



# **Audit report of USDDWIN**

**Prepared By: - Kishan Patel**  
Prepared On: - 02/05/2025.

**Prepared for: Dwin Intertrade 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 Ltd. (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 02/05/2025 – 06/05/2025.

The project has 1 files. It contains approx 300 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

<b>Token Name</b>	USD Dwin
<b>Token Symbol</b>	USDW
<b>Platform</b>	Tron
<b>Order Started Date</b>	02/05/2025
<b>Order Completed Date</b>	06/05/2025

## 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
<b>Critical</b>	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens loss etc.
<b>Medium</b>	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
<b>Low</b>	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 and \_totalSupply is bigger or equal to amount.

```
216     function _burn(address to, uint amount) internal {  infinite gas
217         require(balances[to]>=amount);
218         require(_totalSupply>=amount);
219         balances[to] = balances[to].sub(amount);
220         _totalSupply -= amount;
221     }
```

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

```
224
225     function MintForPeg(address to , uint amount) public returns (bool){
226         require((msg.sender==Minter)|| (msg.sender==Admin));
227         _mint(to,amount);
228         emit PegAsset(msg.sender, to, amount);
229     }
```

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

```
233     function BurnAsset(uint amount) public returns(bool){  infinite gas
234         require((msg.sender==Minter)|| msg.sender==Admin);
235         _burn(msg.sender, amount);
236     }
```

## 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: - approve

```
162
163     function approve(address spender, uint tokens) public returns (bool
164         allowed[msg.sender][spender] = tokens;
165         emit Approval(msg.sender, spender, tokens);
166         return true;
167     }
```

- Here in approve functions smart contract can check that balance of msg.sender is bigger than amount.

#### ○ Function: - transferAnyERC20Token

```
202     // -----
203     function transferAnyERC20Token(address tokenAddress, uint tokens) pu
204         return IERC20(tokenAddress).transfer(Admin, tokens);
205     }
206
```

- Here in transferAnyERC20Token functions smart contract can check that transfer method is successfully called and finished.

## 10. Summary

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

Critical	Medium	Low
0	0	2

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.