# DAO Token & Sale Contracts Manual Audit Report

## Scope of Audit:

The scope of this audit was to analyze **DAO Token & Sale Contracts** smart contracts codebase for quality, security, and correctness.

### Repository:

1- DAOToken.sol
2- SaleContract.sol

## Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC20 transfer() does not return boolean
- ERC20 approve() race
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

# Techniques and Methods

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the Smart contract is structured in a way that will not result in future problems.

Throughout the audit of **DAO Token & Sale Contracts** smart contracts care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

## Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the Smart contract is structured in a way that will not result in future problems.

## Static Analysis

Static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

## Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually

analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

### Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and possibilities of optimization of code to reduce gas consumption.

# Issue Categories

Every issue in this report has been assigned with a severity level. There are four levels of severity and each of them has been explained below.

### High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality and we recommend these issues to be fixed before moving to a live environment.

### Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems and they should still be fixed.

### Low Severity Issues

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

## Informational Issues

1. **DAOToken smart contract**

**Warning: line 158 in change Allowance function**
**Severity: Low**
No need to check the balance for approving tokens.

```
157
158     function changeAllowance(address spender, uint256 newValue) public virtual returns (bool) {
159         require(owned[_msgSender()] >= newValue, "Insufficient balance");
160         _approve(_msgSender(), spender, newValue);
161         return true;
162     }
163
```

## No. of issue per saverity

| Severity | High | Medium | Low |
|---|---|---|---|
| Open | 0 | 0 | 1 |

**Audit Report**

**Over-all contract design - Good.**

2. **SaleContract**

**Warning: line103 - line107**
**Severity: Low**
Variables declared twice. Could be initialized in one line.

```
103     address public tokenAddress;
104     address public multisigAddress;
105     address public BUSDaddress;
106     DAOToken token;
107     IERC20 BUSD;
```

### Gas Optimization Error : line 232 - line 247

**Severity : Low**

'count' is not incremented inside the else block which is increasing the number of iterations for the loop to run which will eventually result in increasing gas cost of the function.

When the 'tokenCount' is > 0 and < 'purchasedtokens', it goes inside the else condition, and the 'tokenCount' is set to 0, but the 'count' is not incremented, so it again checks the condition on the same 'count', which increases the number of iterations which could have been avoided if the count is incremented in the else condition itself.

```
232         do{
233             if(cats[count].tokencount == 0){
234                 count++;
235             }
236             else if(cats[count].tokencount >= purchasedtokens){
237                 tokensToBeTransferred = tokensToBeTransferred + (purchasedtokens * (100 + cats[count].bonus))/100;
238                 categories[count].tokencount = cats[count].tokencount - purchasedtokens;
239                 purchasedtokens = 0;
240                 break;
241             }
242             else {
243                 tokensToBeTransferred = tokensToBeTransferred + (cats[count].tokencount * (100 + cats[count].bonus))
244                 purchasedtokens = purchasedtokens - cats[count].tokencount;
245                 categories[count].tokencount = 0;
246             }
247         }while(count<3);
```

**Warning : line 211**
**Severity : Low**
'referralCode' is only emitted in the 'TokensPurchased' event. It's usage is not clear properly.

```
210  /*Purchases tokens using amount BUSD with the option to approve allowance within this function call*/
211      function purchaseToken(uint256 amount, string calldata referralCode) external nonReentrant() {      You, 4 minutes a
212          require(purchaseEnabled == true, "Sale is inactive");
213          require(amount > 0 , "You cannot buy 0 tokens -_-");
214          address sender = _msgSender();
215          if(preSaleOnly){
216              Category memory cats = categories[0];
217              require(whitelist[sender] != 0, "You are not whitelisted for presale");
218              uint256 purchasedtokens = amount/salePrice;
219              require(cats.tokencount >= purchasedtokens, "Insufficient tokens in presale");
220              categories[0].tokencount = cats.tokencount - purchasedtokens;
221              purchasedtokens = (purchasedtokens * (100 + cats.bonus))/100;  //Get total tokens, including bonus tokens
222              require(BUSD.allowance(sender, address(this)) >= amount,"Insufficient Allowance");
223              require(BUSD.transferFrom(sender,multisigAddress,amount), "BUSD Transfer Failed"); //Transfer BUSD using thi
224              require(token.lockedTransfer(sender, purchasedtokens), "Token Transfer Failed"); // Perform a transfer of lo
225              emit TokensPurchased(amount, purchasedtokens, referralCode, sender);
226          }
```

**Gas Optimization Error : line 270**
**Severity : Low**
'count' is not incremented inside the else block which is increasing the number of iterations for the loop to run which will eventually result in increasing gas cost of the function.

When the 'tokenCount' is > 0 and < 'purchasedtokens', it goes inside the else condition, and the 'tokenCount' is set to 0, but the 'count' is not incremented, so it again checks the condition on the same 'count', which increases the number of iterations which could have been avoided if the count is incremented in the else condition itself.

```
270              while(count<3){
271                  if(cats[count].tokencount == 0){
272                      count++;
273                  }
274                  else if(cats[count].tokencount >= purchasedtokens){
275                      tokensToBeTransferred = tokensToBeTransferred + (purchasedtokens * (100 + cats[count].bonus))/100;
276                      cats[count].tokencount = cats[count].tokencount - purchasedtokens;
277                      purchasedtokens = 0;
278                      break;
279                  }
280                  else {
281                      tokensToBeTransferred = tokensToBeTransferred + (cats[count].tokencount * (100 + cats[count].bonus))
282                      purchasedtokens = purchasedtokens - cats[count].tokencount;
283                      cats[count].tokencount = 0;
284                  }
285              }
```

## No. of issue per saverity

| Severity | High | Medium | Low |
|---|---|---|---|
| Open | 0 | 0 | 4 |

**Audit Report**

**Over-all contract design - Good.**