Audited by

CredShields

Smart Contract Audit

December 18th, 2024 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by BAF - CredShields Technologies PTE. LTD. on behalf of DWIN Intertrade Company Ltd. between December 17th, 2024, and December 18th, 2024. A retest is yet to be performed.

Author

Shashank

Reviewers

Aditya Dixit (Research Team Lead), Shreyas Koli(Auditor), Naman Jain (Auditor), Sanket Salavi (Auditor)

Prepared for

DWIN Intertrade Company Ltd.

Table of Contents

1. Executive Summary --------------------

DWIN Intertrade Company Ltd. engaged BAF - CredShields to perform a smart contract audit from December 17th, 2024, to December 18th, 2024. During this timeframe, 4 vulnerabilities were identified. **A retest is yet to be performed.**

During the audit, no vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "DWIN Intertrade Company Ltd." and should be prioritized for remediation, and fortunately, none were found.

The table below shows the in-scope assets and a breakdown of findings by severity per asset. Section 2.3 contains more information on how severity is calculated.

Assets in Scope	Critical	High	Medium	Low	info	Gas	Σ
DWIN Token	0	0	0	3	0	1	**4**
	0	**0**	**0**	**3**	**0**	**1**	**4**

Table: Vulnerabilities Per Asset in Scope

The BAF - CredShields team conducted the security audit to focus on identifying vulnerabilities in DWIN Token's scope during the testing window while abiding by the policies set forth by DWIN Intertrade Company Ltd.'s team.

State of Security

To maintain a robust security posture, it is essential to continuously review and improve upon current security processes. Utilizing BAF - CredShields' continuous audit feature allows both DWIN Intertrade Company Ltd.'s internal security and development teams to not only identify specific vulnerabilities but also gain a deeper understanding of the current security threat landscape.

To ensure that vulnerabilities are not introduced when new features are added, or code is refactored, we recommend conducting regular security assessments. Additionally, by analyzing the root cause of resolved vulnerabilities, the internal teams at DWIN Intertrade Company Ltd. can implement both manual and automated procedures to eliminate entire classes of vulnerabilities in the future. By taking a proactive approach, DWIN Intertrade Company Ltd. can future-proof its security posture and protect its assets.

2. The Methodology -------------------

DWIN Intertrade Company Ltd. engaged BAF - CredShields to perform a DWIN Token Smart Contract audit. The following sections cover how the engagement was put together and executed.

2.1 Preparation Phase

The BAF - CredShields team meticulously reviewed all provided documents and comments in the smart contract code to gain a thorough understanding of the contract's features and functionalities. They meticulously examined all functions and created a mind map to systematically identify potential security vulnerabilities, prioritizing those that were more critical and business-sensitive for the refactored code. To confirm their findings, the team deployed a self-hosted version of the smart contract and performed verifications and validations during the audit phase.

A testing window from December 17th, 2024, to December 18th, 2024, was agreed upon during the preparation phase.

2.1.1 Scope

During the preparation phase, the following scope for the engagement was agreed upon:

IN SCOPE ASSETS
https://github.com/dwinno-fin/DWINToken/tree/dcddc29388d61ff0f81b3e03ffdd7ecf33cd403b

2.1.2 Documentation

Documentation was not required as the code was self-sufficient for understanding the project.

2.1.3 Audit Goals

BAF - CredShields uses both in-house tools and manual methods for comprehensive smart contract security auditing. The majority of the audit is done by manually reviewing the contract source code, following SWC registry standards, and an extended industry standard self-developed checklist. The team places emphasis on understanding core concepts, preparing test cases, and evaluating business logic for potential vulnerabilities.

2.2 Retesting Phase

DWIN Intertrade Company Ltd. is actively Collabrating with BAF to validate the remediations implemented towards the discovered vulnerabilities.

2.3 Vulnerability classification and severity

BAF - CredShields follows OWASP's Risk Rating Methodology to determine the risk associated with discovered vulnerabilities. This approach considers two factors - Likelihood and Impact - which are evaluated with three possible values - **Low**, **Medium**, and **High**, based on factors such as Threat agents, Vulnerability factors, and Technical and Business Impacts. The overall severity of the risk is calculated by combining the likelihood and impact estimates.

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	None	Low	Medium
		LOW	MEDIUM	HIGH
Likelihood				

Overall, the categories can be defined as described below -

1. Informational

We prioritize technical excellence and pay attention to detail in our coding practices. Our guidelines, standards, and best practices help ensure software stability and reliability. Informational vulnerabilities are opportunities for improvement and do not pose a direct risk to the contract. Code maintainers should use their own judgment on whether to address them.

2. Low

Low-risk vulnerabilities are those that either have a small impact or can't be exploited repeatedly or those the client considers insignificant based on their specific business circumstances.

3. Medium

Medium-severity vulnerabilities are those caused by weak or flawed logic in the code and can lead to exfiltration or modification of private user information. These vulnerabilities can harm the client's reputation under certain conditions and should be fixed within a specified timeframe.

4. **High**

   High-severity vulnerabilities pose a significant risk to the Smart Contract and the organization. They can result in the loss of funds for some users, may or may not require specific conditions, and are more complex to exploit. These vulnerabilities can harm the client's reputation and should be fixed immediately.

5. **Critical**

   Critical issues are directly exploitable bugs or security vulnerabilities that do not require specific conditions. They often result in the loss of funds and Ether from Smart Contracts or users and put sensitive user information at risk of compromise or modification. The client's reputation and financial stability will be severely impacted if these issues are not addressed immediately.

6. **Gas**

   To address the risk and volatility of smart contracts and the use of gas as a method of payment, BAF - CredShields has introduced a "Gas" severity category. This category deals with optimizing code and refactoring to conserve gas.

2.4BAFTeam-ProcessCo-ordinator

The following individual at CredShields managed this engagement and produced this report:

- Joseph, Head of Sales +91 7338799291

Please feel free to contact this individual with any questions or concerns you have about the engagement or this document.

3. Findings Summary -------------------

This chapter contains the results of the security assessment. Findings are sorted by their severity and grouped by the asset and SWC classification. Each asset section will include a summary. The table in the executive summary contains the total number of identified security vulnerabilities per asset per risk indication.

3.1 Findings Overview

3.1.1 Vulnerability Summary

During the security assessment, 4 security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC | Vulnerability Type
Failure to Deduct Fee if Transferring Tokens using Contract	Low	Business Logic Issue
Use safeTransfer/safeTransferFrom instead of transfer/transferFrom	Low	Missing best practices
Use Ownable2Step	Low	Missing Best Practices
Gas Optimization in Require/Revert Statements	Gas	Gas Optimization

Table: Findings in Smart Contracts

3.1.2 Findings Summary

SWC ID	SWC Checklist	Test Result	Notes
SWC-100	Function Default Visibility	Not Vulnerable	Not applicable after v0.5.X (Currently using solidity v >= 0.8.6)
SWC-101	Integer Overflow and Underflow	Not Vulnerable	The issue persists in versions before v0.8.X.
SWC-102	Outdated Compiler Version	Not Vulnerable	Version 0^.8.0 and above is used
SWC-103	Floating Pragma	Not Vulnerable	Contract uses floating pragma
SWC-104	Unchecked Call Return Value	Not Vulnerable	call() is not used
SWC-105	Unprotected Ether Withdrawal	Not Vulnerable	Appropriate function modifiers and require validations are used on sensitive functions that allow token or ether withdrawal.
SWC-106	Unprotected SELFDESTRUCT Instruction	Not Vulnerable	selfdestruct() is not used anywhere
SWC-107	Reentrancy	Not Vulnerable	No notable functions were vulnerable to it.
SWC-108	State Variable Default Visibility	Not Vulnerable	Not Vulnerable
SWC-109	Uninitialized Storage Pointer	Not Vulnerable	Not vulnerable after compiler version, v0.5.0
SWC-110	Assert Violation	Not Vulnerable	Asserts are not in use.
SWC-111	Use of Deprecated Solidity Functions	Not Vulnerable	None of the deprecated functions like block.blockhash(), msg.gas, throw, sha3(), callcode(), suicide() are in use

SWC-112	Delegatecall to Untrusted Callee	Not Vulnerable	Not Vulnerable.
SWC-113	DoS with Failed Call	Not Vulnerable	No such function was found.
SWC-114	Transaction Order Dependence	Not Vulnerable	Not Vulnerable.
SWC-115	Authorization through tx.origin	Not Vulnerable	tx.origin is not used anywhere in the code
SWC-116	Block values as a proxy for time	Not Vulnerable	Block.timestamp is not used
SWC-117	Signature Malleability	Not Vulnerable	Not used anywhere
SWC-118	Incorrect Constructor Name	Not Vulnerable	All the constructors are created using the constructor keyword rather than functions.
SWC-119	Shadowing State Variables	Not Vulnerable	Not applicable as this won't work during compile time after version 0.6.0
SWC-120	Weak Sources of Randomness from Chain Attributes	Not Vulnerable	Random generators are not used.
SWC-121	Missing Protection against Signature Replay Attacks	Not Vulnerable	No such scenario was found
SWC-122	Lack of Proper Signature Verification	Not Vulnerable	Not used anywhere
SWC-123	Requirement Violation	Not Vulnerable	Not vulnerable
SWC-124	Write to Arbitrary Storage Location	Not Vulnerable	No such scenario was found
SWC-125	Incorrect Inheritance Order	Not Vulnerable	No such scenario was found
SWC-126	Insufficient Gas Griefing	Not Vulnerable	No such scenario was found
SWC-127	Arbitrary Jump with Function Type Variable	Not Vulnerable	Jump is not used.

SWC-128	DoS With Block Gas Limit	Not Vulnerable	Not Vulnerable.
SWC-129	Typographical Error	Not Vulnerable	No such scenario was found
SWC-130	Right-To-Left-Override control character (U+202E)	Not Vulnerable	No such scenario was found
SWC-131	Presence of unused variables	Not Vulnerable	No such scenario was found
SWC-132	Unexpected Ether balance	Not Vulnerable	No such scenario was found
SWC-133	Hash Collisions With Multiple Variable Length Arguments	Not Vulnerable	abi.encodePacked() or other functions are not used.
SWC-134	Message call with hardcoded gas amount	Not Vulnerable	Not used anywhere in the code
SWC-135	Code With No Effects	Not Vulnerable	No such scenario was found
SWC-136	Unencrypted Private Data On-Chain	Not Vulnerable	No such scenario was found

4. Remediation Status -----------------

A retest is yet to be performed.

Also, the table shows the remediation status of each finding.

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Failure to Deduct Fee if Transferring Tokens using Contract	Low	**Fixed** [Dec 18th, 2024]
Use safeTransfer/safeTransferFrom instead of transfer/transferFrom	Low	**Pending Fix** [Dec 18th, 2024]
Use Ownable2Step	Low	**Pending Fix** [Dec 18th, 2024]
Gas Optimization in Require/Revert Statements	Gas	**Pending Fix** [Dec 18th, 2024]

Table: Summary of findings and status of remediation

5. Bug Reports ----------------------

Bug ID #1 [Fixed]

Failure to Deduct Fee if Transferring Tokens using Contract

Vulnerability Type
Business Logic Issue

Severity
Low

Description
The documentation for the USDDWIN token mentions that the users are required to pay a small fee of 1% every time they do a transaction or get minted a token, but this is done off-chain.
There's no such mechanism to deduct a fee if the users themselves decide to transfer tokens among themselves, therefore, bypassing the fee deduction mechanism.

Affected Code

- https://github.com/dwinno-fin/DWINToken/blob/dcddc29388d61ff0f81b3e03ffdd7ecf33cd403b/contracts/DWINToken.sol

Impacts
The missing fee deduction logic could allow all the users to bypass paying the fee off-chain.

Remediation
It is recommended to implement a logic inside the contract that deducts a fee percentage on every transaction and sends the amount to a fee receiver wallet, rather than keeping it off-chain.

Retest
The whitepaper has been updated.

Bug ID #2

Use safeTransfer/safeTransferFrom instead of transfer/transferFrom

Vulnerability Type
Missing best practices

Severity
Low

Description
The transfer() and transferFrom() method is used instead of safeTransfer() and safeTransferFrom(), presumably to save gas however OpenZeppelin's documentation discourages the use of transferFrom(), use safeTransferFrom() whenever possible because safeTransferFrom auto-handles boolean return values whenever there's an error.

Affected Code

- https://github.com/dwinno-fin/DWINToken/blob/dcddc29388d61ff0f81b3e03ffdd7ecf33cd403b/contracts/DWINToken.sol#L105
- https://github.com/dwinno-fin/DWINToken/blob/dcddc29388d61ff0f81b3e03ffdd7ecf33cd403b/contracts/DWINToken.sol#L115

Impacts
Using safeTransferFrom has the following benefits -

- It checks the boolean return values of ERC20 operations and reverts the transaction if they fail,
- at the same time allowing you to support some non-standard ERC20 tokens that don't have boolean return values.
- It additionally provides helpers to increase or decrease an allowance, to mitigate an attack possible with vanilla approve.

Remediation
Consider using safeTransfer() and safeTransferFrom() instead of transfer() and transferFrom().

Retest
-

Bug ID #3

Use Ownable2Step

Vulnerability Type
Missing Best Practices

Severity
Low

Description
The "Ownable2Step" pattern is an improvement over the traditional "Ownable" pattern, designed to enhance the security of ownership transfer functionality in a smart contract. Unlike the original "Ownable" pattern, where ownership can be transferred directly to a specified address, the "Ownable2Step" pattern introduces an additional step in the ownership transfer process. Ownership transfer only completes when the proposed new owner explicitly accepts the ownership, mitigating the risk of accidental or unintended ownership transfers to mistyped addresses.

Affected Code

- https://github.com/dwinno-fin/DWINToken/blob/dcddc29388d61ff0f81b3e03ffdd7ecf33cd403b/contracts/DWINToken.sol#L12

Impacts
Without the "Ownable2Step" pattern, the contract owner might inadvertently transfer ownership to an unintended or mistyped address, potentially leading to a loss of control over the contract. By adopting the "Ownable2Step" pattern, the smart contract becomes more resilient against external attacks aimed at seizing ownership or manipulating the contract's behavior.

Remediation
It is recommended to use either Ownable2Step or Ownable2StepUpgradeable depending on the smart contract.

Retest:
-

Bug ID #4

Gas Optimization in Require/Revert Statements

Vulnerability Type
Gas Optimization

Severity
Gas

Description
The **require/revert** statement takes an input string to show errors if the validation fails.
The strings inside these functions that are longer than **32 bytes** require at least one additional MSTORE, along with additional overhead for computing memory offset and other parameters. For this purpose, having strings lesser than 32 bytes saves a significant amount of gas.

Affected Code
- https://github.com/dwinno-fin/DWINToken/blob/dcddc29388d61ff0f81b3e03ffdd7ecf33cd403b/contracts/DWINToken.sol#L61
- https://github.com/dwinno-fin/DWINToken/blob/dcddc29388d61ff0f81b3e03ffdd7ecf33cd403b/contracts/DWINToken.sol#L71

Impacts
Having longer require/revert strings than **32 bytes** cost a significant amount of gas.

Remediation
It is recommended to shorten the strings passed inside **require/revert** statements to fit under **32 bytes**. This will decrease the gas usage at the time of deployment and at runtime when the validation condition is met.

Retest
-

6. The Disclosure ----------------------

The Reports provided by BAF - CredShields are not an endorsement or condemnation of any specific project or team and do not guarantee the security of any specific project. The contents of this report are not intended to be used to make decisions about buying or selling tokens, products, services, or any other assets and should not be interpreted as such.

Emerging technologies such as Smart Contracts and Solidity carry a high level of technical risk and uncertainty. BAF - CredShields does not provide any warranty or representation about the quality of code, the business model or the proprietors of any such business model, or the legal compliance of any business. The report is not intended to be used as investment advice and should not be relied upon as such.

BAF - CredShields Audit team is not responsible for any decisions or actions taken by any third party based on the report.