BLOCK SOLUTIONS

Smart Contract Code Review and Security Analysis Report for USD DWIN ERC20 Token Smart Contract

Request Date: 2026-02-04
Completion Date: 2026-02-06
Language: Solidity

Contents

Commission

Audited Project	USD DWIN ERC20 Token Smart Contract
Smart Contract Address	0x6AEbdDF980Dc724BAbDC7CEd031c8C37934bBA72
Contract Creator	0xc557ee55f5DbDAc7128fc4D378111a7D68dEb3d6
Default Admin Role	0xc557ee55f5DbDAc7128fc4D378111a7D68dEb3d6
Default Admin Role	0x9a1B379e300cB026E74BDE618b033F7f37ED9291
Default Admin Role	0x9d4b93dD4aA203d2Be48Fb515011beACDB740260
Blockchain Network	Arbitrum One Mainnet

Block Solutions was commissioned by USD DWIN ERC20 Token Smart Contract owners to perform an audit of their main smart contract. The purpose of the audit was to achieve the following:

- Ensure that the smart contract functions as intended.
- Identify potential security issues with the smart contract.

The information in this report should be used to understand the risk exposure of the smart contract, and as a guide to improve the security posture of the smart contract by remediating the issues that were identified.

USD DWIN Properties

Contract Token name	USD DWIN
Total supply	1,000,000,000 USDW
Symbol	USDW
Decimals	18
Holder	35
Transfers	1,855
Smart Contract Address	0x6AEbdDF980Dc724BAbDC7CEd031c8C37934bBA72
Contract Creator	0xc557ee55f5DbDAc7128fc4D378111a7D68dEb3d6
Default Admin Role	0xc557ee55f5DbDAc7128fc4D378111a7D68dEb3d6
Default Admin Role	0x9a1B379e300cB026E74BDE618b033F7f37ED9291
Default Admin Role	0x9d4b93dD4aA203d2Be48Fb515011beACDB740260
Blockchain Network	Arbitrum One Mainnet

Contract Functions

Executables (Write Functions)

All state-changing (write) functions. Access control is noted where applicable.

#	Function	Access
1	function initialize(address _oracle, address admin) external initializer	One-time only
2	function mint(address to, uint256 amount) external	MINTER_ROLE
3	function burn(address from, uint256 amount) external	MINTER_ROLE
4	function pause() external	PAUSER_ROLE
5	function unpause() external	PAUSER_ROLE
6	Functionfunction setBlacklist(address account, bool status) external	AccessDEFAULT_ADMIN_ROLE
7	function releaseFromBlacklist(address account) external	DEFAULT_ADMIN_ROLE
8	function releaseBlacklist(address account) external	DEFAULT_ADMIN_ROLE
9	function setLPWhitelist(address account, bool status) external	DEFAULT_ADMIN_ROLE
10	function releaseFromLPWhitelist(address account) external	DEFAULT_ADMIN_ROLE
11	function setPriceMode(PriceMode mode) external	DEFAULT_ADMIN_ROLE
12	function setPoROracle(address _oracle) external	DEFAULT_ADMIN_ROLE
13	function setDexFeed(address _feed) external	DEFAULT_ADMIN_ROLE
14	function setFixedPrice(uint256 _price) external	DEFAULT_ADMIN_ROLE
15	function setDexAdapter(bytes32 id, address feed, bool active, uint16 weight) external	DEFAULT_ADMIN_ROLE
16	function rescueTokens(address token, address to, uint256 amount) external	DEFAULT_ADMIN_ROLE
17	function approveWeb3(address spender) external returns (bool)	Any (msg.sender approves)

Inherited ERC20 (write):

#	Function	Access
18	function approve(address spender, uint256 amount) public returns (bool)	Caller
19	function transfer(address to, uint256 amount) public returns (bool)	Caller
20	function transferFrom(address from, address to, uint256 amount) public returns (bool)	Caller (with allowance)
21	function increaseAllowance(address spender, uint256 addedValue) public returns (bool)	Caller
22	function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool)	Caller

UUPS upgrade (write):

#	Function	Access
23	function upgradeTo(address newImplementation) external	UPGRADER_ROLE
24	function upgradeToAndCall(address newImplementation, bytes memory data) external payable	UPGRADER_ROLE

Quick Stats

Compiler errors	Passed
Possible delays in data delivery	Passed
Timestamp dependence	Passed
Integer Overflow and Underflow	Passed
Race Conditions and Reentrancy	Attention Required
DoS with Revert	Passed
DoS with block gas limit	Passed
Methods execution permissions	Passed
Economy model of the contract	Passed
Private user data leaks	Passed
Malicious Events Log	Passed
Scoping and Declarations	Passed
Uninitialized storage pointers	Passed
Arithmetic accuracy	Passed
Design Logic	Passed
Impact of the exchange rate	Passed
Oracle Calls	N/A
Cross-function race conditions	Passed
Fallback function security	Passed
Safe Open Zeppelin contracts and implementation usage	Passed
Whitepaper-Website-Contract correlation	Not Checked

Testing Summary

Attention Required

BLOCK SOLUTIONS BELIEVES

This smart contract requires attention on several security concerns before being listed on digital assets exchanges

6th FEB, 2026

Executive Summary

According to the standard audit assessment, the USD DWIN ERC20 smart contract requires Attention in several areas before or after deployment.

The audit was based on manual code review of the provided USDW.sol contract (upgradeable ERC20 with Proof-of-Reserve, DEX/fixed price modes, blacklist, LP whitelist, and role-based access). All findings were manually reviewed and applicable vulnerabilities are presented in the Audit Findings and Quick Stats sections.

Severity overview

Severity	**Count**
Critical	0
High	3
Medium	4
Low	3
Informational	4

Code Quality

The USD DWIN Smart Contract (USDW.sol) is a single main contract that uses OpenZeppelin upgradeable base contracts: ERC20Upgradeable ,PausableUpgradeabln=, AccessControlUpgradeable , and UUPSUpgradeable . It also depends on an external interface IUSDWPorOracle (PoR oracle) and an IExternalPriceFeed-style interface for DEX price feeds.
The code structure is moderately complex with approximately 375 lines of code. The contract implements a mintable/burnable ERC20 with three price modes (PoR,DEX, FIXED), blacklist and LP whitelist, and role-based access (DEFAULT_ADMIN_ROLE , PAUSER_ROLE , MINTER_ROLE , UPGRADER_ROLE). The code uses Solidity 0.8.22 with built-in overflow protection. Overall, the code has minimal inline comments; commenting can provide rich documentation for functions, return variables and more.

Documentation

It is recommended to write comments in the smart contract code so anyone can quickly understand the programming flow as well as complex code logic. The code contains some inline comments but lacks comprehensive NatSpec documentation for public/external functions (e.g. `@notice` , `@param` , `@return`).

Use of Dependencies

The contract uses OpenZeppelin upgradeable libraries that are well-known industry-standard open-source projects. The core code blocks are written in a structured way. The smart contract interacts with an external PoR oracle (IUSDWPorOracle) and optional DEX price feeds (IExternalPriceFeed).

Source	Components
OpenZeppelin	ERC20Upgradeable, PausableUpgradeable, AccessControlUpgradeable, UUPSUpgradeable, IERC20Upgradeable
Custom	IUSDWPorOracle (PoR summary), IExternalPriceFeed (latestAnswer for DEX price)

Risk Level Description

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution (e.g. public access to crucial functions or abuse of privileged functions).
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to direct token loss.
LowRisk Level	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets that can't have significant impact Descriptionon execution.
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Audit Findings

Critical

No critical findings identified in this audit.

High

H-01 Hardcoded admin addresses at initialization

Lines 88–90

Three addresses are granted `DEFAULT_ADMIN_ROLE` in `initialize()` :

```
_grantRole(DEFAULT_ADMIN_ROLE, 0xc557ee55f5DbDAc7128fc4D378111a7D68dEb3d6);
_grantRole(DEFAULT_ADMIN_ROLE, 0x9a1B379e300cB026E74BDE618b033F7f37ED9291);
_grantRole(DEFAULT_ADMIN_ROLE, 0x9d4b93dD4aA203d2Be48Fb515011beACDB740260);
```

Impact
Centralization and inflexibility. If keys are lost or compromised, or addresses are wrong (e.g. wrong network), recovery is only via upgrade.

Recommendation
Remove hardcoded addresses. Pass all initial admins as a parameter, or use a single admin that configures others after deployment.

H-02 rescueTokens can drain USDW and lacks safeguards

Lines 365–372

The `rescueTokens` function transfers any ERC20 to a recipient without disallowing the token itself (USDW) . There is no event and no check that `token != address(this)` .

Impact
Admin can rescue USDW from the contract; if the contract is supposed to hold USDW as backing, this can break accounting or trust. Lack of event hinders monitoring.

Recommendation
Add `require(token != address(this), "USDW: cannot rescue USDW");` (or document intentional USDW rescue and add an event). Emit `RescueTokens(token, to, amount)` . Optionally require `amount > 0` .

H-03 approveWeb3 grants unlimited approval to any spender

Lines 358–361

```
function approveWeb3(address spender) external returns (bool) {
    _approve(msg.sender, spender, type(uint256).max);
    return true;
}
```

Impact
Users can be tricked (e.g. via malicious site) into calling `approveWeb3(maliciousContract)` , granting unlimited approval . The spender can then transfer

Recommendation

Prefer removing this and using standard approve(spender, amount) . If kept, add clear warnings in UI/docs and consider a cap or time-limited approval pattern.

Medium

[M-01] Initialize front-running

Lines 66–94

initialize() is protected only by the initializer modifier. Anyone can call it once. If the deployer does not call initialize in the same transaction as
deployment, a front-runner can call it with their own _oracle and admin and take control.

Recommendation
Deploy proxy and call initialize in the same transaction (e.g. via factory or multicall), or use an access-controlled initializer that only the deployer can call once.

[M-02] Redundant blacklist functions

Lines 151–164 releaseFromBlacklist and releaseBlacklist are identical (both set

blacklist[account] = false and emit the same event).

Recommendation
Keep a single function (e.g. releaseFromBlacklist) and remove the duplicate. Update
docs and frontends accordingly.

[M-03] PoR price when reserve is zero

Lines 234–244

In PoR mode, if supply > 0 but oracle.getPorSummary() returns reserve == 0 , priceUSD() returns 0. This can break integrators or lead to incorrect pricing.

Recommendation
Require reserve > 0 when supply > 0, or explicitly return fixedPrice and document the behaviour. Ensure oracle and operations keep reserve consistent.

[M-04] Missing interface in scope

The contract imports ./IUSDWPorOracle.sol . This interface was not in the audit scope.

Recommendation
Include IUSDWPorOracle (and oracle implementation) in future audits for a complete view of PoR and price guarantees.

Low

[L-01] getAllValues return tuple naming

Lines 286–297

The 5th return value is `activePrice` (priceUSD()) and the 6th is `mode` (priceMode). Implementation is correct but ordering can be confusing.

Recommendation
Document the return tuple clearly in NatSpec and API docs.

[L-02] setDexAdapter does not support "removing" an adapter

Lines 216–229

The function requires `feed != address(0)` . To effectively remove an adapter, only `active = false` can be set; feed and weight remain in storage.

Recommendation
Either allow `feed == address(0)` when deactivating (and set active = false, weight = 0) or document that removal is by deactivation only.

[L-03] No event for rescueTokens

Already noted under H-02. Emit `RescueTokens(token, to, amount)` for monitoring and compliance.

Informational

1. **Reentrancy on rescueTokens** — External call to `token.transfer(to, amount)` . Caller is DEFAULT_ADMIN_ROLE; risk is limited. Consider ReentrancyGuard or document that only trusted tokens should be rescued.
2. **getCompositeDexPrice** — In theory, `uint256(ans) * adapter.weight` could overflow for extreme feed values. Solidity 0.8.x reverts on overflow; document that feeds must return sane values or add bounds.
3. **Add NatSpec** — Document all public/external functions with `@notice` , `@param` , `@return` for maintainability and tooling.
4. **Centralization** — Admin roles and hardcoded admins imply significant centralization. Document and disclose admin addresses and consider multisig/timelock for critical actions.

Conclusion

The USD DWIN ERC20 Smart Contract (USDW.sol) code requires attention before or after deployment. The audit was performed on the provided contract code and the findings above should be addressed where applicable.

Key findings summary

Category	**Assessment**
Code Security	⚠ Attention Required
Centralization Risk	High
Code Quality	Moderate
Documentation	Needs Improvement
Gas Efficiency	Adequate

The contract implements an upgradeable ERC20 with Proof-of-Reserve, DEX and fixed price modes, blacklist, LP whitelist, and role-based access. Several security and design concerns were identified:

1. Hardcoded admin addresses and privileged functions (`rescueTokens` , `approveWeb3`) that require safeguards or removal.
2. Initialize front-running risk if not deployed and initialized in the same transaction.
3. Code quality issues such as redundant functions and missing events/NatSpec.

Security state of the reviewed contract: ⚠ Attention Required

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high-level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally, follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.