MO Issuance V2 - EVM - Managed Smart Contract Assessment

MO

HALBORN

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ABOUT THIS REPORT

ASSESSMENT TYPE
Assurance Assessment

SOURCE CODE
/issuance-v2-ffe117511a57c5c9d46dcdf83baf9ba6

DATE OF ENGAGEMENT
Sep 22, 2026 - Sep 29, 2026

SEVERITY BREAKDOWN

3	2	1	0
All Findings	Solved	Acknowledged	Risk Accepted



Introduction

`M0` engaged `Halborn` to perform a security assessment of the M0 Issuance V2 EVM smart contracts from September 23, 2026 to September 29, 2026. The assessment scope was limited to the Solidity contracts under `evm/src` of the `issuance-v2` repository at commit `ffe117511a57c5c9d46dcdf83baf9ba67cf6bfe6`. The Scope section of this report lists the files in scope.

M0 Issuance V2 is modular, upgradeable infrastructure for stablecoin issuance. It separates the stablecoin token logic from compliance enforcement and policy configuration:

- `IssuerGateway`: deploys stablecoin proxies through CREATE3, adopts migrated tokens, upgrades stablecoins, and controls minting, burning, pause, and the mint circuit breaker.
- `Stablecoin` and `RevenueDirect`: the ERC-20 token core. It dispatches a policy hook before every state change and accrues yield for a single yield recipient.
- `RevenueDirectFromMYieldToOne`: the migration implementation that moves an M0 v1 `MYieldToOne` token onto the v2 model.

- `ConfigRegistrar` and `ConfigPanel`: shared runtime state (pause, freeze, whitelist) and the issuer-administered role grants.
- `PolicyRegistry` and the `Policy` family: the catalog of approved, stateless compliance policies that permit or refuse each token operation.

Assessment Summary

`Halborn` assigned one full-time security engineer to assess the security and correctness of the smart contracts in scope. The engineer is a blockchain and smart contract security expert with experience in penetration testing, smart contract exploitation, and protocol design review.
The objectives of this assessment were to:

- Make sure that the gateway, token, and policy contracts operate as intended.
- Identify vulnerabilities in authorization, upgrade, migration, and yield-accounting flows.

In summary, `Halborn` identified some improvements to reduce the likelihood and impact of risks, which the `M0` team solved or acknowledged. The main ones were the following:

- Bind the adoption check to the ProxyAdmin that the proxy enforces, not to the writable ERC-1967 admin slot.
- Reserve the name of an adopted stablecoin so that registerStablecoin cannot register it again.
- Settle accrued yield for the old recipient before a yield-recipient change, or document that the new recipient receives it.

Test Approach and Methodology

`Halborn` used a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy. Manual review found logic and design flaws. Automated analysis and focused tests increased coverage and confirmed each reported issue.
The assessment included these phases:

- Architecture review of the gateway, token, registrar, and policy registry trust boundaries.
- Manual line-by-line review of all externally reachable functions, their role checks, and their pause behavior.

- Upgrade and migration review: UUPS authorization, implementation validation, storage-namespace checks, and the v1 `TransparentUpgradeableProxy` adoption path.
- Accounting review of yield accrual, claims, yield-recipient rotation, and the mint rate limit.
- Policy review of hook dispatch, the deny-by-default base, pause, freeze, whitelist, and forced-transfer rules.
- AI-assisted analysis with Halborn Neo. The engineer verified each candidate against the code and dismissed unconfirmed ones.
- Focused Foundry tests that reproduce each reported issue at the audited commit.

Risk Methodology

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

4.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILITY METRIC (m_e)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A)	1
	Specific (AO:S)	0.2
Attack Cost (AC)	Low (AC:L)	1
	Medium (AC:M)	0.67
	High (AC:H)	0.33
Attack Complexity (AX)	Low (AX:L)	1
	Medium (AX:M)	0.67
	High (AX:H)	0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

4.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (m_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (C:N)	0
	Low (C:L)	0.25
	Medium (C:M)	0.5
	High (C:H)	0.75
	Critical (C:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical (A:C)	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5

IMPACT METRIC (m_I)	METRIC VALUE	NUMERICAL VALUE
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium (Y:M)	0.5
	High (Y:H)	0.75
	Critical (Y:C)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

4.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N)	1
	Partial (R:P)	0.5
	Full (R:F)	0.25
Scope (s)	Changed (S:C)	1.25
	Unchanged (S:U)	1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Critical	High	Medium	Low	Informational
9 - 10	7 - 8.9	4.5 - 6.9	2 - 4.4	0 - 1.9

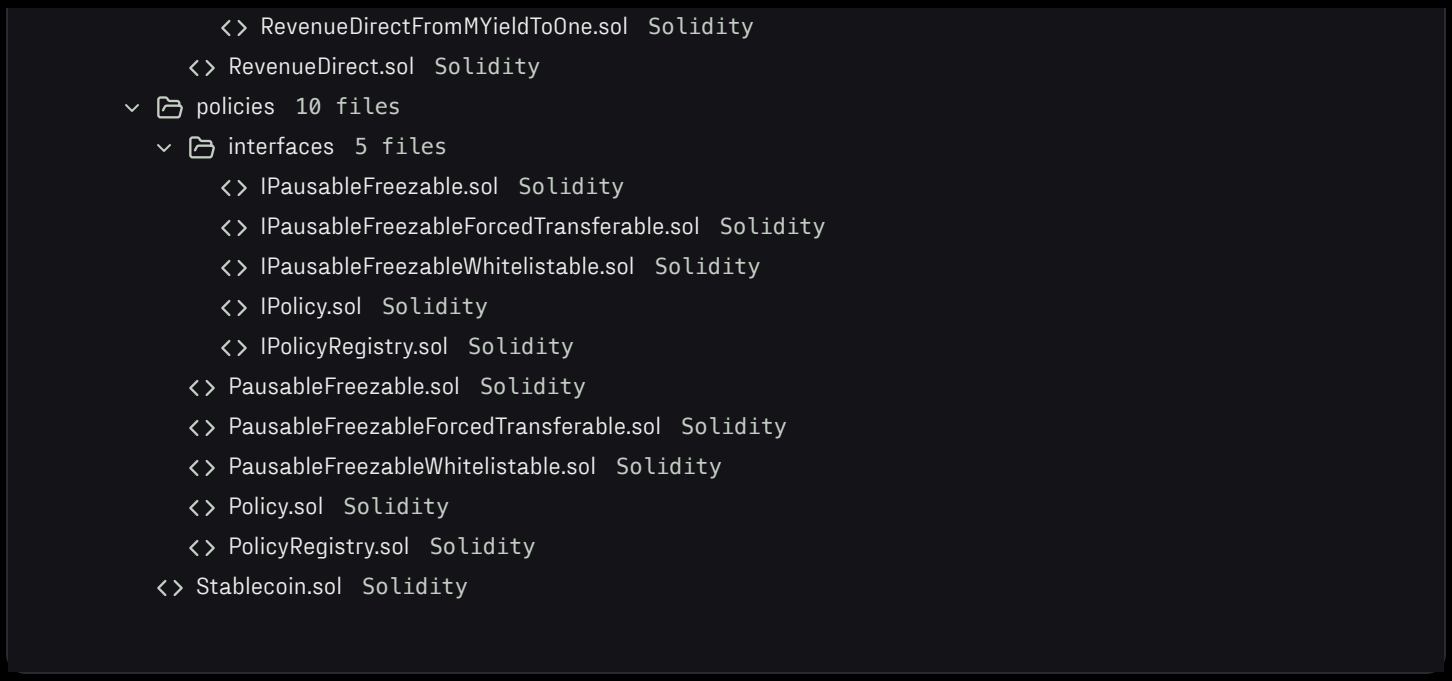
Scope

FILE

(a) Submitted File:
/issuance-v2-ffe117511a57c5c9d46dcdf83baf9ba67cf6bfe6/evm/src/interfaces/IErrors.sol

(b) Items in scope:

- issuance-v2-ffe117511a57c5c9d46dcdf83baf9ba67cf6bfe6/evm/src 26 files
  - interfaces 1 file
    - IErrors.sol Solidity
  - libs 3 files
    - ConfigKeys.sol Solidity
    - ConfigViews.sol Solidity
    - PolicyTypes.sol Solidity
  - platform 22 files
    - config 4 files
      - interfaces 2 files
        - IConfigPanel.sol Solidity
        - IConfigRegistrar.sol Solidity
      - ConfigPanel.sol Solidity
      - ConfigRegistrar.sol Solidity
    - gateway 2 files
      - interfaces 1 file
        - IIssuerGateway.sol Solidity
      - IssuerGateway.sol Solidity
    - interfaces 1 file
      - IStablecoin.sol Solidity
    - models 4 files
      - interfaces 1 file
        - IRevenueDirect.sol Solidity
      - migrations 2 files
        - interfaces 1 file
          - IRevenueDirectFromMYieldToOne.sol Solidity



REMEDIATION COMMIT ID:

- cbf760a2fecea06c2f1981dca0c064a5761105eb
- 28e981fcefcbfa7920a7505e6dff72aaeb80a2f7
- 919ed825cb81ea3d3154e9b3fc0e06b6e421b855

Out-of-Scope: New features/implementations after the remediation commit IDs.

Assessment Summary & Findings Overview

#	TITLE	SEVERITY	SCORE	STATUS
HAL-001	adoptStablecoin Trusts A Writable ERC-1967 Admin Slot	Informational	0.5	SOLVED 10/06/2026
HAL-002	Yield Recipient Change Does Not Settle Accrued Yield	Informational	0.4	ACKNOWLEDGED 10/21/2026
HAL-003	adoptStablecoin Does Not Reserve The Adopted Stablecoin Name	Informational	0.3	SOLVED 10/06/2026

Findings & Tech Details

HAL-001 SOLVED

adoptStablecoin Trusts A Writable ERC-1967 Admin Slot

Informational · 0.5 AO:S/AC:L/AX:M/R:P/S:U/C:N/I:H/A:N/D:N/Y:N

DESCRIPTION

`IssuerGateway._revertIfStillAdministered()` reads the upgrade authority from the ERC-1967 admin slot, but in an OpenZeppelin v5 `TransparentUpgradeableProxy` this slot is informational and the v1 ProxyAdmin owner can clear it while the real admin stays active.

Impact

A v1 ProxyAdmin owner can get a token adopted into the v2 fleet and then upgrade it outside every gateway control, which breaks the invariant that the gateway is the only upgrade authority.

Exploitation Steps

1. The v1 ProxyAdmin owner runs the documented `upgradeAndCall` onto `RevenueDirectFromMYieldToOne` with `migrate(...)`.
2. The owner does not call `renounceOwnership()`. The owner runs a second `upgradeAndCall` onto a helper that writes zero to the admin slot and restores the migration implementation.
3. `Stablecoin.proxyAdmin()` now returns `address(0)`, so `_revertIfStillAdministered()` does not revert.
4. The `CONFIG_MANAGER_ROLE` holder calls `adoptStablecoin()`, and the gateway records the token as registered.
5. The ProxyAdmin owner calls `upgradeAndCall` again. The proxy accepts it because OpenZeppelin v5 stores the admin as an immutable, not in the slot.
6. The token now runs code that no `UPGRADER_ROLE` holder vetted, and it can mint, burn, or move balances without policy hooks.

Code Location

`evm/src/platform/gateway/IssuerGateway.sol:795-805` — `IssuerGateway._revertIfStillAdministered()`
The adoption gate trusts the address that the token reports, and the real ProxyAdmin can change that answer.

SOL 795-805

```
795     /// @dev    Reverts when the proxy's ERC-1967 admin slot names a ProxyAdmin that
796 still has an owner:
797     ///         that owner could upgrade the token past every gateway check. A gateway-
798 deployed proxy
799     ///         has an empty slot; a migrated one must have had its ProxyAdmin renounced
800 first.
801     /// @param  stablecoin The stablecoin being adopted.
802     function _revertIfStillAdministered(address stablecoin) internal view {
803         address proxyAdmin = IStablecoin(stablecoin).proxyAdmin();
804 
805         if (proxyAdmin != address(0) && Ownable(proxyAdmin).owner() != address(0)) {
                revert StablecoinStillAdministered(stablecoin, proxyAdmin);
            }
        }
```

`evm/src/platform/Stablecoin.sol:175-178` — `Stablecoin.proxyAdmin()`

The getter returns the storage slot, which code that runs in the proxy context can overwrite. It does not return the immutable admin that `TransparentUpgradeableProxy` enforces.

SOL 175-178

```
175     /// @inheritdoc IStablecoin
176     function proxyAdmin() external view returns (address) {
177         return ERC1967Utils.getAdmin();
178     }
```

RECOMMENDATION

Do not use the ERC-1967 admin slot to prove that no upgrade authority remains. Make `adoptStablecoin()` take the expected ProxyAdmin address as an argument, and verify that the proxy runtime code embeds this address, for example with an expected codehash. Then require `Ownable(proxyAdmin).owner()` to return `address(0)` before the gateway records the token.

REMEDIATION COMMENT

Addressed in: cbf760a2fecea06c2f1981dca0c064a5761105eb

SOLVED: The `M0` team solved this finding in the specified commit by making `_revertIfStillAdministered()` derive the ProxyAdmin address from the proxy's first CREATE (nonce 1), which is the immutable admin that an OpenZeppelin v5 `TransparentUpgradeableProxy` enforces, and by refusing adoption unless that contract is absent or is an `Ownable` with no owner. The writable ERC-1967 admin slot is no longer used as proof that no upgrade authority remains.

HAL-002 ACKNOWLEDGED

Yield Recipient Change Does Not Settle Accrued Yield

Informational · 0.4 AO:S/AC:L/AX:M/R:P/S:C/C:N/I:N/A:N/D:N/Y:M

DESCRIPTION

`RevenueDirect.setYieldRecipient()` writes the new recipient without a prior claim, so the next `claim()` mints all yield accrued under the old recipient to the new recipient.

Impact

When the yield recipient is a partner and not the issuer, the issuer's `YIELD_MANAGER_ROLE` can move the partner's full unclaimed yield to a new recipient, which M0 v1 prevented because it claimed before the change. The loss is limited to the backlog since the last `claim()`, the partner can limit it with the permissionless `claim()`, and no holder principal is at risk.

Exploitation Steps

1. The stablecoin is registered with a partner, A, as `yieldRecipient`, and yield accrues for a period with no `claim()`.
2. The `YIELD_MANAGER_ROLE` holder calls `IssuerGateway.setYieldRecipient(stablecoin, B)`.
3. Any account calls the permissionless `claim()`.
4. The token mints the full backlog, including yield accrued under A, to B. A receives nothing.

A `claim()` that executes before step 2 pays A. This order gives the correct attribution and is not harmful.

Code Location

`evm/src/platform/models/RevenueDirect.sol:314-324` — `RevenueDirect._setYieldRecipient()`
The function changes the recipient but does not claim the yield that accrued under the old recipient.

SOL 314-324

```sol
function _setYieldRecipient(address yieldRecipient_) internal {
    if (yieldRecipient_ == address(0)) revert ZeroYieldRecipient();

    RevenueDirectStorageStruct storage $ = _getRevenueDirectStorageLocation();

    if ($.yieldRecipient == yieldRecipient_) return;

    $.yieldRecipient = yieldRecipient_;

    emit YieldRecipientSet(yieldRecipient_);
}
```

`evm/src/platform/models/RevenueDirect.sol:87-104` — `RevenueDirect.claim()`

The function is permissionless and pays the full `accruedYield()` to the recipient stored at call time, independent of when the yield accrued.

SOL 87–103

```sol
function claim() external returns (uint256 claimed) {
    claimed = accruedYield();

    if (claimed == 0) return 0;

    RevenueDirectStorageStruct storage $ = _getRevenueDirectStorageLocation();
    address yieldRecipient_ = $.yieldRecipient;

    IPolicy(policy()).beforeClaimChecks(address(this), msg.sender, yieldRecipient_);

    _updateIndex();

    unchecked {
        // NOTE: ...
        $.totalSupply += claimed;
        $.balances[yieldRecipient_] += claimed;
    }
```

RECOMMENDATION

Claim the accrued yield to the current recipient before `_setYieldRecipient()` writes the new recipient, as M0 v1 did. If the policy refuses that claim, for example because the current recipient is frozen, skip the claim so that the recipient change can still execute.

REMEDIATION COMMENT

Addressed in: 28e981fcefcbfa7920a7505e6dff72aaeb80a2f7

ACKNOWLEDGED: The `M0` team acknowledged this finding. In the specified commit, the team made `claim()` permissioned: only the issuer gateway, under `YIELD_MANAGER_ROLE`, or a holder of `CLAIMER_ROLE` scoped to the stablecoin can claim. As part of this change, the team removed the settlement that `setYieldRecipient()` did in commit `53cc4bf`. The yield accrued under the old recipient now goes to the new recipient at the next claim. The team states that this behavior is intended: the yield manager decides whether the old recipient receives the backlog, and claims first in the same `multicall` when needed.

HAL-003 SOLVED

adoptStablecoin Does Not Reserve The Adopted Stablecoin Name

Informational · 0.3 AO:S/AC:L/AX:L/R:P/S:C/C:N/I:L/A:N/D:N/Y:N

DESCRIPTION

`IssuerGateway.adoptStablecoin()` records an existing proxy but does not consume the CREATE3 salt of its name, so `CONFIG_MANAGER_ROLE` can call `registerStablecoin()` with that name and register a second stablecoin that reports the same `name()`.

Impact

The gateway breaks its documented "one name, one stablecoin" property, and `getStablecoinAddress(name)` returns an address that is not the adopted token, so an integrator that resolves by name can use the wrong token. Only the trusted `CONFIG_MANAGER_ROLE` can trigger this, and no gateway accounting or user funds change.

Exploitation Steps

1. The ProxyAdmin owner of a v1 token upgrades it onto `RevenueDirectFromMYieldToOne`, calls `migrate`, and renounces ownership.
2. `CONFIG_MANAGER_ROLE` calls `adoptStablecoin(legacy, mintConfig)`. The gateway records the token but reserves no salt for its name.
3. `CONFIG_MANAGER_ROLE` calls `registerStablecoin()` with `params.name` equal to the name of the adopted token. CreateX deploys a new proxy because the salt is free.
4. As a result, two registered stablecoins report the same `name()` and `symbol()`. `getStablecoinAddress(name)` returns the new proxy, not the adopted token.

Code Location

`evm/src/platform/gateway/IssuerGateway.sol:254-273` — `IssuerGateway.adoptStablecoin()`
The adoption path sets `registeredStablecoins` but never reads `name()` or reserves the name salt that `registerStablecoin()` uses at lines 234-240.

SOL 254-273

```sol
254     function adoptStablecoin(
255         address stablecoin,
256         MintConfig calldata mintConfig
257     ) external onlyRole(CONFIG_MANAGER_ROLE) whenNotPaused {
258         if (stablecoin.code.length == 0) revert InvalidStablecoin(stablecoin);
259
260
```

```
        IssuerGatewayStorageStruct storage $ = _getIssuerGatewayStorageLocation();

        if ($.registeredStablecoins[stablecoin]) revert
StablecoinAlreadyRegistered(stablecoin);

        address implementation = IStablecoin(stablecoin).implementation();

        _validateStablecoinImplementation(implementation);

        address policy = IStablecoin(stablecoin).policy();

        _revertIfPolicyNotRegistered(stablecoin, policy);
        _revertIfStillAdministered(stablecoin);

        $.registeredStablecoins[stablecoin] = true;
```

`evm/src/platform/gateway/IssuerGateway.sol:762-777` — `IssuerGateway._computeStablecoinSalt()`

Name uniqueness depends only on this salt, and the NatSpec states that a name is single-use per gateway.

SOL 762-777

```
    /// @dev    Computes the CREATE3 salt for a stablecoin proxy: the token's ERC-20
name, prefixed
    ///         with this gateway's own address to opt into CreateX's permissioned deploy
protection.
    ///         A name is therefore single-use per gateway, and lands on the same address
on every
    ///         chain where the gateway itself has the same address.
    /// @param  name The token's ERC-20 name.
    /// @return The computed salt.
    function _computeStablecoinSalt(string calldata name) private view returns (bytes32)
{
        return
            bytes32(
                abi.encodePacked(
                    bytes20(address(this)),
                    bytes1(0), // disable cross-chain redeploy protection
                    bytes11(keccak256(abi.encode(name)))
                )
            );
    }
```

RECOMMENDATION

In `adoptStablecoin()`, read `IStablecoin(stablecoin).name()` and store its hash in a reserved-names mapping in the gateway storage. Revert `adoptStablecoin()` if that hash is already reserved, and revert `registerStablecoin()` if the hash of `params.name` is already reserved. With this change, each name maps to exactly one registered stablecoin, whichever path registered it.

REMEDIATION COMMENT

Addressed in: 919ed825cb81ea3d3154e9b3fc0e06b6e421b855

SOLVED: The `M0` team solved this finding in the specified commit by recording every registered or adopted stablecoin under its ERC-20 name in a new `stablecoinsByName` mapping, by reverting with `StablecoinNameTaken` in both `registerStablecoin()` and `adoptStablecoin()` when the name is already held, and by making `getStablecoinAddress()` return the recorded holder before the predicted CREATE3 address.

Disclaimer

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.