SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: XTblock
Date: October 6th, 2021

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for XTblock.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	MasterChef
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/xtblock/binosaur/blob/main/contracts/MasterChef.sol
Commit	47e852c99e12662475c8f979c8822b725ae8397e
Technical Documentation	NO
JS tests	NO
Timeline	24 SEPTEMBER 2021 - 28 SEPTEMBER 2021
Changelog	28 SEPTEMBER 2021 - Initial Audit 06 OCTOBER 2021 - Second Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by XTblock (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contract and its code review conducted between September 24th, 2021 - September 28th, 2021.

Second review conducted on October 6th, 2021.

Scope

The scope of the project is smart contracts in the repository:

Repository:
https://github.com/xtblock/binosaur/blob/main/contracts/MasterChef.sol
Commit:
47e852c99e12662475c8f979c8822b725ae8397e
Technical Documentation: No
JS tests: No
Contracts:
access\Ownable.sol
GSN\Context.sol
math\SafeMath.sol
token\BEP20\BEP20.sol
token\BEP20\IBEP20.sol
token\BEP20\SafeBEP20.sol
utils\Address.sol
utils\Context.sol
MasterChef.sol

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency ▪ Data Consistency
Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart contracts are secured but should be careful with the waitingPoolInfo and poolAllocPointInfo waiting-list array sizes.

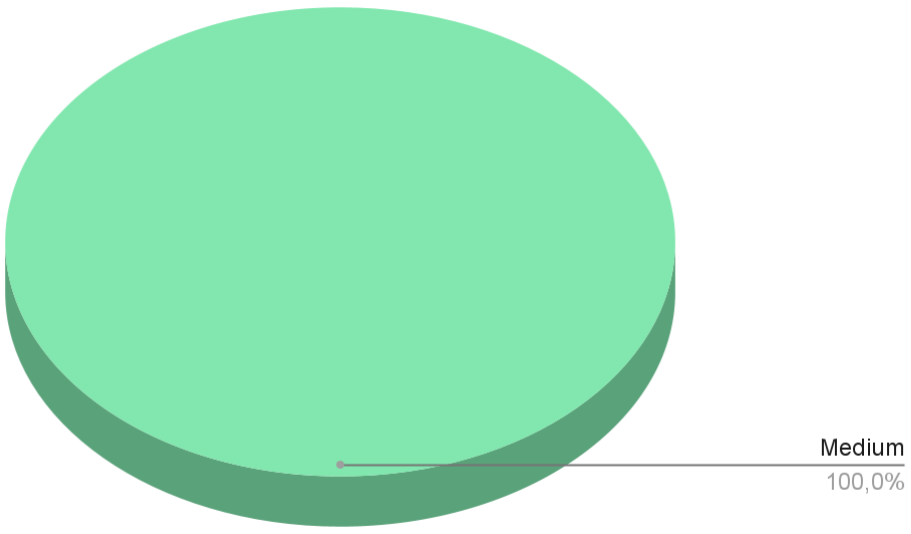
Insecure	Poor secured	Secured	Well-secured
		You are here ↑	

Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. All found issues can be found in the Audit overview section.

As a result of the audit, security engineers found **1** high, **1** medium and **3** low severity issues.

After the second review security engineers found that all main issues were fixed but was added **1** medium severity issue.

Graph 1. The distribution of vulnerabilities after the audit.



Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that can't have a significant impact on execution

Audit overview

🟥 🟥 🟥 🟥 Critical

No critical issues were found.

🟨 🟨 🟨 High

Possible rewards lost or receive more

Changing **allocPoint** in the MasterChef.set method while **_withUpdate** flag set to **false** may lead to rewards lost or receiving rewards more than deserved.

Recommendation: Please call updatePool(_pid) in the case if **_withUpdate** flag is **false** and you don't want to update all pools.

Fixed before the second review

🟩 Medium

1. Privileged ownership

   The owner of the MasterChef contract has permission to updateMultiplier, add new pools, change pool's allocation points and set migrator contract (which will move all LPs from the pool to itself) without community consensus.

   Recommendation: Please consider using one of the following methodologies:

   - Transfer ownership to Time-lock contract with reasonable latency (ie. 24h) so the community may react on changes;
   - Transfer ownership to multi-signature wallet, to prevent single point of failure;
   - Transfer ownership to DAO so the community could deside whether the privileged operations should be executed by voting.

   Status: Created a time-locking feature, so the community now have a minimum of 24h to react to changes.

2. Possibility to get an unreachable contract

   State variables "waitingPoolInfo" and "poolAllocPointInfo" are arrays and not restricted in the length. There is a possiblity when the corresponding "executeAddPools" and "executeUpdateAllocPoint" functions wouldn't be called externally for any reason and those arrays could be filled with a lot of records which will make it impossible to execute corresponding functions because of amount of gas needed will be more than could be taken in the block.

   Recommendation: Please make sure to limit the above arrays. That may be done by checking the array length before pushing a new element and executing some part of the work to decrease its size.

Also, we'd recommend writing a few unit tests to check the gas consumption in the case we have 10-100-1000-10000-100000 records in those arrays.

Lines: MasterChef.sol#166-174

```
waitingPoolInfo.push(PoolInfo({
   lpToken: _lpToken,
   allocPoint: _allocPoint,
   lastRewardBlock: 0,
   accXttPerShare: 0,
   executeTimestamp: _executeTimestamp,
   withUpdate: _withUpdate,
   executed: false
}));
```

Lines: MasterChef.sol#180-183

```
uint256 length = waitingPoolInfo.length;
if(length > 0){
   for (uint256 pid = 0; pid < length; ++pid) {
       PoolInfo storage pool = waitingPoolInfo[pid];
```

Lines: MasterChef.sol#233-239

```
poolAllocPointInfo.push(PoolAllocPointInfo({
   pid: _pid,
   allocPoint: _allocPoint,
   executeTimestamp: _executeTimestamp,
   withUpdate: _withUpdate,
   executed: false
}));
```

Lines: MasterChef.sol#245-248

```
uint256 length = poolAllocPointInfo.length;
if(length > 0){
   for (uint256 index = 0; index < length; ++index) {
       PoolAllocPointInfo storage poolAllocPoint =
poolAllocPointInfo[index];
```

■ Low

1. Unnecessary operations

   When allocPoint is not changed for the pool, there is still an assignment for a new value, which just consumes gas doing nothing.

   Recommendation: Please move "poolInfo[_pid].allocPoint = _allocPoint" assignment inside the *if* block.

   Fixed before the second review

2. Missing Emit Events

   Functions that change critical values should emit events for better off-chain tracking.

Recommendation: Consider adding events when changing critical values, and emit them in the function.

Fixed before the second review

3. A public function that could be declared external

public functions that are never called by the contract should be declared **external** to save gas.

Recommendation: Use the **external** attribute for functions never called from the contract.

Fixed before the second review

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

As a result of the audit, security engineers found **1** high, **1** medium and **3** low severity issues.

After the second review security engineers found that all main issues were fixed but was added **1** medium severity issue.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, 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.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.