Smart contract security audit

DCIP

v.1.2

Table of Contents

1.0 Introduction

1.1 Project engagement

During June of 2021, DCIP engaged CTDSec to audit smart contracts that they created. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts. DCIP provided CTDSec with access to their code repository and whitepaper.

1.2 Disclaimer

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the network's fast-paced and rapidly changing environment, we at CTDSec recommend that DCIP team put in place a bug bounty program to encourage further and active analysis of the smart contract.

2.0 Coverage

2.1 Target Code and Revision

For this audit, we performed research, investigation, and review of the DCIP contract followed by issue reporting, along with mitigation and remediation instructions outlined in this report. The following code files are considered in-scope for the review:

Source:

- DCIP.sol [SHA256] - 46a587b7885be9403efbd9280fc1b7715e9245ae4f43893df629239213b55b2a
- Presale.sol [SHA256] - e867f74f5f93989c87cd1c18245c0701f9e166e74655105c9a2c2c392a26c99e
- Voting.sol [SHA256] - b5f7af6fa4cde92e64e647bb585b953ccfa855ac42ce750850170174a275f562

2.2 Attacks made to the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices.

№	Issue description.	Checking status
1	Compiler warnings.	PASSED
2	Race conditions and Reentrancy. Cross-function race conditions.	PASSED
3	Possible delays in data delivery.	PASSED
4	Oracle calls.	PASSED
5	Front running.	PASSED
6	Timestamp dependence.	PASSED
7	Integer Overflow and Underflow.	PASSED
8	DoS with Revert.	PASSED
9	DoS with block gas limit.	PASSED
10	Methods execution permissions.	PASSED
11	Economy model. If application logic is based on an incorrect economic model, the application would not function correctly and participants would incur financial losses. This type of issue is most often found in bonus rewards systems, Staking and Farming contracts, Vault and Vesting contracts, etc.	HIGH ISSUES - SOLVED BY DEV TEAM
12	The impact of the exchange rate on the logic.	PASSED
13	Private user data leaks.	PASSED
14	Malicious Event log.	PASSED
15	Scoping and Declarations.	PASSED
16	Uninitialized storage pointers.	PASSED
17	Arithmetic accuracy.	PASSED

18	Design Logic.	HIGH ISSUES - SOLVED BY DEV TEAM
19	Cross-function race conditions.	PASSED
20	Safe Zeppelin module.	PASSED
21	Fallback function security.	PASSED
22	Overpowered functions / Owner privileges	OVERPRIVILEGED OWNER - SOLVED BY DEV TEAM

3.0 Security Issues

3.1 High severity issues [3] - Fixed

DCIP Contract:

1. Calculation error Issue:

The function _burnTokenFromWallet() subtracts rBurn from _rOwned only if the account is not excluded.

The function _burnTokenFromWallet() does not decrease allowance of account for the owner.

The function _burnToken subtracts only from _tOwned.

Recommendation: Subtract rBurn from _rOwned in any case.

Dev update: Fixed file DCIPv2.sol [SHA256]:

b5b5744fe577311be3a5fd74cc7cdbefb547ac04a25a693a6971da1c10ff68a9

https://github.com/DCIP-Finance/smart-contracts/blob/main/contracts/DCIP.sol

```
function _burnTokenFromWallet(address sender, uint256 _burnAmount) private {
    if (_burnAmount == 0) {
        return;
    }

    uint256 balance = balanceOf(sender);
    require(balance >= _burnAmount, "DCIP: burn amount exceeds balance");

    uint256 reflectedAmount = _burnAmount.mul(_getRate());

    // remove the amount from the sender's balance first
    _reflectOwned[sender] = _reflectOwned[sender].sub(reflectedAmount);
    if (_isExcluded[sender])
        _takeOwned[sender] = _takeOwned[sender].sub(_burnAmount);

    _burnTokens(sender, _burnAmount, reflectedAmount);
}
```

2. Allowance error Issue:

The owner can burn an unlimited number of tokens from any account.

Recommendation: Check allowance for owner or write about that possibility in your white paper.

Dev update: Fixed file DCIPv2.sol [SHA256]:

b5b5744fe577311be3a5fd74cc7cdbefb547ac04a25a693a6971da1c10ff68a9

https://github.com/DCIP-Finance/smart-contracts/blob/main/contracts/DCIP.sol

Owner can't burn more tokens from external accounts.

3. Economy model issue Issue:

The contract burns tokens by sending them to _tOwned[_burnAddress] and excludes that address from reward. _burnAddress can be included to reward.

_marketingWalletAddress and _communityInvestWalletAddress are excluded but also can be included to reward.

Recommendation: The contract logic is written for distributing tokens to these addresses correctly only if they are excluded.

The function includeInReward() should check and disallow including of these addresses. Or you can rewrite the distribution logic.

Dev update: Fixed file DCIPv2.sol [SHA256]:

b5b5744fe577311be3a5fd74cc7cdbefb547ac04a25a693a6971da1c10ff68a9

https://github.com/DCIP-Finance/smart-contracts/blob/main/contracts/DCIP.sol

```
function includeInReward(address account) external onlyOwner() {
    require(_isExcluded[account], "Account is already included");

    // This is a fictional boundary to prevent out of gass exceptions in the loop below. The excluded list should be tiny.
    require(
        _excluded.length < 1000,
        "There are too many excluded addresses"
    );

    require(
        !_isForeverExcludedFromReward[account],
        "Account can not be included"
    );
    for (uint256 i = 0; i < _excluded.length; i++) {
        if (_excluded[i] == account) {
            _excluded[i] = _excluded[_excluded.length - 1];
            _takeOwned[account] = 0;
            _isExcluded[account] = false;
            _excluded.pop();
            break;
        }
    }
}
```

3.2 Medium severity issues [1] - Fixed

Presale contract

1. Function declaration Issue:

The function withdraw() declared as payable.

Recommendation:

That kind of functions do not receive eth/bnb and should not be payable

Dev update - issue fixed:

```
function withdraw() public {
    require(deposits[msg.sender] > 0, "invalid deposit amount");
    require(whitelist[msg.sender] == true, "invalid withdraw address");

    uint256 tokenAmount = getCalculatedAmount(msg.sender);
    require(tokenAmount > 0, "invalid token amount");
    token.transfer(msg.sender, tokenAmount);
    withdraws[msg.sender] = withdraws[msg.sender].add(tokenAmount);
    emit Withdrawn(msg.sender, tokenAmount);
}
```

3.3 Low severity issues [1] - Fixed

DCIP contract:

1. Out of gas Issue:

The function includeInReward() uses the loop to find and remove addresses from the _excluded list. Function will be aborted with OUT_OF_GAS exception if there will be a long excluded addresses list.

The function _getCurrentSupply also uses the loop for evaluating total supply. It also could be aborted with OUT_OF_GAS exception if there will be a long excluded addresses list.

Recommendation:

Check that the excluded array length is not too big.

Dev update - Fixed:

```
function includeInReward(address account) external onlyOwner() {
    require(_isExcluded[account], "Account is already included");

    // This is a fictional boundary to prevent out of gass exceptions in the loop below. The excluded list should be tiny.
    require(
        _excluded.length < 1000,
        "There are too many excluded addresses"
    );

    require(
        !_isForeverExcludedFromReward[account],
        "Account can not be included"
    );
    for (uint256 i = 0; i < _excluded.length; i++) {
        if (_excluded[i] == account) {
            _excluded[i] = _excluded[_excluded.length - 1];
            _takeOwned[account] = 0;
            _isExcluded[account] = false;
            _excluded.pop();
            break;
        }
    }
}
```

```
 function _getCurrentSupply() private view returns (uint256, uint256) {
     // This is a fictional boundary to prevent out of gass exceptions in the loop below. The excluded list should be tiny.
     require(
         _excluded.length < 1000,
         "There are too many excluded addresses"
     );

     uint256 rSupply = _rTotal;
     uint256 tSupply = _tTotal;
     for (uint256 i = 0; i < _excluded.length; i++) {
         if (
             _reflectOwned[_excluded[i]] > rSupply ||
             _takeOwned[_excluded[i]] > tSupply
         ) return (_rTotal, _tTotal);
         rSupply = rSupply.sub(_reflectOwned[_excluded[i]]);
         tSupply = tSupply.sub(_takeOwned[_excluded[i]]);
     }
     if (rSupply < _rTotal.div(_tTotal)) return (_rTotal, _tTotal);
     return (rSupply, tSupply);
 }
```

3.4 Informational severity issues [5] - Fixed

DCIP contract:

1. Constant variables Issue:

Functions calculateCommunityFee(), calculateMarketingFee(), calculateBurnFee() and calculateTaxFee() calculate fees dependent on _holderToTimestamp list and use different numbers inside of these functions.

Recommendation: Create two types of constant variables. First type will be used when the current block.timestamp is less than (_holderToTimestamp[_msgSender()] + 24 hours), second for the rest.

Dev update - Fixed:

```
function calculateCommunityFee(uint256 _amount, bool _heldLessThan24Hours)
    private
    pure
    returns (uint256)
{
    if (_heldLessThan24Hours) {
        return _amount.mul(6).div(10**2);
    } else {
        return _amount.mul(2).div(10**2);
    }
}
```

Presale contract:

2. Contract's decimals

According to logic in getCalculatedAmount() function token that used in presale should have the same decimal as BNB.

Dev update - Fixed:

```
function getCalculatedAmount(address _address)
    public
    view
    returns (uint256)
{
    uint256 totalAmount = deposits[_address] * rate;

    if (
        now > presaleEndTimestamp.add(2 days) && withdraws[msg.sender] ==
    ) {
        return totalAmount.div(5);
    } else if (
        now > presaleEndTimestamp.add(32 days) &&
        withdraws[msg.sender] == totalAmount.div(5)
    ) {
        return totalAmount.div(5);
    } else if (
        now > presaleEndTimestamp.add(62 days) &&
        withdraws[msg.sender] == totalAmount.div(5).mul(2)
    ) {
        return totalAmount.div(5);
    } else if (
        now > presaleEndTimestamp.add(92 days) &&
        withdraws[msg.sender] == totalAmount.div(5).mul(3)
    ) {
        return totalAmount.div(5);
    } else if (
        now > presaleEndTimestamp.add(122 days) &&
        withdraws[msg.sender] == totalAmount.div(5).mul(4)
    ) {
        return totalAmount.div(5);
    }
    return 0;
}
```

3. Unused variables

The variable presale is unused. The variable Expired in VotingState structure is unused.

Dev update - Variable is deleted. Fixed.

Voting contract:

4. Unused code Issue:

The Voter structure has unused variables: vote and weight.

The function vote() calculates voter weight by dividing voter balance to total supply of the myToken. Total supply of the myToken should always be greater than or equal to the balance of a certain address. There is only one case when voter weight value equals to 1 - voter has all tokens of myToken contract.

The variable endedAt is unused.

The variable presale is unused. Recommendation: Remove unused code.

Dev update - unused variables are deleted.

5. Access error

The onlyOwner() modifier has uncommon logic.

If the owner address is equal to 0x9bF6Fbd80DBE0dFa0f05B3cBc111D46cbb1D055a, any account will be able to call functions with onlyOwner modifier.

Dev update - Fixed:

```
modifier onlyOwner() {
    require(_owner == _msgSender(), "Ownable: caller is not the owner");
    _;
}
```

4.0 Owner Privileges

DCIP:

Owner can burn an unlimited number of tokens from any account. FIXED.

Owner can change excluded from fee and excluded from reward lists.

Owner can change tax and liquidity percent.

Owner can change the max transaction amount by passing percent of total supply.

Owner can enable and disable swap and liquify logic.

Owner can lock and unlock. FIXED.

Voting:

Owner(chairperson) can add proposal.

Owner can start voting.

Owner can force terminate voting.

Presale:

Owner can withdraw any tokens from the contract address to himself. FIXED.

Owner can withdraw BNB from the contract to himself even if presale in active status. FIXED.

Owner can change:

1. white list;
2. presale address;

5.0 Summary of the audit

Development team improved and fixed all issues that were found in v1. Contract does not contain any issue and it's safe to be deployed.

Fixed file DCIPv2.sol [SHA256]:

b5b5744fe577311be3a5fd74cc7cdbefb547ac04a25a693a6971da1c10ff68a9

Fixed file Presalev2.sol [SHA256]:
f68ae6f7ae64bc68956d849c488ff59eae4ba6bd77d6d873ec2ff1b079f62470

Fixed file Votingv2.sol [SHA256]:
b9176f6ce3a0f35f24571546e601248ebc4bc44324a0c8670559530c11fb6068