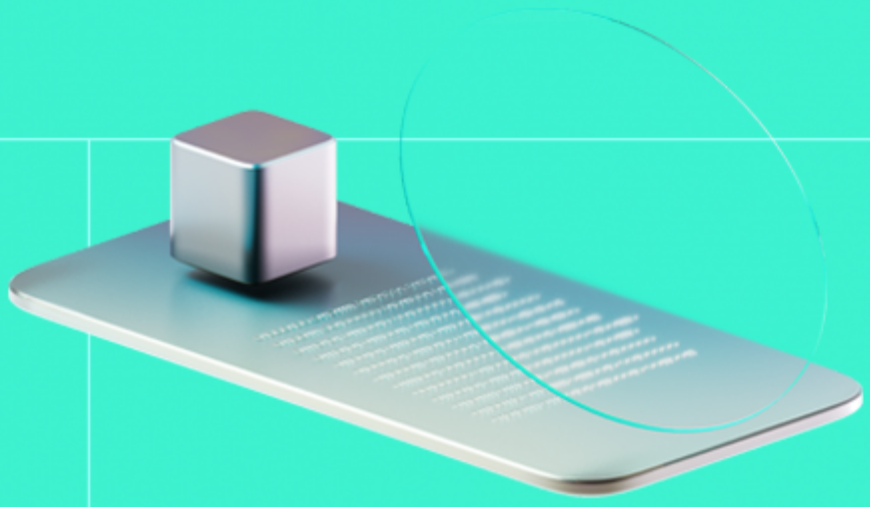
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: Gold Token SA

Date: 12/03/2026

We express our gratitude to the Gold Token SA team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

The GTSA Base is an upgradeable Optimism-compatible bridged ERC20 (DGLD) that combines bridge-controlled mint/burn mechanics with role-based governance, pause controls, and blacklist-based compliance restrictions.

Document

Name	Smart Contract Code Review and Security Analysis Report for Gold Token SA
Audited By	Kornel Światłowski
Approved By	Khrystyna Tkachuk
Website	https://www.gtsa.io/
Changelog	05/03/2026 - Preliminary Report
	12/03/2026 - Final Report
Platform	Base
Language	Solidity
Tags	Permit Token, Fungible Token, Upgradable
Methodology	https://docs.hacken.io/methodologies/smart-contracts

Review Scope

Repository	https://github.com/goldtokensa/gtsa-base-smart-contracts
Commit	82cbf9c76064e5666dece5e88021f3b1e913d5158d5b
Final Commit	88c21fdc95d182706004c49e2ff64882dc6570a6

Audit Summary

The system users should acknowledge all the risks summed up in the risks section of the report

6	5	1	0
Total Findings	Resolved	Accepted	Mitigated

Findings by Severity

Severity	Count
Critical	0
High	0
Medium	1
Low	4

Vulnerability	Severity	Status
F-2026-15301 - Unintended Default_Admin_Role Renouncement During Role Transfer	Medium	Fixed
F-2026-15295 - Missing Validation of Initialization Parameters in DGLDToken	Low	Fixed
F-2026-15296 - Absence of Granular Role Separation	Low	Accepted
F-2026-15298 - Lack of Two Step DEFAULT_ADMIN_ROLE Role Transfer Mechanism	Low	Fixed
F-2026-15300 - Permanent Loss Of DEFAULT_ADMIN_ROLE Control	Low	Fixed
F-2026-15302 - Redundant Role Admin Configuration In Initialize Function	Info	Fixed

Documentation quality

- Functional requirements are detailed.
  - Project overview is detailed
  - All roles in the system are described.
  - Use cases are described and detailed.
  - For each contract, all futures are described
- Technical description is detailed.
  - Run instructions are provided.
  - Technical specification is provided.
  - NatSpec is sufficient.

Code quality

- The development environment is configured.
- NatSpec covers the code and is detailed.

Test coverage

Code coverage of the project is **50%** (branch coverage).

- Deployment and basic user interactions are covered with tests.

Table of Contents

System Overview **6**

Privileged Roles 6

Potential Risks **8**

Findings **9**

Vulnerability Details 9

Disclaimers 20

Appendix 1. Definitions **21**

Severities 21

Potential Risks 21

Appendix 2. Scope **22**

Appendix 3. Additional Valuables **23**

System Overview

The GTSA Base is an upgradeable Optimism-compatible bridged `ERC20` (DGLD) that combines bridge-controlled mint/burn mechanics with role-based governance, pause controls, and blacklist-based compliance restrictions. It contains the following contracts:

`UpgradeableOptimismMintableERC20` contract provides the bridge-specific token foundation. It extends upgradeable `ERC20` + permit behavior and stores core bridge configuration values such as the local bridge address, remote token address, and token decimals. Its key security boundary is that minting and burning are restricted to the configured bridge through an `onlyBridge` check. It also includes backward-compatible legacy getters and a `Permit2` integration pattern where allowance to Permit2 is treated as effectively unlimited by design.

`Blacklistable` contact is the compliance-oriented control layer. It introduces blacklist storage, a dedicated blacklister role, and enforcement through a reusable notBlacklisted modifier. This module is intended to be inherited by the main token so transfer and approval flows can deny interactions from restricted accounts.

`DGLDToken` contract integrates these components into one operational contract. It inherits the bridge mint/burn model from `UpgradeableOptimismMintableERC20`, combines it with OpenZeppelin's pausable `ERC20` behavior, uses access control roles for governance and operations, and uses UUPS for upgrades. During initialization, it assigns admin, pauser, and blacklister permissions and sets the role hierarchy. In normal operation, user balance-changing actions and allowance flows are constrained by pause state and blacklist checks, while bridge mint and burn remain gated by bridge authorization from the parent contract. The result is an upgradeable bridged `ERC20` with explicit operational safety controls for production token management.

Privileged roles

The system uses `AccessControlUpgradeable` and custom access-control checks. The system defines the following privileged roles:

The `DEFAULT_ADMIN_ROLE` can call:

- grantRole(bytes32 role, address account) - grants a role to an account (including admin/pauser/blacklister roles, since admin is role-admin).
- revokeRole(bytes32 role, address account) - revokes a role from an account.
- changeRolesAdmin(address newAdmin) - transfers admin control by granting `DEFAULT_ADMIN_ROLE` to newAdmin and revoking it from the caller.
- upgradeToAndCall(address newImplementation, bytes data) - inherited UUPS upgrade entrypoint that upgrades implementation and optionally executes setup call.
- setBridge(address _bridgeAddress) - sets bridge address.
- startChangeRolesAdmin(address newAdmin) - starts the transfer of the admin role to a new address.

The `PAUSER_ROLE` can call:

- pause() - pauses the contract (halts token interactions guarded by pause checks).
- unpause() - unpauses the contract.

The `BLACKLISTER_ROLE` can call:

- blacklist(address account) - marks an account as blacklisted.
- unBlacklist(address account) - removes an account from the blacklist.

Additional privileged authority (not an AccessControl role):

- Bridge authority (BRIDGE() address) can call mint(address to, uint256 amount) and burn(address from, uint256 amount) in the base bridge model, subject to blacklist/pause behavior in `DGLDToken`.

Potential Risks

- **Unlimited Allowance Granted to Predeployed Permit2 Contract:** The Permit2 address defined in the Preinstalls library is treated as having unlimited token allowance, with the allowance() function returning type(uint256).max whenever the spender is the designated Permit2 address, regardless of the token holder. This effectively grants unrestricted transfer rights to that address. In the event of compromise, misconfiguration, or unauthorized control of the Permit2 address, an attacker could transfer tokens from any holder without limitation, potentially resulting in a total loss of user funds.
- **Single Entity Upgrade Authority:** The token ecosystem grants a single entity the authority to implement upgrades or changes. This centralization of power risks unilateral decisions that may not align with the community or stakeholders' interests, undermining trust and security.
- **Flexibility and Risk in Contract Upgrades:** The project's contracts are upgradable, allowing the administrator to update the contract logic at any time. While this provides flexibility in addressing issues and evolving the project, it also introduces risks if upgrade processes are not properly managed or secured, potentially allowing for unauthorized changes that could compromise the project's integrity and security.
- **Absence of Upgrade Window Constraints:** The contract suite allows for immediate upgrades without a mandatory review or waiting period, increasing the risk of rapid deployment of malicious or flawed code, potentially compromising the system's integrity and user assets.
- **Insufficient Multi-signature Controls for Critical Functions:** The lack of multi-signature requirements for key operations centralizes decision-making power, increasing vulnerability to single points of failure or malicious insider actions, potentially leading to unauthorized transactions or configuration changes.
- **Centralized Minting and Burning Authority via Bridge:** The DGLDToken deployed on L2 allows the designated bridge address to mint and burn tokens from any account, a privilege required for proper cross-chain bridging. If the bridge address or the off-chain system managing cross-chain communication is compromised, an attacker could arbitrarily mint or burn tokens from any account, potentially resulting in total loss of funds and severe disruption of token supply integrity.
- **Role/Governance Concentration:** `DEFAULT_ADMIN_ROLE` can manage other privileged roles. Compromise, coercion, or operational error at this layer can lead to full control over minting/burning operators, pause authority, and blacklist authority.
- **Mint/Burn Operational Trust and Collateralization:** `MINTER_ROLE` and `BURNER_ROLE` control issuance/redemption execution. Users must trust off-chain processes to correctly validate bar custody, bar identity, and redemption workflow; on-chain logic cannot independently prove physical collateral quality or existence.
- **Pause-Induced Availability:** `PAUSER_ROLE` can stop core token actions. This is useful for incident response but can also create temporary loss of transferability and reduced protocol liveness during pauses.
- **Blacklist Censorship/Freezing:** `BLACKLISTER_ROLE` can block addresses from interacting with token functions guarded by blacklist checks. This supports compliance objectives but introduces explicit censorship/funds-mobility restrictions for affected users.

Findings

Vulnerability Details

F-2026-15301 - Unintended Default_Admin_Role Renouncement During Role Transfer - Medium

Description:

The `changeRolesAdmin()` function is responsible for transferring `DEFAULT_ADMIN_ROLE` to a new address provided as a function argument. The function first grants `DEFAULT_ADMIN_ROLE` to the provided address and then revokes the role from `_msgSender()`.

```
function changeRolesAdmin(address newAdmin) external onlyRole(DEFAULT_ADMIN_ROLE) {
    if (newAdmin == address(0)) revert ZeroAddress();

    _grantRole(DEFAULT_ADMIN_ROLE, newAdmin);
    _revokeRole(DEFAULT_ADMIN_ROLE, _msgSender());
}
```

However, if the provided address is equal to the current `_msgSender()`, the function grants and subsequently revokes `DEFAULT_ADMIN_ROLE` from the same address. As a result, instead of transferring the role, the function effectively renounces it. This behavior may unintentionally remove the only existing `DEFAULT_ADMIN_ROLE`, leading to permanent loss of administrative control.

Additionally, the return values of `_grantRole()` and `_revokeRole()` are not validated. Since both functions return a boolean value indicating success or failure, lack of validation may allow silent failures and result in inconsistent role state.

Assets:

- src/DGLDToken.sol [https://github.com/goldtokensa/gtsa-base-smart-contracts]

Status: Fixed

Classification

Impact Rate: 5/5

Likelihood Rate: 3/5

Exploitability:	Dependent
Complexity:	Simple
Severity:	Medium

Recommendations

Remediation: It is recommended to validate that `newAdmin` differs from `_msgSender()` before executing the role transfer logic. Additionally, the return values of `_grantRole()` and `_revokeRole()` should be verified to ensure successful execution and prevent inconsistent role states.

Resolution: The issue was fixed in commit `88c21fd`. A validation was added to the `startChangeRolesAdmin()` function to ensure that `newAdmin` is different from the current `msg.sender`, who holds the `DEFAULT_ADMIN_ROLE`. Additionally, checks of the return values of `_grantRole()` and `_revokeRole()` were added to the `completeChangeRolesAdmin()` function, where these functions are invoked.

F-2026-15295 - Missing Validation of Initialization Parameters in DGLDToken - Low

Description:

The `DGLDToken.initialize(...)` function does not validate the `_bridge`, `_remoteToken`, and `_decimals` parameters. The same parameters are assigned without validation in the parent initializer `UpgradeableOptimismMintableERC20.__UpgradeableOptimismMintableERC20_init(...)`.

As a result, invalid configuration values may be stored during initialization. Because initialization can typically be executed only once in upgradeable deployments, misconfiguration at this stage may require a contract upgrade.

If default or incorrect values are provided and stored, the following conditions may occur:

- If `_bridge == address(0)`, functions gated by `onlyBridge`, such as `mint()` and `burn()`, become permanently unusable, breaking core bridge functionality.
- If `_remoteToken` is incorrect or set to the zero address, the token may reference an invalid L1 counterpart, leading to integration failures, incorrect bridge assumptions, or loss of interoperability.
- If `_decimals` is incorrectly configured, token representation across off-chain systems may become inconsistent, potentially causing pricing discrepancies, indexing mismatches, and accounting inconsistencies.

Because these parameters define fundamental bridge and token properties, the lack of validation introduces configuration risk with potentially long-term operational consequences.

Assets:

- src/DGLDToken.sol [https://github.com/goldtokensa/gtsa-base-smart-contracts]
- src/UpgradeableOptimismMintableERC20.sol [https://github.com/goldtokensa/gtsa-base-smart-contracts]

Status: Fixed

Classification

Impact Rate: 4/5

Likelihood Rate: 2/5

Exploitability: Dependent

Complexity: Simple

Severity: Low

Recommendations

Remediation: It is recommended to introduce explicit validation checks during initialization to ensure that `_decimals`, `_bridge` and `_remoteToken` are non-zero values, thereby preventing the permanent storage of invalid configuration values.

Resolution: The issue was fixed in commit `88c21fd`. Validation checks against the zero address for `_bridge` and `_remoteToken`, as well as a check ensuring `_decimals` is not zero, were added to the `initialize()` function of the `DGLDToken` contract.

F-2026-15296 - Absence of Granular Role Separation - Low

Description:

The `DGLDToken` contract implements the `DEFAULT_ADMIN_ROLE`, which currently grants extensive privileges across multiple administrative operations. The wide scope of authority assigned to this role introduces a security concern when it is used for day-to-day administrative tasks. For example:

```
function _authorizeUpgrade(address newImplementation) internal override
    onlyRole(DEFAULT_ADMIN_ROLE) {}
```

A compromise of the private key controlling this role could result in full unauthorized access to key protocol functions. This may lead to unauthorized configuration changes, disruption of system behavior, or full control over protocol parameters. The current design lacks role segregation, where distinct operational and administrative responsibilities are assigned to separate roles with limited scopes of authority.

Assets:

- src/DGLDToken.sol [https://github.com/goldtokensa/gtsa-base-smart-contracts]

Status: Accepted

Classification

Impact Rate: 4/5

Likelihood Rate: 2/5

Exploitability: Dependent

Complexity: Simple

Severity: Low

Recommendations

Remediation: It is recommended to introduce granular role segregation by creating separate role like `UPGRADER_ROLE` for contract upgrades. The `DEFAULT_ADMIN_ROLE` should be reserved exclusively for roles management.

Resolution: The issue was accepted by the Gold Token SA team.

F-2026-15298 - Lack of Two Step DEFAULT_ADMIN_ROLE Role Transfer Mechanism - Low

Description:

The `DGLDToken` contract utilizes `AccessControlUpgradeable` for role management. The contract includes a non-standard function `changeRolesAdmin()` that revokes the `DEFAULT_ADMIN_ROLE` from the caller and assigns it to the address provided as a function argument:

```
function changeRolesAdmin(address newAdmin) external onlyRole(DEFAULT_ADMIN_ROLE) {
    if (newAdmin == address(0)) revert ZeroAddress();

    _grantRole(DEFAULT_ADMIN_ROLE, newAdmin);
    _revokeRole(DEFAULT_ADMIN_ROLE, _msgSender());
}
```

This approach allows immediate role transfer, but entering an incorrect address can permanently assign ownership to an unintended or inaccessible address. This may result in permanent loss of administrative control.

Assets:

- src/DGLDToken.sol [https://github.com/goldtokensa/gtsa-base-smart-contracts]

Status: Fixed

Classification

Impact Rate: 4/5

Likelihood Rate: 2/5

Exploitability: Dependent

Complexity: Simple

Severity: Low

Recommendations

Remediation: It is recommended to implement a two-step "claimable" ownership transfer mechanism:

1. Introduce a pending owner variable to store the designated new `DEFAULT_ADMIN_ROLE`.
2. Require the designated address to explicitly claim the role to complete the transfer.
3. Maintain the current `DEFAULT_ADMIN_ROLE` until the claim is confirmed.

This ensures that the new `DEFAULT_ADMIN_ROLE` address is accurate and actively accepts ownership, reducing the risk of accidental loss of administrative control.

Resolution: The issue was fixed in commit `88c21fd`. The `changeRolesAdmin()` function was replaced with the newly added `startChangeRolesAdmin()` and `completeChangeRolesAdmin()` functions. The flow now requires the current `DEFAULT_ADMIN_ROLE` to initiate the process via `startChangeRolesAdmin()`, and the new admin must accept the role by calling `completeChangeRolesAdmin()`.

F-2026-15300 - Permanent Loss Of DEFAULT_ADMIN_ROLE Control - Low

Description: The `DGLDToken` contract inherits `AccessControlUpgradeable`, which allows the holder of the `DEFAULT_ADMIN_ROLE` to call `renounceRole(DEFAULT_ADMIN_ROLE, account)` or `revoke(DEFAULT_ADMIN_ROLE, account)` on their own address. If either function is executed—accidentally or maliciously—there is no on-chain mechanism to reassign `DEFAULT_ADMIN_ROLE`. This results in permanent loss of administrative privileges. Consequences include the inability to grant or revoke roles, rotate compromised keys, or manage operational roles such as `BLACKLISTER_ROLE` and `PAUSER_ROLE`, and the ability for future contract updates.

Assets:

- src/DGLDToken.sol [https://github.com/goldtokensa/gtsa-base-smart-contracts]

Status: Fixed

Classification

Impact Rate: 5/5

Likelihood Rate: 1/5

Exploitability: Dependent

Complexity: Simple

Severity: Low

Recommendations

Remediation: It is recommended to restrict `renounceRole()` and `revoke()` calls where the provided `role` argument is `DEFAULT_ADMIN_ROLE` and the `account` argument is the current `msg.sender`. This prevents accidental or malicious self-revocation, preserves administrative control, and ensures the ability to manage roles, rotate keys, and perform operational functions. This approach also guarantees that at least one `DEFAULT_ADMIN_ROLE` exists throughout the contract lifecycle.

Resolution: The issue was fixed in commit `88c21fd`. The `renounceRole()` and `revokeRole()` functions were restricted by overriding them and adding validation that prevents `DEFAULT_ADMIN_ROLE` from self-revoking or renouncing the role,

ensuring that there is always at least one address holding the `DEFAULT_ADMIN_ROLE`.

This role can still be transferred through the `completeChangeRolesAdmin()` function, which assigns the role to a new address, or revoked via the `revokeRole()` function when called by a different address.

F-2026-15302 - Redundant Role Admin Configuration In Initialize Function - Info

Description:

The `initialize()` function of the `DGLDToken` contract configures the initial `ERC20` parameters and performs privileged role assignments. The function grants roles such as `DEFAULT_ADMIN_ROLE`, `PAUSER_ROLE`, and `BLACKLISTER_ROLE` to addresses provided as function arguments.

```
function initialize(...) external virtual initializer {
    // ...

    // Set up role hierarchy: DEFAULT_ADMIN_ROLE is the admin for all custom
roles
    _setRoleAdmin(PAUSER_ROLE, DEFAULT_ADMIN_ROLE);
    _setRoleAdmin(BLACKLISTER_ROLE, DEFAULT_ADMIN_ROLE);

    // ...
}
```

Additionally, the `initialize()` function explicitly sets the admin role of `PAUSER_ROLE` and `BLACKLISTER_ROLE` to `DEFAULT_ADMIN_ROLE`. This configuration is redundant because, by default, `DEFAULT_ADMIN_ROLE` acts as the admin for all roles within the AccessControl framework unless explicitly modified.

As a result, unnecessary `_setRoleAdmin()` calls increase gas consumption during initialization without providing additional security or functional benefits.

Assets:

- src/DGLDToken.sol [https://github.com/goldtokensa/gtsa-base-smart-contracts]

Status: Fixed

Classification

Impact Rate: 1/5

Likelihood Rate: 5/5

Exploitability: Dependent

Complexity: Simple

Severity: Info

Recommendations

Remediation: It is recommended to remove redundant `_setRoleAdmin()` calls that assign `DEFAULT_ADMIN_ROLE` as the admin of roles already governed by it by default. This reduces gas consumption during initialization and simplifies the role configuration logic without affecting security or functionality.

Resolution: The issue was fixed in commit `88c21fd`. The redundant admin role assignments of `DEFAULT_ADMIN_ROLE` to the `PAUSER_ROLE` and `BLACKLISTER_ROLE` roles with the `_setRoleAdmin()` were removed, since `DEFAULT_ADMIN_ROLE` is the admin of these roles by default.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with 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).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

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.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

As part of Hacken's ongoing quality assurance process, we may conduct re-audits of select projects. These re-audits are performed independently from the original audit and are intended solely for internal quality control and improvement. Updated reports resulting from such re-audits will be shared privately with the respective clients and may be published on the Hacken website only with their explicit consent.
The sole authoritative source for finalized and most up-to-date versions of all reports remains the Audits section at https://hacken.io/audits/.

Technical Disclaimer

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 vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Definitions

Severities

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution.

Potential Risks

The "Potential Risks" section identifies issues that are not direct security vulnerabilities but could still affect the project's performance, reliability, or user trust. These risks arise from design choices, architectural decisions, or operational practices that, while not immediately exploitable, may lead to problems under certain conditions. Additionally, potential risks can impact the quality of the audit itself, as they may involve external factors or components beyond the scope of the audit, leading to incomplete assessments or oversight of key areas. This section aims to provide a broader perspective on factors that could affect the project's long-term security, functionality, and the comprehensiveness of the audit findings.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details	
Repository	https://github.com/goldtokensa/gtsa-base-smart-contracts
Commit	82cbf9c76064e5666dece5e88021f3b1e913d5158d5b
Final Commit	88c21fdc95d182706004c49e2ff64882dc6570a6
Whitepaper	N/A
Requirements	NatSpec
Technical Requirements	README.md

Asset	Type
src/Blacklistable.sol [https://github.com/goldtokensa/gtsa-base-smart-contracts]	Smart Contract
src/DGLDToken.sol [https://github.com/goldtokensa/gtsa-base-smart-contracts]	Smart Contract
src/UpgradeableOptimismMintableERC20.sol [https://github.com/goldtokensa/gtsa-base-smart-contracts]	Smart Contract

Appendix 3. Additional Valuables

Additional Recommendations

The smart contracts in the scope of this audit could benefit from the introduction of automatic emergency actions for critical activities, such as unauthorized operations like ownership changes or proxy upgrades, as well as unexpected fund manipulations, including large withdrawals or minting events. Adding such mechanisms would enable the protocol to react automatically to unusual activity, ensuring that the contract remains secure and functions as intended.

To improve functionality, these emergency actions could be designed to trigger under specific conditions, such as:

- Detecting changes to ownership or critical permissions.
- Monitoring large or unexpected transactions and minting events.
- Pausing operations when irregularities are identified.

These enhancements would provide an added layer of security, making the contract more robust and better equipped to handle unexpected situations while maintaining smooth operations.

Frameworks and Methodologies

This security assessment was conducted in alignment with recognised penetration testing standards, methodologies and guidelines, including the NIST SP 800-115 – Technical Guide to Information Security Testing and Assessment, and the Penetration Testing Execution Standard (PTES), These assets provide a structured foundation for planning, executing, and documenting technical evaluations such as vulnerability assessments, exploitation activities, and security code reviews. Hacken's internal penetration testing methodology extends these principles to Web2 and Web3 environments to ensure consistency, repeatability, and verifiable outcomes.