expelee

Building the Futuristic **Blockchain Ecosystem**

Audit Report FOR

Qroniswap

OVERVIEW

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

Audit Result	Passed
KYC Verification	To Be Done
Audit Date	7 Sep 2022

Be aware that smart contracts deployed on the blockchain aren't resistant to internal exploit, external vulnerability, or hack. For a detailed understanding of risk severity, source code vulnerability, functional hack, and audit disclaimer, kindly refer to the audit.

- Team Expelee

PROJECT DESCRIPTION

Qroniswap

Qroniswap is a DeFi protocol and ecosystem covering cross-chain crypto trading and swapping, NFT ticketings, & fiat on-ramp payments. Qroniswap's robust ecosystem is such that allows for deep liquidity and seamless crypto transactions, all spiced up with incentivized earning models which allow users' assets to create passive streams of income while they engage in other activities on the protocol.

🌐 qroni.io

qroniswap

qroniswap

It's always good to check the social profiles of the project, before making your investment.

- Team Expelee

CONTRACT DETAILS

Contract Name
MASTERCHEF

Optimization
No with 200 runs

Contract Address
0xcCED9aA097bcE7c23156f0E35e4F3D594EF8A25c

Network
BSC

Language
Solidity

Compiler
v0.6.12+commit.27d51765

License
MIT license

AUDIT METHODOLOGY

Audit Details

Our comprehensive audit report provides a full overview of the audited system's architecture, smart contract codebase, and details on any vulnerabilities found within the system.

Audit Goals

The audit goal is to ensure that the project is built to protect investors and users, preventing potentially catastrophic vulnerabilities after launch, that lead to scams and rugpulls.

Code Quality

Our analysis includes both automatic tests and manual code analysis for the following aspects:

- Exploits
- Back-doors
- Vulnerability
- Accuracy
- Readability

Tools

- DE
- Open Zeppelin
- Code Analyzer
- Solidity Code
- Complier
- Hardhat

VULNERABILITY CHECKLIST

Design Logic	Passed
Compiler warnings.	Passed
Private user data leaks	Passed
Timestamp dependence	Passed
Integer overflow and underflow	Passed
Race conditions & reentrancy. Cross-function race conditions	Passed
Possible delays in data delivery	Passed
Oracle calls	Passed
Front running	Passed
DoS with Revert	Passed
DoS with block gas limit	Passed
Methods execution permissions	Passed
Economy model	Passed
Impact of the exchange rate on the logic	Passed
Malicious Event log	Passed
Scoping and declarations	Passed
Uninitialized storage pointers	Passed
Arithmetic accuracy	Passed
Cross-function race conditions	Passed
Safe Zeppelin module	Passed
Fallback function security	Passed

RISK CLASSIFICATION

When performing smart contract audits, our specialists look for known vulnerabilities as well as logical and access control issues within the code. The exploitation of these issues by malicious actors may cause serious financial damage to projects that failed to get an audit in time. We categorize these vulnerabilities by the following levels:

High Risk

Issues on this level are critical to the smart contract's performance/functionality and should be fixed before moving to a live environment.

Medium Risk

Issues on this level are critical to the smart contract's performance/functionality and should be fixed before moving to a live environment.

Low Risk

Issues on this level are minor details and warning that can remain unfixed.

Informational

Information level is to offer suggestions for improvement of efficacy or security for features with a risk free factor.

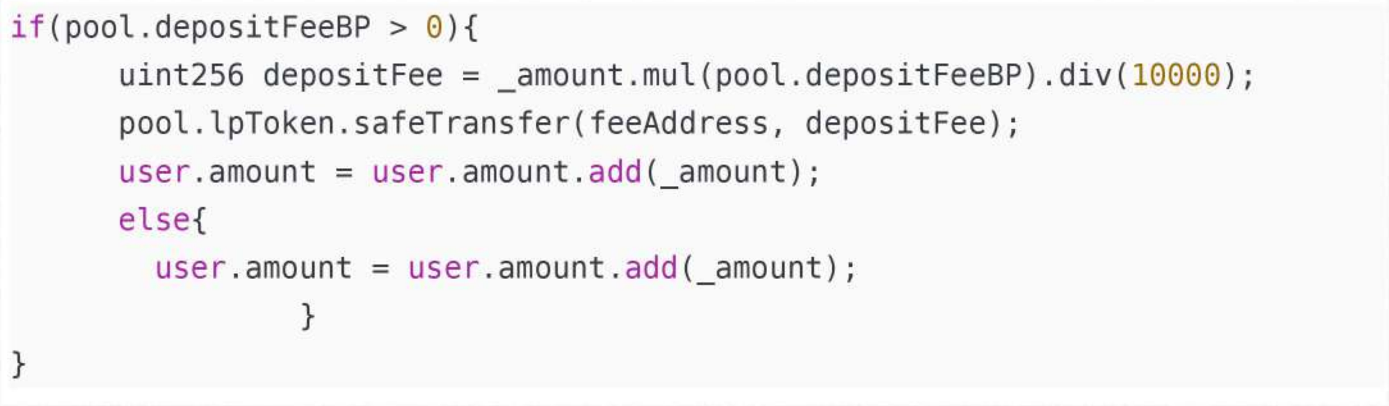
MANUAL AUDIT

Staking Contract

Contract is forked from Sushi Masterchef contract:
https://etherscan.io/address/0xc2edad668740f1aa35e4d8f227fb8e17dca888cd
main parts of contract are same as Sushi Masterchef contract, there is some changes in **deposit, withdraw, add, set** functions which has been audited here.

Centralization Issues:

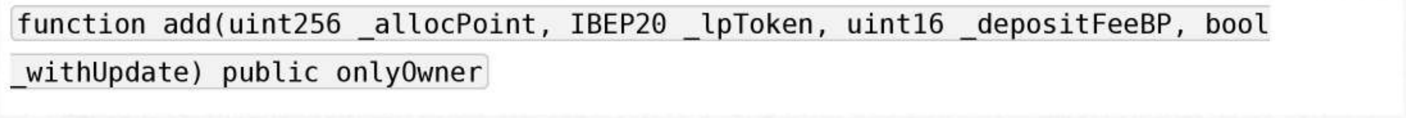
High: Owner is able to set a fee for every pool, there is no limit for this fee and can be up to 100%, this fees will be deducted from a wallet's deposited amount and sent to feeAddress



```
if(pool.depositFeeBP > 0){
      uint256 depositFee = _amount.mul(pool.depositFeeBP).div(10000);
      pool.lpToken.safeTransfer(feeAddress, depositFee);
      user.amount = user.amount.add(_amount);
      else{
        user.amount = user.amount.add(_amount);
                }
}
```

Status : Resolved!, Qroni team applied a limit for depositFee, and now this value can't be more than **20%**

Low - Owner is able to use any token to create a pool and set a high allocation point, even if token doesn't have liqudity.



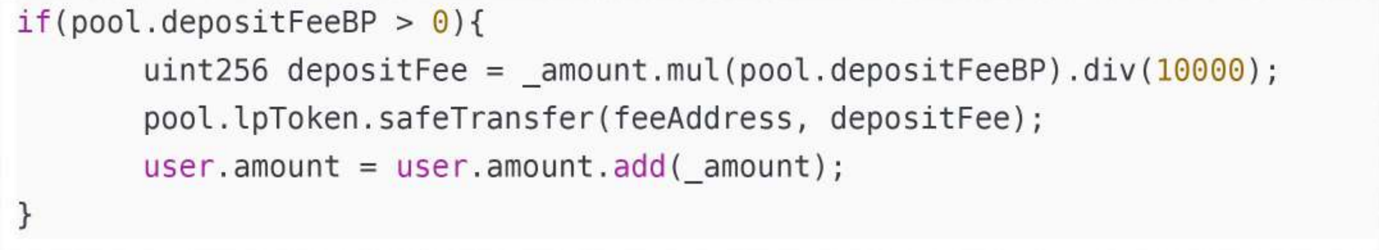
```
function add(uint256 _allocPoint, IBEP20 _lpToken, uint16 _depositFeeBP, bool
_withUpdate) public onlyOwner
```

Low- a malicious owner is able to set stakingWallet to an address that doesn't hold any QNI tokens, hence disabling all rewards and withdraws (emergency withdraw doesn't get affected)

Low- a malicious owner is able to use **updateEmissionRate** and set qniPerBlock to a very big number or even 0

Logical Issues:

Critical - depositFeeBP fee is not deducted from deposited amount, this can disable a wallet from withdrawing his tokens, in this code user.amount is assigned to _amount without first deducting depositFee from _amount



```
if(pool.depositFeeBP > 0){
        uint256 depositFee = _amount.mul(pool.depositFeeBP).div(10000);
        pool.lpToken.safeTransfer(feeAddress, depositFee);
        user.amount = user.amount.add(_amount);
}
```

Status: Resolved!, Qroni team noticed this issue and fixed it.

Low- accidently setting feeAddress to a wrong address is irreversible since only feeAddress owner is able to change feeAddress

Gas Optimizations

GO-1 : devaddr is not used inside contract, delete this variable as well as **dev** function **(applied)**
GO-2: use solidity > 0.8.0 and delete SafeMath to save gas **(applied)**
GO-4: change **getMultiplier** state mutability from view to pure **(applied)**
GO-5 : define **startBlock** as immutable **(applied)**
GO-6 : define this functions as external: u**pdateEmissionRate, setStakingWallet, setFeeAddress, dev (applied)**

Suggestions

- emit an event from this functions **(applied):**
  add, set, updatePool, setFeeAddress, setStakingWallet, updateEmissionRate
- use compiler with version > 0.8.0 **(applied)**

ABOUT EXPELEE

Expelee is a product-based aspirational Web3 Start-up. Coping up with numerous solutions for blockchain Security and constructing a Web3 Ecosystem from Deal making platform to developer hosting open platform, while also developing our own commercial and sustainable blockchain.

www.expelee.com

expeleeofficial

Expelee

expelee_official

expelee

expelee

expelee-co

expelee

Building the Futuristic **Blockchain Ecosystem**

DISCLAIMER

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment. Team provides no guarantees against the sale of team tokens or the removal of liquidity by the project audited in this document.

Always Do your own research and protect yourselves from being scammed. The Expelee team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools.

Under no circumstances did Expelee receive a payment to manipulate those results or change the awarding badge that we will be adding in our website. Always Do your own research and protect yourselves from scams.

This document should not be presented as a reason to buy or not buy any particular token. The Expelee team disclaims any liability for the resulting losses.