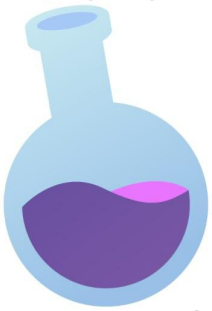
InterFi NETWORK

@INTERFINETWORK

SMART CONTRACT SECURITY AUDIT OF

MasterChefV2

(EDXA DeFi Laboratory)



SMART CONTRACT AUDIT | TEAM KYC | PROJECT EVALUATION

RELENTLESSLY SECURING THE PUBLIC BLOCKCHAIN | MADE IN CANADA

Summary

Auditing Firm	InterFi Network
Architecture	InterFi "Echelon" Auditing Standard
Approved By	Chris | Blockchain Specialist at InterFi Network
Platform	Solidity
Audit Check (Mandatory)	Static, Software, Auto Intelligent & Manual Analysis
Project Check (Optional)	KYC, Website & Socials Analysis (Not Applicable)
Consultation Request Date	November 01, 2021
Report Date	November 04, 2021

Audit Summary

InterFi team has performed a line-by-line manual analysis and automated review of the smart contract. The smart contract was analyzed mainly for common smart contract vulnerabilities, exploits, and manipulation hacks. According to the smart contract audit:

- **MasterChef's smart contract source code has LOW RISK SEVERITY.**
- **MasterChef has PASSED the smart contract audit.**
- **MasterChefV2 is a part of EDXA DeFi Laboratory.**
- **MasterChef's smart contract owner can mint tokens and add it to the total supply.**

For the detailed understanding of risk severity, source code vulnerability, and functional test, kindly refer to the audit.

Table Of Contents

Project Overview

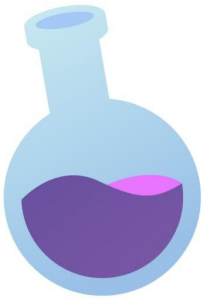
InterFi was consulted by EDXA on November 01, 2021 to conduct a smart contract security audit of their MasterChef source code.

About EDXA Project

EDXA is a type of Decentralize Exchange (DEX) known as an automated market maker (AMM). This essentially means that there are no order books, bid/ask system or limit/market orders. Instead, users trading on the platform automatically draw liquidity from one or more liquidity pools, which then rebalance after the trade is complete. EDXA is the native utility token, is used for a variety of purposes within the growing landscape. Its main functions are yield farming, staking, participating in the pools and launchpad.

Project	MasterChef
Blockchain	Binance Smart Chain
Language	Solidity
Contract	0x0463587C9D3C39130C59f6Cf96739a15b27D2a3d
Block Explorer	https://bscscan.com/address/0x0463587C9D3C39130C59f6Cf96739a15b27D2a3d
Website	https://edxa.co/
Medium	https://edxalabs.medium.com/
Telegram	https://t.me/edxa_group_official
Twitter	https://twitter.com/EdxaLabs
GitHub	https://github.com/EdxaLabs

Public logo



Solidity Source Code On Blockchain (BscScan Verified Contract Source Code)

https://bscscan.com/address/0x0463587C9D3C39130C59f6Cf96739a15b27D2a3d#code

Contract Name: MasterChef

Compiler Version: v0.8.9+commit.e5eed63a

Optimization Enabled: Yes with 200 runs

Solidity Source Code On InterFi GitHub

https://github.com/interfinetwork/audited-codes/blob/main/MasterChef-EDXA.sol

GitHub Commits

Solidity source code committed at: cdd51472e6de685964bfa57fa73bf8c9bdf151b7

SHA-1 Hash

Solidity source code audited at #e5207de8c431ecff0c41b51fc1bb9c872e54f13f

Audit Scope & Methodology

The scope of this report is to audit the smart contract source code of MasterChef. The source code can be viewed in its entirety on

https://bscscan.com/address/0x0463587C9D3C39130C59f6Cf96739a15b27D2a3d#code

InterFi has scanned the contract and reviewed the project for common vulnerabilities, exploits, hacks, and back-doors. Below is the list of commonly known smart contract vulnerabilities, exploits, and hacks:

Category	
Smart Contract Vulnerabilities	❖ Re-entrancy (RE) ❖ Unhandled Exceptions (UE) ❖ Transaction Order Dependency (TO) ❖ Integer Overflow (IO) ❖ Unrestricted Action (UA) ❖ Ownership Takeover
Source Code Review	❖ Gas Limit and Loops ❖ Deployment Consistency ❖ Repository Consistency ❖ Data Consistency ❖ Token Supply Manipulation
Functional Assessment	❖ Access Control and Authorization ❖ Operations Trail and Event Generation ❖ Assets Manipulation ❖ Liquidity Access

InterFi's Echelon Audit Standard

The aim of InterFi's "Echelon" standard is to analyze the smart contract and identify the vulnerabilities and the hacks in the smart contract. Mentioned are the steps used by ECHELON-1 to assess the smart contract:

1. Solidity smart contract source code reviewal:
   - ❖ Review of the specifications, sources, and instructions provided to InterFi to make sure we understand the size, scope, and functionality of the smart contract.
   - ❖ Manual review of code, which is the process of reading source code line-byline to identify potential vulnerabilities.
2. Static, Manual, and Automated AI analysis:
   - ❖ Test coverage analysis, which is the process of determining whether the test cases are covering the code and how much code is exercised when we run those test cases.
   - ❖ Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts

Automated 3P frameworks used to assess the smart contract vulnerabilities

- ❖ Slither
- ❖ Consensys MythX
- ❖ Consensys Surya
- ❖ Open Zeppelin Code Analyzer
- ❖ Solidity Code Complier

InterFi's Risk Classification

Smart contracts are generally designed to manipulate and hold funds denominated in ETH/BNB. This makes them very tempting attack targets, as a successful attack may allow the attacker to directly steal funds from the contract. Below are the typical risk levels of a smart contract:

Vulnerable: A contract is vulnerable if it has been flagged by a static analysis tool as such. As we will see later, this means that some contracts may be vulnerable because of a false-positive.

Exploitable: A contract is exploitable if it is vulnerable and the vulnerability could be exploited by an external attacker. For example, if the "vulnerability" flagged by a tool is in a function which requires to own the contract, it would be vulnerable but not exploitable.

Exploited: A contract is exploited if it received a transaction on the main network which triggered one of its vulnerabilities. Therefore, a contract can be vulnerable or even exploitable without having been exploited.

Risk severity	Meaning
! Critical	This level vulnerabilities could be exploited easily, and can lead to asset loss, data loss, asset manipulation, or data manipulation. They should be fixed right away.
! High	This level vulnerabilities are hard to exploit but very important to fix, they carry an elevated risk of smart contract manipulation, which can lead to critical risk severity
! Medium	This level vulnerabilities are should be fixed, as they carry an inherent risk of future exploits, and hacks which may or may not impact the smart contract execution.
! Low	This level vulnerabilities can be ignored. They are code style violations, and informational statements in the code. They may not affect the smart contract execution

Smart Contract – Overview

Query	Result
Name	MasterChefV2
Owner	0x0aea551571876cd264b6dda625a415a25ab9d09f
Dev Address	0x5c07a65e525445d9b549a2d3cee4a24b04e5b292

Smart Contract – Static Analysis

Symbol	Meaning
🛑	Function can be modified
💵	Function is payable
🔒	Function is locked
🔐	Function can be accessed
!	Important functionality

```
| **ReentrancyGuard** | Implementation |  |||
| └ | <Constructor> | Internal 🔒 | 🛑  | | |
||||||
| **Context** | Implementation |  |||
| └ | _msgSender | Internal 🔒 |  | | |
| └ | _msgData | Internal 🔒 |  | | |
||||||
| **Ownable** | Implementation | Context |||
| └ | <Constructor> | Internal 🔒 | 🛑  | | |
| └ | owner | Public ! |  |NO! |
| └ | renounceOwnership | Public ! | 🛑  | onlyOwner |
| └ | transferOwnership | Public ! | 🛑  | onlyOwner |
||||||
| **IReferral** | Interface |  |||
| └ | recordReferral | External ! | 🛑  |NO! |
| └ | recordReferralCommission | External ! | 🛑  |NO! |
| └ | getReferrer | External ! |  |NO! |
||||||
| **Address** | Library |  |||
| └ | isContract | Internal 🔒 |  | | |
| └ | sendValue | Internal 🔒 | 🛑  | | |
| └ | functionCall | Internal 🔒 | 🛑  | | |
| └ | functionCall | Internal 🔒 | 🛑  | | |
| └ | functionCallWithValue | Internal 🔒 | 🛑  | | |
| └ | functionCallWithValue | Internal 🔒 | 🛑  | | |
| └ | functionStaticCall | Internal 🔒 |  | | |
| └ | functionStaticCall | Internal 🔒 |  | | |
| └ | _verifyCallResult | Private 🔐 |  | | |
||||||
| **IBEP20** | Interface |  |||
| └ | totalSupply | External ! |  |NO! |
```

└	decimals	External ❗		NO❗
└	symbol	External ❗		NO❗
└	name	External ❗		NO❗
└	getOwner	External ❗		NO❗
└	balanceOf	External ❗		NO❗
└	transfer	External ❗	🛑	NO❗
└	allowance	External ❗		NO❗
└	approve	External ❗	🛑	NO❗
└	transferFrom	External ❗	🛑	NO❗
SafeMath	Library			
└	add	Internal 🔒		
└	sub	Internal 🔒		
└	sub	Internal 🔒		
└	mul	Internal 🔒		
└	div	Internal 🔒		
└	div	Internal 🔒		
└	mod	Internal 🔒		
└	mod	Internal 🔒		
BEP20	Implementation	Context, IBEP20, Ownable		
└	<Constructor>	Public ❗	🛑	NO❗
└	getOwner	External ❗		NO❗
└	name	Public ❗		NO❗
└	symbol	Public ❗		NO❗
└	decimals	Public ❗		NO❗
└	totalSupply	Public ❗		NO❗
└	balanceOf	Public ❗		NO❗
└	transfer	Public ❗	🛑	NO❗
└	allowance	Public ❗		NO❗
└	approve	Public ❗	🛑	NO❗
└	transferFrom	Public ❗	🛑	NO❗
└	increaseAllowance	Public ❗	🛑	NO❗
└	decreaseAllowance	Public ❗	🛑	NO❗
└	mint	Public ❗	🛑	onlyOwner
└	_transfer	Internal 🔒	🛑	
└	_mint	Internal 🔒	🛑	
└	_burn	Internal 🔒	🛑	
└	_approve	Internal 🔒	🛑	
└	_burnFrom	Internal 🔒	🛑	
TheToken	Implementation	BEP20		
└	mint	Public ❗	🛑	onlyOwner
└	delegates	External ❗		NO❗
└	delegate	External ❗	🛑	NO❗
└	delegateBySig	External ❗	🛑	NO❗
└	getCurrentVotes	External ❗		NO❗
└	getPriorVotes	External ❗		NO❗
└	_delegate	Internal 🔒	🛑	
└	_moveDelegates	Internal 🔒	🛑	

└	_writeCheckpoint	Internal 🔒	🛑		
└	safe32	Internal 🔒			
└	getChainId	Internal 🔒			
SafeBEP20	Library				
└	safeTransfer	Internal 🔒	🛑		
└	safeTransferFrom	Internal 🔒	🛑		
└	safeApprove	Internal 🔒	🛑		
└	safeIncreaseAllowance	Internal 🔒	🛑		
└	safeDecreaseAllowance	Internal 🔒	🛑		
└	_callOptionalReturn	Private 🔐	🛑		
MasterChefV2	Implementation	Ownable, ReentrancyGuard			
└	<Constructor>	Public ❗	🛑	NO❗	
└	poolLength	External ❗		NO❗	
└	add	Public ❗	🛑	onlyOwner nonDuplicated	
└	set	Public ❗	🛑	onlyOwner	
└	getMultiplier	Public ❗		NO❗	
└	pendingToken	External ❗		NO❗	
└	massUpdatePools	Public ❗	🛑	NO❗	
└	updatePool	Public ❗	🛑	NO❗	
└	deposit	Public ❗	🛑	nonReentrant	
└	withdraw	Public ❗	🛑	nonReentrant	
└	emergencyWithdraw	Public ❗	🛑	nonReentrant	
└	safeTokenTransfer	Internal 🔒	🛑		
└	updateEmissionRate	Public ❗	🛑	onlyOwner	
└	dev	Public ❗	🛑	NO❗	
└	updateDevTokenRate	Public ❗	🛑	NO❗	
└	setFeeAddress	Public ❗	🛑	NO❗	
└	setTokenReferral	Public ❗	🛑	onlyOwner	
└	setReferralCommissionRate	Public ❗	🛑	onlyOwner	
└	payReferralCommission	Internal 🔒	🛑		

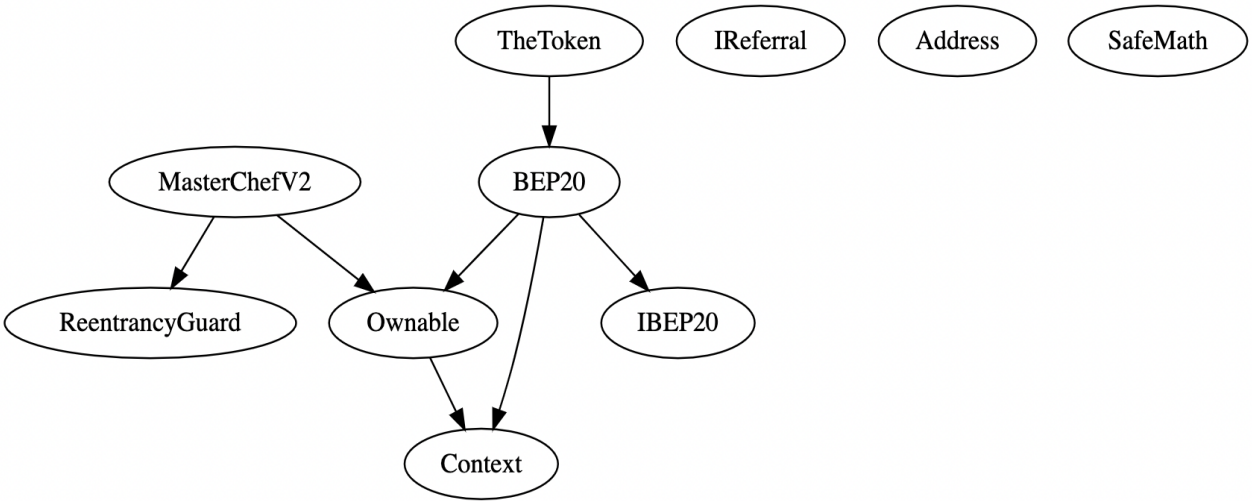
Smart Contract – Software Analysis

Function Signatures

```
16279055  =>  isContract(address)
39509351  =>  increaseAllowance(address,uint256)
119df25f  =>  _msgSender()
8b49d47e  =>  _msgData()
8da5cb5b  =>  owner()
715018a6  =>  renounceOwnership()
f2fde38b  =>  transferOwnership(address)
0c7f7b6b  =>  recordReferral(address,address)
dc1694b8  =>  recordReferralCommission(address,uint256)
4a9fefc7  =>  getReferrer(address)
24a084df  =>  sendValue(address,uint256)
a0b5ffb0  =>  functionCall(address,bytes)
241b5886  =>  functionCall(address,bytes,string)
2a011594  =>  functionCallWithValue(address,bytes,uint256)
d525ab8a  =>  functionCallWithValue(address,bytes,uint256,string)
c21d36f3  =>  functionStaticCall(address,bytes)
dbc40fb9  =>  functionStaticCall(address,bytes,string)
18c2c6a2  =>  _verifyCallResult(bool,bytes,string)
18160ddd  =>  totalSupply()
313ce567  =>  decimals()
95d89b41  =>  symbol()
06fdde03  =>  name()
893d20e8  =>  getOwner()
70a08231  =>  balanceOf(address)
a9059cbb  =>  transfer(address,uint256)
dd62ed3e  =>  allowance(address,address)
095ea7b3  =>  approve(address,uint256)
23b872dd  =>  transferFrom(address,address,uint256)
771602f7  =>  add(uint256,uint256)
b67d77c5  =>  sub(uint256,uint256)
e31bdc0a  =>  sub(uint256,uint256,string)
c8a4ac9c  =>  mul(uint256,uint256)
a391c15b  =>  div(uint256,uint256)
b745d336  =>  div(uint256,uint256,string)
f43f523a  =>  mod(uint256,uint256)
71af23e8  =>  mod(uint256,uint256,string)
a457c2d7  =>  decreaseAllowance(address,uint256)
a0712d68  =>  mint(uint256)
30e0789e  =>  _transfer(address,address,uint256)
4e6ec247  =>  _mint(address,uint256)
6161eb18  =>  _burn(address,uint256)
104e81ff  =>  _approve(address,address,uint256)
a22b35ce  =>  _burnFrom(address,uint256)
40c10f19  =>  mint(address,uint256)
587cde1e  =>  delegates(address)
```

```
5c19a95c  =>  delegate(address)
c3cda520  =>  delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32)
b4b5ea57  =>  getCurrentVotes(address)
782d6fe1  =>  getPriorVotes(address,uint256)
a28a42b3  =>  _delegate(address,address)
955f9fd8  =>  _moveDelegates(address,address,uint256)
ee59e77f  =>  _writeCheckpoint(address,uint32,uint256,uint256)
869d1f83  =>  safe32(uint256,string)
3408e470  =>  getChainId()
71ef09a2  =>  safeTransfer(IBEP20,address,uint256)
fee2e02e  =>  safeTransferFrom(IBEP20,address,address,uint256)
a93f2c02  =>  safeApprove(IBEP20,address,uint256)
e6ae193b  =>  safeIncreaseAllowance(IBEP20,address,uint256)
4988652c  =>  safeDecreaseAllowance(IBEP20,address,uint256)
a096fbe2  =>  _callOptionalReturn(IBEP20,bytes)
081e3eda  =>  poolLength()
f71bf9f7  =>  add(uint256,IBEP20,uint16,bool)
d9638422  =>  set(uint256,uint256,uint16,bool)
8dbb1e3a  =>  getMultiplier(uint256,uint256)
48e43af4  =>  pendingToken(uint256,address)
630b5ba1  =>  massUpdatePools()
51eb05a6  =>  updatePool(uint256)
8dbdbe6d  =>  deposit(uint256,uint256,address)
441a3e70  =>  withdraw(uint256,uint256)
5312ea8e  =>  emergencyWithdraw(uint256)
301d29db  =>  safeTokenTransfer(address,uint256)
0ba84cd2  =>  updateEmissionRate(uint256)
8d88a90e  =>  dev(address)
f7c3b31d  =>  updateDevTokenRate(uint256)
8705fcd4  =>  setFeeAddress(address)
d7d22929  =>  setTokenReferral(IReferral)
55dbc826  =>  setReferralCommissionRate(uint16)
63ec308e  =>  payReferralCommission(address,uint256)
```

Inheritance Graph



Smart Contract – Manual Analysis

Function	Description	Tested	Verdict
Total Supply	provides information about the total token supply	Yes	**Passed**
Balance Of	provides account balance of the owner's account	Yes	**Passed**
Transfer	executes transfers of a specified number of tokens to a specified address	Yes	**Passed**
Approve	allow a spender to withdraw a set number of tokens from a specified account	Yes	**Passed**
Allowance	returns a set number of tokens from a spender to the owner	Yes	**Passed**
Buy Back	is an action in which the project buys back its tokens from the existing holders usually at a market price	NA	NA
Burn	executes transfers of a specified number of tokens to a burn address	NA	NA
Mint	executes creation of a specified number of tokens and adds it to the total supply	Yes	**Passed**
Rebase	circulating token supply adjusts (increases or decreases) automatically according to a token's price fluctuations	NA	NA
Blacklist	stops specified wallets from interacting with the smart contract function modules	NA	NA
Lock	stops or locks all function modules of the smart contract	NA	NA

Note

- Active Owner: **0x0aea551571876cd264b6dda625a415a25ab9d09f**
- **When the smart contract has an active owner address, some of the smart contract functions can be edited, modified or altered.**
- The smart contract uses "MINT" function, the contract owner can mint any number of tokens and add it to the total supply.

```
contract TheToken is BEP20('EDXA Labs Token', 'EDXA') {

/// @notice Creates `_amount` token to `_to`. Must only be called by the owner (MasterChef).
function mint(address _to, uint256 _amount) public onlyOwner {
    _mint(_to, _amount);
    _moveDelegates(address(0), _delegates[_to], _amount);
}
```

- MasterChef's smart contract utilizes "SafeMath" function to avoid common smart contract vulnerabilities.

```
library SafeMath {
function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    require(c >= a, 'SafeMath: addition overflow');

    return c;
  }
  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    return sub(a, b, 'SafeMath: subtraction overflow');
  }
  uint256 c = a * b;
    require(c / a == b, 'SafeMath: multiplication overflow');

    return c;
  }
```

- MasterChef's smart contract utilizes "ReentrancyGuard" to prevent reentrant calls to a function. ReentrancyGuard is a contract module that helps prevent reentrant calls to a function. Inheriting from ReentrancyGuard will make the nonReentrant modifier available, which can be applied to functions to make sure there are no nested (reentrant) calls to them.
- MasterChef's smart contract has 1 low severity issue which may or may not create any functional vulnerability.

{

"resource": " /MasterChef.sol",

"owner": "_generated_diagnostic_collection_name_#0",

"severity": 8, (! Low Severity)

" Expected pragma, import directive or contract/interface/library definition",

"source": "solc",

}

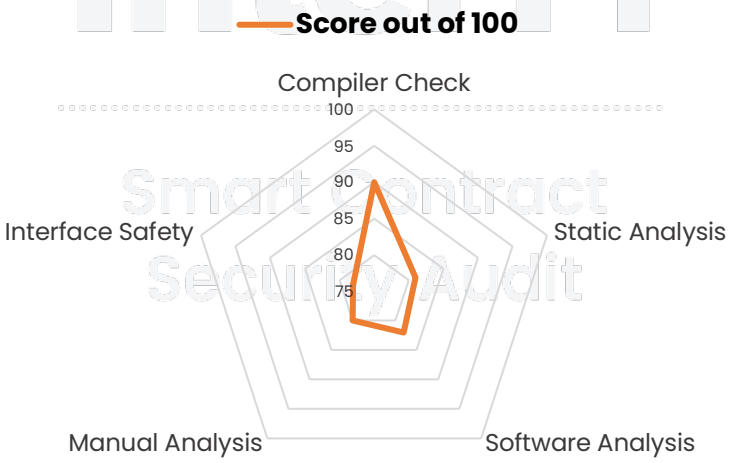
Smart Contract – SWC Attacks

SWC ID	Description	Verdict
SWC-101	Integer Overflow and Underflow	**Passed**
SWC-102	Outdated Compiler Version	**Passed**
SWC-103	Floating Pragma	**! Low**
SWC-104	Unchecked Call Return Value	**Passed**
SWC-105	Unprotected Ether Withdrawal	**Passed**
SWC-106	Unprotected SELFDESTRUCT Instruction	**Passed**
SWC-107	Re-entrancy	**Passed**
SWC-108	State Variable Default Visibility	**Passed**
SWC-109	Uninitialized Storage Pointer	**Passed**
SWC-110	Assert Violation	**Passed**
SWC-111	Use of Deprecated Solidity Functions	**Passed**
SWC-112	Delegate Call to Untrusted Callee	**Passed**
SWC-113	DoS with Failed Call	**Passed**
SWC-114	Transaction Order Dependence	**Passed**
SWC-115	Authorization through tx.origin	**Passed**
SWC-116	Block values as a proxy for time	**Passed**
SWC-117	Signature Malleability	**Passed**
SWC-118	Incorrect Constructor Name	**Passed**

SWC-119	Shadowing State Variables	**Passed**
SWC-120	Weak Sources of Randomness from Chain Attributes	**Passed**
SWC-121	Missing Protection against Signature Replay Attacks	**Passed**
SWC-122	Lack of Proper Signature Verification	**Passed**
SWC-123	Requirement Violation	**Passed**
SWC-124	Write to Arbitrary Storage Location	**Passed**
SWC-125	Incorrect Inheritance Order	**Passed**
SWC-126	Insufficient Gas Griefing	**Passed**
SWC-127	Arbitrary Jump with Function Type Variable	**Passed**
SWC-128	DoS With Block Gas Limit	**Passed**
SWC-129	Typographical Error	**Passed**
SWC-130	Right-To-Left-Override control character (U+202E)	**Passed**
SWC-131	Presence of unused variables	**Passed**
SWC-132	Unexpected Ether balance	**Passed**
SWC-133	Hash Collisions With Multiple Variable Length Arguments	**Passed**
SWC-134	Message call with hardcoded gas amount	**Passed**
SWC-135	Code With No Effects (Irrelevant/Dead Code)	**Passed**
SWC-136	Unencrypted Private Data On-Chain	**Passed**

Smart Contract - Risk Status & Radar Chart

Risk Severity	Status
! Critical	None critical severity issues identified
! High	None high severity issues identified
! Medium	None medium severity issues identified
! Low	**1 low severity issue identified**
Passed	**41 functions and instances verified and passed**



Compiler Check	90
Static Analysis	81
Software Analysis	82
Manual Analysis	80
Interface Safety	78

Auditor's Verdict

InterFi team has performed a line-by-line manual analysis and automated review of the smart contract. The smart contract was analyzed mainly for common smart contract vulnerabilities, exploits, and manipulation hacks.

MasterChef's smart contract source code has LOW RISK SEVERITY.

MasterChef has PASSED the smart contract audit.

MasterChefV2 is a part of EDXA DeFi Laboratory.

Auditor's Note:

- **Be aware that active smart contract owner privileges constitute an elevated impact to smart contract's safety and security.**
- **Smart contract owner can mint tokens and add it to the total supply.**
- Project owner's KYC isn't checked and verified due to out of scope.
- Project's liquidity pair isn't checked and verified due to out of scope.
- Project website is not checked due to out of scope. The website hasn't been reviewed for SSL and lighthouse report.

Important Disclaimer

InterFi Network provides contract auditing and project verification services for blockchain projects. The purpose of the audit is to analyse the on-chain smart contract source code, and to provide basic overview of the project. **This report should not be transmitted, disclosed, referred to, or relied upon by any person for any purposes without InterFi's prior written consent.**

InterFi provides the easy-to-understand assessment of the project, and the smart contract (otherwise known as the source code). The audit makes no statements or warranties on the security of the code. It also cannot be considered as an enough assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have used all the data at our disposal to provide the transparent analysis, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. **Be aware that smart contracts deployed on a blockchain aren't resistant from external vulnerability, or a hack. Be aware that active smart contract owner privileges constitute an elevated impact to smart contract's safety and security. Therefore, InterFi does not guarantee the explicit security of the audited smart contract.**

The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

This report should not be considered as an endorsement or disapproval of any project or team. The information provided on this report does not constitute investment advice, financial advice, trading advice, or any other sort of advice and you should not treat any of the report's content as such. Do conduct your own due diligence and consult your financial advisor before making any investment decisions.

About InterFi Network

InterFi Network provides intelligent blockchain solutions. InterFi is developing an ecosystem that is seamless and responsive. Some of our services: Blockchain Security, Token Launchpad, NFT Marketplace, etc. **InterFi's mission is to interconnect multiple services like Blockchain Security, DeFi, Gaming, and Marketplace under one ecosystem that is seamless, multi-chain compatible, scalable, secure, fast, responsive, and easy-to-use.**

InterFi is built by a decentralized team of UI experts, contributors, engineers, and enthusiasts from all over the world. Our team currently consists of 6+ core team members, and 10+ casual contributors. **InterFi provides manual, static, and automatic smart contract analysis, to ensure that project is checked against known attacks and potential vulnerabilities.**

To learn more, visit https://interfi.network

To view our audit portfolio, visit https://github.com/interfinetwork

To book an audit, message https://t.me/interfiaudits

@INTERFINETWORK

RELENTLESSLY SECURING THE PUBLIC BLOCKCHAIN | MADE IN CANADA