Coinsult

Request your audit at coinsult.net

Advanced Manual
Smart Contract Audit

November 21, 2022

CoinsultAudits

info@coinsult.net

coinsult.net

Audit requested by

Adeys Advantage

0xAC19A9c36169374E0f90aa039c8B897b727fd092

Table of Contents

Audit Summary

Project Name	Adeys Advantage
Website	https://adeysadvantage.com/
Blockchain	Binance Smart Chain
Smart Contract Language	Solidity
Contract Address	0xAC19A9c36169374E0f90aa039c8B897b727fd092
Audit Method	Static Analysis, Manual Review
Date of Audit	21 November 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

Coinsult was comissioned by Adeys Advantage to perform an audit based on the following code:

https://bscscan.com/address/0xac19a9c36169374e0f90aa039c8b897b727fd092#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

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

SWC Attack Analysis

The Smart Contract Weakness Classification Registry (SWC Registry) is an implementation of the weakness classification scheme proposed in EIP-1470. It is loosely aligned to the terminologies and structure used in the Common Weakness Enumeration (CWE) while overlaying a wide range of weakness variants that are specific to smart contracts.

ID	Description	Status
SWC-100	Function Default Visibility	Passed
SWC-101	Integer Overflow and Underflow	Passed
SWC-102	Outdated Compiler Version	Passed
SWC-103	Floating Pragma	Passed
SWC-104	Unchecked Call Return Value	Passed
SWC-105	Unprotected Ether Withdrawal	Passed
SWC-106	Unprotected SELFDESTRUCT Instruction	Passed
SWC-107	Reentrancy	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	Delegatecall 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	Passed
SWC-136	Unencrypted Private Data On-Chain	Passed

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	8	0	8	0
● Medium-Risk	0	0	0	0
● 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

```
struct depoite{
    uint256 amount;
    uint256 depositeTime;
    bool isToken;
    uint256 checkPointWBNB;
    uint256 checkPointBUSD;
}
```

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 &amp;&amp; investment&gt;0){
        amount=investment;
        require(amount&lt;=token.allowance(msg.sender, address(this)),&quot;Insufficient Allowence 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.deposites.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.deposites.push(depoite(amount.sub(tax), block.timestamp,true,0,block.timestamp));
}else{
    users.refferAddress=reffer;
    investor[msg.sender].refferalRewardsBUSD+=refferalFee;
    users.deposites.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`, 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 withdrawRewardBUSD()public {
    (uint256 totalRewards,)=calclulateReward(msg.sender);
    require(totalRewards&gt;0,"No Rewards Found");
    require(totalRewards&lt;=getContractBUSDBalacne(),&quot;Not Enough Token for withdrwal from contrac
    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].ref
    for(uint256 i=0;i&lt;investor[msg.sender].deposites.length;i++){
        if(investor[msg.sender].deposites[i].isToken) investor[msg.sender].deposites[i].checkPointBUSD=
    }
    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 &amp;&amp; time40 days){
    reward6+=users.deposites[i].amount.mul(percentage[2]).div(divider).mul(time.sub(40 days)).div(1 day
    reward5+=users.deposites[i].amount.mul(percentage[1]).div(divider).mul(20 days).div(1 days);
    reward4+=users.deposites[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&gt;=800 &amp;&amp; _withdrawTax&lt;=2000,&quot;Withdraw Fees Must be in the ra
    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.

Error Code	Description
CS: 071	Using safemath in Solidity 0.8.0+

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

Using safemath in Solidity 0.8.0+

SafeMath is generally not needed starting with Solidity 0.8, since the compiler now has built in overflow checking.

```
library SafeMath {
/**
 * @dev Returns the addition of two unsigned integers, with an overflow flag.
 *
 * _Available since v3.4._
 */
function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
    unchecked {
        uint256 c = a + b;
        if (c < a) return (false, 0);
        return (true, c);
    }
}

/**
 * @dev Returns the substraction of two unsigned integers, with an overflow flag.
```

Recommendation

Check if you really need SafeMath and consider removing it.

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 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

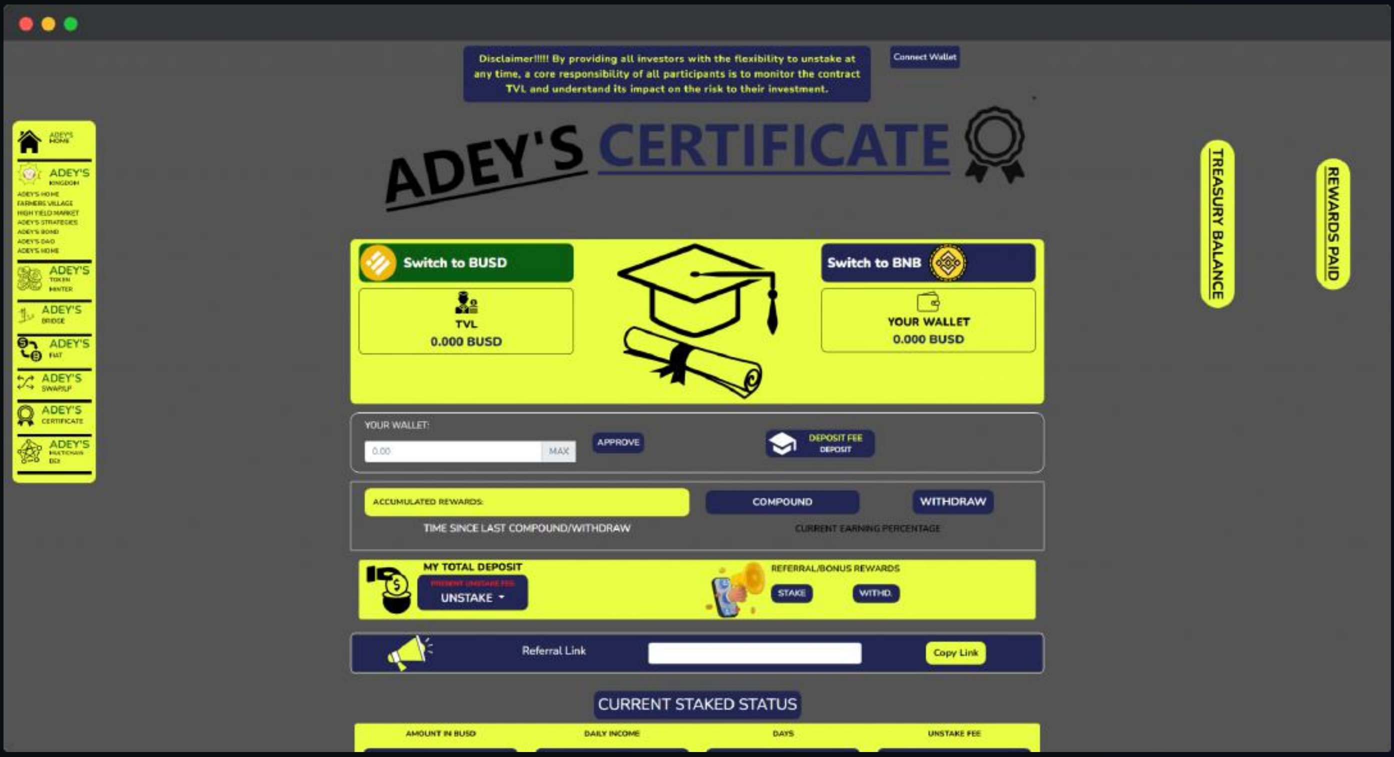
Contract Snapshot

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.

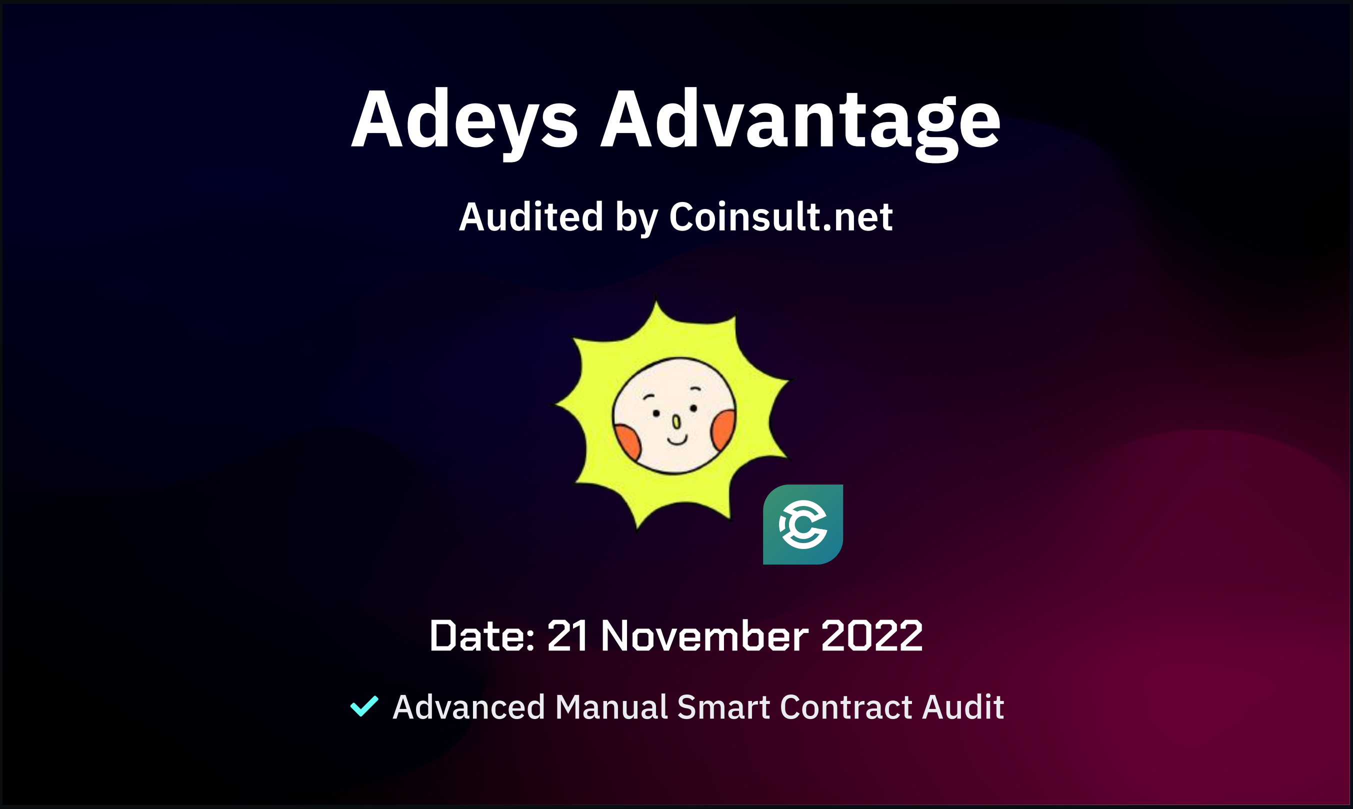


Disclaimer!!!!! By providing all investors with the flexibility to unstake at any time, a core responsibility of all participants is to monitor the contract TVL and understand its impact on the risk to their investment.

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 November 2022

✓ Advanced Manual Smart Contract Audit

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.

End of report
Smart Contract Audit

CoinsultAudits

info@coinsult.net

coinsult.net

Request your smart contract audit / KYC

t.me/coinsult_tg