BEOSIN
Blockchain Security

DP

Smart Contract Security Audit

V1.0

No. 202210242230

Oct 24th, 2022

Contents

Summary of Audit Results **1**

1 Overview **3**

1.1 Project Overview 3

1.2 Audit Overview 3

2 Findings **4**

[DP-1] Centralization risk 5

[DP-2] The remaining USDT cannot be withdrew 6

[DP-3] The deadFeePercent value limit is incorrect 7

[DP-4] Insufficient reward blacklist settings 8

[DP-5] Unreasonable setting of key parameters 10

[DP-6] The _totalSupply was not updated when the token was destroyed 11

[DP-7] Missing event trigger 12

[DP-8] Redundant Code 14

3 Appendix **15**

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts 15

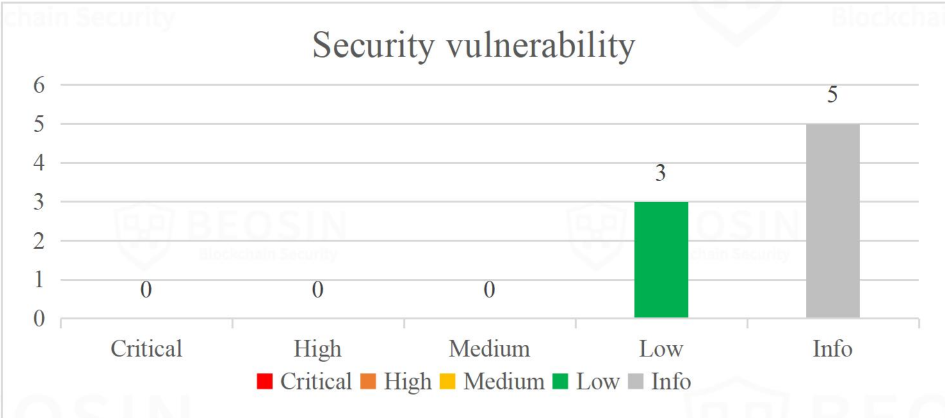
3.2 Audit Categories 17

3.3 Disclaimer 19

3.4 About BEOSIN 20

Summary of Audit Results

After auditing, 3 Low-risk and 5 Info items were identified in the DP project. Specific audit details will be presented in the **Findings section**. Users should pay attention to the following aspects when interacting with this project:



***Notes:**

- **Risk Description:**

1. This contract designs reward mechanism that rewards should be given during each transfer operation. However, since the reward factor is equal to zero, the reward will not be applied. User should pay attention that they will not receive the reward in the current contract.
2. The implementation of the numerical limit logic of deadFeePercent is incorrect. User should pay attention to the value of deadFeePercent when transferring.
3. There is asset centralization risk that this project mint all token to one address during contract construction, and there is no additional way to increase supply.
4. A small amount of USDT remains in this contract and cannot be withdrawn.
5. The LP token obtained by adding liquidity to the corresponding pair will be sent to the lpReceiver address, which is specified by the owner.

- **Project Description:**

1. Business overview

The DP is a BEP-20 token issued on BNB Chain. The total supply of DP is 210 thousand, which can not be minted and can be burned (transfer to the dead address). The contract will mint the total supply of tokens to the deployer address when the contract is deployed. The deployer will be granted owner permission when the contract is deployed. The owner can set important variables such as whitelist, fee rate, etc. This contract has a reward mechanism. During each transfer operation, the contract takes a percentage of the fees and transfers them to a different address, including the current contract address. When the balance of the current contract exceeds a certain threshold, the liquidity increase operation will be triggered.

2. Basic Token Information

Token name	DPToken
Token symbol	DP
Decimals	18
Pre-mint	210,000 (All to deployer)
Total supply	210,000 (burnable)
Token type	BEP-20

Table 1 Basic information of DP token

1 Overview

1.1 Project Overview

Project Name	DP
Platform	BNB Chain
Contract Address	0x69A19e89689AF92A431D5391D15ee9ece854d8fF

1.2 Audit Overview

Audit work duration：October 20, 2022 - October 24, 2022

Audit methods: Formal Verification, Static Analysis, Typical Case Testing and Manual Review.

Audit team: Beosin Security Team.

2 Findings

Index	Risk description	Severity level	Status
DP-1	Centralization risk	Low	Acknowledged
DP-2	The remaining USDT cannot be withdrew	Low	Acknowledged
DP-3	The deadFeePercent value limit is incorrect	Low	Acknowledged
DP-4	Insufficient reward blacklist settings	Info	Acknowledged
DP-5	Unreasonable setting of key parameters	Info	Acknowledged
DP-6	The _totalSupply was not updated when the token was destroyed	Info	Acknowledged
DP-7	Missing event trigger	Info	Acknowledged
DP-8	Redundant Code	Info	Acknowledged

Status Notes:

1. DP-1 is unfixed and will cause centralization risk.

2. DP-2 is unfixed and will cause a certain number of USDT be locked in the contract forever.

3. DP-3 is unfixed. The deadFeePercent value limit is incorrect, which may cause the destruction ratio to be inconsistent with the expectation.

4. DP-4 is unfixed and will cause the actual number of tokens destroyed will be higher than expected.

5. DP-5 is unfixed and will cause to users not being rewarded.

6. DP-6 is unfixed and will not cause any issues.

7. DP-7 is unfixed and will not cause any issues.

8. DP-8 is unfixed and will not cause any issues.

Finding Details:

[DP-1] Centralization risk

Severity Level Low

Type Business Security

Lines DP.sol #L1582

Description When the contract is deployed, all tokens are allocated to the deployer's account through the *_mint* function, which has the risk of centralization of token allocation.

```
1579        whitelist[msg.sender] = true;
1580        emit AddWhitelist(msg.sender);
1581
1582        _mint(msg.sender, 210000 * BASE_RATIO);
1583
1584
1585        bytes memory bytecode = type(SmartVault).creationCode;
1586        bytes32 salt = keccak256(abi.encodePacked(address(this)));
1587        address _smartVault;
1588
```

Figure 1 The source code of related code

Recommendations It is recommended to use multi-signature wallet or DAO governance to manage assets in the deployer address.

Status Acknowledged.

[DP-2] The remaining USDT cannot be withdrew

Severity Level	Low
Type	Business Security
Lines	DP.sol #L1860-1876
Description	The *swapAndLiquidy* function converts half of the contractTokenBalance DP tokens to USDT. The other half of DP tokens and part of the converted USDT are deposited into the DP-USDT pool on PancakeSwap as liquidity. For every *swapAndLiquify* function call, a small amount of USDT leftover in the contract. This is because the price of DP drops after swapping the first half of DP tokens into USDT, and the other half of DP tokens require less than the converted USDT to be paired with it when adding liquidity. The contract doesn't appear to provide a way to withdraw those USDT, and they will be locked in the contract forever.

```
1860    function swapAndLiquify() private lockTheSwap {
1861        uint256 contractTokenBalance = balanceOf(address(this));
1862        uint256 half = contractTokenBalance.div(2);
1863        uint256 otherHalf = contractTokenBalance.sub(half);
1864
1865        uint256 initialBalance = usdtToken.balanceOf(smartVault);
1866
1867        swapTokensForToken(half, address(this), address(usdtToken), smartVault);
1868
1869        uint256 newBalance = usdtToken.balanceOf(smartVault).sub(
1870            initialBalance
1871        );
1872
1873        addLiquidity(newBalance, otherHalf);
1874
1875        emit SwapAndLiquify(half, newBalance, otherHalf);
1876    }
1877
```

Figure 2 The source code of *swapAndLiquify* function

Recommendations	It is recommended to add the function of drawing USDT in the contract.
Status	Acknowledged.

[DP-3] The deadFeePercent value limit is incorrect

Severity Level	**Low**
Type	Business Security
Lines	DP.sol #L1636-1642 DP.sol #L1511-1518 DP.sol #L1660-1664
Description	Irrelevant parameter input in *setDeadFeePercent* function. devFeePercent as input parameter is not relevant to deadFeePercent. Meanwhile, deadFeePercent is not included in CurrentAllFee. It doesn't make sense that devFeePercent as input of *checkMaxFeeLimit* modifier to check the condition and update the value of deadFeePercent.

```
1636    function setDeadFeePercent(uint256 percent)
1637        external
1638        onlyOwner
1639        checkMaxFeeLimit(devFeePercent, percent)
1640    {
1641        deadFeePercent = percent;
1642    }
```

Figure 3 The source code of *setDeadFeePercent* function

```
1511    uint256 public fundFeePercent = (1 * BASE_RATIO) / 100;
1512    uint256 public marketFeePercent = (9 * BASE_RATIO) / 100;
1513    uint256 public devFeePercent = (0 * BASE_RATIO) / 100;
1514    uint256 public deadFeePercent = (0 * BASE_RATIO) / 100;
1515    uint256 public liquidityFeePercent = (0 * BASE_RATIO) / 100;
1516    uint256 public nftPoolFeePercent = (0 * BASE_RATIO) / 100;
1517    uint256 public currentAllFee =
1518        fundFeePercent + marketFeePercent + devFeePercent + liquidityFeePercent + nftPoolFeePercent;
1519
```

Figure 4 The source code of related code

```
1660    modifier checkMaxFeeLimit(uint256 oldFee, uint256 newFee) {
1661        _;
1662        currentAllFee = currentAllFee.sub(oldFee).add(newFee);
1663        require(currentAllFee < MAX_FEE, "trade fee exceeded maximum limit");
1664    }
```

Figure 5 The source code of *checkMaxFeeLimit* modifier

Recommendations	It is recommended to correctly write the *setDeadFeePercet* function and set a reasonable range for the deadFeePercet according to the business situation.
Status	Acknowledged.

[DP-4] Insufficient reward blacklist settings

Severity Level	Info
Type	Business Security
Lines	DP.sol #L1793-1803 DP.sol #L1361-1369 DP.sol #L1972-1977 DP.sol #L1958-1970 DP.sol #L1575-1577
Description	Rewards will still be calculated for tokens transferred to 0xdEaD for destruction, and the actual number of tokens destroyed will be higher than expected.

```
1793        if(from != liquidity && to != liquidity){
1794            deadFee = amount.mul(deadFeePercent).div(BASE_RATIO);
1795        }
1796        else{
1797            deadFee = amount.mul(deadFeePercent).div(BASE_RATIO);
1798        }
1799
1800        if (dev != address(0) && deadFee > 0) {
1801            realAmount = realAmount.sub(deadFee);
1802            super._transfer(account, dead, deadFee);
1803        }
```

Figure 6 The source code of related code

```
1361    function _transfer(
1362        address sender,
1363        address recipient,
1364        uint256 amount
1365    ) internal virtual {
1366        require(sender != address(0), "ERC20: transfer from the zero address");
1367        require(recipient != address(0), "ERC20: transfer to the zero address");
1368
1369        _beforeTokenTransfer(sender, recipient, amount);
```

Figure 7 The source code of *_transfer* function

```
1972    function _beforeTokenTransfer(
1973        address from,
1974        address to,
1975        uint256 amount
1976    ) internal virtual override calculateReward(from) calculateReward(to) {}
1977 }
```

Figure 8 The source code of *_beforeTokenTransfer* function

```
modifier calculateReward(address account) {
    if (account != address(0)) {
        uint256 reward = getReward(account);
        if (reward > 0) {

            _balances[account] = _balances[account].add(reward);
            extraSupply = extraSupply.add(reward);
        }

        lastUpdateTime[account] = lastTime();
    }
    _;
}
```

Figure 9 The source code of *calculateReward* modifier

```
function getReward(address account) public view returns (uint256) {

    if (lastUpdateTime[account] == 0 || rewardBlacklist[account]) {
        return 0;
    }
    return
        _balances[account].mul(SPY).div(BASE_RATIO).mul(
            lastTime().sub(lastUpdateTime[account])
        );
}
```

Figure 10 The source code of *getReward* function

```
        setRewardBlacklist(liquidity, true);
        setRewardBlacklist(address(this), true);
        setRewardBlacklist(msg.sender,true);
```

Figure 11 The source code of related code

Recommendations	It is recommended to add the destruction address to the reward blacklist.
Status	Acknowledged.

[DP-5] Unreasonable setting of key parameters

Severity Level	Info
Type	Business Security
Lines	DP.sol #L1504 DP.sol #L1942-1951

Description

The initial value of constant SPY is 0 and can't be update. This constant is used when calculating rewards, which will result in the reward value always being 0.

```
1504    uint256 public constant SPY = (0 * BASE_RATIO) / 10000 / 1 days;
```

Figure 12 The source code of related code

```
1942  function getReward(address account) public view returns (uint256) {
1943
1944      if (lastUpdateTime[account] == 0 || rewardBlacklist[account]) {
1945          return 0;
1946      }
1947      return
1948          _balances[account].mul(SPY).div(BASE_RATIO).mul(
1949              lastTime().sub(lastUpdateTime[account])
1950          );
1951  }
```

Figure 13 The source code of *getReward* function

Recommendations	It is recommended to set a reasonable SPY value according to the business situation.
Status	Acknowledged.

[DP-6] The _totalSupply was not updated when the token was destroyed

Severity Level Info

Type Business Security

Lines DP.sol #L1793-1803

Description The token transferred to 0xdEaD for destruction is not recorded. It cause the displayed total supply to be inconsistent with the actual.

```
1793        if(from != liquidity && to != liquidity){
1794            deadFee = amount.mul(deadFeePercent).div(BASE_RATIO);
1795        }
1796        else{
1797            deadFee = amount.mul(deadFeePercent).div(BASE_RATIO);
1798        }
1799
1800        if (dev != address(0) && deadFee > 0) {
1801            realAmount = realAmount.sub(deadFee);
1802            super._transfer(account, dead, deadFee);
1803        }
1804
```

Figure 14 The source code of related code

Recommendations It is recommended to add logic to update _totalSupply when burning token.

Status Acknowledged.

[DP-7] Missing event trigger

Severity Level	Info
Type	Coding Conventions
Lines	DP.sol #L1598-1642 DP.sol #L1644-1658 DP.sol #L1666-1695
Description	Event record is not triggered when several important parameters are changed.

```
1598    function setMinSwapAndLiquifyLimit(uint256 min) external onlyOwner {
1599        minSwapAndLiquifyLimit = min;
1600    }
1601
1602    function setMinSwapLimit(uint256 min) external onlyOwner {
1603        minSwapLimit = min;
1604    }
1605
1606    function setCanTransfer(bool enable) external onlyOwner {
1607        canTransfer = enable;
1608    }
1609    function setCanSwap(bool enable) external onlyOwner {
1610        canSwap = enable;
1611    }
1612
1613    function setFundFeePercent(uint256 percent)
1614        external
1615        onlyOwner
1616        checkMaxFeeLimit(fundFeePercent, percent)
1617    {
1618        fundFeePercent = percent;
1619    }
1620
1621    function setMarketFeePercent(uint256 percent)
1622        external
1623        onlyOwner
1624        checkMaxFeeLimit(marketFeePercent, percent)
1625    {
1626        marketFeePercent = percent;
1627    }
1628
1629    function setDevFeePercent(uint256 percent)
1630        external
1631        onlyOwner
1632        checkMaxFeeLimit(devFeePercent, percent)
1633    {
1634        devFeePercent = percent;
1635    }
1636    function setDeadFeePercent(uint256 percent)
1637        external
1638        onlyOwner
1639        checkMaxFeeLimit(devFeePercent, percent)
1640    {
1641        deadFeePercent = percent;
1642    }
```

Figure 15 The source code of related functions

```
    function setLiquidityFeePercent(uint256 percent)
        external
        onlyOwner
        checkMaxFeeLimit(liquidityFeePercent, percent)
    {
        liquidityFeePercent = percent;
    }

    function setNFTPoolFeePercent(uint256 percent)
        external
        onlyOwner
        checkMaxFeeLimit(nftPoolFeePercent, percent)
    {
        nftPoolFeePercent = percent;
    }
```

Figure 16 The source code of related functions

```
    function setReferralHandle(address _referralContract) external onlyOwner {
        referralHandle = IReferral(_referralContract);
    }


    function setRouter(IRouter _router) external onlyOwner {
        router = _router;
    }

    function setNFTPool(address _nftPool) external onlyOwner {
        nftPool = _nftPool;
    }

    function setFund(address _fund) external onlyOwner {
        fund = _fund;
    }

    function setLpReceiver(address _lpReceiver) external onlyOwner {
        lpReceiver = _lpReceiver;
    }

    function addWhitelist(address _addr) external onlyOwner {
        whitelist[_addr] = true;
        emit AddWhitelist(_addr);
    }

    function delWhitelist(address _addr) external onlyOwner {
        delete whitelist[_addr];
        emit DelWhitelist(_addr);
    }
```

Figure 17 The source code of related functions

Recommendations	It is recommended to declare and trigger the corresponding event.
Status	Acknowledged.

[DP-8] Redundant Code

Severity Level: Info

Type: Coding Conventions

Lines: DP.sol #L251-255
DP.sol #L1757-1762

Description: Redundant code not used. And the code blocks in “if” and “else” are the same, so adding an “if- else” selection structure is meaningless.

```
249  pragma solidity ^0.8.0;
250
251  interface IDayOfRightsClub {
252      function mint(address _recipient) external;
253
254      function dispatchHandle() external view returns (address);
255  }
```

Figure 18 The source code of related functions

```
1757            if(from != liquidity && to != liquidity){
1758                nftFee = amount.mul(nftPoolFeePercent).div(BASE_RATIO);
1759            }
1760            else{
1761                nftFee = amount.mul(nftPoolFeePercent).div(BASE_RATIO);
1762            }
```

Figure 19 Partial source code of related code

Recommendations: It is recommended to delete them.

Status: Acknowledged.

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1 (Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Impact / Likelihood	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	High	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.4 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.5 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Compiler Version Security
		Deprecated Items
		Redundant Code
		require/assert Usage
		Gas Consumption
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		call/delegatecall Security
		Returned Value Security
		tx.origin Usage
		Replay Attack
		Overriding Variables
		Third-party Protocol Interface Consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Asset Tradability
		Arbitrage Attack

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Solidity language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The project party and users need to stay vigilant at all times.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in Blockchain.

3.4 About BEOSIN

BEOSIN is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions.BEOSIN has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, BEOSIN has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

Official Website

https://www.beosin.com

Telegram

https://t.me/+dD8Bnqd133RmNWNl

Twitter

https://twitter.com/Beosin_com

Email

Contact@beosin.com