Adevar Labs

Prepared for:

M0 Foundation

October 29, 2026

Issuance V2 EVM Audit Report

Version: after fix review

Table of Contents

1. Executive Summary 4

About M0 Issuance V2 4

Audit Summary 4

Overall Security Posture 4

Launch Recommendations 5

Audit Scope 5

2. Assumptions and Considerations 7

Audit Assumptions 7

Centralization Risks 7

Trust Assumptions 7

3. Severity Definitions 8

Impact 8

Likelihood 8

Severity Classification Matrix 8

4. Findings 10

M01: adoptStablecoin can be passed while the v1 ProxyAdmin keeps upgrade rights 10

L01: Blocking revocations while a spender is frozen will let it spend a holder's stale allowance after unfreeze 14

L02: Missing scoped-only enforcement will let a plain role holder edit all whitelists in ConfigRegistrar 18

5. Enhancement Opportunities 20

E01: Submitting targeted freezes privately will improve containment by removing the pre-inclusion window to move funds 20

E02: An optional per-stablecoin mint pause could improve operational control in IssuerGateway 22

E03: Add mToken to the Migrated event so a log consumer can identify the swept asset 24

© 2026 Adevar Labs. All rights reserved.

About Us **25**

About Adevar Labs **25**

Audit Methodology 25

Confidentiality Notice 27

Legal Disclaimer 27

© 2026 Adevar Labs. All rights reserved.

1. Executive Summary

About M0 Issuance V2

M0 Issuance V2 is modular, upgradeable infrastructure for permissioned stablecoin issuance, designed for GENIUS Act compliance. M0 deploys one instance per issuer, made of an Issuer Gateway, a Config Registrar, a Config Panel and a Policy Registry. The gateway deploys and adopts stablecoins, mints through a delayed proposal flow or a direct path under a per-token rate limit, and upgrades token code. Before every state change, each token calls a stateless policy that reads pause, freeze and whitelist state from the registrar. Yield streams to one recipient per token, and holders receive none. This report covers the EVM contracts. The SVM program is covered in a separate report.

Audit Summary

The table below summarizes identified vulnerabilities by severity and remediation status.

Severity	Count	Fixed	Acknowledged
Critical	0	0	0
High	0	0	0
Medium	1	1	0
Low	2	1	1

Enhancement opportunities: 3

Overall Security Posture

The codebase is well structured and carefully documented. Roles are narrowly scoped, minting is bounded by a rate limit and a circuit breaker, with a delay on proposed mints, and the compliance controls hold once committed. We found no Critical or High issues: 1 Medium issue in the v1 adoption check, 2 Low issues and 3 enhancement opportunities.

Before Fix Review Summary

The Medium issue let the owner of a v1 ProxyAdmin keep upgrade rights through adoption, because **adoptStablecoin** checked the writable ERC-1967 admin slot instead of the admin the proxy enforces. The 2 Low issues are a plain whitelist role that can edit every stablecoin's whitelist at the registrar, and a holder who cannot revoke an allowance while the spender is frozen, so the spender can use it right after the unfreeze. The enhancements name the swept token in the **Migrated** event, add a per-stablecoin mint pause, and suggest submitting targeted freezes privately.

© 2026 Adevar Labs. All rights reserved.

After Fix Review Summary

M0 fixed the Medium issue and the allowance revocation issue as recommended, and implemented the **Migrated** event and per-stablecoin mint pause enhancements. The whitelist role issue is acknowledged: M0 documented that only the Config Panel should hold the plain role, and that granting it to another account is the registrar admin's decision. Private submission of freezes is acknowledged as an operational choice for each issuer and chain.

M0 also made **claim()** permissioned in PR #63: only the issuer gateway, under **YIELD_MANAGER_ROLE**, or a holder of **CLAIMER_ROLE** scoped to the stablecoin can claim yield, and **setYieldRecipient** no longer claims on its own.

Launch Recommendations

Before the fix review, we recommended:

1. Check the admin a v1 proxy actually enforces before adopting it, not the writable admin slot.
2. Let holders revoke an allowance at any time, whatever the freeze state.
3. Keep the plain **WHITELIST_MANAGER_ROLE** with the Config Panel only.
4. Submit freezes and unwhitelists through a private path on chains with a public mempool.

After the fix review, items 1 and 2 are implemented.

Before launch, we recommend:

- After each deployment and role change, check that the Config Panel is the only holder of the plain **WHITELIST_MANAGER_ROLE**.
- Follow the migration runbook: seed the v1 freeze list before the upgrade, pass the genuine **M** token and M0's solver to **migrate**, and before **adoptStablecoin**, confirm the proxy's real ProxyAdmin (for example from the **AdminChanged** event in its deployment transaction) and check that its owner has renounced.
- Brief issuers on submitting compliance actions privately, as M0 plans to.

After launch, alert on grants of **WHITELIST_MANAGER_ROLE** and transfers of **DEFAULT_ADMIN_ROLE**, on **Migrated** events, on mint configuration changes and mint pauses, and on stablecoin upgrades.

Audit Scope

- **Repository:** https://github.com/m0-platform/issuance-v2
- **Before Fix Review Commit Hash:** **07519df4a4582cb22da40d008d62f508cdd0e0a3**
- **After Fix Review Commit Hash:** **28e981fcefcbfa7920a7505e6dff72aaeb80a2f7**
- **Files/Modules in Scope:**
  - **evm/src**

© 2026 Adevar Labs. All rights reserved.

- **Exclusions:**
  - **evm/lib**, including **m0-foundation/common**, a previously audited dependency reviewed only where the platform uses it

All remediation pull requests (#39, #42, #43, #44, #45, #47 and #63) have been merged into **main**. Commit **28e981fcefcbfa7920a7505e6dff72aaeb80a2f7**, the merge of #63, is the last of them. Later commits on **main**, up to October 29, 2026, change **evm/src** only in its license headers (PR #69), and add upgrade scripts outside the scope (PR #67), which we did not review.

© 2026 Adevar Labs. All rights reserved.

2. Assumptions and Considerations

Audit Assumptions

I. Honest earner rate: The issuer sets **earnerRate** to match the yield earned on its off-chain reserve. In V2 the rate moved from M0 to the issuer, and the contracts cannot check it. M0 confirmed this assumption.

II. Freeze list seeded first: Before the migration upgrade, the operator copies the full v1 freeze list into the v2 registrar. M0 confirmed that this is the first step of the migration runbook, with the list derived from the v1 token's freeze events.

III. Trusted migration procedure: M0 runs each v1 migration. The v1 **MYieldToOne** storage layout matches what **migrate** reads, and the **mToken** and **mRecipient** that M0 passes are the genuine **M** token and M0's solver.

IV. Audited policies: M0 registers only audited policies in the Policy Registry.

Centralization Risks

- The issuer admin (**DEFAULT_ADMIN_ROLE** on the gateway and the registrar) grants every operational role and upgrades the gateway and the registrar. A stolen admin key controls the issuer's whole fleet.
- **UPGRADER_ROLE** sets the stablecoin implementation and upgrades tokens with arbitrary initialization data, which runs with the gateway as caller.
- Issuer roles mint, burn, pause, freeze, whitelist and seize. M0 controls the Policy Registry, and so which policies a token can use.

Trust Assumptions

- Off-chain reserves back the supply 1:1. Nothing on-chain proves it.
- The issuer, M0 and the partner project a coin is issued for do not collude.
- **m0-foundation/common** (index math, ERC-20 extensions, permits and authorizations) behaves as previously audited. This review covered how the platform uses it.

© 2026 Adevar Labs. All rights reserved.

3. Severity Definitions

Each issue identified in this report is assigned a severity level based on two dimensions: **Impact** and **Likelihood**. These dimensions help convey both the potential consequences of a vulnerability and how likely a vulnerability is to be discovered and exploited in the real world.

Note: Enhancements represent non-blocking improvements, typically usability, observability, or defense-in-depth tweaks that do not pose an immediate asset risk but would improve the product's reliability and/or user experience if implemented.

Impact

Impact reflects the potential consequences of the issue, particularly on **project funds**, **user funds**, and the **availability or integrity** of the protocol.

- **High Impact**: Successful exploitation could result in a complete loss of user or protocol funds, disruption of core protocol functionality, or permanent loss of control over critical components.
- **Medium Impact**: Exploitation could cause significant disruption or partial loss of funds, but not a total compromise. May impact some users or non-core functionality.
- **Low Impact**: The issue has minor or negligible consequences. It may affect edge cases, expose metadata, or degrade performance slightly without putting funds or core logic at serious risk.

Likelihood

Likelihood reflects how easy a vulnerability is to discover and exploit by an attacker, as well as how economically attractive the exploit is to an attacker.

- **High Likelihood**: The vulnerability is trivially exploitable. This means it can be exploited by a wide range of actors without privileged access rights, with minimal capital requirements and low financial risks.
- **Medium Likelihood**: This type of vulnerability can be found and exploited with moderate effort. It might require a significant capital investment, but with manageable financial risk.
- **Low Likelihood**: Exploitation of these vulnerabilities is often technically infeasible or requires highly specialized conditions. They may require extraordinary effort or a significant financial risk for an attacker, with a high chance of failure and minimal potential return.

Severity Classification Matrix

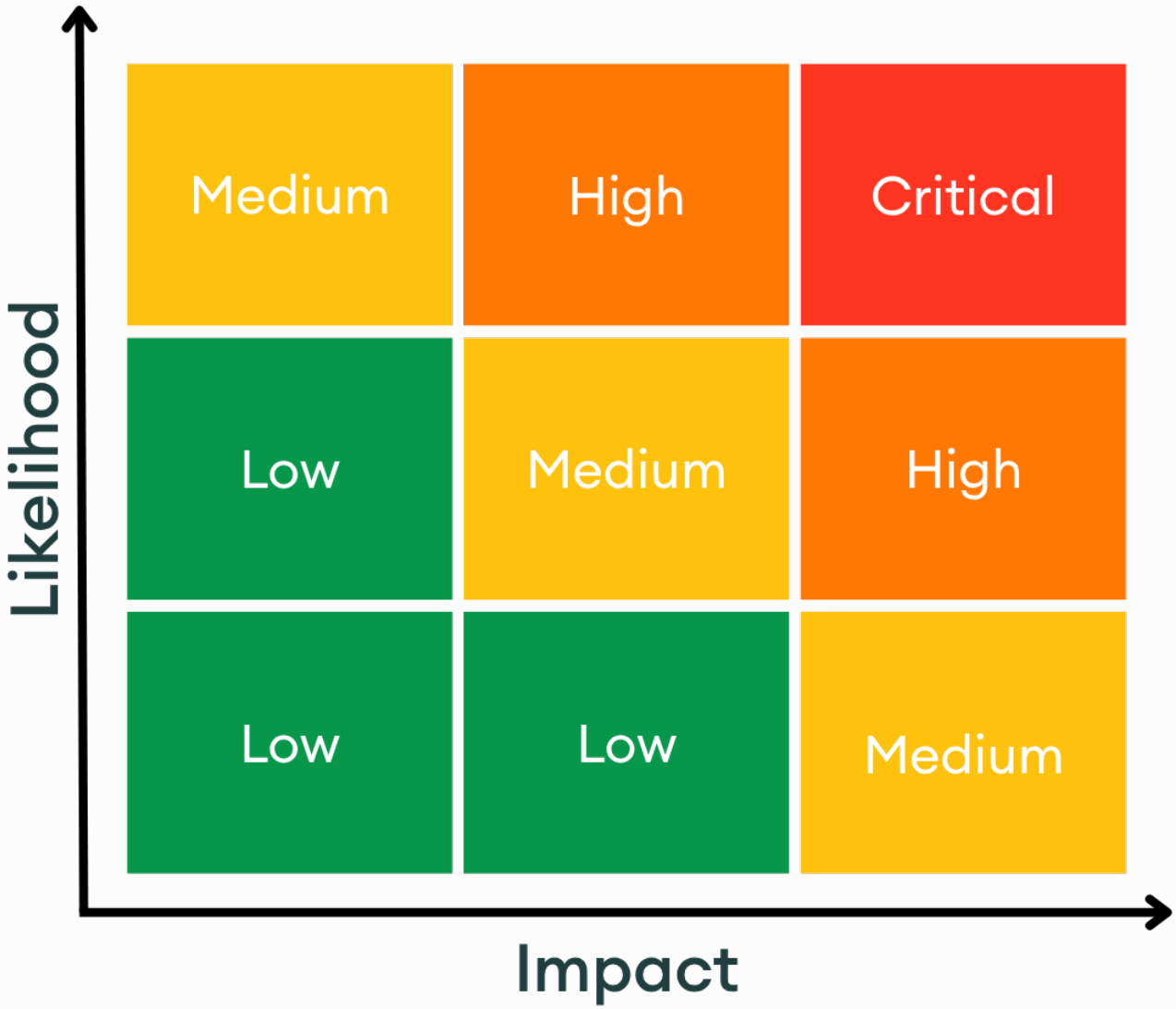
By combining **Impact** and **Likelihood**, we assign a severity level using the matrix below:

- **Critical**: High impact + high likelihood (e.g. a bug that could allow anyone to drain a substantial amount of protocol funds with minimal effort)
- **High**: High impact with medium likelihood, or medium impact with high likelihood
- **Medium**: Moderate impact and/or discoverability

© 2026 Adevar Labs. All rights reserved.

- **Low**: Minimal impact or unlikely to be exploited

This structured approach helps teams prioritize fixes and mitigate the most dangerous threats first.



Severity Matrix

© 2026 Adevar Labs. All rights reserved.

4. Findings

M01: adoptStablecoin can be passed while the v1 ProxyAdmin keeps upgrade rights



LOCATION

issuance-v2/07519df4/evm/src/platform/gateway/IssuerGateway.sol:L736-L744

RELEVANT SNIPPET

>_ IssuerGateway.sol — SOLIDITY

```
734: /// @dev    Reverts when the proxy's ERC-1967 admin slot names a ProxyAdmin that still
       has an owner.
735: /// @param  stablecoin The stablecoin being adopted.
736: function _revertIfStillAdministered(address stablecoin) internal view {
737:     address proxyAdmin = IStablecoin(stablecoin).proxyAdmin();
738:
739:     // NOTE: OZ v5 can never clear the admin slot, so a migrated token passes only onc
       e its ProxyAdmin
740:     //       is ownerless; an owned one could upgrade the token past every gateway che
       ck.
741:     if (proxyAdmin != address(0) && Ownable(proxyAdmin).owner() != address(0)) {
742:         revert StablecoinStillAdministered(stablecoin, proxyAdmin);
743:     }
744: }
745:
746: /// @dev    Reverts unless a stablecoin is in this gateway's fleet.
```

OTHER LOCATIONS

- issuance-v2/07519df4/evm/src/platform/Stablecoin.sol:L166-L168

© 2026 Adevar Labs. All rights reserved.

DESCRIPTION

adoptStablecoin adds an already-deployed proxy to the gateway's stablecoins. Its main use is a token migrated from M0 v1, such as Citrea USD, which sits behind an OpenZeppelin v5 **TransparentUpgradeableProxy**. Before adoption, the v1 ProxyAdmin owner is expected to call **renounceOwnership()** so that the gateway becomes the token's only upgrade authority.

The gateway enforces this in **_revertIfStillAdministered**:

```solidity
address proxyAdmin = IStablecoin(stablecoin).proxyAdmin(); // ERC1967Utils.getAdmin()
if (proxyAdmin != address(0) && Ownable(proxyAdmin).owner() != address(0)) {
    revert StablecoinStillAdministered(stablecoin, proxyAdmin);
}
```

proxyAdmin() reads the ERC-1967 admin storage slot. An OZ v5 Transparent proxy does not use that slot to authorize upgrades. It stores the admin as **address private immutable _admin** and checks **msg.sender == _proxyAdmin()** against that immutable. The constructor writes the slot once, only as information. The OZ documentation warns about this directly: the admin slot "can still be overwritten by the implementation logic pointed to by this proxy", and the slot can then differ "from the actual admin".

Because the ProxyAdmin owner decides which implementation the proxy runs, it can run code that sets the admin slot to zero, or to any contract whose **owner()** returns zero, and never renounce. The check then passes, and the proxy still accepts **upgradeToAndCall** from the original ProxyAdmin.

This contradicts the documented intent. The project documentation says an owned ProxyAdmin "could upgrade the token past every gateway check, so the invariant that the gateway is the only upgrade authority is enforced at the door rather than left to an ops step". The **adoptStablecoin** NatSpec says it reverts unless the token has "no owned ProxyAdmin". With a modified slot, an owned ProxyAdmin passes that check.

Once adopted, the token is treated as a normal member. Its old ProxyAdmin owner can replace its code at any time and skip every gateway control: the rate limit, mint delay, pause, circuit breaker, policy registry checks and the **ModelStorageLocationMismatch** guard.

The v1 ProxyAdmin owner, which is a privileged party, has to act on purpose. Nothing in the adoption flow detects it, and the config manager sees a normal successful adoption.

PROOF OF CONCEPT

The actor is the owner of the v1 ProxyAdmin **PA** for the Citrea USD proxy.

1. **PA.upgradeAndCall(proxy, RevenueDirectFromMYieldToOne, abi.encodeCall(migrate, (...)))**. **migrate** checks **msg.sender == ERC1967Utils.getAdmin()**. The slot still holds **PA**, so it passes.
2. **PA.upgradeAndCall(proxy, helper, abi.encodeCall(Helper.run, ()))**, where **Helper.run** runs in the proxy's storage context:

© 2026 Adevar Labs. All rights reserved.

```solidity
function run() external {
       StorageSlot.getAddressSlot(ERC1967Utils.ADMIN_SLOT).value = address(0);
       StorageSlot.getAddressSlot(ERC1967Utils.IMPLEMENTATION_SLOT).value = REVENUE_DIRECT_FRO
  M_M_YIELD_TO_ONE;
   }
```

The proxy now runs the valid implementation, the admin slot is zero, the immutable **_admin** is still **PA**, and **PA.owner()** is still the attacker.
3. The config manager calls **adoptStablecoin(proxy, mintConfig)**. **implementation()** passes **_validateStablecoinImplementation**, **policy()** is the registry's current entry, and **proxyAdmin()** returns zero, so **_revertIfStillAdministered** does not revert. The token is recorded in **registeredStablecoins**.
4. **PA.upgradeAndCall(proxy, maliciousImpl, data)** still succeeds, because the proxy checks the immutable **_admin**. The adopted token now runs any code the attacker picks.

For comparison, the existing fork test (**evm/test/fork/CitreaMigration.t.sol**, **_renounceThenAdopt**) only covers the honest case, where the slot still equals the real admin.

RECOMMENDATION

Derive the real admin instead of reading the ERC-1967 admin slot. An OZ v5 **TransparentUpgradeableProxy** creates its own ProxyAdmin in its constructor (**_admin = address(new ProxyAdmin(initialOwner))**). Unless the initializer deploys a contract first, this is the proxy's first CREATE, so the ProxyAdmin lives at the CREATE address of the proxy with nonce **1**. Implementation code cannot change that address, and it is the address the proxy checks upgrades against. For the Citrea USD proxy **0x8D82...6d5D**, this gives **0x8023...480f**, which is the **PROXY_ADMIN** used in **CitreaMigration.t.sol**.

```diff
function _revertIfStillAdministered(address stablecoin) internal view {
-    address proxyAdmin = IStablecoin(stablecoin).proxyAdmin();
+    // An OZ v5 TransparentUpgradeableProxy deploys its ProxyAdmin as its first CREATE (nonce
   1)
+    // and authorizes upgrades against that address, not against the writable ERC-1967 admin
  slot.
+    address proxyAdmin = address(
+        uint160(uint256(keccak256(abi.encodePacked(bytes1(0xd6), bytes1(0x94), stablecoin,
  bytes1(0x01)))))
+    );

-    if (proxyAdmin != address(0) && Ownable(proxyAdmin).owner() != address(0)) {
+    if (proxyAdmin.code.length != 0 && Ownable(proxyAdmin).owner() != address(0)) {
         revert StablecoinStillAdministered(stablecoin, proxyAdmin);
     }
 }
```

After this change, a rewritten admin slot no longer lets an owned ProxyAdmin pass. An honest migration still passes once the owner has renounced. Gateway-deployed **ERC1967Proxy** tokens create no contract, so the derived address has no code and the check is skipped for them.

© 2026 Adevar Labs. All rights reserved.

The derivation only covers OZ v5 Transparent proxies, which are the documented migration source. A custom proxy with its own upgrade path would still need the config manager to verify it before adoption. Keeping an allowlist of expected proxy and ProxyAdmin pairs would enforce that on-chain.

FIX REVIEW NOTES

Fixed according to the recommendation: **_revertIfStillAdministered** derives the ProxyAdmin from the proxy's nonce-1 CREATE address instead of reading the ERC-1967 admin slot. One edge case remains. When the initializer passed as **_data** to the proxy's constructor deploys a contract, that contract takes nonce 1 and the ProxyAdmin lands at nonce 2. The check then looks at the wrong address, and an owned admin can pass **adoptStablecoin**. Catching this on-chain would require reading the real **_admin** from the proxy's bytecode. M0 documented the edge case in PR #47 and in its migration runbook.

© 2026 Adevar Labs. All rights reserved.

L01: Blocking revocations while a spender is frozen will let it spend a holder's stale allowance after unfreeze



LOCATION

issuance-v2/07519df4/evm/src/platform/policies/PausableFreezable.sol:L48-L62

RELEVANT SNIPPET

>_ PausableFreezable.sol — SOLIDITY

```
46:  }
47:
48:  /// @dev    Rejects an approval while either party is frozen, revocations included.
49:  ///         Pause does not apply.
50:  /// @param  stablecoin The stablecoin the check is performed for.
51:  /// @param  owner      The address whose tokens would be approved for spending.
52:  /// @param  spender    The address that would be allowed to spend the tokens.
53:  function _beforeApproveChecks(
54:      address stablecoin,
55:      address,
56:      address owner,
57:      address spender,
58:      uint256
59:  ) internal view virtual override {
60:      _revertIfFrozen(stablecoin, owner);
61:      _revertIfFrozen(stablecoin, spender);
62:  }
63:
64:  /// @dev    Rejects a mint while the stablecoin is paused or the recipient is frozen.
```

OTHER LOCATIONS

- issuance-v2/07519df4/evm/src/platform/Stablecoin.sol:L217-L221
- issuance-v2/07519df4/evm/src/platform/config/ConfigPanel.sol:L100-L104
- issuance-v2/07519df4/evm/src/platform/config/ConfigPanel.sol:L132-L136
- issuance-v2/07519df4/evm/test/integration/Compliance.t.sol:L320-L346

DESCRIPTION

PausableFreezable._beforeApproveChecks reverts whenever the owner or the spender is frozen, whatever the amount. **Stablecoin._approve** calls it before writing the allowance, and every holder-side allowance write goes through **_approve**: **approve** and both **permit** overloads in **ERC20ExtendedUpgradeable** (**lib/common** at **81ec16e**). There is no **decreaseAllowance**. So while a spender is frozen, the holder cannot set its allowance to zero. The NatSpec says this is intended ("revocations included"), and three tests assert it.

© 2026 Adevar Labs. All rights reserved.

The frozen spender cannot use the allowance either, since **_beforeTransferChecks** rejects a frozen caller, and a reverted **transferFrom** does not consume it. The gap opens at unfreeze. **ConfigPanel.unfreeze** only clears the freeze-list entry. It does not touch allowances and gives the holder no window to revoke first. From the block the unfreeze lands in, the old allowance is live again, and the holder's revocation races the spender's **transferFrom**. A spender that watches the freeze list and gets ordered first can take **min(allowance, balance)**.

The holders who hit this are the ones who tried to revoke, usually because the spender was frozen for a reason (a phishing address, a compromised contract). The same race applies when the holder is the frozen party and is later unfrozen.

Two mitigations exist, but nothing enforces them. A holder who is not frozen can move its balance to a fresh address during the freeze. The issuer can keep the token paused across the unfreeze, since approvals ignore pause. For the spender to succeed, the token must be unpaused and the holder, spender and recipient must all be unfrozen (and whitelisted, where the policy requires it) when the **transferFrom** executes.

PROOF OF CONCEPT

Executed against the unmodified audit commit with Foundry 1.8.3 and solc 0.8.36, reusing the **ComplianceIntegrationTest** setup:

SOLIDITY

```
// evm/test/integration/FrozenRevocationPoC.t.sol
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.8.36;

import { IPausableFreezable } from "../../src/platform/policies/interfaces/IPausableFreezable.
  sol";

import { ComplianceIntegrationTest } from "./Compliance.t.sol";

contract FrozenRevocationPoC is ComplianceIntegrationTest {
    function test_poc_spenderFrontRunsRevocationAfterUnfreeze() external {
        address attackerRecipient = makeAddr("attackerRecipient");

        // 1. alice approves carol.
        vm.prank(alice);
        tokenA.approve(carol, amount);

        // 2. carol is frozen.
        _freeze(stablecoinA, carol);

        // 3. alice tries to revoke while carol is frozen. The revocation reverts and the allo
  wance stays.
        vm.expectRevert(abi.encodeWithSelector(IPausableFreezable.AccountFrozen.selector, stab
  lecoinA, carol));
        vm.prank(alice);
        tokenA.approve(carol, 0);

        assertEq(tokenA.allowance(alice, carol), amount);

        // 4. carol is unfrozen. Nothing touches the allowance.
```

© 2026 Adevar Labs. All rights reserved.

... continued SOLIDITY

```
        _unfreeze(stablecoinA, carol);

        assertEq(tokenA.allowance(alice, carol), amount);

        // 5. carol is ordered before alice's renewed revocation and spends the full allowance
  .
        vm.prank(carol);
        tokenA.transferFrom(alice, attackerRecipient, amount);

        // 6. alice's revocation now lands, too late.
        vm.prank(alice);
        tokenA.approve(carol, 0);

        assertEq(tokenA.balanceOf(attackerRecipient), amount);
        assertEq(tokenA.balanceOf(alice), 0);
    }
}
```

```
[PASS] test_poc_spenderFrontRunsRevocationAfterUnfreeze() (gas: 506221)
```

Steps 1 to 4 match the existing **test_freeze_blocksRevocation**. That test has **alice** revoke first after the unfreeze. Step 5 shows that nothing guarantees that ordering.

RECOMMENDATION

Let a zero-amount approval through regardless of freeze state, and keep rejecting nonzero approvals with a frozen party:

DIFF

```
-    /// @dev    Rejects an approval while either party is frozen, revocations included.
+    /// @dev    Rejects an approval while either party is frozen. A revocation (zero amount)
  is always allowed.
     ///         Pause does not apply.
     function _beforeApproveChecks(
         address stablecoin,
         address,
         address owner,
         address spender,
-        uint256
+        uint256 amount
     ) internal view virtual override {
+        if (amount == 0) return;
+
         _revertIfFrozen(stablecoin, owner);
         _revertIfFrozen(stablecoin, spender);
     }
```

With this change the non-fork suite passes apart from the three tests that assert the current behaviour: **test_beforeApproveChecks_spenderFrozenRevocation** (unit), **test_freeze_blocksRevocation**

© 2026 Adevar Labs. All rights reserved.

(integration) and phase 11 of **test_lifecycle_stablecoin**. These would need to flip to expect a successful revocation.

If blocking revocations is a compliance requirement, an alternative is an unfreeze procedure that keeps the token paused until affected holders have had a chance to revoke.

FIX REVIEW NOTES

Fixed according to the recommendation: **_beforeApproveChecks** now lets a zero-amount approval through regardless of freeze state.

© 2026 Adevar Labs. All rights reserved.

L02: Missing scoped-only enforcement will let a plain role holder edit all whitelists in ConfigRegistrar



LOCATION

issuance-v2/07519df4/evm/src/platform/config/ConfigRegistrar.sol:L119-L132

RELEVANT SNIPPET

>_ ConfigRegistrar.sol — SOLIDITY

```
117:
118: /// @inheritdoc IConfigRegistrar
119: function setStablecoinListEntry(
120:     address stablecoin,
121:     bytes32 list,
122:     address account,
123:     bool included
124: ) external returns (bool) {
125:     _revertIfZeroStablecoin(stablecoin);
126:     _revertIfZeroList(list);
127:     _revertIfZeroAccount(account);
128:
129:     _revertIfUnauthorized(stablecoin, getListManagerRole(list));
130:
131:     return _setListEntry(stablecoin, list, account, included);
132: }
133:
134: /// @inheritdoc IConfigRegistrar
```

OTHER LOCATIONS

- issuance-v2/07519df4/evm/src/platform/config/ConfigRegistrar.sol:L109-L116

DESCRIPTION

The docs say each stablecoin's whitelist is managed only by that coin's own managers. **ConfigKeys** describes **WHITELIST_MANAGER_ROLE** as "held scoped", and **IConfigPanel** says "only the stablecoin-scoped id authorizes anything, a plain grant confers nothing". The panel follows this rule, since each whitelist function in **ConfigPanel** calls **_revertIfUnauthorizedScoped**, which accepts only the role scoped to the target stablecoin.

The registrar does not apply the same rule. **setStablecoinListEntry** calls **_revertIfUnauthorized**, which returns early when the caller holds the plain role, and for the whitelist that plain role is **WHITELIST_MANAGER_ROLE**. Any holder of the plain role can therefore edit any stablecoin's whitelist by calling the registrar directly, which contradicts the docs quoted above. The panel relies on this

© 2026 Adevar Labs. All rights reserved.

check: **ConfigRegistrar.initialize** grants the panel the plain role, and the panel writes to the registrar as itself after checking the caller's scoped role.

Other accounts can hold the plain role as well. **AccessControlDefaultAdminRulesUpgradeable** limits only **DEFAULT_ADMIN_ROLE** to one holder, and the registrar's **grantRole** override only rejects the zero address, so the admin can grant **WHITELIST_MANAGER_ROLE** to a second account while the panel keeps it. That account can then add or remove accounts on every stablecoin's whitelist. Under **PausableFreezableWhitelistable**, a removed account can no longer send, receive, mint or claim the token, and an added account skips the approval of that coin's own whitelist manager.

Pause and freeze work differently. **ConfigViews** combines their global and per-coin entries, so a global manager already affects every coin by design. The whitelist has no global entry, so only the panel should hold the plain role.

RECOMMENDATION

Store the panel address in **initialize**, which already receives and validates it. Then tighten **_revertIfUnauthorized**:

SOLIDITY

```
function _revertIfUnauthorized(address stablecoin, bytes32 role) internal view {
    if (msg.sender == _configPanel && hasRole(role, msg.sender)) return;
    if (hasStablecoinRole(stablecoin, role, msg.sender)) return;

    revert UnauthorizedForStablecoin(msg.sender, stablecoin, role);
}
```

If M0 prefers to keep the registrar key-blind, at minimum document in **IConfigRegistrar** that the plain **WHITELIST_MANAGER_ROLE** must be held only by the panel. Also add a post-deployment check that no other account holds it.

DEVELOPER RESPONSE

Acknowledged. M0 documented that the registrar accepts a plain **WHITELIST_MANAGER_ROLE** for every stablecoin's whitelist, that the Config Panel holds that role by construction, and that granting it to any other account is the registrar admin's decision (PR #42).

© 2026 Adevar Labs. All rights reserved.

5. Enhancement Opportunities

E01: Submitting targeted freezes privately will improve containment by removing the pre-inclusion window to move funds

LOCATION

issuance-v2/07519df4/evm/src/platform/config/ConfigPanel.sol:L84-L88

RELEVANT SNIPPET

>_ ConfigPanel.sol SOLIDITY

```
82:
83: /// @inheritdoc IConfigPanel
84: function freeze(address account) external {
85:     _revertIfUnauthorizedGlobally(FREEZE_MANAGER_ROLE);
86:
87:     _setGlobalFreeze(account, true);
88: }
89:
90: /// @inheritdoc IConfigPanel
```

OTHER LOCATIONS

- issuance-v2/07519df4/evm/src/platform/config/ConfigPanel.sol:L116-L120
- issuance-v2/07519df4/evm/src/platform/config/ConfigPanel.sol:L164-L168
- issuance-v2/07519df4/evm/src/platform/Stablecoin.sol:L198-L211
- issuance-v2/07519df4/evm/src/platform/Stablecoin.sol:L227-L239

DESCRIPTION

ConfigPanel.freeze, **unwhitelist** and their batch variants take the target address as calldata, and the restriction only exists once the transaction is included. Until then, **Stablecoin._transfer** checks the committed state. A targeted holder who sees the pending transaction can move its whole balance to another address it controls, as long as that address is not frozen and, under the whitelist policy, is already whitelisted. The freeze then lands on an empty address.

For tokens on the forced-transfer policy, the follow-up seizure fails too. **_forceTransfer** from the now-frozen address passes the frozen-source check but reverts in **_revertIfInsufficientBalance**, and the successor address has to be found, frozen and seized separately.

The contracts enforce every restriction correctly once it is committed, so this is not a defect in the code. It comes from broadcasting compliance actions publicly, applies to freeze functions on any chain with observable pending transactions, and needs the token to be unpaused. We didn't see in the repository how M0 or issuers plan to submit these transactions. Are target-bearing freezes and unwhitelists sent through a private path with no public fallback, and on which chains?

POTENTIAL BENEFIT

The targeted holder cannot learn about a freeze or unwhitelist before it takes effect. The restriction then covers the full balance at the targeted address, and on seizure-enabled tokens that balance stays recoverable with one **forceTransfer** instead of a chase across successor addresses.

© 2026 Adevar Labs. All rights reserved.

RECOMMENDATION

Submit target-bearing compliance transactions (**freeze**, **freezeAccounts**, **unwhitelist**, **unwhitelistAccounts**, and any **forceTransfer** that follows) through a private submission path, such as a private RPC or builder relay, with no automatic fallback to the public mempool. Document the expected path per deployment chain, since some L2s have no public mempool and may already meet this.

Where no private path is available, pausing the token first with **ConfigPanel.pause(stablecoin)** does not reveal the target, and the freeze can be applied before unpausing. The pause is itself public and stops every holder, so it suits urgent cases rather than routine ones.

DEVELOPER RESPONSE

Acknowledged. Submission is an operational choice for each issuer and chain, not a contract matter. M0 will make sure issuers know the best practices for broadcasting compliance actions.

© 2026 Adevar Labs. All rights reserved.

E02: An optional per-stablecoin mint pause could improve operational control in IssuerGateway

LOCATION

issuance-v2/07519df4/evm/src/platform/gateway/IssuerGateway.sol:L42

RELEVANT SNIPPET

>_ IssuerGateway.sol — SOLIDITY

```
40: /// @custom:storage-location erc7201:M0.storage.IssuerGateway
41: struct IssuerGatewayStorageStruct {
42:     bool mintPaused;
43:     uint48 mintNonce;
44:     address stablecoinImplementation;
```

OTHER LOCATIONS

- issuance-v2/07519df4/evm/src/platform/gateway/IssuerGateway.sol:L426-L433

DESCRIPTION

IssuerGateway stores one mintPaused flag for the whole fleet. GUARDIAN_ROLE flips it through pauseMint, and proposeMint, both mint overloads and nothing else check it through _revertIfMintPaused. If one stablecoin has a problem (a compromised issuer key for that token, a reserve mismatch, a bad policy), the guardian can only stop minting of every stablecoin the gateway serves.

The gateway offers two workarounds, and both have costs:

- CONFIG_MANAGER_ROLE calls setMintConfig with rateLimitCapacity = 1 and rateLimitRefillPerSecond = 0. _setMintConfig refuses a zero capacity with ZeroRateLimitCapacity, so one base unit stays mintable. The call is whenNotPaused, so it fails once the guardian has paused the gateway, and it needs CONFIG_MANAGER_ROLE instead of the guardian. It also drains the bucket for good: when CONFIG_MANAGER_ROLE later restores the real config, _setMintConfig refills the remainder at the outgoing config (capacity 1, zero refill) and clamps it, so the token restarts with at most one unit of capacity and waits for a full refill.
- A registrar pauser pauses the token itself. PausableFreezable then blocks transfers, burns and claims as well, so holders cannot move or redeem while the issuer only wanted to stop new supply.

The interface describes pauseMint as the tool that "pauses every mint path, leaving burns and configuration live", and a scoped version of the same switch would give the guardian that behavior for one token.

POTENTIAL BENEFIT

The guardian can halt minting for one stablecoin in one call, while the gateway is paused or not, without freezing holders or touching the other tokens' mint capacity.

© 2026 Adevar Labs. All rights reserved.

RECOMMENDATION

Add a per-stablecoin flag next to the global one and check both on every mint path. Append the mapping at the end of the ERC-7201 struct so the storage layout stays compatible.

DIFF

```
struct IssuerGatewayStorageStruct {
         ...
         mapping(address stablecoin => bool registered) registeredStablecoins;
+        mapping(address stablecoin => bool paused) stablecoinMintPaused;
     }

+    function pauseMint(address stablecoin) external onlyRole(GUARDIAN_ROLE) {
+        _revertIfStablecoinNotRegistered(stablecoin);
+        _setStablecoinMintPaused(stablecoin, true);
+    }
+
+    function unpauseMint(address stablecoin) external onlyRole(GUARDIAN_ROLE) {
+        _setStablecoinMintPaused(stablecoin, false);
+    }

-    function _revertIfMintPaused() internal view {
-        if (_getIssuerGatewayStorageLocation().mintPaused) revert MintCircuitBreakerTripped()
  ;
+    function _revertIfMintPaused(address stablecoin) internal view {
+        IssuerGatewayStorageStruct storage $ = _getIssuerGatewayStorageLocation();
+        if ($.mintPaused || $.stablecoinMintPaused[stablecoin]) revert
  MintCircuitBreakerTripped();
     }
```

FIX REVIEW NOTES

Fixed according to the recommendation: a per-stablecoin **pauseMint**/**unpauseMint** flag, appended to the storage struct, is checked with the fleet-wide one on every mint path.

© 2026 Adevar Labs. All rights reserved.

E03: Add mToken to the Migrated event so a log consumer can identify the swept asset

LOCATION

issuance-v2/07519df4/evm/src/platform/models/migrations/RevenueDirectFromMYieldToOne.sol:L79-L81

RELEVANT SNIPPET

>_ RevenueDirectFromMYieldToOne.sol — SOLIDITY

```
77:
78:     // NOTE: M reverts on failure rather than returning false, so the result is not ch
      ecked.
79:     if (mHeld != 0) IERC20(mToken).transfer(mRecipient, mHeld);
80:
81:     emit Migrated(mRecipient, mHeld);
82: }
83:
```

DESCRIPTION

migrate sweeps the token's whole balance of the caller-supplied **mToken** to **mRecipient**, then emits **Migrated(mRecipient, mHeld)**. The event records where the balance went and how much, not which token moved. A migration run with an unintended **mToken** emits an event indistinguishable from a correct one. Confirming which asset left requires decoding the transaction calldata.

POTENTIAL BENEFIT

An indexer can confirm each migration swept the intended backing asset, and filter on it, without decoding calldata.

RECOMMENDATION

Add **mToken** as an indexed parameter.

SOLIDITY

```
event Migrated(address indexed mToken, address indexed mRecipient, uint256 amount);
```

FIX REVIEW NOTES

Fixed according to the recommendation: **Migrated** now emits **mToken** as its first indexed parameter.

© 2026 Adevar Labs. All rights reserved.

About Us

About Adevar Labs

Adevar Labs is a boutique blockchain security firm specializing in Web3 audits.

The firm was built by a mix of experienced professionals from traditional enterprise security and crypto natives who have contributed to some of the most critical projects in blockchain infrastructure.

Our team's background spans companies like Bitdefender, Asymmetric Research, Quantstamp, Chainproof, and Juicebox, and includes experience securing smart contracts, bridges, and L1 and L2 protocols across ecosystems like Solana, Ethereum, Polkadot, Cosmos, and MultiversX.

With over 100 audits completed and a portfolio that includes custom fuzzers, exploit modeling, and runtime testing frameworks, Adevar Labs brings both depth and precision to every engagement.

Our auditors have discovered critical vulnerabilities, built high-impact tooling, and placed in top positions in premier audit competitions including Code4rena and Sherlock.

Team members hold distinctions such as PhDs in software protection, key roles at Fortune 500 companies, and leadership of flagship conferences like ETH Bucharest.

With team members whose publications have earned over 1,100 academic citations, and with the trust of projects securing over $500M in on-chain value, Adevar Labs blends elite technical rigor with real-world security impact.

We also collaborate with some of the best independent security researchers in the Web3 space.

Projects may optionally request specific contributors to be part of their audit.

Audit Methodology

Our audit methodology is designed to provide thorough security assessments of Ethereum smart contracts. We focus explicitly on rigorous manual analysis, detailed threat modeling, and careful validation of implemented fixes to ensure your smart contracts operate securely and as intended.

1. Smart Contract Context and Architecture Analysis

Our auditors begin by deeply examining your Ethereum smart contract's documentation and intended functionality. We meticulously map out contract interactions, transaction processing flows, and state management logic. Special attention is given to understanding how your smart contract interfaces with critical EVM standards, such as ERC-20, ERC-721, or ERC-4626 tokens, and governance/timelock modules. Last but not least we check external integrations with other DeFi protocols and verify if the inputs and return values are handled properly.

© 2026 Adevar Labs. All rights reserved.

2. Threat Modeling

We conduct targeted threat modeling tailored specifically for the EVM execution environment. We carefully define attacker capabilities and identify potential vulnerabilities that may arise from EVM-specific issues, including but not limited to:

- Unauthorized state variable modification
- Improper Access Control, role checks (**onlyOwner**, **require(hasRole)**) or **msg.sender** verification
- Misuse of low-level calls (**call**, **delegatecall**) or proxy/upgradeability patterns (e.g., storage collisions)
- Incorrect use of external calls, including reentrancy vectors
- Risks stemming from proxy initialization logic or denial-of-service related to gas limitations

3. In-depth Manual Security Review

Our experienced auditors perform an extensive manual security review of your Solidity-based Ethereum smart contracts. This involves a comprehensive line-by-line inspection of source code, focusing on common EVM vulnerabilities including, but not limited to:

- Missing or insufficient Access Control and Role Checks
- Inadequate msg.sender and transaction origin checks
- Incorrect handling of external calls and invocation privileges, particularly potential reentrancy risks
- Arithmetic and integer overflow or underflow errors
- Unsafe use of ABI Encoding/Decoding or low-level assembly
- Improper ERC-20/Token Handling (e.g., approval logic, flash loan vectors, token locking)
- Logic flaws in financial operations or state transitions
- Edge-case handling in function input validation
- Potential denial-of-service vectors related to gas usage and transaction execution

During this stage, we clearly document any discovered vulnerabilities, including detailed descriptions, precise severity ratings, and recommendations for secure implementation. In situations where it is not clear how the vulnerability might be exploited we may also include a detailed proof-of-concept exploit including code snippets and instructions on how the exploit could be performed.

4. Detailed Fix Review and Validation

After the initial audit and your team's subsequent remediation efforts, we perform a comprehensive fix review to ensure the vulnerabilities identified have been effectively resolved.

© 2026 Adevar Labs. All rights reserved.

We verify each fix individually, confirming that:

- Corrections effectively eliminate the security risks
- Changes do not inadvertently introduce new vulnerabilities or regressions
- The fixes align closely with EVM best practices and secure coding guidelines

Our detailed validation ensures that security improvements are robust, complete, and aligned with best practices specific to the EVM development ecosystem.

Confidentiality Notice

This report, including its content, data, and underlying methodologies, is subject to the confidentiality and feedback provisions in your agreement with Adevar Labs. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Adevar Labs.

Legal Disclaimer

The review and this report are provided by Adevar Labs on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Adevar Labs disclaims all warranties, express or implied, in connection with this report, its content, and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

You agree that access to and/or use of the report and other results of the review, including any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time of the engagement.

You acknowledge that blockchain technology remains under development and is subject to unknown risks and flaws. Adevar Labs does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open-source or third-party software, code, libraries, materials, or information accessible through the report. As with the purchase or use of a product or service in any environment, you should use your best judgment and exercise caution where appropriate.

© 2026 Adevar Labs. All rights reserved.