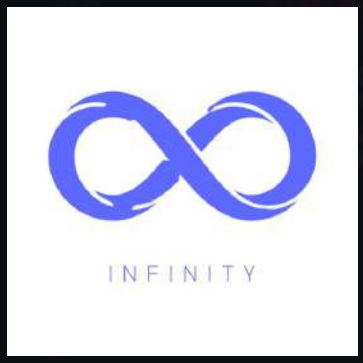
Coinsult

Advanced Manual Smart Contract Audit



Project: Infinity Trust

Website: https://infinitytrusts.com/

Low-Risk	Medium-Risk	High-Risk
7 low-risk code issues found	0 medium-risk code issues found	0 high-risk code issues found

Contract Address

0x233DeA98806F91d175d23068cea548aDBb1876fE

Disclaimer: Coinsult is not responsible for any financial losses. Nothing in this contract audit is financial advice, please do your own research.

Disclaimer

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.

Tokenomics

Rank	Address	Quantity (Token)	Percentage
1	0x9cbf9fd973f9d6df2bf66742a7b6c26caea143c8	598,000,000	59.8000%
2	0x4e708abb624f2813fc0b010b6ad2ad4b1c98d0e4	402,000,000	40.2000%

Source Code

Coinsult was comissioned by Infinity Trust to perform an audit based on the following smart contract:

https://bscscan.com/address/0x233DeA98806F91d175d23068cea548aDBb1876fE#code

Manual Code Review

In this audit report we will highlight all these issues:

Low-Risk

7 low-risk code
issues found

Medium-Risk

0 medium-risk code
issues found

High-Risk

0 high-risk code
issues found

The detailed report continues on the next page...

🟢 **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).

More information: Slither

```
function _transferFrom(
    address sender,
    address recipient,
    uint256 amount
) internal returns (bool) {
    // antiwhale
    if ((sender == pair) || (recipient == pair)) {
        if (
            !isAntiWhaleEnded() &amp;amp;
            !(addLPaddress[sender] || addLPaddress[recipient])
        ) {
            require(amount &lt;= antiWhaleAmount, &quot;AntiWhale&quot;);
        }
    }
    // antiwhale

    if (inSwap) {
        return _basicTransfer(sender, recipient, amount);
    }
    if (shouldRebase()) {
        rebase();
    }
```

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.

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

Avoid relying on block.timestamp

block.timestamp can be manipulated by miners.

```
router.swapExactTokensForETHSupportingFeeOnTransferTokens(
            amountToSwap,
            0,
            path,
            address(this),
            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.

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

No zero address validation for some functions

Detect missing zero address validation.

```
function setTrashBin(address newTrashBin) public onlyOwner {
    _trashBin = newTrashBin;
}
```

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.

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

Functions that send Ether to arbitrary destinations

Unprotected call to a function sending Ether to an arbitrary address.

```
function swapBack() internal swapping {
    uint256 amountToSwap = _gonBalances[address(this)].div(
        _gonsPerFragment
    );

    if (amountToSwap == 0) {
        return;
    }

    uint256 balanceBefore = address(this).balance;
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = router.WETH();

    router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        amountToSwap,
        0,
        path,
        address(this),
        block.timestamp
    );
```

Recommendation

Ensure that an arbitrary user cannot withdraw unauthorized funds.

Exploit scenario

```
contract ArbitrarySend{
    address destination;
    function setDestination(){
        destination = msg.sender;
    }

    function withdraw() public{
        destination.transfer(this.balance);
    }
}
```

Bob calls `setDestination` and `withdraw`. As a result he withdraws the contract's balance.

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

Unchecked transfer

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

```
function cleanPhishingToken(address _token) public onlyOwner {
        IERC20(_token).transfer(
            msg.sender,
            IERC20(_token).balanceOf(address(this))
        );
    }

    //burn bnb sent to contract
    function sweepBNB(uint256 _amount) public onlyOwner {
        uint256 _balance = address(this).balance;
        require(_balance &gt;= _amount);
        payable(_trashBin).transfer(_amount);
    }
}
```

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..

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

Write after write

Variables that are written but never read and written again.

```
function swapBack() internal swapping {
    uint256 amountToSwap = _gonBalances[address(this)].div(
        _gonsPerFragment
    );

    if (amountToSwap == 0) {
        return;
    }

    uint256 balanceBefore = address(this).balance;
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = router.WETH();

    router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        amountToSwap,
        0,
        path,
        address(this),
        block.timestamp
    );
```

Recommendation

Fix or remove the writes.

Exploit scenario

```
```solidity
contract Buggy{
    function my_func() external initializer{
        // ...
        a = b;
        a = c;
        // ..
    }
}
```

`a` is first asigned to `b`, and then to `c`. As a result the first write does nothing.
```

🟢 **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.

```
uint256 times = deltaTime.div(6 minutes);
        uint256 epoch = times.mul(6);
```

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.

Owner privileges

- Owner cannot set fees higher than 25%
- Owner cannot pause trading
- Owner can change max transaction amount
- Owner can exclude from fees
- Owner can set time between transaction

Extra notes by the team

No notes

Contract Snapshot

```
contract InfinityFinance is ERC20Detailed {
using SafeMath for uint256;

event LogRebase(uint256 indexed epoch, uint256 totalSupply);

string public constant _name = "Infinity Trust";
string public constant _symbol = "INF";
uint8 public constant _decimals = 5;

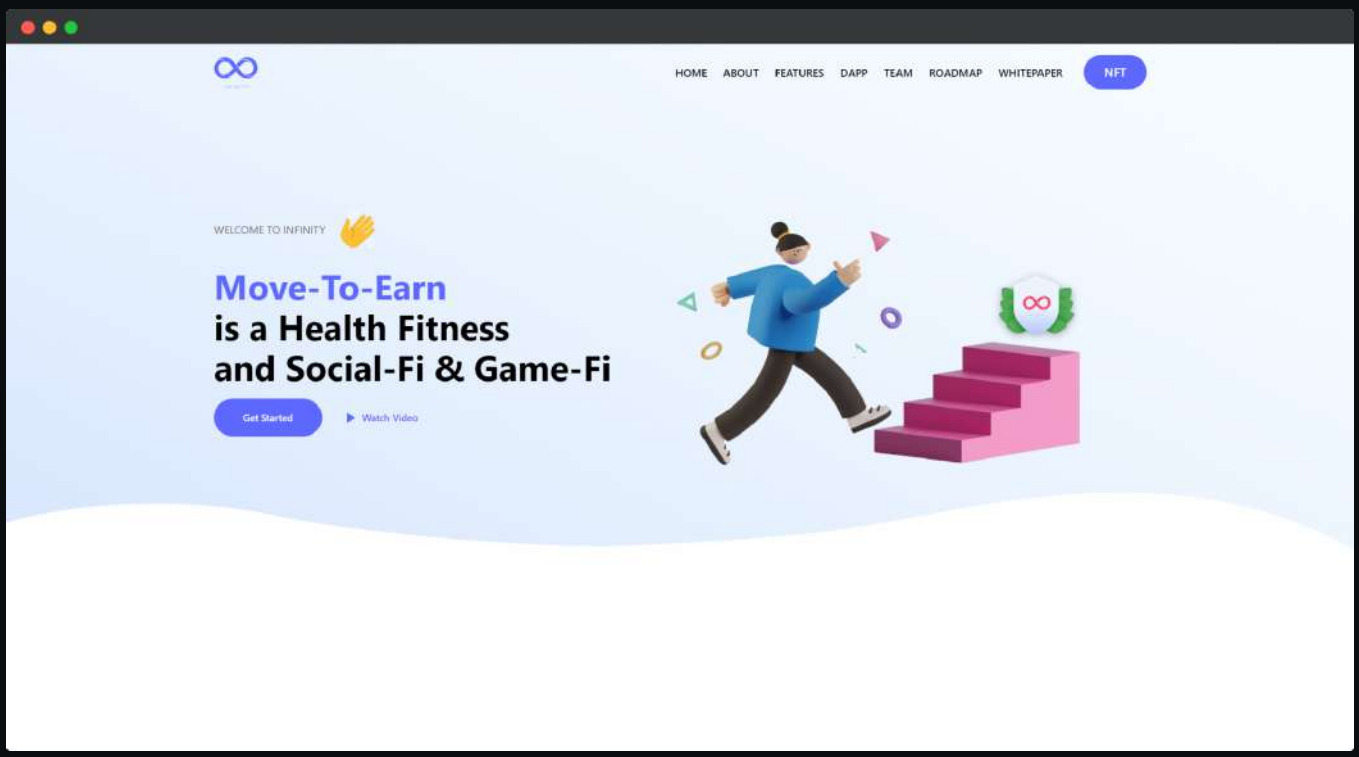
IPancakeSwapPair public pairContract;
mapping(address =&gt; bool) _isFeeExempt;

modifier validRecipient(address to) {
    require(to != address(0x0));
    _;
}

uint256 public constant DECIMALS = 5;
uint256 public constant MAX_UINT256 = ~uint256(0);
uint8 public constant RATE_DECIMALS = 7;
```

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.



- Mobile Friendly
- Does not contain jQuery errors
- SSL Secured
- No major spelling errors

Project Overview

Not KYC verified by Coinsult