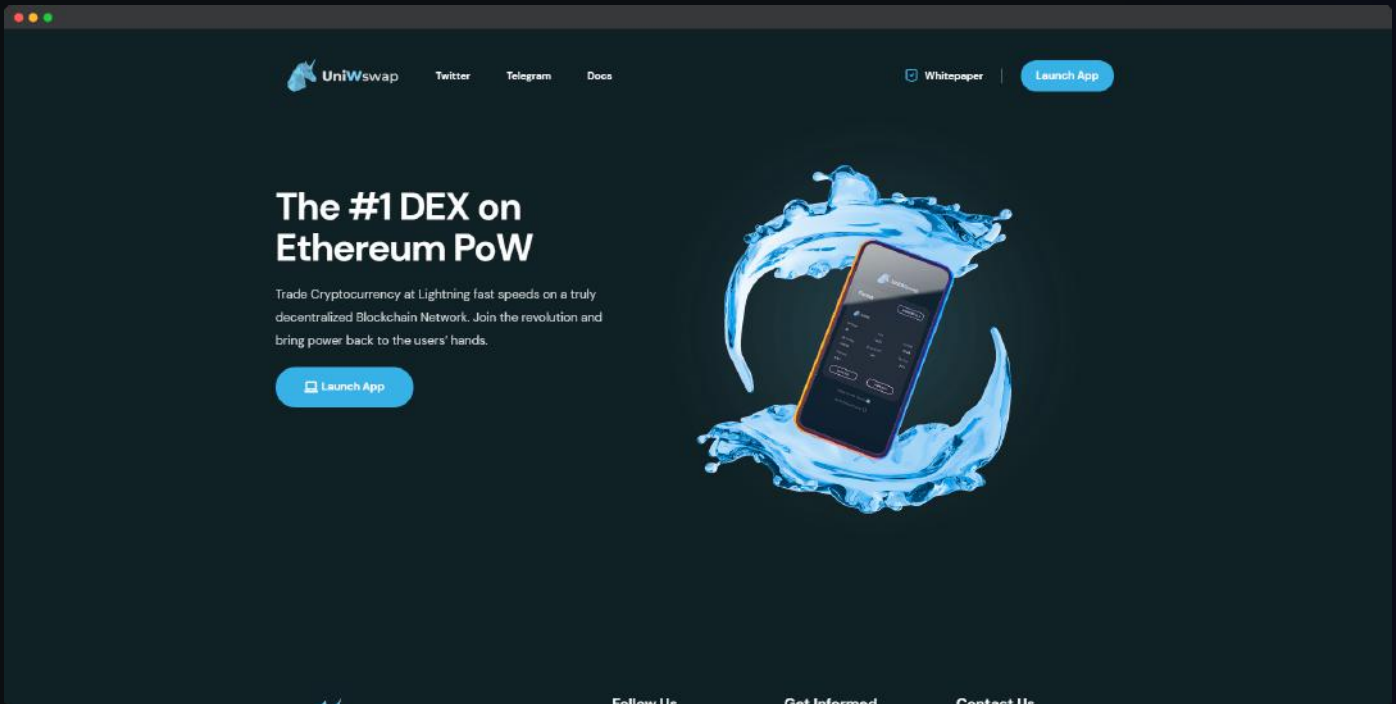
    struct NFTDetails {
        address nftAddress;
        uint256 tokenId;
    }

    // The TOKEN!
    IToken public token;
    // Tokens created per block.
    uint256 public tokenPerBlock;

    // Info of each pool.
    PoolInfo[] public poolInfo;
    // Info of each user that stakes LP tokens.
    mapping(uint256 => mapping(address => UserInfo)) public userInfo;
```

Website Review

Coinsult checks the website completely manually and looks for visual, technical and textual errors. We also look at the security, speed and accessibility of the website. In short, a complete check to see if the website meets the current standard of the web development industry.



Type of check	Description
Mobile friendly?	● The website is mobile friendly
Contains jQuery errors?	● The website does not contain jQuery errors
Is SSL secured?	● The website is SSL secured
Contains spelling errors?	● The website does not contain spelling errors

Certificate of Proof

● Not KYC verified by Coinsult

UniWswap MasterChef

Audited by Coinsult.net



Date: 23 September 2022

✓ Advanced Manual Smart Contract Audit

End of report
Smart Contract Audit

Request your smart contract audit / KYC

t.me/coinsult_tg