Coinsult

Request your audit at coinsult.net

Advanced Manual
Smart Contract Audit

October 26, 2022

CoinsultAudits

info@coinsult.net

coinsult.net

Audit requested by

Eternity Staking

0x21ef3122c35301f9eff3c150640dfe13709022fc

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 Eternity Staking

7. Contract Snapshot

8. Website Review

9. Certificate of Proof

Audit Summary

Project Name	Eternity Staking
Website	https://www.eternityprotocol.net/
Blockchain	Binance Smart Chain
Smart Contract Language	Solidity
Contract Address	0x21ef3122c35301f9eff3c150640dfe13709022fc
Audit Method	Static Analysis, Manual Review
Date of Audit	26 October 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.

Audit Scope

Source Code

Coinsult was comissioned by Eternity Staking to perform an audit based on the following code:

https://bscscan.com/address/0x21ef3122c35301f9eff3c150640dfe13709022fc#code

Note that we only audited the code available to us on this URL at the time of the audit. If the URL is not from any block explorer (main net), it may be subject to change. Always check the contract address on this audit report and compare it to the token you are doing research for.

Ownership held by SAFU dev Trynos

Tokenomics

Not available

Audit Method

Coinsult's manual smart contract audit is an extensive methodical examination and analysis of the smart contract's code that is used to interact with the blockchain. This process is conducted to discover errors, issues and security vulnerabilities in the code in order to suggest improvements and ways to fix them.

Automated Vulnerability Check

Coinsult uses software that checks for common vulnerability issues within smart contracts. We use automated tools that scan the contract for security vulnerabilities such as integer-overflow, integer-underflow, out-of-gas-situations, unchecked transfers, etc.

Manual Code Review

Coinsult's manual code review involves a human looking at source code, line by line, to find vulnerabilities. Manual code review helps to clarify the context of coding decisions. Automated tools are faster but they cannot take the developer's intentions and general business logic into consideration.

Used Tools

- Slither: Solidity static analysis framework
- Remix: IDE Developer Tool
- CWE: Common Weakness Enumeration
- SWC: Smart Contract Weakness Classification and Test Cases
- DEX: Testnet Blockchains

Risk Classification

Coinsult uses certain vulnerability levels, these indicate how bad a certain issue is. The higher the risk, the more strictly it is recommended to correct the error before using the contract.

Vulnerability Level	Description
Informational	Does not compromise the functionality of the contract in any way
Low-Risk	Won't cause any problems, but can be adjusted for improvement
Medium-Risk	Will likely cause problems and it is recommended to adjust
High-Risk	Will definitely cause problems, this needs to be adjusted

Coinsult has four statuses that are used for each risk level. Below we explain them briefly.

Risk Status	Description
Total	Total amount of issues within this category
Pending	Risks that have yet to be addressed by the team
Acknowledged	The team is aware of the risks but does not resolve them
Resolved	The team has resolved and remedied the risk

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

Error Code	Description
CS-01	Wrong comments

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

Wrong comments

```
//30 days
pooldata[7].lockupDuration = 7;
pooldata[7].returnPer = 15000; // 150%

//90 days
pooldata[30].lockupDuration = 30;
pooldata[30].returnPer = 30000; // 300%
```

Recommendation

Add the right comments, or change the values of the lockup duration.

Error Code	Description
SLT: 078	Conformance to numeric notation best practices

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

Too many digits

Literals with many digits are difficult to read and review.

```
uint256 stakeTime = block.timestamp.sub(orderInfo.starttime);
uint256 totalReward =  orderInfo.amount.mul(stakeTime).mul(orderInfo.returnPer).div(10000*365*86400);
uint256 claimAvailableNow =  totalReward.sub(orderInfo.claimedReward);
return claimAvailableNow;
```

Recommendation

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

Exploit scenario

```
contract MyContract{
    uint 1_ether = 10000000000000000000;
}
```

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

Error Code	Description
SLT: 056	Missing Zero Address Validation

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

No zero address validation for some functions

Detect missing zero address validation.

```
function setToken(IERC20 _token) external onlyOwner {
    token = _token;
}
```

Recommendation

Check that the new address is not zero.

Exploit scenario

```
contract C {

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

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

Bob calls `updateOwner` without specifying the `newOwner`, soBob loses ownership of the contract.

Error Code	Description
SWC-104	CWE-252: Unchecked Return Value

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

Unchecked transfer

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

```
function withdrawOtherTokens(IERC20 _token) external onlyOwner {
   require(IERC20(_token) != IERC20(token), "Can't withdraw reward token!");
   uint256 contract_balance = IERC20(_token).balanceOf(address(this));
   IERC20(_token).transfer(address(owner()) , contract_balance);
}
```

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..

Error Code	Description
SLT: 054	Missing Events Arithmetic

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

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function setToken(IERC20 _token) external onlyOwner {
    token = _token;
}
```

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

Error Code	Description
CEN-100	Centralization: Operator Priviliges

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

⚠️ Owner can set emergency withdraw fees to 35% max

⚠️ Owner can set token

⚠️ Owner can toggle staking

⚠️ Owner can withdraw tokens from the contract, not the reward token

Notes

Notes by Eternity Staking

No notes provided by the team.

Notes by Coinsult

No notes provided by Coinsult

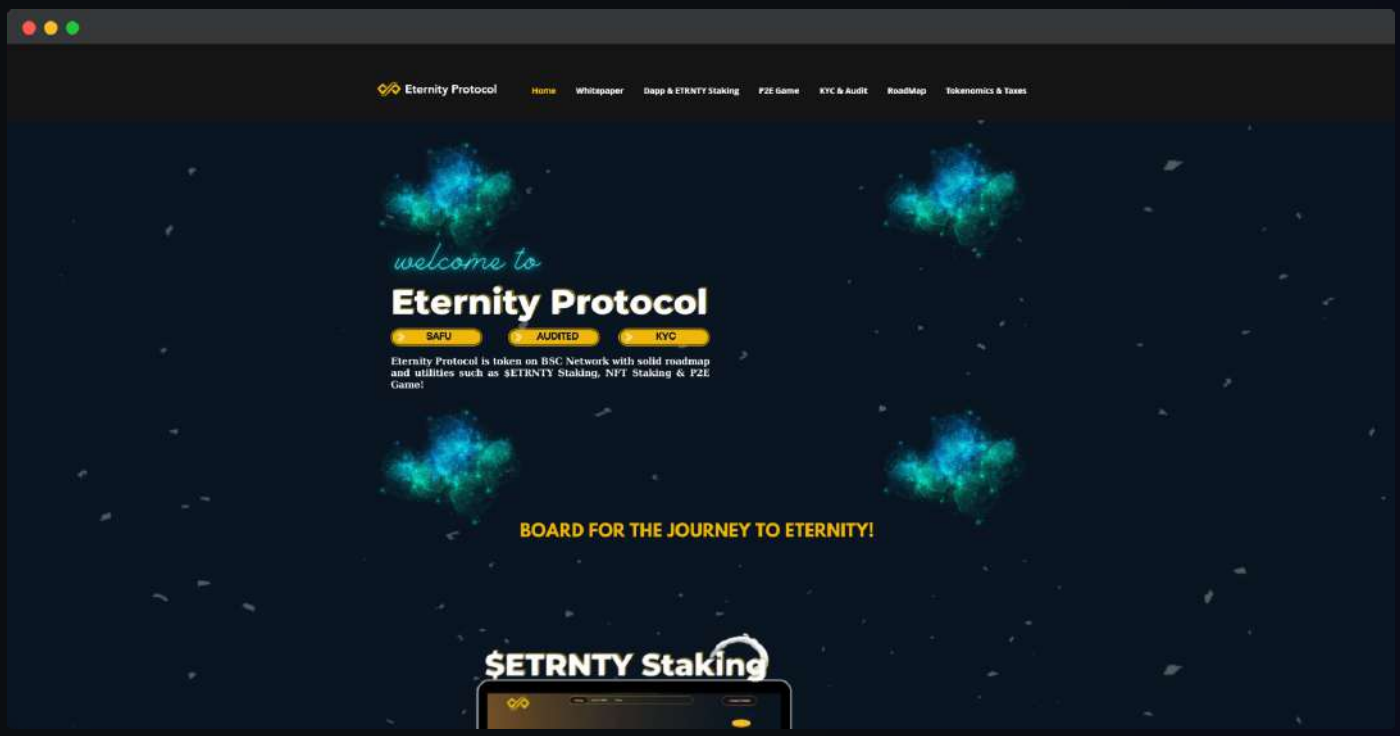
Contract Snapshot

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

```
contract EternityStaking is ReentrancyGuard, Ownable {
 using SafeMath for uint256;
 using SafeERC20 for IERC20;

 struct PoolInfo {
     uint256 lockupDuration;
     uint256 returnPer;
 }
 struct OrderInfo {
     address beneficiary;
     uint256 amount;
     uint256 lockupDuration;
     uint256 returnPer;
     uint256 starttime;
     uint256 endtime;
     uint256 claimedReward;
     bool claimed;
 }
 IERC20 public token;
 bool public started = true;
 uint256 private latestOrderId;
 uint256 public emergencyWithdrawFees; // 10% ~ 1000
 uint256 public totalStake;
 uint256 public totalWithdrawal;
 uint256 public totalRewardsDistribution;
 uint256 public totalRewardPending;

 mapping(uint256 =&gt; PoolInfo) public pooldata;
 /// @dev balanceOf[investor] = balance
 mapping(address =&gt; uint256) public balanceOf;
 mapping(address =&gt; uint256) public totalRewardEarn;
 mapping(uint256 =&gt; OrderInfo) public orders;
 mapping(address =&gt; uint256[]) private orderIds;

 constructor(
    address _token,
    bool _started,
```

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

Coinsult

coinsult.net

End of report
Smart Contract Audit

CoinsultAudits

info@coinsult.net

coinsult.net