

Audit report of DWIN_GOLD

Prepared By: - Kishan Patel
Prepared On: - 06/02/2025.

**Prepared for: DWIN INTERTRADE
COMPANY LTD.**

Table of contents

- 1. Disclaimer**
- 2. Introduction**
- 3. Project information**
- 4. List of attacks checked**
- 5. Severity Definitions**
- 6. Good things in code**
- 7. Critical vulnerabilities in code**
- 8. Medium vulnerabilities in code**
- 9. Low vulnerabilities in code**
- 10. Summary**

THIS AUDIT REPORT WILL CONTAIN CONFIDENTIAL INFORMATION ABOUT THE SMART CONTRACT AND INTELLECTUAL PROPERTY OF THE CUSTOMER AS WELL AS INFORMATION ABOUT POTENTIAL VULNERABILITIES OF THEIR EXPLOITATION.

THE INFORMATION FROM THIS AUDIT REPORT CAN BE USED INTERNALLY BY THE CUSTOMER OR IT CAN BE DISCLOSED PUBLICLY AFTER ALL VULNERABILITIES ARE FIXED - UPON THE DECISION OF THE CUSTOMER.

1. Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions). Because the total numbers of test cases are unlimited, the audit makes no statements or warranties on the security of the code.

It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.

2. Introduction

Kishan Patel (Consultant) was contacted by DWIN INTERTRADE COMPANY LIMITED. (Customer) to conduct a Smart Contracts Code Review and Security Analysis. This report presents the findings of the security assessment of Customer's smart contracts and its code review conducted between 10/01/2025 – 11/01/2025.

The project has 1 files. It contains approx 250 lines of Solidity code. All the functions and state variables are well commented on using the natspec documentation, but that does not create any vulnerability.

3. Project information

Token Name	DWIN_Gold
Token Symbol	OZ
Platform	Binance Smart Chain
Order Started Date	06/02/2025
Order Completed Date	07/02/2025

4. List of attacks checked

- Over and under flows
- Short address attack
- Visibility & Delegate call
- Reentrancy / TheDAO hack
- Forcing BNB to a contract
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference


5. Severity Definitions

Risk	Level Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens loss etc.
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution

6. Good things in code

- **Good required condition in functions:-**


- Here smart contract is checking that balance of to address is bigger or equal to amount, _totalSupply is bigger than amount.

```
220     function _burn(address to, uint amount) internal {  infinite gas
221         require(balances[to]>=amount);
222         require(_totalSupply>=amount);
223         balances[to] = balances[to].sub(amount);
224         _totalSupply -= amount;
```

- Here smart contract is checking that msg.sender is Minter or Admin address.

```
229     function MintForPeg(address to , uint amount) public returns (bool){
230         require((msg.sender==Minter)|| (msg.sender==Admin));
231         _mint(to,amount);
232         emit PegAsset(msg.sender, to, amount);
233         emit PegAsset(msg.sender, to, amount);
```

- Here smart contract is checking that msg.sender is Minter or Admin address.

```
237     function BurnAsset(uint amount) public returns(bool){  infinite gas
238         require((msg.sender==Minter)|| msg.sender==Admin);
239         _burn(msg.sender, amount);
240
```


7. Critical vulnerabilities in code

- No Critical vulnerabilities found

8. Medium vulnerabilities in code

- No Medium vulnerabilities found

9. Low vulnerabilities in code

9.1. Suggestions to add code validations:-

=> You have implemented required validation in contract.

=> There are some places where you can improve validation and security of your code.

=> These are all just suggestion it is not bug.

○ Function: - setAdmin

```
122
123     function setAdmin(address _admin) public onlyOwner{
124
125         Admin = _admin;
126
127     }
128 }
```

26992 gas

- Here in setAdmin function smart contract can check that _admin value is not already same as Admin.

○ Function: - setMinter

```
129
130     function setMinter(address _minter) public onlyAdmin{
131
132         Minter = _minter;
133     }
134 }
```

26972 gas

- Here in setAdmin function smart contract can check that _minter value is not already same as Minter.

- **Function: - transfer**

```
157 //
158 function transfer(address to, uint tokens) public returns (bool succ
159     balances[msg.sender] =balances[msg.sender].sub(tokens);
160     balances[to] +=tokens;
161     emit Transfer(msg.sender, to, tokens);
162     return true;
163 }
```

- Here in transfer function smart contract can use safeMath library for calculating balance of to address.

- **Function: - approve**

```
166
167 function approve(address spender, uint tokens) public returns (bool
168     allowed[msg.sender][spender] = tokens;
169     emit Approval(msg.sender, spender, tokens);
170     return true;
171 }
```

- Here in approve function smart contract can check that balance of msg.sender is bigger than or equal to tokens.

- **Function: - transferFrom**

```
176
177 function transferFrom(address from, address to, uint tokens) public
178     balances[from] =balances[from].sub(tokens);
179     allowed[from][msg.sender] = allowed[from][msg.sender].sub(tokens
180     balances[to] +=tokens;
181     emit Transfer(from, to, tokens);
182 }
```

- Here in transferFrom function smart contract can use safeMath library for calculating balance of to address.

- **Function: - approveAndCall**

```
195
196 function approveAndCall(address spender, uint tokens, bytes memory d
197     allowed[msg.sender][spender] = tokens;
198     emit Approval(msg.sender, spender, tokens);
199     ApproveAndCallFallback(spender).receiveApproval(msg.sender, toke
200     return true;
201 }
```


- Here in approveAndCall function smart contract can check that balance of msg.sender is bigger than or equal to tokens.

○ **Function: - transferAnyERC20Token**

```
207 function transferAnyERC20Token(address tokenAddress, uint tokens) public {
208     return IBEP20(tokenAddress).transfer(Admin, tokens);
209 }
```


- Here in transferAnyERC20Token function in smart contract can have protection that admin or owner can only call this method.
- **Currently anyone is able to call this method and token will be transferred to Admin address.**

○ **Function: - _mint**

```
214 function _mint(address to, uint amount) internal {  infinite gas
215     balances[to] = balances[to].add(amount);
216     _totalSupply += amount;
```

- Here in _mint function smart contract can use safeMath library to calculate _totalSupply.

○ **Function: - _burn**

```
220 function _burn(address to, uint amount) internal {  infinite gas
221     require(balances[to] >= amount);
222     require(_totalSupply >= amount);
223     balances[to] = balances[to].sub(amount);
224     _totalSupply -= amount;
```

- Here in _burn function smart contract can use safeMath library to calculate _totalSupply.

10. Summary

- **Number of problems in the smart contract as per severity level**

Critical	Medium	Low
0	0	9

According to the assessment, the smart contract code is well secured. The code is written with all validation and all security is implemented. Code is performing well and there is no way to steal funds from this contract.

- **Good Point:** Code performance and quality are good. All kind of necessary validation added into smart contract and all validations are working as expected.
- **Suggestions:** Please try to implement suggested code validations.