

Facility Contracts Audit Report

Table of Contents

1. Executive Summary4

About PACT Protocol Facility Contracts4

Audit Summary4

Overall Security Posture4

After Fix Review Summary4

Before Fix Review Summary4

Launch Recommendations5

Audit Scope5

2. Assumptions and Considerations6

Audit Assumptions6

Centralization Risks6

Trust Assumptions6

3. Severity Definitions7

Impact7

Likelihood7

Severity Classification Matrix7

4. Findings9

L01: configureWaterfall() cannot atomically replace recipient allowlists9

L02: Unrestricted holder growth can render distribute() uncallable12

5. Enhancement Opportunities14

E01: Complete the receivable custody lifecycle with an explicit outbound handling path14

E02: Add address(0) validation15

E03: Document or enforce EOA-only signer assumptions in AttestationRegistry16

E04: Document or enforce the expected vehicle / facility deployment invariant in FundingManager.initialize()17

E05: Consider enforcing freshness against payload evaluation timestamps, not only attestation submission time	18
E06: Consider additional ownership-management hardening for AttestationRegistry	19
E07: Immutable admin registry reference across core contracts	20
E08: forceCloseCapitalCall does not emit CapitalClaimed events during claim loop	21
E09: cancelCapitalCall sets active = false after _clearCallState, inconsistent with other termination paths	22
About Us	24
About Adevar Labs	24
Audit Methodology	24
Confidentiality Notice	26
Legal Disclaimer	26

1. Executive Summary

About PACT Protocol Facility Contracts

PACT Protocol Facility Contracts are a smart-contract system for managing private credit facilities. The in-scope contracts cover facility and vehicle logic, capital-call and funding flows, share-class accounting, waterfall distributions, and attestation-backed covenant and borrowing-base checks.

Audit Summary

The table below summarizes identified vulnerabilities by risk level and remediation status.

Risk Level	Count	Fixed	Acknowledged
Critical	0	0	0
High	0	0	0
Medium	0	0	0
Low	2	0	0

Enhancement Opportunities: 9

Overall Security Posture

The PACT Protocol Facility Contracts present a solid baseline security posture for the scoped facility workflow. No critical, high, or medium-severity issues were identified during this review, though several low-severity hardening and operational items should be addressed before production deployment.

After Fix Review Summary

All reported findings and enhancements have been addressed. The updated codebase is modular and generally well-structured across the facility, funding, waterfall, and attestation flows, with no outstanding issues remaining from this review.

Before Fix Review Summary

The codebase is modular and generally well-structured across the facility, funding, waterfall, and attestation flows. The main improvements relate to hardening administrative safety and better enforcing or documenting intended operating assumptions.

Launch Recommendations

Before launch, the team should complete the identified hardening work, document supported deployment configurations, and place governance and upgrade authority behind a well-controlled multisig. After deployment, attestation timeliness and distribution operations should be monitored closely.

Audit Scope

- **Repository:** <https://github.com/Lucid-Fi/pact-evm-contracts>
- **Before Fix Review Commit Hash:** [4f58fd3a2cb8d41af8a4f496428982fb0847ec45](#)
- **After Fix Review Commit Hash:** [040b5a0da795929f5c1e2674394aca19978504b6](#)
- **Files/Modules in Scope:**
 - [src/Facility.sol](#)
 - [src/Vehicle.sol](#)
 - [src/EnumeratingShareClass.sol](#)
 - [src/CapitalCallManager.sol](#)
 - [src/FundingManager.sol](#)
 - [src/WaterfallExecutor.sol](#)
 - [src/AttestationRegistry.sol](#)
 - [src/ScopedWhitelist.sol](#)
 - [src/interfaces/IFacility.sol](#)
 - [src/interfaces/IFacilityBorrowingBase.sol](#)
 - [src/interfaces/IVehicle.sol](#)
 - [src/interfaces/IVault.sol](#)
 - [src/interfaces/IShareClass.sol](#)
 - [src/interfaces/ICapitalCallManager.sol](#)
 - [src/interfaces/IFundingManager.sol](#)
 - [src/interfaces/IWaterfallExecutor.sol](#)
 - [src/interfaces/IAttestationRegistry.sol](#)
 - [src/interfaces/IScopedWhitelist.sol](#)
 - [src/types/Types.sol](#)
 - [src/lib/SafeSupportsInterface.sol](#)

2. Assumptions and Considerations

Audit Assumptions

The following assumptions and design constraints materially affect the protocol's risk profile.

- Attestation freshness depends partly on operational timing: **submittedAt** is assumed to be a reasonable proxy for when the underlying analysis was performed.
- The attester signer model is effectively EOA-only: contract-wallet / ERC-1271 signers are not supported without additional validation logic.
- Unsolicited receivable NFTs are assumed to be economically inert: direct transfers outside the intended pledge flow are not expected to affect accounting or collateral status.
- **EnumeratingShareClass** is intended for a small, operationally bounded holder set. It assumes holder growth remains constrained: distributions iterate the live holder set onchain.
- Payment-token integrations assume standard ERC-20 behavior: fee-on-transfer, rebasing, hook-enabled, or other non-standard tokens may break accounting or lifecycle flows.

Centralization Risks

- Governance and upgrade roles are powerful: privileged actors can change critical configuration, authorize upgrades, and affect protocol safety and liveness.
- **AttestationRegistry** is a key centralization point: its owner controls signer management, thresholds, pause state, and upgrades.
- Vehicle and funding administration remains privileged: admin-controlled configuration can materially affect distributions, recipient allowlists, and share-class behavior.

Trust Assumptions

- The attester network is trusted to produce accurate and timely off-chain analyses.
- Identity registries and recipient allowlists are trusted to reflect the intended compliance state.
- Operators are trusted to deploy, wire, and administer the system correctly.

3. Severity Definitions

Each issue identified in this report is assigned a severity level based on two dimensions: **Impact** and **Likelihood**. These dimensions help project our team's understanding of 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 unfeasible 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.

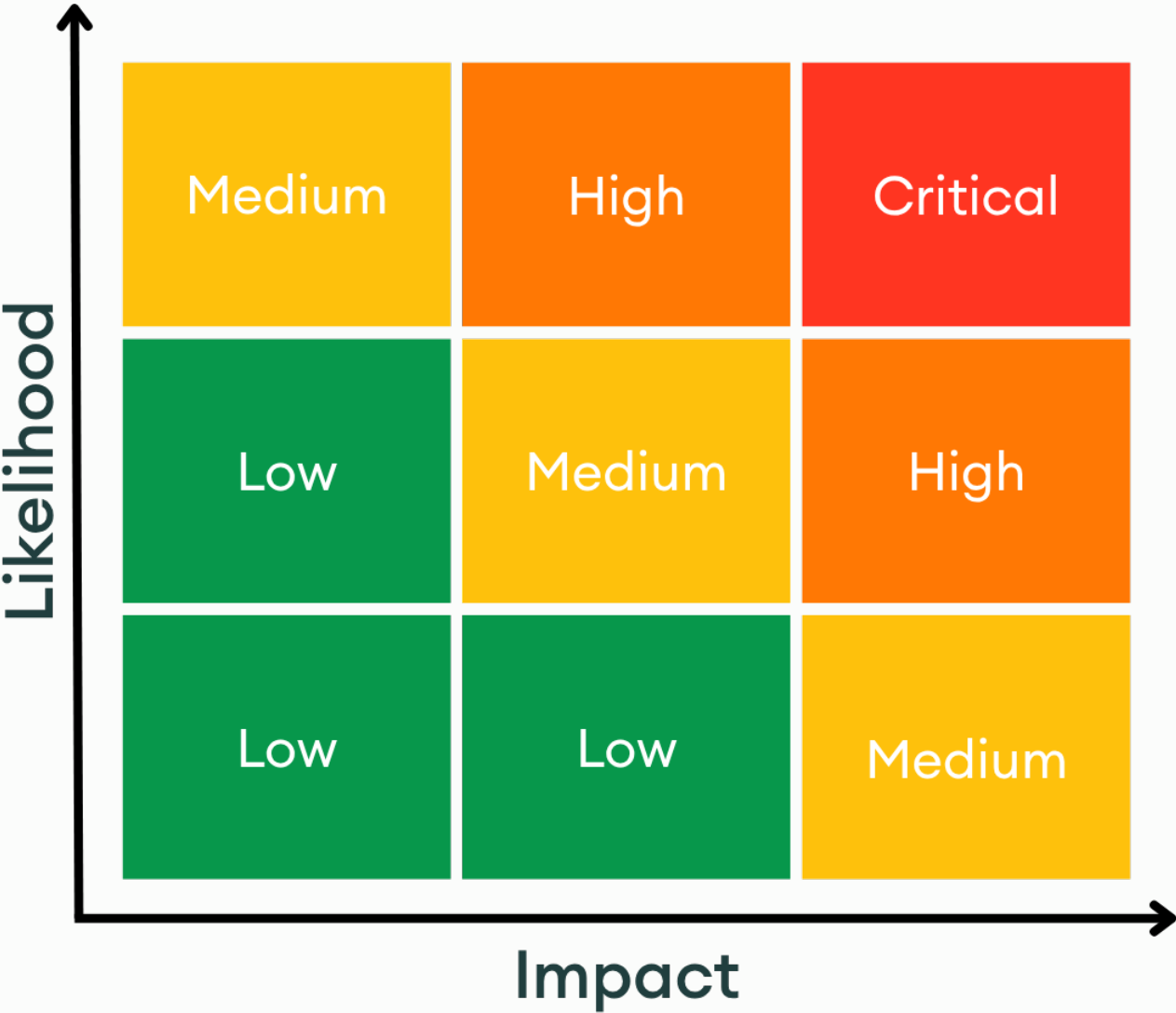
Severity Classification Matrix

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

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

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



Severity Matrix

4. Findings

L01: `configureWaterfall()` cannot atomically replace recipient allowlists



LOCATION

pact-evm-contracts/4f58fd3a/src/WaterfallExecutor.sol:L173-L175 [↗](#)

RELEVANT SNIPPET

```
> _WaterfallExecutor.sol SOLIDITY
171: _validWaterfallIds[waterfallId] = true;
172:
173: for (uint256 i; i < allowedRecipients.length; ++i) {
174:     _recipientWhitelist.addToScope(waterfallId, allowedRecipients[i]);
175: }
176:
177: emit WaterfallConfigured(waterfallId, allowedAccounts);
```

DESCRIPTION

`configureWaterfall()` overwrites the allowed-account bitmap, but recipient updates are append-only. New recipients are added to the scoped whitelist, but previously authorized recipients are never removed. As a result, reconfiguring a waterfall with a narrower recipient set does not revoke omitted addresses, they remain eligible in the whitelist and can still receive disbursements.

This also makes recipient rotation operationally fragile. If the new configuration includes any address already present in the scope, `ScopedWhitelist.sol` reverts with `AlreadyInScope()`, so admins cannot safely reuse `configureWaterfall()` as an idempotent “set current recipients” operation.

```
SOLIDITY
if (_whitelist[scope][addr]) revert AlreadyInScope();
```

PRACTICAL IMPACT

The admin may unknowingly leave deauthorized recipients eligible to receive waterfall disbursements. An attester-crafted distribution plan targeting a stale recipient would pass the whitelist check and execute.

RECOMMENDATION

Make recipient updates explicit and atomic. Because the external scoped whitelist does not support enumeration, the executor cannot infer which previously authorized recipients should be revoked during a reconfiguration. A low-impact fix is to add an overloaded `configureWaterfall()` that accepts `removedRecipients`, removes those addresses first, and treats duplicate additions/removals idempotently. This preserves the current initialization flow while allowing safe recipient rotation in one transaction.

DIFF

```
diff --git a/src/interfaces/IWaterfallExecutor.sol b/src/interfaces/IWaterfallExecutor.sol
--- a/src/interfaces/IWaterfallExecutor.sol
+++ b/src/interfaces/IWaterfallExecutor.sol
@@ -37,6 +37,17 @@ interface IWaterfallExecutor {
    address[] calldata allowedRecipients
    ) external;

+   /// @notice Reconfigure a waterfall ID, updating allowed accounts and recipient deltas
+   /// atomically.
+   /// @dev Admin only. Because the external scoped whitelist has no on-chain enumeration,
+   /// callers must explicitly supply any recipients that should be removed.
+   /// @param waterfallId The waterfall template identifier.
+   /// @param allowedAccounts Collection account types allowed for this waterfall.
+   /// @param allowedRecipients Addresses to add to the allowed-recipient scope.
+   /// @param removedRecipients Addresses to revoke from the allowed-recipient scope.
+   function configureWaterfall(
+       bytes32 waterfallId,
+       AccountType[] calldata allowedAccounts,
+       address[] calldata allowedRecipients,
+       address[] calldata removedRecipients
+   ) external;
+
+   /// @notice Set a pluggable validator for a waterfall ID. address(0) disables validation.
+   /// @dev Admin only. The caller must ensure `validator` implements IWaterfallValidator;
+   /// no on-chain interface check is performed.
diff --git a/src/WaterfallExecutor.sol b/src/WaterfallExecutor.sol
--- a/src/WaterfallExecutor.sol
+++ b/src/WaterfallExecutor.sol
@@ -152,8 +152,32 @@ contract WaterfallExecutor is
    /// @inheritdoc IWaterfallExecutor
    /// @dev Reconfiguring accounts for an existing waterfallId invalidates any pending
    attestations
    /// that reference the old account configuration.
    function configureWaterfall(
        bytes32 waterfallId,
        AccountType[] calldata allowedAccounts,
        address[] calldata allowedRecipients
    ) external onlyAdmin {
+       _configureWaterfall(waterfallId, allowedAccounts, allowedRecipients, new address[](0)
    );
+   }
+
+   /// @inheritdoc IWaterfallExecutor
+   function configureWaterfall(
+       bytes32 waterfallId,
```

... continued

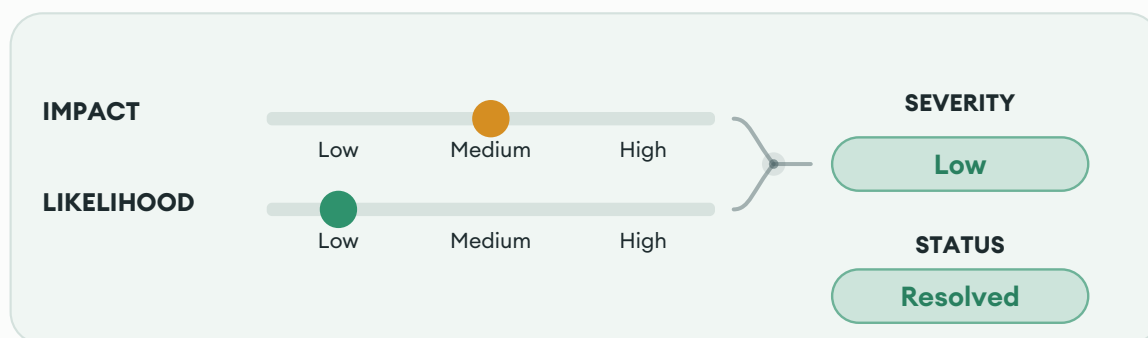
DIFF

```
+     AccountType[] calldata allowedAccounts,
+     address[] calldata allowedRecipients,
+     address[] calldata removedRecipients
+ ) external onlyAdmin {
+     _configureWaterfall(waterfallId, allowedAccounts, allowedRecipients,
+ removedRecipients);
+ }
+
+ function _configureWaterfall(
+     bytes32 waterfallId,
+     AccountType[] memory allowedAccounts,
+     address[] memory allowedRecipients,
+     address[] memory removedRecipients
+ ) internal {
+     if (waterfallId == bytes32(0)) revert ZeroWaterfallId();
+     if (allowedAccounts.length == 0) revert EmptyAccounts();
+
+     uint8 bitmap;
+     for (uint256 i; i < allowedAccounts.length; ++i) {
@@ -167,8 +191,18 @@ contract WaterfallExecutor is
+         _allowedAccountsBitmap[waterfallId] = bitmap;
+         _validWaterfallIds[waterfallId] = true;
+
+     for (uint256 i; i < removedRecipients.length; ++i) {
+         address recipient = removedRecipients[i];
+         if (_recipientWhitelist.isInScope(waterfallId, recipient)) {
+             _recipientWhitelist.removeFromScope(waterfallId, recipient);
+         }
+     }
+
+     for (uint256 i; i < allowedRecipients.length; ++i) {
-         _recipientWhitelist.addToScope(waterfallId, allowedRecipients[i]);
+         address recipient = allowedRecipients[i];
+         if (!_recipientWhitelist.isInScope(waterfallId, recipient)) {
+             _recipientWhitelist.addToScope(waterfallId, recipient);
+         }
+     }
+
+     emit WaterfallConfigured(waterfallId, allowedAccounts);
```

FIX REVIEW NOTES

resetRecipientAllowlist was introduced as an argument that increases the current scope nonce before adding addresses to the scope.

L02: Unrestricted holder growth can render `distribute()` uncallable



LOCATION

[pact-evm-contracts/4f58fd3a/src/EnumeratingShareClass.sol:L367-L377](#) 

RELEVANT SNIPPET

[illegible]

DESCRIPTION

When `identityRegistry_` is `address(0)` (permissionless mode, an intentionally supported feature per the NatSpec: `address(0) = no checks`), any share holder can transfer 1 wei to an arbitrary number of addresses.

_update() adds every non-zero-balance recipient to the **_holders EnumerableSet**.

distribute() linearly iterates the entire set with storage reads and external **_tryTransfer** calls per holder.

Each transfer adds a new entry to `_holders`. Since there is no cap on the set size and no mechanism to remove holders with non-zero balances, inflating the set past the block gas limit DoSes all `distribute()` calls.

The attack cost is negligible (only gas for 1-wei transfers), and the effect is not recoverable through current functionality.

PRACTICAL IMPACT

If `_holders` grows too large, intentionally or unintentionally, `distribute()` will become unusable (iterating will require too much gas for the tx to succeed) and there's no backup mechanism (other than manual intervention by upgrading the contract and adding new functionality).

RECOMMENDATION

The permissionless transfer mode is a valid feature. The issue is that `_holders` iteration in `distribute()` is not safe under that mode. Three interconnected options which can avoid this issue:

- Cap the holder set. Add `if (_holders.length() >= MAX_HOLDERS) revert HolderLimitReached();` in `_update()` before `_holders.add(to)`.
- Add an only-owner batched distribution function. A `manualDistribute(address[] calldata recipients, uint256[] calldata amounts)` bypasses the full `_holders` iteration and restores liveness even if the set is already bloated.
- Allow admin to remove holders. An admin-only `removeHolder(address holder)` that force-transfers or burns dust balances enables cleanup of spam entries.

FIX REVIEW NOTES

The `maxHolders` cap prevents unbounded `_holders` growth and therefore prevents the reported `distribute()` DoS condition, at the cost of making new-holder transfers revert once the configured limit is reached.

5. Enhancement Opportunities

E01: Complete the receivable custody lifecycle with an explicit outbound handling path

LOCATION

pact-evm-contracts/4f58fd3a/src/Vehicle.sol:L254-L259 [↗](#)

RELEVANT SNIPPET

>_ Vehicle.sol		SOLIDITY
252:	// IVehicle: Receivable Pledging	
253:		
254:	/// @inheritdoc IVehicle	
255:	function pledgeReceivable(address collection, uint256 receivableId) external nonReentrant {	
256:	if (!_canPledge(msg.sender)) revert NotAuthorizedToPledge(msg.sender);	
257:	IERC721(collection).safeTransferFrom(msg.sender, address(this), receivableId);	
258:	emit ReceivablePledged(collection, receivableId);	
259:	}	
260:		
261:	/// @dev Returns true if `caller` is authorized to pledge receivables. Default: admin only.	

DESCRIPTION

The vehicle can receive receivable NFTs, including unsolicited direct transfers, but the current codebase does not appear to expose a corresponding release, substitution, liquidation, foreclosure, or admin recovery path. Even if unsolicited NFTs are assumed to be economically inert, this leaves the contract in an ambiguous custody state and makes the receivable workflow appear incomplete.

POTENTIAL BENEFIT

Improves operational clarity, avoids stranded receivables in vehicle custody, and makes it easier to distinguish intentional pledged assets from incidental NFT transfers.

RECOMMENDATION

Consider adding an explicit outbound receivable-handling flow, together with explicit tracking for protocol-recognized pledges if only certain inbound transfers should be treated as meaningful.

FIX REVIEW NOTES

releaseReceivable was introduced.

E02: Add **address(0)** validation

LOCATION

pact-evm-contracts/4f58fd3a/src/ScopedWhitelist.sol:L40-L44 [↗](#)

RELEVANT SNIPPET

>_ ScopedWhitelist.sol	SOLIDITY
<pre>38: 39: /// @inheritdoc IScopedWhitelist 40: function addToScope(bytes32 scope, address addr) external onlyOwner { 41: if (_whitelist[scope][addr]) revert AlreadyInScope(); 42: _whitelist[scope][addr] = true; 43: emit AddedToScope(scope, addr); 44: } 45: 46: /// @inheritdoc IScopedWhitelist</pre>	

DESCRIPTION

Three entry points accept recipient addresses without **address(0)** validation:

- **ScopedWhitelist.addToScope(scope, addr)**: no check on **addr**
- **WaterfallExecutor.addRecipient(waterfallId, recipient)**: no check on **recipient**
- **WaterfallExecutor.configureWaterfall(waterfallId, _, allowedRecipients)**: no check on array elements

POTENTIAL BENEFIT

Prevent accidental admin errors.

RECOMMENDATION

Apply the **address(0)** guard.

FIX REVIEW NOTES

The check was added in **ScopedWhitelist.addToScope**.

E04: Document or enforce the expected vehicle / facility deployment invariant in `FundingManager.initialize()`

LOCATION

pact-evm-contracts/4f58fd3a/src/FundingManager.sol:L62-L74 [↗](#)

RELEVANT SNIPPET

```
>_ FundingManager.sol SOLIDITY
60:  /// @param p The initialization parameters.
61:  function initialize(InitParams calldata p) external initializer {
62:      _initializeCCM(p.vehicle);
63:
64:      if (p.facility == address(0)) revert ZeroFacility();
65:      if (p.originatorRegistry == address(0)) revert IVehicle.ZeroAddress("originatorReg
istry");
66:      if (p.originator == address(0)) revert IVehicle.ZeroAddress("originator");
67:      if (p.capitalCallPeriodStart >= p.capitalCallPeriodEnd) {
68:          revert InvalidCapitalCallPeriod(p.capitalCallPeriodStart, p.capitalCallPeriodE
nd);
69:      }
70:      if (p.capitalCallPeriodEnd > p.recyclePeriodEnd) {
71:          revert InvalidRecyclePeriod(p.capitalCallPeriodEnd, p.recyclePeriodEnd);
72:      }
73:
74:      _facility = p.facility;
75:      _originatorRegistry = IAccessControl(p.originatorRegistry);
76:      _originator = p.originator;
```

DESCRIPTION

`FundingManager.initialize()` accepts both vehicle and facility addresses independently. In the current facility architecture, the deployed **Facility** contract appears to serve both roles, and the tests initialize **FundingManager** with the same address for each. Explicitly documenting or enforcing this invariant would reduce deployment ambiguity and make the intended integration model clearer.

POTENTIAL BENEFIT

Reduces configuration ambiguity and helps prevent unsupported deployments.

RECOMMENDATION

If **FundingManager** is only intended to operate against a **Facility** that also serves as its vehicle, consider either: documenting that vehicle and facility must be the same deployed proxy, or enforcing that invariant during initialization

FIX REVIEW NOTES

facility is no longer an argument for `initialize()` and **FundingManager** now only uses **vehicle**.

E05: Consider enforcing freshness against payload evaluation timestamps, not only attestation submission time

LOCATION

pact-evm-contracts/4f58fd3a/src/Vehicle.sol:L314-L320 [🔗](#)

RELEVANT SNIPPET

```
>_ Vehicle.sol SOLIDITY
312: function _covenantVariant() internal pure virtual returns (bytes4);
313:
314: /// @dev Reverts if the attestation is stale or from the future. Skips if staleness ==
315:     0 (disabled).
316: function _validateAttestationFreshness(IAttestationRegistry.Attestation memory att) in
317:     ternal view {
318:     if (att.submittedAt > block.timestamp) revert AttestationFromFuture();
319:     if (_attestationStaleness != 0 && block.timestamp - att.submittedAt > _attestation
320:         Staleness) {
321:         revert AttestationStale();
322:     }
323: }
324:
325: // ERC-721 Receiver
```

DESCRIPTION

Freshness checks for covenant and waterfall attestations are currently enforced against the registry-level **submittedAt** timestamp rather than the timestamp inside the signed payload that represents when the underlying analysis was actually performed.

For example, covenant payloads already include an **evaluatedAt** field, but **Vehicle** only validates freshness using the attestation's **submittedAt**. Similarly, **WaterfallExecutor** checks **submittedAt** rather than the plan's own evaluation timestamp.

This means the on-chain contracts currently assume that attestations are submitted promptly after off-chain computation, and that **submittedAt** is therefore a reliable proxy for analysis freshness. If that operational assumption ever weakens, stale off-chain analyses could still appear fresh on-chain.

POTENTIAL BENEFIT

Using payload-level **evaluatedAt** for freshness checks would align on-chain enforcement with the actual age of the off-chain analysis, reducing reliance on operational timing assumptions in the attester pipeline.

RECOMMENDATION

Consider validating freshness against the signed payload's own evaluation timestamp in addition to the registry-level **submittedAt** timestamp for covenant and waterfall attestations.

FIX REVIEW NOTES

Fixed according to the recommendation.

E06: Consider additional ownership-management hardening for AttestationRegistry

LOCATION

pact-evm-contracts/4f58fd3a/src/AttestationRegistry.sol:L22 [↗](#)

RELEVANT SNIPPET

> _AttestationRegistry.sol		SOLIDITY
20:	Initializable,	
21:	UUPSUpgradeable,	
22:	OwnableUpgradeable,	
23:	PausableUpgradeable,	
24:	EIP712Upgradeable	

DESCRIPTION

AttestationRegistry is a root-governance contract whose owner controls signer management, threshold configuration, pause/unpause, and upgrades. Because this ownership is expected to be held and rotated manually, additional ownership-management safeguards may improve operational safety.

Two related hardening measures are particularly relevant here:

- adopting **Ownable2StepUpgradeable** would require the proposed new owner to explicitly accept the role before the transfer becomes effective, reducing the chance of transferring governance to the wrong or unprepared address
- disabling **renounceOwnership()** would prevent accidental transition to an ownerless state where owner-controlled governance actions and upgrades are no longer available

POTENTIAL BENEFIT

Reduces the risk of accidental governance misconfiguration during manual ownership rotation, including mistaken ownership transfer and unintended transition to an ownerless state.

RECOMMENDATION

Consider applying additional ownership-management safeguards to AttestationRegistry, such as:

- adopting **Ownable2StepUpgradeable** for manual ownership rotation
- disabling **renounceOwnership()** if ownerless operation is not an intended steady state

FIX REVIEW NOTES

Ownable2StepUpgradeable was adopted.

E07: Immutable admin registry reference across core contracts

LOCATION

pact-evm-contracts/4f58fd3a/src/Vehicle.sol:L92 [↗](#)

RELEVANT SNIPPET

>_ Vehicle.sol		SOLIDITY
90:	<code>_paymentToken = IERC20(paymentToken_);</code>	
91:	<code>_attestationRegistry = IAttestationRegistry(attestationRegistry_);</code>	
92:	<code>_adminRegistry = IAccessControl(adminRegistry_);</code>	
93:	<code>_isCovenantsCompliant = true;</code>	
94:	<code>_attestationStaleness = attestationStaleness_;</code>	

DESCRIPTION

`_adminRegistry` is set once during `initialize()` in **Vehicle**, later **CapitalCallManager** and **WaterfallExecutor** initialise this variable by referencing the **Vehicle** contract

If the admin registry contract needs to be replaced (compromise, migration to a new access control model), every dependent contract must undergo a UUPS upgrade

POTENTIAL BENEFIT

Allow hot-swapping the access control registry without forcing a full UUPS upgrade on every dependent contract, reducing operational risk and downtime.

RECOMMENDATION

Consider adding an admin-gated `setAdminRegistry()` function to allow registry migration without requiring contract upgrades.

FIX REVIEW NOTES

`setAdminRegistry` was introduced in **Vehicle**.

E08: **forceCloseCapitalCall** does not emit **CapitalClaimed** events during claim loop

LOCATION

pact-evm-contracts/4f58fd3a/src/CapitalCallManager.sol:L374-L380 [↗](#)

RELEVANT SNIPPET

> _CapitalCallManager.sol

SOLIDITY

```
372: uint256 totalClaimed;
373: address vehicleToken = IVehicle(_vehicle).paymentToken();
374: for (uint256 i; i < len; ++i) {
375:     address sc = _callShareClasses[i];
376:     uint256 fulfilledAmt = _fulfilled[sc];
377:     if (fulfilledAmt > 0) {
378:         IShareClass(sc).claimCompletedCapital(fulfilledAmt);
379:         totalClaimed += fulfilledAmt;
380:     }
381: }
382:
```

DESCRIPTION

fulfillCapitalCall() auto-completion emits a **CapitalClaimed** event for each share class during the claim loop:

SOLIDITY

```
if (fulfilledAmt > 0) {
    IShareClass(sc).claimCompletedCapital(fulfilledAmt);
    totalClaimed += fulfilledAmt;
    emit CapitalClaimed(vehicleToken, fulfilledAmt);
}
```

forceCloseCapitalCall() performs the same claim loop but omits the event.

POTENTIAL BENEFIT

Consistent event emission across both claim paths

RECOMMENDATION

Add the **CapitalClaimed** event inside **forceCloseCapitalCall** claim loop:

FIX REVIEW NOTES

Fixed according to the recommendation.

E09: `cancelCapitalCall` sets `active = false` after `_clearCallState`, inconsistent with other termination paths

LOCATION

pact-evm-contracts/4f58fd3a/src/CapitalCallManager.sol:L344-L357 [↗](#)

RELEVANT SNIPPET

```
>_ CapitalCallManager.sol SOLIDITY
342:  ///      Early close (partial-fill acceptance) is intentionally omitted to reduce att
      ck
343:  ///      surface. Only zero-fill cancellation is supported.
344:  function cancelCapitalCall() external onlyVehicleOrAdmin nonReentrant {
345:      if (!_currentCall.active) revert NoActiveCall();
346:
347:      // H-1: Only allow cancel if no share class has been partially filled.
348:      uint256 len = _callShareClasses.length;
349:      for (uint256 i; i < len; ++i) {
350:          if (_fulfilled[_callShareClasses[i]] > 0) revert CannotCancelPartiallyFilled()
      ;
351:      }
352:
353:      _clearCallState();
354:      _currentCall.active = false;
355:
356:      emit CapitalCallCancelled(block.number);
357:  }
358:
359:  /// @inheritdoc ICapitalCallManager
```

DESCRIPTION

`cancelCapitalCall()` calls `_clearCallState()` while `_currentCall.active` is still `true`, then sets `active = false`:

```
_clearCallState();
_currentCall.active = false;
```

However, both `forceCloseCapitalCall()` and `fulfillCapitalCall()` auto-completion set `active = false` before `_clearCallState()`:

POTENTIAL BENEFIT

Consistent ordering across all three termination paths

RECOMMENDATION

Set `active = false` before `_clearCallState()` in `cancelCapitalCall`, matching the other two paths:

```
_currentCall.active = false;
_clearCallState();
```

FIX REVIEW NOTES

Fixed according to the recommendation.

About Us

About Adevar Labs

Adevar Labs is a boutique blockchain security firm specializing in web3 audits.

Built by a mix of experienced professionals in traditional enterprise 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, and key roles at Fortune 500 companies, and leadership of flagship conferences like ETH Bucharest.

With team members having publications with over 1,100 academic citations and trusted by 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 specialized 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.

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.

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, expressed 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 provided.

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.