



Advanced Manual Smart Contract Audit

October 21, 2022

 [CoinsultAudits](#)

 info@coinsult.net

 coinsult.net

Audit requested by



Adeys Advantage

0x00f7c7dc31c26C5Bf319a691DA42dbBec88abC99

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 Adeys Advantage

7. Contract Snapshot

8. Website Review

9. Certificate of Proof

Audit Summary

Project Name	Adeys Advantage
Website	https://adeysadvantage.com/
Blockchain	Binance Smart Chain
Smart Contract Language	Solidity
Contract Address	0x00f7c7dc31c26C5Bf319a691DA42dbBec88abC99
Audit Method	Static Analysis, Manual Review
Date of Audit	21 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 commissioned by Adeys Advantage to perform an audit based on the following code:

<https://bscscan.com/address/0x00f7c7dc31c26C5Bf319a691DA42dbBec88abC99#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.

Staking Contract

Tokenomics

– No tokenomics

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	7	0	7	0
 Medium-Risk	1	0	0	1
 High-Risk	0	0	0	0

Error Code	Description
CS-01	Typos in variables

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

Typos in variables

```
uint256 public depoiteTax=1000;

struct depoite{
    uint256 amount;
    uint256 depositTime;
    bool isToken;
    uint256 checkPointBNB;
    uint256 checkPointBUSD;
}

event Compund(address indexed user, uint256 amount);

function getContractBNBBalacne() public view returns(uint256 totalBNB){
    totalBNB=address(this).balance;
}
```

Recommendation

Fix the typos for better readability.

Error Code	Description
SWC-107	CWE-841: Improper Enforcement of Behavioral Workflow

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

Contract contains Reentrancy vulnerabilities

Additional information: This combination increases risk of malicious intent. While it may be justified by some complex mechanics (e.g. rebase, reflections, buyback).

```
function invest(uint256 investment,address reffer) public payable {
    user storage users = investor[msg.sender];
    uint256 amount;
    if(msg.value==0 && investment>0){
        amount=investment;
        require(amount<=token.allowance(msg.sender, address(this)),"Insufficient Allowance to t
    uint256 tax=amount.mul(depoiteTax).div(divider);
    uint256 refferalFee=amount.mul(refferalTax).div(divider);

    token.transferFrom(msg.sender, treasury, tax);
    token.transferFrom(msg.sender, address(this), amount.sub(tax));

    if(reffer==address(0) || reffer==msg.sender||reffer==address(this)){
        users.deposits.push(depoite(amount.sub(tax), block.timestamp,true,0,block.timestamp));
    }else{
        users.refferAddress=reffer;
```

Recommendation

Apply the check-effects-interactions pattern.

Exploit scenario

```
function withdrawBalance(){
    // send userBalance[msg.sender] Ether to msg.sender
    // if mgs.sender is a contract, it will call its fallback function
    if( ! (msg.sender.call.value(userBalance[msg.sender])() ) ){
        throw;
    }
    userBalance[msg.sender] = 0;
}
```

Bob uses the re-entrancy bug to call withdrawBalance two times, and withdraw more than its initial deposit to the contract.

Error Code	Description
SWC-116	CWE-829: Inclusion of Functionality from Untrusted Control Sphere

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

Avoid relying on `block.timestamp`

`block.timestamp` can be manipulated by miners.

```
if(reffer==address(0) || reffer==msg.sender||reffer==address(this)){
  users.deposits.push(depoite(amount.sub(tax), block.timestamp,true,0,block.timestamp));
}else{
  users.refferAddress=reffer;
  investor[msg.sender].refferalRewardsBUSD+=refferalFee;
  users.deposits.push(depoite(amount.sub(tax), block.timestamp,true,0,block.timestamp));
}
```

Recommendation

Do not use `block.timestamp`, `now` or `blockhash` as a source of randomness

Exploit scenario

```
contract Game {

    uint reward_determining_number;

    function guessing() external{
        reward_determining_number = uint256(block.blockhash(10000)) % 10;
    }
}
```

Eve is a miner. Eve calls `guessing` and re-orders the block containing the transaction. As a result, Eve wins the game.

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 setWallet( address _treasury, address _devWallet) public onlyOwner{
    treasury=_treasury;
    devWallet=_devWallet;
}
```

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, so Bob 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 withdrawRewardBUSD()public {
    (uint256 totalRewards,)=calculateReward(msg.sender);
    require(totalRewards>0,"No Rewards Found");
    require(totalRewards<=getContractBUSDBalacne(),"Not Enough Token for withdrwal from contract");
    uint256 tax=totalRewards.mul(rewardTax).div(divider);
    uint256 taxR=totalRewards.mul(withdrawRTax).div(divider);
    totalDevRewardsBUSD+=tax;
    token.transfer(msg.sender, totalRewards.sub(taxR));
    if(investor[msg.sender].refferAddress!=address(0)) investor[investor[msg.sender].refferAddress].refferAddress=msg.sender;
    for(uint256 i=0;i<investor[msg.sender].deposites.length;i++){
        if(investor[msg.sender].deposites[i].isToken) investor[msg.sender].deposites[i].checkPointBUSD=block.timestamp;
    }
    investor[msg.sender].totalRewardWithdrawBUSD+=totalRewards;
    investor[msg.sender].checkBusd=block.timestamp;
    totalWithdrawBUSD+=totalRewards;
    emit RewardWithdraw(msg.sender, totalRewards);
}
```

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: 038	Imprecise arithmetic operations order

● **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.

```
if(time=21 days && time<40 days){
    reward6+=users.deposits[i].amount.mul(percentage[2]).div(divider).mul(time.sub(40 days)).div(1 day);
    reward5+=users.deposits[i].amount.mul(percentage[1]).div(divider).mul(20 days).div(1 days);
    reward4+=users.deposits[i].amount.mul(percentage[0]).div(divider).mul(20 days).div(1 days);
}
```

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.

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 setTax(uint256 _withdrawTax) public onlyOwner{
    require(_withdrawTax>=800 && _withdrawTax<=2000,"Withdraw Fees Must be in the range 800-2000");
    withdrawTax=_withdrawTax;
}
```

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 Privileges

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

 Owner can set taxes between 8 and 20 percent

Notes

Notes by Adeys Advantage

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 staking is Ownable {  
    using SafeMath for uint256;  
  
    address public devWallet;  
  
    address public treasury;  
  
    uint256 private divider=10000;  
  
    uint256 public depoiteTax=1000;  
  
    uint256 public withdrawTax=800;  
  
    uint256 public withdrawRTax=500;
```

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

Adeys Advantage

Audited by Coinsult.net




Date: 21 October 2022

✓ Advanced Manual Smart Contract Audit

End of report

Smart Contract Audit

 CoinsultAudits

 info@coinsult.net

 coinsult.net

Request your smart contract audit / KYC

t.me/coinsult_tg