Coinsult

Advanced Manual Smart Contract Audit

Project: BeeCoin

Website: https://www.beecoin.in/

Low-Risk	Medium-Risk	High-Risk
3 low-risk code issues found	1 medium-risk code issues found	0 high-risk code issues found

Contract Address

0x248f2fd8A34303d8F1B7Ba65A1848eF90ae4F72F

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	0x2c37687624cb313efbc6931776ddd682f08c1586	283,342,065.609275240428867031	56.6684%
2	BeeCoin: Deployer	148,515,101.175877812618351194	29.7030%
3	PancakeSwap V2: BCO 23	33,220,174.005900422757405068	6.6440%
4	0xf6545ee42c7d2d37a2845e6699f1aaba23606090	15,534,913.516976297245355541	3.1070%
5	0x64aaf1401447eaf1af4a8b425ab3a7d75c6228fb	15,389,792.004858992171623579	3.0780%

Source Code

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

https://bscscan.com/address/0x248f2fd8A34303d8F1B7Ba65A1848eF90ae4F72F#code

Manual Code Review

In this audit report we will highlight all these issues:

🟢 Low-Risk

3 low-risk code
issues found

🟡 Medium-Risk

1 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.

Divide before multiply

Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.

```
_rOwned[owner] = _rTotal.div(1000).mul(995);
_rOwned[feeWallet] = _rTotal.div(1000).mul(5);
```

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.

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

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function setFeesPercentage(uint256 feesPercentage) external onlyOwner() checkIsFeesValid(feesPercent
    _feesPercentage = feesPercentage;
}
```

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.

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

Costly operations inside a loop

Costly operations inside a loop might waste gas, so optimizations are justified.

```
function includeAccount(address account) external onlyOwner() {
    require(_isExcluded[account], "Account is already excluded");
    for (uint256 i = 0; i &lt; _excluded.length; i++) {
        if (_excluded[i] == account) {
            _excluded[i] = _excluded[_excluded.length - 1];
            _tOwned[account] = 0;
            _isExcluded[account] = false;
            _excluded.pop();
            break;
        }
    }
}
```

Recommendation

Use a local variable to hold the loop computation result.

Exploit scenario

```
contract CostlyOperationsInLoop{

    function bad() external{
        for (uint i=0; i < loop_count; i++){
            state_variable++;
        }
    }

    function good() external{
      uint local_variable = state_variable;
      for (uint i=0; i < loop_count; i++){
        local_variable++;
      }
      state_variable = local_variable;
    }
}
```

Incrementing `state_variable` in a loop incurs a lot of gas because of expensive `SSTORE`s, which might lead to an `out-of-gas`.

🟡 **Medium-Risk:** Should be fixed, could bring problems.

Wrong error code

```
function includeAccount(address account) external onlyOwner() {
    require(_isExcluded[account], "Account is already excluded");
    for (uint256 i = 0; i &lt; _excluded.length; i++) {
        if (_excluded[i] == account) {
            _excluded[i] = _excluded[_excluded.length - 1];
            _tOwned[account] = 0;
            _isExcluded[account] = false;
            _excluded.pop();
            break;
        }
```

Recommendation

Change the comment ‘excluded’ to included to prevent misconceptions.

Owner privileges

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

Extra notes by the team

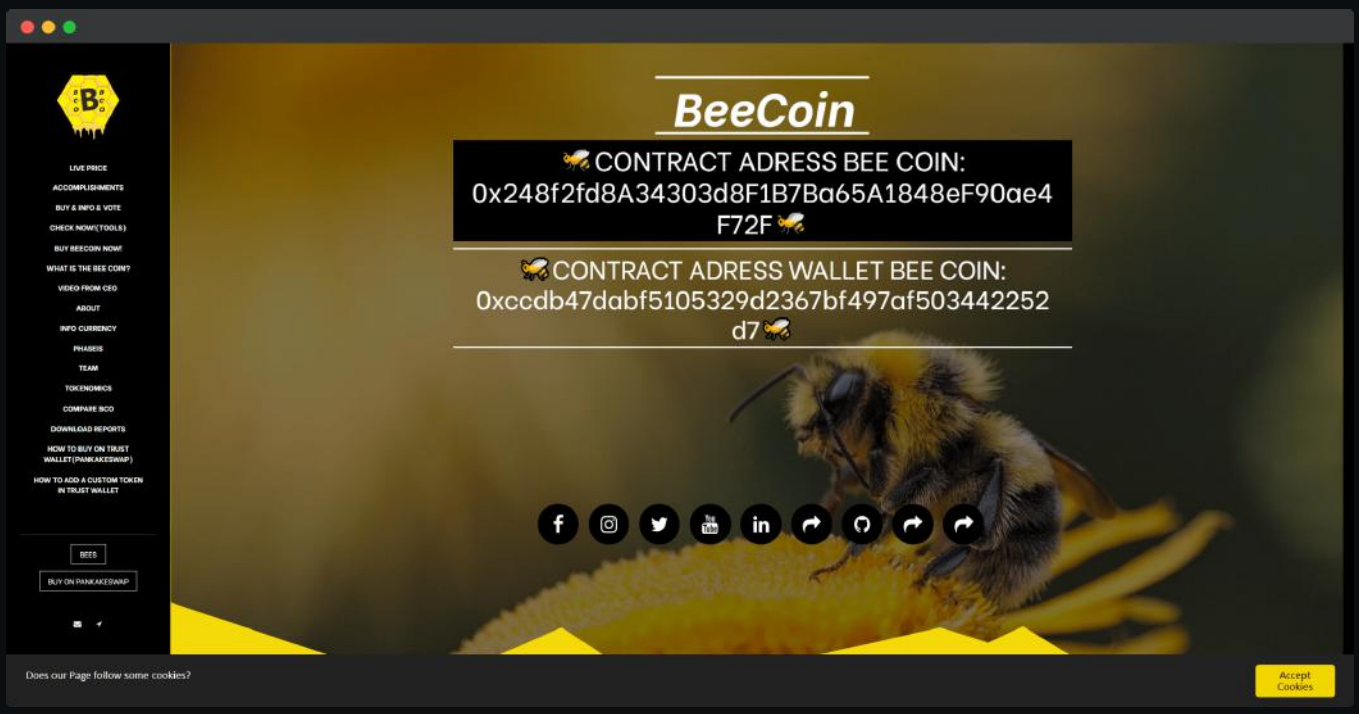
No notes

Contract Snapshot

```
contract TaxableTeamToken is IERC20, Context, Ownable {
using SafeMath for uint256;
using Address for address;

mapping (address =&gt; uint256) private _rOwned;
mapping (address =&gt; uint256) private _tOwned;
mapping (address =&gt; mapping (address =&gt; uint256)) private _allowances;

mapping (address =&gt; bool) private _isExcluded;
address[] private _excluded;

uint256 private constant MAX = ~uint256(0);
uint256 private _tTotal;
uint256 private _rTotal;
uint256 private _tFeeTotal;

string private _name;
string private _symbol;
uint8 private _decimals;
uint256 private _feesPercentage;

modifier checkIsAddressValid(address ethAddress)
{
    require(ethAddress != address(0), "[Validation] invalid address");
    require(ethAddress == address(ethAddress), "[Validation] invalid address");
    _;
}
```

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

KYC verified by Coinsult

BeeCoin

Completed KYC Verification at Coinsult.net

Date: 11 June 2022

✔ Project Owner Identified

✔ Contract: 0x248f2fd8A34303d8F1B7Ba65A1848eF90ae4F72F

BeeCoin

Audited by Coinsult.net

Date: 11 June 2022

✔ Advanced Manual Smart Contract Audit