SECURITY ASSESSMENT

for

Trustline

on

Jun 01, 2026



FIDESIUM

Table of Contents

Executive Summary **3**
Root Causes 6
Risk Issues **8**
Governance & Trust Model **11**
Remediation Roadmap **13**
Vulnerabilities **15**
Critical 15
High 16
Medium 21
Low 25
Info 27
Integrator-Surface & Distribution 29
Appendix **45**
Appendix A: Architecture Diagram 45
Appendix B: Authority Structure 46
Appendix C: Public Entry Points 47
Appendix D: Storage Layout 49
Appendix E: Adversarial Scenarios 50
Appendix F: Differential Review vs Upstream 52
Appendix G: Cross-Configuration Differential Testing 56
Appendix H: Positive Observations 57
Appendix I: Configuration State Machine 58
Appendix J: Threat Model - Maintainer 59
Appendix K: Threat Model - Integrator 60
Appendix L: Finding × Surface Matrix 62
Disclaimer **63**

Executive Summary

Report

23
/100
TOTAL
Low risk
Jun 26, 2026

48
/100
TOTAL
Medium risk
Jun 01, 2026

Abstract

Trustline engaged Fidesium to assess the `EVMSDK` and `validation-engine-sanctions-only` programs. This assessment was conducted in May 2026.

Program Links:

1. github.com/TrustLine-id/evmsdk

   Commit Hash: `967387946156c99b3ae344e23393dfda2077d9af`. Corresponds to npm version `1.0.6`

   Follow up review: `9f180bf8e914fe5831ca3047c310837234f652c0`. No npm version published.

   Files in Scope:

   - Trustlined.sol
   - IValidationEngine.sol
   - IValidationEngineInitializer.sol

2. github.com/TrustLine-id/validation-engine-sanctions-only

   Commit Hash: `127f10357b57fe574b0f52c74b61ac5fee66757a`

   Follow up review: `51d99518adea872c12670c39f7df9bcb12ecaa74`

Key Findings

Scope of assurance. This audit assures the *mechanism* around the sanctions oracle. It does **not**, and cannot, establish that Trustlined means regulatory compliant. A boolean address-membership oracle cannot express the legal obligation (OFAC 50%-ownership, entity/identity resolution, jurisdiction/list selection, designation freshness), The SDK attests address-list screening. The integrator's obligation is strictly broader. See finding **I10**.

Primary risk areas: The proxy's `DEFAULT_ADMIN_ROLE` is a single point of failure for the compliance guarantee. A malicious upgrade can mutate state on every `requireTrustline` call because the interface declares it as non-view (C1, H6).

Significant risk: The pinned `Trustlined.sol` validates a passed proxy/logic only with `code.length > 0` (H2) and uses string-based selector encoding for `initialize` (H1). A reverting or gas-bombing sanctions oracle DoSs every integrator entry point (H3). An attacker who front-runs an uninitialized adopt-path proxy seizes admin (H4).

Integrator-borne risk: because the SDK is non-custodial, the bulk of real-world risk lands on the *integrator*. Fidesium identified: a fail-open default that ships the gate disabled (H5, High), a silently-ignorable check return (M5), the compliance gate doubling as a reentrancy vector (H6), a recipient/counterparty screening blind spot (M3b), an immutable VE binding with no rotation path (C1), ignored value/calldata (M6b), and an account-abstraction blind spot (M3c).

Bugs surfaced by automated methods:

- **Slither:** `calls-loop` in the per-address sanctions screening confirms M4; `reentrancy-no-eth` on the proxy deployment is benign (I6).
- **Fidesium App:** run against both code bases. The analyzer corroborated centralization (C1), front-runnable initialization (H4), the unbounded sanctions loop (M4), oracle dependence (H3), single-step/no-event reconfiguration, and dormant state (I1).
- **Foundry PoCs:** 88 tests
- **Differential review:** H1 is fixed at EVMSDK HEAD (`596c425`); H2 is partially mitigated (`e6e7147`) but a fallback-truthy proxy still bypasses (see Appendix F).

Scope

Source Files	5 (+ 1 informational deploy script)
Lines of code (in-scope)	315 total / ~194 code
External entry points	22 (proxy) + 1 (Trustlined)
State Variables (VE)	7 declared, 2 live

Codebase Profile

`evmsdk/contracts/`	~96 lines
`validation-engine.../validation/`	~109 lines
`validation-engine.../implementations/`	~24 lines
`script/` (informational)	~19 lines

Executive Summary

Issue Summary

Critical	High	Medium	Low	Info
~~1~~ 0 Issues	~~6~~ 1 Issues	~~16~~ 5 Issues	~~5~~ 1 Issues	~~12~~ 8 Issues

Engagement

Report Version	1.1 — Integrator-surface findings
Engagement Period	May 25 – Jun 01, 2026
Delivery Date	May 30, 2026

Team

Lead Security Researcher	Abraham Polishchuk @skilurus

Root Causes

The 28 Critical/High/Medium findings are facets of **six root causes**.

Root cause	Findings	Recommendation
Unbounded admin & upgrade authority	C1, H6, M1, M2, I12, M16	Hold `DEFAULT_ADMIN_ROLE` behind multisig + timelock; add a last-admin floor, upgrade target validation, config events, and a fail-closed freeze.
Weak / spoofable engine binding	H1, H2, H4, M13, M14	Type-checked init encoding; verify the bound engine by codehash/registry (not `code.length`); atomic init; a rotation path; reject 7702/delegatecall-host binds.
Oracle is an unconstrained dependency	H3, M4, M7, M8, M10	Constrain `ISanctionsList`: conformance-check at set time, freshness/heartbeat with fail-closed staleness, bounded gas, typed fail modes; treat output as trusted-but-verified.
Incomplete screening scope	H5, M3, M5, M6, M9	Ship enabled by default; enforce via `requireTrustline`; screen the full beneficiary set (recipients, AA owners, inner/batch actors); re-screen at settlement.
Operation surveils users	M12, M15	Send the oracle only what it needs (not full `msg.data`); avoid shared engines for privacy-sensitive flows; document the operator's vantage.
Shipped artifact ≠ audited source	M11	Pin toolchain + deps with a committed lockfile and npm provenance; verify on-chain bytecode against the audited tuple.

Methodology

- Manual source code review
- Threat modeling (actors, assets, trust assumptions, entry points)
- System design analysis
- Protocol-interaction & composition analysis
- Static analysis
- Storage-layout inspection
- Configuration state-machine review
- Selector-collision audit (proxy surface)
- Distribution & supply-chain review
- Cross-config differential testing across the audited (solc 0.8.34 / osaka), the cancun, and the shipped-SDK (0.8.28 / cancun) configurations. Byte-identical gas for every gas-grief finding (see Appendix G).
- Adversarial role analysis
- Integrator-side risk assessment
- Integrator-centric (Risk-Bearer) severity model
- Differential review vs upstream HEAD

Severity Definitions

Critical	The issue can cause large economic losses, large-scale data disorder or loss of control of authority management.
High	The issue puts users' sensitive information at risk or is likely to lead to catastrophic financial implications.
Medium	The issue puts a subset of users' sensitive information at risk, reputation damage or moderate financial impact.
Low	The risk is relatively small and could not be exploited on a recurring basis, or is low-impact to the client's business.
Informational	The issue does not pose an immediate risk but is relevant to security best practices or defense in depth.

Integrator-centric scoring. The in-scope contracts are non-custodial; impact is therefore measured on the party that *bears* it.

Risk Issues

Active New Acknowledged Remediated Resolved

Vulnerability	Description	Risk	Probability	Status
C1: Centralization: Unconstrained Admin	One unconstrained `DEFAULT_ADMIN_ROLE` can renounce and permanently brick the engine, (b) defeat all screening, seize admin.	Critical	Low	Remediated
H1: Selector-String `initialize` Encoding	`abi.encodeWithSignature("initialize(address)", initialOwner)` matches selector blindly.	High	Medium	Resolved
H2: Weak Proxy/Logic Validation	An integrator can bind to any contract.	High	Medium	Remediated
H3: Sanctions-Oracle DoS Bricks Every Integrator	`SanctionsOnlyValidationEngine._isSanctionsApproved` calls `sanctionsOracle.isSanctioned(...)` inside a loop with no `try/catch` and no gas budget.	High	Medium	Remediated
H4: Adopt-Uninitialized-Proxy Hijack	If Trustline (or anyone) ever deploys an `ERC1967Proxy(logic, "")`, an attacker can race them.	High	Low	Remediated
H5: Fail-Open Default (Integrator-surface)	A fresh VE ships with `sanctionsOracleEnabled = false` and screening fails **open**.	High	High	Resolved
H6: Non-View `requireTrustline` Interface	`IValidationEngine.requireTrustline` is declared non-view, so integrators emit `CALL` not `STATICCALL`.	High	Low	Active
M1: Integrator-Deployer Becomes Admin	The address that calls the integrator's constructor (`msg.sender`) silently becomes `DEFAULT_ADMIN_ROLE` and `AUDITOR_ROLE`.	Medium	Medium	Active
M2: `setValidationConfiguration` Emits No Event	Admin can disable sanctions or swap the oracle without producing any on-chain log.	Medium	High	Resolved
M3: End-User Bypass via Clean Intermediary	The SDK only screens `msg.sender` + a caller-supplied `addresses[]`.	Medium	High	Remediated
M4: Unbounded `addresses[]` Gas-Amplification DoS	~3.8K gas per array element (linear loop over `sanctionsOracle`). A user-controlled array lets a caller drive gas costs to the block limit, making honest use of the integrator uneconomical.	Medium	Medium	Resolved
M5: Ignored `checkTrustlineStatus` Return	The boolean query enforces nothing; discarding its result.	Medium	Medium	Acknowledged
M6: `ValidationMode` Accepted but Discarded	The interface markets 4 modes (incl. ERC3643); all are ignored.	Medium	Medium	Acknowledged
M7: Oracle Never Conformance-Checked	`setValidationConfiguration` only null-checks the oracle.	Medium	Medium	Resolved
M8: Oracle Selective Censorship	A compromised/coerced oracle returns true for chosen victims, denying availability while looking healthy.	Medium	Low	Acknowledged

Risk Issues

Vulnerability	Description	Risk	Probability	Status
M9: Screen/Settle TOCTOU	Deferred-settlement flows screen at submission, settle later with no re-check; an actor sanctioned in between is still paid.	Medium	Medium	Active
M10: No Oracle Freshness	`isSanctioned` returns a bare `bool` with no timestamp/heartbeat.	Medium	Medium	Active
M11: Supply Chain: Unpinned, Unverifiable Build & Distribution	The build is unpinned and unverifiable. Shipped artifact ≠ audited.	Medium	High	Remediated
M12: Calldata Surveillance via Shared Proxy	Full `msg.data` is forwarded to the VE on every guarded call; under the shared-proxy model a single operator harvests every integrator's complete calldata, resulting in a transaction-graph / deanonymization dataset.	Medium	Medium	Active
M13: EIP-7702 Engine-Identity Defeat	A 7702-delegated EOA (live on osaka) passes code.length/conformance/code-read at bind, then the controller re-delegates to permissive code via a type-4 tx, silently flipping the bound engine with no Upgraded/ERC-1967 event.	Medium	Medium	Remediated
M14: Diamond / Delegatecall Slot-0 Redirection	Used as a Diamond facet / delegatecall target, `Trustlined.validationEngine` collides with the host's slot 0. Extends the Appendix F wrong-base-class advisory.	Medium	Low	Acknowledged
M15: Oracle-Operator Vantage / Intent MEV	The oracle receives the screened address on every guarded call.	Medium	Medium	Active
M16: Privileged Actor Sanctioned / Seized	Admin or oracle operator is themselves designated/indicted/key-seized leads to frozen governance, possibly no compliant operator.	Medium	Low	Acknowledged
L1: Initial Owner Holds Dormant `AUDITOR_ROLE`	No in-scope function consumes `AUDITOR_ROLE`, yet it is granted to the initial owner at init.	Low	Medium	Active
L2: Floating Pragma `^0.8`	SDK does not pin a compiler version.	Low	Low	Remediated
L3: Disable-Path Ignores Oracle Address Parameter	`setValidationConfiguration(false, someAddress)` accepts the call but blanks `sanctionsOracle` to `address(0)`.	Low	Low	Resolved
L4: Cross-Chain Canonical-Address Verification Fragility	CREATE-based deploy makes the canonical logic address nonce-dependent (not cross-chain-deterministic)	Low	Medium	Remediated
L5: Out-of-Range ValidationMode Hard-Reverts	A `ValidationMode` ≥ 4 reverts at ABI-decode (empty returndata, not Panic 0x21).	Low	Low	Acknowledged
I1: Dormant State Variables in VE	`validationRegistry`, `trustlineOracleEnabled`, `AUTO_APPROVED_VALIDITY_PERIOD`, `MANUAL_APPROVED_VALIDITY_PERIOD`, `_txMetadata` are declared and set but never read in the sanctions-only flow.	Info	Info	Active
I2: Hardcoded Dormant `VALIDATION_REGISTRY`	`address private constant VALIDATION_REGISTRY = 0x34e3fe42...1dd` is set at init but never read by any in-scope function.	Info	Info	Active
I3: Three-Layer Init Pattern in Non-Upgradeable Abstract	`Trustlined` (pinned) uses constructor → `__Trustlined_init` → `__Trustlined_init_unchained`. The OZ-style pattern is unnecessary for the non-upgradeable contract. **Flattened at HEAD**.	Info	Info	Active

Risk Issues

Vulnerability	Description	Risk	Probability	Status
I4: `_authorizeUpgrade` Body Empty; `newImpl` Unvalidated	Admin can upgrade to `address(0)` or to any contract.	Info	Info	Remediated
I5: Permissionless Sanctions Enumeration	Unauthenticated, arbitrary-subject `checkTrustlineStatus` lets anyone read any address's sanctioned status for free and enumerate the list / probe oracle availability.	Info	High	Active
I6: Untyped Revert Reasons	Sanctioned-sender, sanctioned-counterparty, and oracle-failure are not distinguishable on-chain (plain string / propagated reverts).	Info	Medium	Active
I7: Unauthenticated Event Emission	Events are emitter-unauthenticated; anyone emits `ValidationEngineDeployed` with the canonical topic + arbitrary args. Indexers/dashboards keyed on topic (not `log.address`) can be shown fabricated/sanctioned integrators as "Trustline-protected.	Info	Medium	Active
I8: Compliance-Model Adequacy	A boolean address-membership oracle cannot express the legal obligation (OFAC 50% rule, identity/entity resolution, jurisdiction/list, designation freshness).	Info	High	Active
I9: On-Chain Compliance Record (Privacy/GDPR)	Routing every screen on-chain creates a permanent public record of compliance decisions about users (GDPR right-to-erasure tension).	Info	Medium	Active
I10: Adopt Path Emits No `ValidationEngineDeployed` Event	The adopt-existing branch emits no event, so off-chain indexers undercount adopt-path integrators.	Info	High	Resolved

Governance & Trust Model

Trust Model Overview

The system's compliance guarantee is that `requireTrustline` reverts when any of `{sender}Uaddresses[]` is sanctioned. This depends on the following trust relationships:

1. **The address holding** `DEFAULT_ADMIN_ROLE` on each deployed VE proxy. *(Not enforced:**C1, H4, M1, M2, M7, M16**)*
2. **The sanctions oracle** at `sanctionsOracle` being correctly maintained, live, and honest. *(Not enforced: **H3, M7, M8, M10, M15**)*
3. **The** `TrustlineSanctionsOnlyVE` **logic address** being verifiable by integrators before they bind to it, and remaining what was verified. *(Not enforced: **H1, H2, C1, H6, M13, M14, M3, L4, M11**)*
4. **The oracle's address-boolean model** adequately expressing the legal sanctions obligation. *(Cannot be enforced: **I10**)*

Admin Control Surface

The `DEFAULT_ADMIN_ROLE` holder on a VE proxy can:

Function	What it controls	Blast radius
`setValidationConfiguration(bool, address)`	Toggle sanctions screening + replace oracle pointer	Atomic disable of all sanctions enforcement for every integrator bound to this proxy. No event emitted (M2). Atomic oracle swap to any contract.
`upgradeToAndCall(address, bytes)`	Replace VE logic via UUPS	Complete behavioral override. Permissive replacement defeats screening (C1b). `requireTrustline` harvests end-user gas to silently grant admin (C1c).
`grantRole / revokeRole`	Manage `DEFAULT_ADMIN_ROLE` and `AUDITOR_ROLE`	Admin can promote a co-conspirator or revoke other admins. `RoleGranted/RoleRevoked` events fire.
`renounceRole`	Self-removal of role	If the last admin renounces, the proxy is permanently bricked (C1).

No Bounds on Admin Actions

The contracts do not enforce any of:

- A timelock between proposing and executing a configuration change or upgrade (**C1**).
- A two-step admin transfer (propose + accept) (no standalone finding; covered by **C1**'s remedy).
- A floor on the number of admins (last-admin protection) (**C1**).
- Codehash or registry validation on the upgrade target (**I4**, weaponized by **C1**).
- Bounds on the oracle address (**M7, L3**).

Every admin action is single-signature, single-block, and irrevocable in its effects. Combined with the silent `setValidationConfiguration` path (M2).

How Admin Is Assigned

When the integrator's constructor runs `Trustlined(logic, address(0))`, the SDK auto-deploys a fresh VE proxy and captures `msg.sender` as `initialOwner`. That address receives both `DEFAULT_ADMIN_ROLE` and `AUDITOR_ROLE`.

The only signal is the `RoleGranted` events from `AccessControl`. If the integrator is deployed via a factory contract (CREATE/CREATE2 deployer, account-abstraction bundler, proxy admin), the factory becomes admin.(M1).

Shared-Proxy Risk

The documentation invites multiple integrators to share a single VE proxy ("Multiple dapps can share the same ValidationEngine contract"). A compromise of that shared proxy's admin compromises every integrator using it.

Governance & Trust Model

Maximum Damage Chain

A compromised admin can complete a permanent takeover in three transactions, all within a single block:

1. **Tx 1:** Deploy the malicious logic, approving everyone or allow granting admin to an attacker. **C1b/C1c**
2. **Tx 2:** `upgradeToAndCall(newImpl, "")`. Proxy now serves permissive answers, or harvests admin to the attacker on the next honest call.
3. **Tx 3:** `renounceRole(DEFAULT_ADMIN_ROLE, self)`, preventing recovery

The only on-chain trace is one `Upgraded(address)` event and one `RoleRevoked` event.

Integrator User Trust Assumptions

End users of an integrator dApp implicitly trust that:

1. The VE proxy's `DEFAULT_ADMIN_ROLE` is held by a properly governed multisig with a timelock.
2. The integrator developer correctly chose between deploy-fresh (auto-becomes admin) and adopt-existing.
3. If adopting, the proxy address came from a verified Trustline-published source and was already initialized.
4. The sanctions oracle is honest and operational.
5. The integrator passes the right actor set to `requireTrustline` (e.g. M3: `tx.origin` for end-user screening when relay patterns are in scope).

None of these assumptions are verifiable on-chain by the end user. C1 (timelock + last-admin protection), M2 (configuration events).

Remediation Roadmap

Priority	Finding	Action	Rationale	Effort
Immediate	C1: Centralization (brick + permissive upgrade + state-leech)	Adopt OZ's `AccessControlDefaultAdminRulesUpgradeable` (last-admin floor + two-step transfer); hold admin behind a multisig + timelock; bound `_authorizeUpgrade` to a codehash allowlist / registry.	Single unconstrained admin is the single point of failure; the floor + custody is what drops this from Critical to High.	Small–Medium: swap inherited contract + governance setup
Immediate	M2: Silent Config (observability)	Emit `ValidationConfigurationChanged(bool, address)` from `setValidationConfiguration`.	Observability is the only line of defense against admin compromise (supports C1).	Small: single event
Short-term	H6: Non-View Interface	Make `requireTrustline view` in the interface, forcing all integrators to emit `STATICCALL`.	Eliminates the entire class of upgrade-via-call attacks (C1c) against existing integrators.	Medium: breaking interface change.
Short-term	H1: Selector-String Encoding	Already fixed upstream (`abi.encodeCall(IValidationEngineInitializer.initialize, (initialOwner))` at `596c425`).	The code fix exists; residual risk is *adoption*.	Small (publish + advisory); integrator: bump + redeploy
Short-term	H2: Weak Proxy Validation	Drive integrators to upgrade `@trustline.id/evmsdk` to a release with HEAD's `_assertValidationEngine`. Also verify an expected codehash or query a Trustline-maintained registry of canonical proxy addresses.	HEAD fix catches reverting fakes but not fallback-truthy contracts. Registry/codehash closes the residual gap.	Small (publish + integrator upgrade). Moderate (registry/codehash)
Short-term	H3: Oracle DoS	Wrap `sanctionsOracle.isSanctioned(...)` in `try/catch`.	Preserves compliance integrity while preventing total bricking.	Small
Short-term	H4: Adopt-Path Hijack	Never deploy a VE proxy with empty init bytes (operational).	Operational discipline + an optional in-code positive check.	Small
Short-term	Integrator usage contract: H5, M3, M5, M6, L5	Ship the gate **enabled by default** (fail-closed), not disabled (H5). Make integrators enforce via `requireTrustline`, never the discardable bool (M5); screen the **full** address set(recipients, smart-account owners, inner batch beneficiaries, counterfactual addresses (M3)); provide a guarded VE-rotation path in the base contract (C1); document that mode/value/calldata are forwarded but ignored (M6, L5) and that injection can grief screened sets (M3f).	Risk lands on the integrator. Today the defaults and usage patterns fail open or silently no-op.	Medium: defaults + docs + a rotation setter; SDK API changes.
Medium-term	M1: Factory Footgun	Accept an explicit `initialOwner` parameter in `Trustlined`'s constructor (with `msg.sender` as default).	Removes the factory footgun.	Small: constructor signature change
Medium-term	I12: Adopt-Path Event	Emit `ValidationEngineAdopted(address indexed client, address indexed engineProxy)`.	Closes the indexer visibility gap.	Small
Medium-term	M4: Unbounded Array	Cap loop length in the SDK.	Defends against per-call gas-amplification DoS.	Small

Remediation Roadmap

Priority	Finding	Action	Rationale	Effort
Medium-term	M3: End-User Bypass	Recommend integrators pass `tx.origin` in the address array.	Architectural; cannot be fixed in scope.	Small (docs)
Medium-term	Oracle trust model: H3, M7, M8, M9, M10, M15	Constrain the `ISanctionsList` contract: conformance-check the oracle at set time (M7); add a freshness/heartbeat and enforce max-staleness **fail-closed** (M10); adopt typed fail modes (H3). Treat the oracle as trusted-but-verifiable against censorship / lies-clean / empty-coverage (M8) and the operator-vantage exposure (M15). Re-screen at settlement for deferred/cross-domain flows (M9).	Oracle is a fully-trusted external dependency	Medium: `ISanctionsList` + engine changes.
Medium-term	Supply chain. M11, M13, M14, L4	Pin `solc` + `evm_version` + OZ/SDK deps exactly with a committed lockfile and npm `--provenance`; verify on-chain bytecode against the audited tuple (M11). Add base-class / slot guards (M13, M14). Treat upgrades as storage-layout-checked (I1); document cross-chain address non-determinism (L4).	The shipped artifact is not provably the audited one, and binding/identity assumptions break under delegatecall / 7702 / CREATE	Small - Medium: build/release hygiene
Low	L1–L3, I1–I6	Address per individual writeups. Prioritize L2 (pragma pinning) and I1/I2 (dead state) early to keep the storage layout intentional.	Defensive hardening.	Variable.
Document	Compliance-model & governance limits: I10, M9, M16, I7, I8, I9, I11	State plainly that "Trustlined ⇒ compliant" is **unsatisfiable** by an address-boolean oracle (I10) and that a source-domain screen binds nothing on the destination domain (M9). Add multi-party governance with compliant succession and a fail-closed targeted-freeze primitive (M16). Add typed revert reasons (I8); decide the enumeration / event / on-chain-record posture (I7, I9, I11).	These cannot be fixed by code alone.	Doc + one governance primitive (freeze).

Vulnerabilities Critical

C1: Centralization: Unconstrained Admin Bricks and Defeats the Engine

Vulnerability severity:	Critical	Vulnerability probability:	Low	Status:	Remediated

Remediated at commit `c3363f5` by switching to OpenZeppelin's `AccessControlDefaultAdminRulesUpgradeable`, enforcing a two-phase, 3-day-timelocked default-admin transfer. The timelock governs only the admin *handover*.

! Fix: Ensure `_authorizeUpgrade`/`upgradeToAndCall`, `setValidationConfiguration`, and other privileged mutators are also subject to timelock. Hold `DEFAULT_ADMIN_ROLE` behind a `TimelockController` and/or bound `_authorizeUpgrade` to a codehash/registry allowlist. Close the state-leech leg (C1-c) by declaring `requireTrustline view` in the interface (H6) — until then a permissive upgrade can still mutate state through every integrator's `CALL`.

`SanctionsOnlyValidationEngine` grants an unconstrained `DEFAULT_ADMIN_ROLE` the power to brick, reconfigure, and arbitrarily replace the engine, with no timelock, no last-admin floor, and no codehash/registry bound on upgrades.

The unconstrained role manifests as three distinct symptoms:

- **(a) Renounce-last-admin brick (liveness).** `renounceRole(DEFAULT_ADMIN_ROLE, self)` from the only admin freezes oracle config and upgrades permanently. `AccessControlUpgradeable` is inherited with no `renounceRole` override, no `AccessControlDefaultAdminRules`, and no last-admin floor. If `sanctionsOracleEnabled = true` and the oracle later misbehaves (H3), every integrator on the proxy is permanently DoS'd. *Trigger:* a single honest mistake *or* hostile call. *Recovery:* none (full redeploy + migration).
- **(b) Permissive-upgrade screening defeat (integrity).** `TrustlineSanctionsOnlyVE._authorizeUpgrade(address)` has an empty body gated only by `onlyRole(DEFAULT_ADMIN_ROLE)`. The admin deploys a `PermissiveVE` that returns `true` for every check and calls `upgradeToAndCall(newImpl, "")`; the only trace is one `Upgraded(address)` event. *Trigger:* key compromise or insider malice. *Recovery:* a counter-upgrade. Can be made immortal through (a).
- **(c) State-leech / admin-seizure via upgraded `requireTrustline` (integrity).** Because integrator callsites emit `CALL` rather than `STATICCALL` (H6), a malicious upgrade can make `requireTrustline` mutate state on every invocation, granting `DEFAULT_ADMIN_ROLE` to an attacker.*Trigger:* key compromise or insider malice. *Recovery:* as (b).

Symptoms (b) and (c) can be chained and then frozen by (a) in three transactions within a single block (worst-case G.1/G.4 in the threat model).

Integrators cannot escape a rogue engine. The SDK's `Trustlined` binds `validationEngine` exactly once (`require(validationEngine == address(0))`, no setter), so a non-upgradeable integrator bound to a compromised proxy has **no rotation path**.

```
PermissiveVE permImpl = new PermissiveVE();
vm.prank(admin);
UUPSUpgradeable(address(ve)).upgradeToAndCall(address(permImpl), "");
// sanctioned user now passes silently
vm.prank(sanctioned);
integrator.action();
```

Vulnerabilities Critical

CONTINUED: C1: Centralization: Unconstrained Admin Bricks and Defeats the Engine

Recommendations:

- **Last-admin floor + two-step transfer (closes a).** Adopt OZ's `AccessControlDefaultAdminRulesUpgradeable` (enforces a delayed two-step admin transfer and prevents renunciation while a single admin remains), or override `renounceRole` to `require(getRoleMemberCount(DEFAULT_ADMIN_ROLE) > 1, "last admin")`.
- **Multisig + timelock custody (constrains b/c).** Hold `DEFAULT_ADMIN_ROLE` behind a multisig + 24–72h timelock with a non-zero voting delay (no same-block propose+execute); avoid borrowable-vote governance tokens. Gives observability and a reaction window.
- **Bound the upgrade target (narrows b/c).** Validate `newImplementation` against a codehash allowlist or a Trustline-maintained registry in `_authorizeUpgrade`.
- **Observability.** Emit `ValidationConfigurationChanged(bool, address)` (M2) so config drift is detectable off-chain.
- **Eliminate the callsite vector (c).** Make `requireTrustline view` in the interface (H6) so integrators emit `STATICCALL` and a mutating upgrade reverts.

Vulnerabilities High

H1: Selector-String `initialize` Encoding

Vulnerability severity:	High	Vulnerability probability:	Medium	Status:	Resolved

Resolved at commit `a416c27` by replacing the string-matched `abi.encodeWithSignature("initialize(address)", initialOwner)` with the type-checked `abi.encodeCall(IValidationEngineInitializer.initialize, (initialOwner))`.

`Trustlined.sol:53` (pinned) initializes the freshly-deployed VE proxy with:

```
bytes memory data = abi.encodeWithSignature("initialize(address)", initialOwner);
address proxy_ = address(new ERC1967Proxy(logic, data));
```

The selector `0xc4d66de8` is matched by string, with no compile-time check that the logic actually exposes that signature. If the logic's `initialize` diverges (e.g., `initialize(address,address)`), the call hits the fallback (if any).

A logic contract with a permissive fallback accepts the call silently and serves `true` for every subsequent screening request.

Upstream status: Fixed at EVMSDK HEAD by commit `596c425`:

```
bytes memory data = abi.encodeCall(
    IValidationEngineInitializer.initialize, (initialOwner)
);
```

Recommendations:

Drive integrators to upgrade `@trustline.id/evmsdk` to a release with the type-checked `abi.encodeCall(IValidationEngineInitializer.initialize, ...)` encoding (`596c425`) and **redeploy**: `Trustlined` is npm-vendored and compile-time-inherited. Alternatively, re-pin the audit scope to `596c425` or later and downgrade this finding to Informational.

Vulnerabilities High

H2: Weak Proxy/Logic Validation

Vulnerability severity:	High	Vulnerability probability:	Medium	Status:	Remediated

Remediated at commit `7004cc7` (with VE `80117c1`) by adding an EIP-165 `_assertValidationEngine` conformance check plus a genuine-contract / non-7702 check, replacing the bare `code.length > 0`. EOAs and non-conformant fakes are rejected, but a fallback-truthy or conformant-but-malicious engine still passes.

! Fix: Bind the proxy/logic to a Trustline-maintained per-chain registry or an expected `EXTCODEHASH`, and probe the enforcing selector (`requireTrustline`), not only EIP-165 shape.

`Trustlined.sol:46, 50` (pinned) validates the passed proxy/logic only by code presence:

```
require(proxy.code.length > 0, "Proxy is not a contract");
// ...
require(logic.code.length > 0, "Logic is not a contract");
```

Any contract passes this check.

Upstream status: Partially mitigated at HEAD by `e6e7147`:

```
function _assertValidationEngine(address candidate) private view {
    IValidationEngine engine = IValidationEngine(candidate);
    try engine.checkTrustlineStatus(address(this), uint256(0), bytes(""))
        returns (bool) {}
    catch { revert("Invalid validation engine"); }
}
```

This catches outright-reverting fakes. It **does not** catch a fallback-based "always true" proxy.

The HEAD probe validates a *different selector than it enforces* `_assertValidationEngine` probes view `checkTrustlineStatus(address,uint256,bytes)`, but runtime enforcement flows through `requireTrustline(...)`. An adopted/rogue engine can implement a **conformant** `checkTrustlineStatus` while implementing `requireTrustline` as a silent no-op.

Recommendations:

Adopt the HEAD `_assertValidationEngine` first (eliminates the easiest fakes). For strong assurance against fallback-truthy contracts:

- **Codehash verification:** `require(candidate.codehash == EXPECTED_CODEHASH)` for the proxy's runtime bytecode. *Pin to the **canonical** engine's known codehash rather than a bind-time snapshot, and reject any target whose code begins with the `0xef0100` 7702 designator prefix; a bind-time snapshot is defeated by later type-4 re-delegation (M13).*
- **Registry lookup:** consult a Trustline-maintained on-chain registry of canonical proxy addresses per chain. *This is the 7702-resistant remedy: a registry of deployed-contract addresses cannot be flipped by EOA delegation (M13).*
- **Probe the enforcing selector:** if a conformance probe is kept, it must exercise `requireTrustline` (the enforced path).
- **EIP-165 is recommended but currently absent.** The contracts do *not* implement `supportsInterface` for `IValidationEngine` (only AccessControl's interfaces + ERC-165 itself are advertised). `supportsInterface(type(IValidationEngine).interfaceId)` returns `false`. Once added, override `supportsInterface` to advertise the VE interface, then integrators can use EIP-165 as a sanity check.

Vulnerabilities High

H3: Sanctions-Oracle DoS Bricks Every Integrator Entry Point

Vulnerability severity:	High	Vulnerability probability:	Medium	Status:	Remediated

Remediated at commit `50e873e` by wrapping every oracle call in `try/catch` and treating any failure as sanctioned (fail-closed), so a reverting oracle can no longer brick entry points. There is no per-call gas budget, cached last-good answer, or circuit-breaker, so a gas-exhaustion/return-bomb oracle or a deliberate deny-all remains possible.

! Fix: Add a fixed `{gas:}` budget per oracle call, a max-staleness cached last-good answer, and an admin circuit-breaker / redundant oracle so a single induced failure is not a global deny.

`SanctionsOnlyValidationEngine._isSanctionsApproved` calls `sanctionsOracle.isSanctioned(...)` directly:

```
function _isSanctionsApproved(address sender, address[] memory addresses)
    private view returns (bool)
{
    if (!sanctionsOracleEnabled) return true;
    if (sanctionsOracle.isSanctioned(sender)) return false;
    uint256 n = addresses.length;
    for (uint256 i = 0; i < n;) {
        if (sanctionsOracle.isSanctioned(addresses[i])) {
            return false;
        }
        unchecked { ++i; }
    }
    return true;
}
```

No `try/catch`or per-call gas budget. A hostile or compromised oracle can:

- **Revert on every call** via `requireTrustline`/`checkTrustlineStatus`.
- **Consume all gas**.
- **Return-data bomb** by abusing Solidity's high-level call copying the full returndata into memory to ABI-decode it, and returning a multi-KiB blob inflates.

Vulnerabilities High

CONTINUED: H3: Sanctions-Oracle DoS Bricks Every Integrator Entry Point

Recommendations:

Wrap each oracle call in `try/catch` and treat failure as **fail-closed** (sanctioned).

Naive fail-closed is itself a griefing primitive. Anyone who can induce an oracle revert (or the oracle operator alone) flips every integrator to deny-all, and a fixed per-call gas budget intermittently trips a heavier oracle call:

- Fail-closed only on *distinguished* causes.
- Serve the last-good cached answer up to a max-staleness (ties M10).
- An admin circuit-breaker to switch fail-open/closed deliberately during an incident.
- A redundant second oracle so one revert is not a global deny.

```
try sanctionsOracle.isSanctioned{gas: ORACLE_GAS_BUDGET}(addr)
    returns (bool sanctioned)
{
    if (sanctioned) return false;
} catch {
    emit OracleCallFailed(addr);
    return false; // fail-closed
}
```

Emit an event on each failure.

Vulnerabilities High

H4: Adopt-Uninitialized-Proxy Hijack

Vulnerability severity:	High	Vulnerability probability:	Low	Status:	Remediated

Remediated at commit `4da9517` (doc `7edbde8`) by gating the adopt branch on `IAccessControlDefaultAdminRules(proxy).defaultAdmin() == msg.sender`, so a proxy pre-initialised by an attacker (admin ≠ adopter) is rejected. The check is bind-once and provenance is still inferred from runtime state, not an authenticated registry.

! Fix: Verify the proxy against a per-chain canonical-address registry / expected `EXTCODEHASH`, and reword the stale `Trustlined.sol` "multiple dapps can share the same ValidationEngine" NatSpec to "shared logic / same-operator proxy reuse."

The adopt-existing-proxy branch in `Trustlined.sol` (both pinned and HEAD) accepts the proxy with only a code-length check and never calls `initialize`:

```
if (proxy != address(0)) {
    require(proxy.code.length > 0, "Proxy is not a contract");
    validationEngine = IValidationEngine(proxy);
}
```

If `ERC1967Proxy(logic, "")` (empty init bytes) is deployed in a separate tx from `initialize`, an attacker can race them:

1. Trustline deploys `ERC1967Proxy(canonicalLogic, "")`.
2. Attacker observes the not-yet-initialized proxy and calls `initialize(attacker)` first. Attacker now holds `DEFAULT_ADMIN_ROLE`.
3. An integrator adopts this proxy via `Trustlined(address(0), proxyAddr)`. The attacker controls the engine they're calling.

HEAD's `_assertValidationEngine` does not catch this.

Recommendations:

- **Operational (Trustline):** never deploy a VE proxy with empty init bytes — always atomic init via the `ERC1967Proxy(logic, data)` constructor. Document this in deployment runbooks.
- **In code:** have the adopt path require an expected-admin parameter and check `hasRole(DEFAULT_ADMIN_ROLE, expected)` before binding:

```
function __Trustlined_init(address logic, address proxy, address expectedAdmin)
    internal { ...
    if (proxy != address(0)) {
        require(
            IAccessControl(proxy).hasRole(0x00, expectedAdmin),
            "Proxy admin mismatch"
        );
        validationEngine = IValidationEngine(proxy);
    } ...
}
```

Vulnerabilities High

H6: Non-View `requireTrustline` Interface rendering STATICCALL guarantee unenforceable

Vulnerability severity:	High	Vulnerability probability:	Low	Status:	Active

`IValidationEngine.requireTrustline(...)` is declared `external` (non-view) even though every in-scope implementation is `view`. Integrators' callsites compile to `CALL`, not `STATICCALL`: they treat as a read-only guard a function the type system permits to mutate state.

A future upgrade that makes `requireTrustline` stateful, for instance emitting a per-call audit event or caching a result, **silently strips the static-call guarantee from every already-deployed integrator**, turning the surface into a **reentrancy vector**.

Recommendations:

Decide intent in the interface.

- Option A (recommended): Declare `requireTrustline` as `view` in `IValidationEngine.sol`. Solidity will then emit `STATICCALL` from integrators and any upgrade attempting to mutate state will revert with `StateChangeDuringStaticCall`. *(Verified: a STATICCALL into a mutating `requireTrustline` reverts and leeches no admin, while the CALL path does.)*
- Option B: Document explicitly and have integrators use a thin `staticcall` wrapper for view-sensitive flows. Preserves future flexibility (events on each call) at the cost of remaining exposed.

Option A is a breaking interface change but eliminates the attack against existing integrators. Coordinate the H1/H2 (and the M1/I12 SDK-API changes) fixes into a single SDK version bump and drive integrators to upgrade to it.

Integrator defense-in-depth (until Option A ships). Treat `requireTrustline` as an untrusted external interaction: apply a `nonReentrant` guard on protected functions, order it within checks-effects-interactions, and prefer the `view checkTrustlineStatus` (STATICCALL — cannot reenter) wherever the flow permits.

Vulnerabilities Medium

M1: Integrator-Deployer Becomes Admin

Vulnerability severity:	Medium	Vulnerability probability:	Medium	Status:	Active

`Trustlined.__Trustlined_init_unchained` captures `msg.sender` as `initialOwner`:

```
address initialOwner = msg.sender;
bytes memory data = abi.encodeWithSignature("initialize(address)", initialOwner);
```

When the integrator contract is deployed by an EOA, the EOA becomes admin. When the integrator is deployed via a **factory contract** (CREATE2 deployers, account-abstraction bundlers, proxy admins, etc), the **factory's address** becomes admin . Most factories have no admin-transfer API, leaving the role stranded, and with no entry points.

Recommendations:

Accept an explicit `initialOwner` parameter in `Trustlined`'s constructor and **require it to be non-zero**:

```
constructor(address logic, address proxy, address initialOwner) {
    require(initialOwner != address(0), "Trustlined: zero owner");
    __Trustlined_init(logic, proxy, initialOwner);
}
```

M2: `setValidationConfiguration` Emits No Event

Vulnerability severity:	Medium	Vulnerability probability:	High	Status:	Resolved

Resolved at commit `2f9c047` by emitting `ValidationConfigurationChanged(bool sanctionsEnabled, address sanctionsOracle)` from `setValidationConfiguration`, restoring off-chain observability of enable/disable and oracle swaps.

`SanctionsOnlyValidationEngine.setValidationConfiguration` mutates the oracle pointer and the enable flag with no event:

```
function setValidationConfiguration(bool sanctionsEnabled, address sanctionsOracle_)
    external onlyRole(DEFAULT_ADMIN_ROLE) {
    if (sanctionsEnabled) {
        require(address(sanctionsOracle_) != address(0), "...");
        sanctionsOracle = ISanctionsList(sanctionsOracle_);
    } else {
        sanctionsOracle = ISanctionsList(address(0));
    }
    sanctionsOracleEnabled = sanctionsEnabled;
    // no event
}
```

Vulnerabilities Medium

CONTINUED: M2: `setValidationConfiguration` Emits No Event

Admin can disable sanctions screening or swap the oracle to a permissive list with no eventing or offchain observability.

It is not exploitable alone, but it removes the alerting that allows incident response.

Recommendations:

Emit a single event covering both fields:

```
event ValidationConfigurationChanged(
    bool sanctionsEnabled, address indexed oracle
);
```

Vulnerabilities Medium

M3: Address Screening Misses Indirect / Derived Actors

Vulnerability severity:	Medium	Vulnerability probability:	High	Status:	Remediated

Remediated at commit `2056dd0` by additionally screening `tx.origin`. This only catches the case where the originating EOA is itself listed; the relay/fresh-EOA, no-arg-overload, AA/4337, batch, counterfactual-TOCTOU and over-inclusion-DoS sub-cases remain, and unconditional `tx.origin` screening adds an AA/relayer availability hazard.

! Fix: Move counterparty screening to the explicit `addresses[]` overload, screen per-leg with check-and-exclude (do not hard-revert an attacker-joinable set), re-screen at value transfer, and treat `tx.origin` as advisory only.

The SDK screens the immediate `msg.sender` and any caller-supplied `addresses[]`. It has no view of fund provenance, upstream identity, the code at an address, or actors reached indirectly: the screened address is not the beneficial actor, and a sanctions-clean verdict binds neither identity nor settlement.

M3a: Clean intermediary (relay / fresh EOA / chosen array)

A sanctioned actor calls through a relay contract whose address is unlisted. **Mitigation:** for high-value flows screen `tx.origin` alongside `msg.sender`

M3b: No-arg overload screens only the caller

The no-argument `requireTrustline()` inspects only the caller, so a payments integrator can pay a sanctioned *recipient* and still pass. The EVMSDK README's `transfer(recipient, amount)` example uses exactly this sender-only call. **Mitigation:** pass every value-receiving counterparty via the `addresses[]` overload.

M3c: Account-abstraction / composition blind spot

A sanctioned actor behind a smart-contract wallet (ERC-4337 account, proxy, multisig) presents the wallet's clean address; and any integrator reached through a router, aggregator, or Permit2-style intermediary is screened on that clean contract, never the end-user. Under the `osaka` target, EIP-7702 (M13) further means a plain `tx.origin == msg.sender` no longer implies a human EOA, so the `tx.origin` mitigation is a weak heuristic here, not a remedy. **Mitigation:** screen the controlling owner/signer at the application layer; treat `tx.origin` as advisory.

M3d: Batch / multicall inner actors

An integrator exposing a batch/multicall entrypoint screens the outer caller once; inner sub-actions with different beneficiaries execute unscreened in the same transaction. **Mitigation:** screen per sub-action, treat a batch as N independent flows.

M3e: Counterfactual / identity TOCTOU

Address-membership binds neither the code at an address nor a not-yet-deployed identity. An actor pre-screens a codeless CREATE2 address, passes, then deploys controlled code to it; the verdict is invariant to what later occupies the address. **Mitigation:** re-screen at the moment of value transfer and bind to expected codehash.

M3f: Adversarial over-inclusion (forced-reversion injection DoS)

The inverse direction, with a *perfectly honest oracle and zero privilege*: `_isSanctionsApproved` returns `false` the moment *any* address in `{sender} ∪ addresses[]` is listed, and `requireTrustline` reverts hard. So an attacker who gets a known-sanctioned address *into* a screened set bricks the function for everyone. A payments/pool/escrow integrator that derives `addresses[]` from on-chain counterparties (LPs, depositors, signers, royalty recipients, settlement members) can be permanently frozen by the "Tornado-dust" pattern. **Worst case:** permanent denial of service / fund-freeze for all honest participants of an attacker-joinable set, for the cost of dust. **Mitigation:** do not feed a hard-reverting screen a flat set an attacker can join, screen per-leg (skip/quarantine the offending counterparty rather than revert the whole operation)

A check-and-exclude design (screen each leg, exclude rather than revert) is the only structure that resolves the tension.

Vulnerabilities Medium

M4: Unbounded addresses[] Gas-Amplification DoS

Vulnerability severity:	Medium	Vulnerability probability:	Medium	Status:	Resolved

Resolved at commit `951612a` by bounding the screened set with `require(addresses.length <= MAX_ADDRESSES_PER_CALL)` (`MAX_ADDRESSES_PER_CALL = 32`), capping per-call oracle iterations.

`SanctionsOnlyValidationEngine._isSanctionsApproved` loops over the user-supplied `addresses[]` with one external oracle call per element and no upper bound. Each element adds ~3.8K gas on top of fixed overhead:

```
gas for 1 address:    9,964
gas for 200 addrs:  767,991
delta (199 calls):  758,027  →  ~3.8K gas per call
```

If an integrator passes a user-controlled array through to `requireTrustline`, a malicious caller can scale the gas cost arbitrarily within the block limit.

Recommendations:

Cap the loop length inside the SDK:

```
require(addresses.length <= MAX_ADDRESSES_PER_CALL, "too many addresses");
```

Vulnerabilities Low

L1: Initial Owner Silently Holds Dormant `AUDITOR_ROLE`

Vulnerability severity:	Low	Vulnerability probability:	Medium	Status:	Active

`SanctionsOnlyValidationEngine.__SanctionsOnlyValidationEngine_init_unchained` grants both `DEFAULT_ADMIN_ROLE` and `AUDITOR_ROLE` to `initialOwner`:

```
_grantRole(DEFAULT_ADMIN_ROLE, initialOwner);
_grantRole(AUDITOR_ROLE, initialOwner);
```

No in-scope function consumes `AUDITOR_ROLE`. Future upgrades that introduce AUDITOR_ROLE-gated functions will **retroactively** empower the initial owner (typically the integrator deployer).

Recommendations:

Either:

- Remove the `AUDITOR_ROLE` grant until a function actually consumes it.

L2: Floating Pragma `^0.8`

Vulnerability severity:	Low	Vulnerability probability:	Low	Status:	Remediated

Remediated at commit `a70bfd1` by pinning the VE contracts to `pragma solidity 0.8.34` and the toolchain in `foundry.toml`. The four evmsdk SDK files still declare `pragma solidity ^0.8` with no Foundry pin, and the SDK hardhat config (`0.8.28/cancun`) contradicts the canonical-toolchain manifest.

! Fix: Pin the SDK contracts to a tested range (e.g. `^0.8.20`) and align `hardhat.config.js` with the canonical 0.8.34/prague toolchain.

All in-scope `.sol` files declare `pragma solidity ^0.8;`. This accepts any 0.8.x release.

The VE repo's `foundry.toml` pins `solc_version = "0.8.34"`, which mitigates for VE deployments. The SDK side has no such pin, so an integrator's package manager will resolve whatever 0.8.x is current. Behavioral differences between, say, 0.8.0 and 0.8.34 are documented but not exhaustive.

Recommendations:

Pin to a tested range, e.g., `pragma solidity ^0.8.20;` or `>=0.8.20 <0.9.0`. Apply to all in-scope files and to `foundry.toml` / `hardhat.config.js`.

Vulnerabilities Low

L3: Disable-Path Ignores Oracle Address Parameter

Vulnerability severity:	Low	Vulnerability probability:	Low	Status:	Resolved

Resolved at commit `061ac9e` by assigning `sanctionsOracle = ISanctionsList(sanctionsOracle_)` unconditionally (the disable branch no longer discards the supplied address), so a swap-and-disable preserves the operator's oracle reference.

`setValidationConfiguration(false, address(0xDEADBEEF))` accepts the call but silently discards the supplied oracle:

```
if (sanctionsEnabled) {
    require(address(sanctionsOracle_) != address(0), "...");
    sanctionsOracle = ISanctionsList(sanctionsOracle_);
} else {
    sanctionsOracle = ISanctionsList(address(0));  // sanctionsOracle_ ignored
}
```

An admin attempting to swap-and-disable in one transaction loses their oracle reference. Minor UX/correctness concern.

Recommendations:

Either preserve the supplied address (for "disabled but ready to re-enable"), or reject ambiguous calls:

```
require(
    sanctionsEnabled || sanctionsOracle_ == address(0),
    "no oracle when disabled"
);
```

Vulnerabilities Info

I1: Dormant State Variables in VE

Vulnerability severity:	Info	Vulnerability probability:	Info	Status:	Active

Five state variables in `SanctionsOnlyValidationEngine` are declared and (in most cases) set, but never read in the sanctions-only flow:

- `validationRegistry` — set at init to a meaningful value, never read
- `trustlineOracleEnabled` — **permanently `false`**: written once (to `false`) in `__SanctionsOnlyValidationEngine_init_unchained`, no other writer exists anywhere in the contract, and never read. Distinct from the others below — they hold real-but-unread values; this one is structurally unflippable.
- `AUTO_APPROVED_VALIDITY_PERIOD` — set to 30 minutes, never read
- `MANUAL_APPROVED_VALIDITY_PERIOD` — set to 5 days, never read
- `_txMetadata` — never written nor read

They consume storage slots 0–4 (see Appendix D), reducing the available `__gap` window for descendants. Slither also flags the unused-state-warning paths as dead code.

Vulnerabilities Info

CONTINUED: I1: Dormant State Variables in VE

Recommendations:

The safe way to clean this up depends on the deployment lifecycle. On a contract that already has a live proxy, deleting or reordering storage shifts every slot after it — and because `sanctionsOracleEnabled`/`sanctionsOracle` share slot 1 with the dormant vars (slots 0–4), a naive "cleanup" can flip the gate **fail-open**:

- **On any live (already-deployed) proxy:** never delete or reorder. Keep each dormant variable at its exact slot and width and achieve the cleanup with *layout-neutral* changes only. Change `public → private` to remove the misleading getters, rename in place to a `__deprecated_*` prefix, and annotate with `/// @dev dead: retained for storage-layout stability`. Gate every upgrade on a storage-layout diff (`slither-check-upgradeability`) in CI.
- **Greenfield only:** deletion plus resizing `__gap` upward to preserve total storage is safe and preferable.

I2: Hardcoded Dormant `VALIDATION_REGISTRY`

Vulnerability severity:	Info	Vulnerability probability:	Info	Status:	Active

`TrustlineSanctionsOnlyVE` hardcodes:

```
address private constant VALIDATION_REGISTRY = 0x34e3fe42e212aB9a9f5C22be5DAa98652DF151dd;
```

This is passed to `__SanctionsOnlyValidationEngine_init` and stored in `validationRegistry`, but no in-scope function reads it. The constant assumes cross-chain identity at this address: that a contract owned by Trustline exists at `0x34e3fe42...1dd` on every chain this engine deploys to. The assumption is not validated in code.

Recommendations:

Either document the intent (and validate cross-chain) or remove the constant until it is consumed.

Vulnerabilities Info

I3: Three-Layer Init Pattern in Non-Upgradeable Abstract (Pinned)

Vulnerability severity:	Info	Vulnerability probability:	Info	Status:	Active

`Trustlined` (pinned) uses the OZ-style three-layer init: `constructor` → `__Trustlined_init` → `__Trustlined_init_unchained`. This pattern is necessary for upgradeable contracts (where descendants need to chain initializers), but `Trustlined` is non-upgradeable.

Upstream status: Flattened at HEAD (the constructor body is inlined). No backport needed if the client adopts HEAD.

Recommendations:

Adopt the HEAD flattening

I4: `_authorizeUpgrade` Body Empty; `newImpl` Unvalidated

Vulnerability severity:	Info	Vulnerability probability:	Info	Status:	Remediated

Remediated at commit `ce614a6` by replacing the empty body with `_assertValidationEngineImplementation(newImplementation)` (an EIP-165 probe), rejecting EOA/empty/non-conformant upgrade targets. EIP-165 is trivially spoofable and there is no codehash/registry allowlist or per-call timelock, so a conformant permissive logic still installs (the C1-b residual).

! Fix: Bound `_authorizeUpgrade` to a codehash allowlist / Trustline registry and place the upgrade behind a timelock.

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

The `address newImplementation` parameter is unused. There is no codehash check, no zero-address check, no registry consultation. An admin can upgrade to `address(0)` (bricks the proxy via UUPS's `proxiableUUID` safety check) or to any contract satisfying the UUPS interface.

OZ's `_upgradeToAndCallUUPS` already verifies that `newImplementation` implements `proxiableUUID` correctly, so a completely random contract will be rejected. But any contract intentionally claiming UUPS conformance is acceptable. This is the root mechanism behind C1 (symptom b).

Recommendations:

`require(newImplementation.code.length > 0)` and consider a registry-based allowlist for additional defense in depth. The dominant mitigation remains the timelock recommended in C1.

Integrator-Surface & Distribution Findings

Why these are scored from the integrator's seat

The in-scope contracts are **non-custodial infrastructure**: they hold no funds and have no users of their own. A classic "impact-to-the-audited-contract" lens therefore scores almost everything here as negligible, which is misleading. For an *embedded compliance SDK whose entire purpose is to be inherited*, the unit of impact is the **integrator and its end-users**, because that is where the value, the user base, and the legal/sanctions obligation actually sit.

Two modifiers push severity **up**:

- **Silence** — the SDK emits no events on configuration/enable/disable, so a broken guarantee is undetectable on-chain.
- **Irrecoverability** — several states (renounced admin, immutable VE binding) have no remedy short of full redeploy.

Integrator-Surface High

H5: Fail-Open Default — the Compliance Gate Ships Disabled

Vulnerability severity:	High	Vulnerability probability:	High	Status:	Resolved
Resolved at commit `061ac9e` by initialising `sanctionsOracleEnabled = true` and treating an unset oracle as sanctioned, so every `requireTrustline` reverts (fail-closed) until an oracle is configured.					

A freshly deployed VE has `sanctionsOracleEnabled = false` (set in `__SanctionsOnlyValidationEngine_init_unchained`), and the screening helper returns *approved* when disabled:

```
if (!sanctionsOracleEnabled) return true;   // fail OPEN
```

Between deployment and a **separate** `setValidationConfiguration(true, oracle)` transaction, every `requireTrustline` passes. An integrator that goes live on deploy, or never runs the enable step, screens nobody, and because `setValidationConfiguration` emits no event (M2), there is no on-chain signal that the gate is off.

See **Appendix I** for the full reachable-state enumeration (S0 is this default) and the proof that no on-chain invariant forces screening-before-usable.

Recommendations:

Make the primitive fail **closed**: require a non-zero oracle and enabled state at initialization, or have `requireTrustline` revert until a configuration has been explicitly set.

Integrator-Surface Medium

M5: `checkTrustlineStatus` Return Value Is Silently Ignorable

Vulnerability severity:	Medium	Vulnerability probability:	Medium	Status:	Acknowledged
Acknowledged at commit `2b0390b` by documenting (README role table) that `checkTrustlineStatus` is a non-enforcing query that must be wrapped in `require(...)` and `requireTrustline` is the enforcing call. The query was not renamed and no compile-time guard was added.					
! Fix: Rename the query (e.g. `isTrustlineOk`) so a discarded return value is self-evidently non-enforcing.					

The SDK exposes two near-identically named helpers: the enforcing `requireTrustline` (reverts) and the query `checkTrustlineStatus` (returns a `bool`). An integrator that calls the query form and discards the result gets no enforcement.

Recommendations:

Document unambiguously that `checkTrustlineStatus` is a *query* that must be wrapped in `require(...)`, and that `requireTrustline` is the enforcing call. Consider renaming the query (e.g., `isTrustlineOk`).

M6: `ValidationMode` Accepted but Discarded

Vulnerability severity:	Medium	Vulnerability probability:	Medium	Status:	Acknowledged
Acknowledged at commit `afd1428` by documenting that `ValidationMode`, `value` and `data` are accepted for interface compatibility but ignored; only `sender`/`tx.origin`/`addresses` are screened. The unused modes remain in the interface with no per-mode validation.					
! Fix: Remove the unsupported modes from the interface (or revert on them) until per-mode policy is implemented.					

The interface accepts parameters that the sanctions-only engine silently discards, so integrators build policy on guarantees that do not exist. Two instances:

`ValidationMode` accepted but discarded

`IValidationEngine` advertises four modes (`Dapp`, `UniswapV4`, `MorphoV2`, `ERC3643`) and both mode-taking functions accept the parameter, but the argument is unnamed and unused. **Worst case:** an ERC-3643 security-token integrator selects `ERC3643` mode expecting transfer-restriction / identity-registry enforcement and receives only sanctions-list membership.

M6b: `msg.value` / `msg.data` ignored

`Trustlined` forwards `msg.value` and `msg.data` to the engine, but the sanctions-only VE ignores both: the decision is purely sanctions-membership and value-independent.

Recommendations:

Either implement per-mode validation, or **remove the modes from the interface** until they are implemented and `revert` on any unsupported mode.

Integrator-Surface Medium

M7: Oracle Is Never Conformance-Checked — an Honest Admin Can Brick All Screening

Vulnerability severity:	Medium	Vulnerability probability:	Medium	Status:	Resolved

Resolved at commit `dc53f2e` by validating the oracle on enable: `setValidationConfiguration` now requires `sanctionsOracle_.code.length > 0` and runs a `try/catch isSanctioned` conformance probe before binding.

The SDK validates the validation engine before binding (`code.length` on the pinned path, `_assertValidationEngine` on HEAD), but `setValidationConfiguration` only null-checks the oracle:

```
sanctionsOracle = ISanctionsList(sanctionsOracle_);   // no codesize, no probe
```

If a single fat-fingered or malicious admin transaction takes **100% of entry points offline** for every integrator on that VE. Paired with C1 (renounce), it is permanent.

This is state S3 (BRICKED) in the **Appendix I** configuration state machine.

Recommendations:

In `setValidationConfiguration`, when enabling, require `sanctionsOracle_.code.length > 0` and perform a `try/catch` conformance probe (a trial `isSanctioned` call) — the same standard already applied to the engine.

Integrator-Surface Medium

M8: Oracle Output Is Trusted Blindly

Vulnerability severity:	Medium	Vulnerability probability:	Medium	Status:	Acknowledged

Acknowledged at commit `ecb67cf` by documenting that the engine does not attest oracle trustworthiness and that integrators are responsible for choosing a trustworthy oracle. No canary/coverage check or empty-list fail-closed control was added.

! Fix: Require a known-sanctioned canary to screen true at configuration time, treat an empty/low-population list as fail-closed, and reconcile determinations off-chain.

The engine treats `sanctionsOracle.isSanctioned(...)` as a truth function with no defense against the answer being wrong.

M8a: Targeted censorship

A compromised or coerced oracle returns `true` for chosen victims denying them availability while looking healthy.

M8b: False-negative / lies-clean

An oracle that returns `false` for a genuinely-sanctioned actor lets sanctioned parties transact freely and silently across every integrator.

M8c: Honest-but-empty

A perfectly honest, live, responsive oracle with **no data** for this chain or asset class returns `false` for everyone → silent fail-open while every health check is green.

Recommendations:

Verify the Oracle. Log every determination and reconcile against the authoritative list off-chain; monitor decision *rates* for anomalous denials and alert on any address flipping sanctioned→clean without an official de-listing; second-source high-value flows. Validate coverage at configuration time (a known-sanctioned canary must screen `true` before enabling) and treat an empty/low-population list as fail-closed.

Integrator-Surface Medium

M9: Screening Decision Not Bound to Settlement Time or Domain

Vulnerability severity:	Medium	Vulnerability probability:	Medium	Status:	Active

`requireTrustline` is evaluated in the calling transaction and produces no artifact bound to the settled action, a later time, or another chain. A pass at screen-time therefore imposes nothing on what executes afterward or elsewhere.

M9a: Temporal (screen-at-submission vs settle-later)

Any two-step or async flow: queued/timelocked withdrawals, escrow release, intent/order settlement, governance execution, L2 delayed messages, etc, screens at submission, then settles later when sanction status or the oracle may differ. An actor passes at submission, is listed before settlement, and the deferred action still executes; the validity window equals the settlement delay. **Mitigation:** re-screen at execution time, not only at submission.

M9b:Cross-domain (screen on source, settle on destination)

For intent-based bridging (Across-style), L1→L2 delayed messages, or any "screen on source, settle on destination" flow, the screen on chain A imposes nothing on the action that executes on chain B. There is no on-chain exploit to reproduce; the gap is the absence of a domain-bound attestation. **Mitigation:** require a destination-bound attestation, or re-screen at settlement on the executing domain.

Recommendations:

- **Implement the bound-certificate model it already markets.** The README advertises a Trustline oracle that issues/verifies *validation certificates*, which the sanctions-only engine does not implement. A screen should produce a signed artifact bound to `{subjects, actionHash, chainId, issuedAt, expiry}`, verified at the *point of use* rather than only at submission. `expiry` closes M9a (a stale certificate fails at settlement, forcing a re-screen); `chainId` closes M9b (a chain-A certificate won't verify on chain B); `actionHash` ties the screen to the specific settled action.
- **Provide the M9b primitives.** An integrator cannot bind a screen across domains: deploy the canonical VE + a fresh oracle on every settlement chain (so re-screen-at-destination is even possible), and sign destination-bound attestations the source can hand to the destination.
- **Make re-screening meaningful** by adding a freshness signal to `ISanctionsList` (`lastUpdated()`/heartbeat) and enforcing max-staleness (M14) — otherwise a settlement-time re-screen runs against a possibly-frozen feed.

Integrator-Surface Medium

M10: No Oracle Freshness / Staleness Validation (Stale-Oracle Laundering)

Vulnerability severity:	Medium	Vulnerability probability:	Medium	Status:	Active

The engine trusts `sanctionsOracle.isSanctioned(addr)` with **no freshness check**, and the `ISanctionsList` interface returns a bare `bool` with **no timestamp / heartbeat**. Therefore:

- **Stale / frozen-feed laundering.** A snapshot or cached oracle whose update feed has stalled (operator stops updating, or an attacker DoSs the updater) keeps every "not sanctioned" answer frozen. An actor designated *after* the snapshot reads as clean and launders during the window.
- **Update-lag front-running.** An actor who sees the pending oracle update that will list them races a transaction ahead of it.
- **Cross-chain freshness divergence.** The oracle on an L2 may lag mainnet; an address can be live-sanctioned on one chain and stale-clean on another.

Recommendations:

Extend `ISanctionsList` with a freshness signal (e.g., `lastUpdated()` / a heartbeat), and have the engine enforce a maximum staleness — **fail-closed** past the threshold (treat a stale oracle like a reverting one, per H3). Document the chosen staleness bound. Off-chain: require the oracle's data pipeline to be attested/monitored for freeze.

Integrator-Surface Medium

M11: Supply Chain — Unpinned, Unverifiable Build & Distribution (Audited Source ≠ Shipped Artifact)

Vulnerability severity:	Medium	Vulnerability probability:	High	Status:	Remediated

Remediated at commit `a70bfd1` + `7ef234f` (VE) by pinning solc 0.8.34 / `prague` / OZ 5.4.0 exactly, reproducible-bytecode settings, a committed lockfile, `provenance: true`, verification scripts and per-chain `EXTCODEHASH` in `releases/`. This covers the VE repo only — the `@trustline.id/evmsdk` package integrators install still floats OZ `^5.1.0`, ships gitignored prebuilt artifacts with no lockfile/provenance, and compiles `0.8.28/cancun`, contradicting the canonical-toolchain manifest.

! Fix: Apply the same pinning/lockfile/provenance/reproducible build to the evmsdk package, and publish the audited build as the version integrators actually resolve.

M11a: Unpinned toolchain (non-reproducible compile)

Compiler pragma `^0.8` compiles under any 0.8.x the integrator's toolchain resolves (tracked as **L2**). `evm_version = "osaka"` targets a not-yet-finalized fork: bytecode compiled against its opcode/gas semantics may behave differently than tested, or revert on chains that have not activated it. The audit reproducer deliberately builds with `cancun`, so the exact `osaka` bytecode Trustline ships was **not** the bytecode under test. **Mitigation:** narrow the pragma (e.g. `>=0.8.20 <0.9.0`) and set `evm_version` to a finalized fork per target chain.

M11b: Floating dependencies, no lockfile (inbound substitution + base drift)

Both repos declare `@openzeppelin/contracts(-upgradeable): "^5.1.0"` (the reproducer pins exactly `5.4.0`) and the engine declares `@trustline.id/evmsdk: "^1.0.7"`. The VE repo commits a `package-lock.json`, but the package integrator imports `@trustline.id/evmsdk` (currently `1.0.8`), which has **no committed lockfile** and floats OZ at `^5.1.0`. Two consequences: *(1) An integrator resolves a different OZ 5.x than was audited; a future minor that changes `AccessControlUpgradeable`/`UUPSUpgradeable`/`Initializable` internals (ERC-7201 slot derivation, `_disableInitializers`) invalidates the storage-layout / upgrade-safety conclusions in Appendix D/I for the shipped version. (2) Caret ranges auto-pull the newest matching release; an npm account takeover or malicious publish of `@trustline.id/evmsdk` (or an OZ-adjacent package) propagates to every integrator's next install with no code change on their side.* ***Mitigation:*** *commit a lockfile to the SDK package, install with `npm ci`, and pin OZ to the exact audited `5.4.0` and the SDK dependency exactly.*

M11c: Gitignored, prebuilt, unaudited published artifact (no provenance)

The audit reviewed ***Solidity source*** *at `9673879` (npm `1.0.6`). What an integrator `npm install`s is* ***not that source*** *— `index.js` exports compiled `./artifacts/contracts/*.json` (ABI/bytecode) and `package.json:files` ships `artifacts/`+`dist/`. Those are* ***gitignored*** *(along with `package-lock.json`, `.gitignore` line 2), generated at publish time, never committed, never audited. No lockfile, no CI, no `.npmignore`, no provenance/SLSA — the published tarball is whatever the publisher's machine produced.* ***Three mismatched build configs across one system, none matching what's shipped:*** *the SDK compiles with hardhat → **solc 0.8.28 / cancun**; the VE with foundry → **0.8.34 / osaka**; the audit reproducer uses **0.8.34 / osaka** (matching the VE). Demonstration: `forge inspect TrustlineSanctionsOnlyVE bytecode` under the same solc, varying only `evm_version`, yields **different bytecode** (cancun `788c1f28…` vs osaka `68360ae1…`). So there is **no single artifact** an integrator can hash and match against "the audited thing." Version drift compounds it: audited `1.0.6`, `package.json` now `1.0.8`, VE imports `^1.0.7` — a third version, all via caret ranges.*

Integrator-Surface Medium

M11: Supply Chain — Unpinned, Unverifiable Build & Distribution (cont.)

M11d: Interface / ABI version skew → selector-mismatch DoS

The VE imports its own `IValidationEngine` from the npm package (pinned `^1.0.7`) while integrators import from a possibly different published version (`^1.0.8+`) — two independently-versioned halves of one trust boundary held together only by caret ranges. A minor release that reorders or adds an interface parameter changes the function selector; the integrator's `requireTrustline` call then lands on a non-existent selector → revert (total DoS) or hits the wrong overload. A reordered-param selector reverts against the audited engine while the correct one resolves. This is a ***correctness/availability*** *failure.*

Recommendations:

- *Narrow the pragma and set a finalized `evm_version` per chain*
- *Commit a lockfile to the SDK package and install with `npm ci`, pinning OZ to the exact audited `5.4.0` and the SDK dependency exactly*
- *Make builds reproducible (pin solc + full settings + `evmVersion` identically across SDK and VE), publish with provenance (`npm publish --provenance` / SLSA), sign releases, and publish the expected `EXTCODEHASH` / canonical (solc, evm_version, OZ) tuple per chain*

M12: Calldata Surveillance via a Shared Proxy

Vulnerability severity:	Medium	Vulnerability probability:	Medium	Status:	Active

`Trustlined` forwards full `msg.data` to the VE on every guarded call. Under the SDK's invited ***shared-proxy*** *model (many dApps adopting one VE), a single VE operator — or a C1-upgraded logic — receives the* ***complete calldata of every integrator bound to it, on every call****. That is a ready-made transaction-graph / deanonymization dataset. For a privacy-sensitive compliance product this is a mass-surveillance vector.*

Worst case: *the proxy operator reconstructs cross-dApp user activity (amounts, recipients, intents) for every integrator on the shared engine — a deanonymization dataset that compounds with I7 enumeration.*

Recommendations:

Do not forward full `msg.data` — pass only what the policy needs (the screened addresses). Discourage proxy sharing across unrelated integrators.

Integrator-Surface Medium

M13: EIP-7702-Delegated EOA Defeats Engine-Binding Verification & Proxy Monitoring

Vulnerability severity:	Medium	Vulnerability probability:	Medium	Status:	Remediated

Remediated at commit `1c4aeaf` by rejecting delegated-EOA targets — `_assertContractAccount` requires `EIP7702Utils.fetchDelegate(target) == address(0)` on both bind paths. Binding is still bind-once (later re-delegation/upgrade is not re-verified) and there is no registry.

! Fix: Add a per-chain registry of Trustline-controlled addresses and treat the engine binding as re-verifiable rather than bind-once.

*A 7702-delegated EOA has code — the 23-byte designator `0xef0100‖address` — and a defined codehash, while executing the delegated implementation in its own context. As a bound "engine" it therefore **passes every weak binding check**: `proxy.code.length > 0` (H2 pinned), the HEAD `_assertValidationEngine` conformance probe (if the delegate implements a conformant `checkTrustlineStatus`), and a human code-read. Additionally, the controlling EOA can submit a **new type-4 authorization at any later block**, re-delegating to permissive code (or to `address(0)` to clear it). The integrator's `validationEngine` still points at the same address.*

***Two consequences:** (1) a new bind-target class for the H2/B5 social-engineering scenario — "adopt this Trustline engine at 0x…" pointing at an attacker's 7702 EOA that looks perfectly conformant at bind time; (2) the **codehash assumption is unsound** — codehash is no longer a stable property: it is `keccak(0xef0100‖delegate)` and changes on re-delegation, so a bind-time snapshot is defeated by a later type-4 tx.*

Recommendations:

*Reject delegated-EOA targets: require the bind target to be a genuine contract account. Check its code does not begin with the `0xef0100` 7702 designator prefix. Do this in addition to a codehash/registry check. Prefer a **per-chain registry of addresses Trustline controls** over "has code / conformant / codehash-snapshot" heuristics, and treat the engine binding as re-verifiable, not bind-once (codehash/monitoring must account for account-level delegation changes, which emit no proxy event).*

Integrator-Surface Medium

M14: Diamond / Delegatecall Slot-0 Active Engine Redirection

Vulnerability severity:	Medium	Vulnerability probability:	Low	Status:	Acknowledged

Acknowledged at commit `9f180bf` by documenting that only direct inheritance into a contract that owns slot-0 storage is supported (Diamond facets / delegatecall routers unsupported) — one of the recommended options. No ERC-7201-namespaced storage variant was added.

! Fix: Provide an ERC-7201-namespaced `validationEngine` storage variant if diamond/delegatecall hosts must be supported.

*The wrong-base-class advisory covers `Trustlined` behind a proxy where the constructor never runs (validationEngine stays `address(0)`). It does **not** cover `Trustlined` mixed into a live `delegatecall` context that actively uses slot 0 such as a Diamond (EIP-2535) facet or a delegatecall router. `validationEngine` sits at a fixed inherited slot with **no ERC-7201 namespacing and no `__gap`** (unlike the VE side, P3/P4). In such a host, slot 0 collides with whatever the host stores there, so the read engine address becomes a **host-controlled (attacker-influenceable) value**.*

Recommendations:

State the storage-safety contract for non-standard inheritance hosts: `Trustlined` must not be used as a Diamond facet / delegatecall target without dedicated (ERC-7201-namespaced) storage for `validationEngine`. Add a namespaced-storage variant for upgradeable/diamond hosts, or document that only direct inheritance in a contract that owns slot 0 is supported.

M15: Oracle-Operator Vantage — Real-Time Screening Feed & Intent-Before-Settlement MEV

Vulnerability severity:	Medium	Vulnerability probability:	Medium	Status:	Active

*Every guarded call routes the screened address(es) to the oracle. Therefore: (1) a **continuous deanonymization / transaction-graph feed** accrues to the oracle operator. (2) because screening is a **precondition to a value action**, anyone who observes the oracle call (the oracle operator, or a mempool watcher of the gated tx) **learns intent before settlement** → front-running / sandwiching — an MEV surface.*

Recommendations:

Minimize what reaches the oracle (avoid sending the full counterparty set where a hash/commitment suffices); avoid sharing one engine/oracle across unrelated integrators (M12); for MEV-sensitive flows, decouple the screen from the value action (commit–reveal or private mempool) so the oracle call does not telegraph intent.

Integrator-Surface Medium

M16: Governance / Legal Primitives Absent

Vulnerability severity:	Medium	Vulnerability probability:	Medium	Status:	Acknowledged

Acknowledged at commit `6c0927d` by recommending multisig + `TimelockController` custody with a documented succession migration. No emergency-freeze / targeted per-address or per-integrator block primitive was added.

! Fix: Add a fail-closed `pause()` (multisig+timelock, with a no-timelock emergency freeze-only role) and a targeted block list invokable under legal order without an upgrade.

M16a: Privileged key-holder is themselves sanctioned / seized

A privileged key-holder may be designated, indicted, or have their key seized. Single-admin VE (C1) + admin OFAC-listed → governance is legally frozen with possibly no compliant party able to operate it, and fail-closed-on-renounce (C1's recommendation) makes it worse. The oracle operator being seized/compelled is modeled only as censorship (M8); the "operator must cease operation → silent staleness/halt" path is not. ***Mitigation:*** *DAR held by a multi-jurisdiction multisig with a documented compliant-successor procedure; a governance path that does not require the (possibly-designated) incumbent to act; an oracle continuity plan for operator seizure. Avoid single-admin custody.*

M16b: No emergency-freeze / legal-order primitive

The inability to honor a court/regulator emergency-freeze order is itself a finding. There is no targeted freeze, no per-integrator kill, no incident "deny-all-now" that fails closed, so a regulator directing a freeze against a specific integrator or address set cannot be obeyed. ***Mitigation:*** *add a fail-**closed** `pause()` (multisig+timelock, with a no-timelock freeze-only emergency role) that makes `requireTrustline` deny-all, plus a targeted per-address / per-integrator block list independent of the oracle, invokable under legal order without an upgrade.*

Recommendations:

Add multi-party governance with compliant succession, and a fail-closed targeted freeze.

Integrator-Surface Low

L4: Cross-Chain Canonical-Address Verification Fragility

Vulnerability severity:	Low	Vulnerability probability:	Medium	Status:	Remediated

Remediated at commit `4b98242` by documenting CREATE address non-determinism (`Trustlined.sol` NatSpec) and publishing per-chain canonical addresses + `EXTCODEHASH` in the VE `releases/` manifest. The deploy script still uses `new …()` (CREATE), so addresses are not reproducible across chains by construction.

! Fix: Deploy the canonical logic via CREATE2 with a pinned salt + init-code from a single attested deployer so the address is deployer-bound and reproducible per chain.

*The deploy script uses `new TrustlineSanctionsOnlyVE()` (**CREATE — nonce-dependent**), so the canonical logic address is **not** guaranteed identical across chains (two deployments by one deployer land at different addresses). Integrators must therefore track a different correct address per chain, and the registry constant assumes an identity it cannot guarantee.*

Recommendations:

*Deploy the canonical logic/registry via `CREATE2` with a pinned salt + pinned init-code from a single attested deployer (so the address is reproducible and deployer-bound across chains), and publish a **per-chain** registry of canonical addresses + expected `EXTCODEHASH`. Validate the `VALIDATION_REGISTRY` constant per chain before any future engine reads it.*

L5: Out-of-Range `ValidationMode` Hard-Reverts at ABI-Decode

Vulnerability severity:	Low	Vulnerability probability:	Low	Status:	Acknowledged

Acknowledged at commit `afd1428` alongside the M6 mode-handling note: the modes are documented as ignored, but they remain in the interface, so an out-of-range `ValidationMode` still hard-reverts at ABI-decode.

! Fix: Remove the modes from the interface (see M6) to eliminate the out-of-range decode surface.

*The mode-taking overloads take `ValidationMode` (enum, 0–3). M6 notes the mode is ignored in the body, but a value ≥ 4 reverts at **ABI-decode, before the body runs**. "Mode is ignored" and "mode out of range is a hard revert"*

Recommendations:

If the modes are not implemented, remove them from the interface (see M6).

Integrator-Surface Info

I7: Permissionless Sanctions Enumeration (Privacy / Recon)

Vulnerability severity:	Info	Vulnerability probability:	High	Status:	Active

`checkTrustlineStatus` / `requireTrustline` are unauthenticated, `view`, and take an arbitrary `sender`/`addresses[]`. Anyone can query the sanctioned-status of any address for free through the VE, enumerate the underlying list, or probe oracle availability. A random caller learns the status of arbitrary subjects.

Worst case: the oracle's dataset (which may be proprietary/curated) is freely enumerable on-chain; the VE doubles as a reconnaissance primitive (probe who is/ isn't listed, or whether the oracle is live).

Recommendations:

If the sanctions data is intended to be public, accept this. If it is sensitive, the read surface needs access control or rate-limiting at the oracle, and integrators should be aware the VE leaks list membership.

I8: Untyped Revert Reasons Hamper Incident Response

Vulnerability severity:	Info	Vulnerability probability:	Medium	Status:	Active

A sanctioned sender and a sanctioned counterparty both revert with the identical string `"Transaction involves sanctioned addresses"`. Separately, oracle-failure modes (H3 revert, M7 no-code) propagate as yet-different non-typed reverts. There is no typed-error taxonomy, so integrators wanting differentiated fail-open/fail-closed handling and monitors wanting to alert have no programmatic signal to branch on.

Recommendations:

Replace the string with custom errors carrying cause + culprit (e.g., `SanctionedAddress(address)`, `OracleUnavailable()`).

Integrator-Surface Info

I9: Unauthenticated Event Emission (Indexer / Compliance-Dashboard Poisoning)

Vulnerability severity:	Info	**Vulnerability probability:**	Medium	**Status:**	Active
Status unchanged (Active). The I12 remediation added a second unauthenticated, forgeable event `ValidationEngineAdopted(...)`, modestly enlarging this indexer-poisoning surface; the recommended consumer-side `log.address` filtering still applies.					

Events are not authenticated by emitter. Anyone can deploy a contract that emits `ValidationEngineDeployed(client, engineProxy, logic, initialOwner)` with the exact topic hash and arbitrary args. An off-chain indexer / compliance dashboard that keys on the topic ***without filtering*** *`log.address ∈ {known integrators}` can be fed fabricated "integrator → VE" relationships, or made to show a sanctioned/fake integrator as "Trustline-protected.".*

Recommendations:

Indexers/dashboards must filter on `log.address` against a trusted set (the Trustline registry / known integrator addresses), not on topic alone. Consider emitting from a single canonical registry contract so there is one authenticated emitter to trust.

Integrator-Surface Info

I10: Compliance-Model Adequacy — Address-Boolean Screening Cannot Express the Legal Obligation

Vulnerability severity:	Info	Vulnerability probability:	High	Status:	Active

The audit assures the mechanism around the oracle, which is treated as the "truth function" (Appendix B). A boolean `isSanctioned(address)` *oracle* ***cannot satisfy the obligation the product markets****, regardless of correct implementation:*

- ***OFAC 50% rule.*** *Entities ≥50%-owned by SDNs are blocked even if the specific address is unlisted. Address-membership cannot express ownership/control derivation — a fresh wallet of a 50%-owned entity is "clean".*
- ***Address ≠ identity ≠ entity.*** *Sanctions designate persons/entities/vessels; the chain has addresses. Even with zero adversary, the data model is a lossy projection of the obligation.*
- ***Under-typed answer.*** *One bool — no list-source (OFAC vs EU vs UN), no jurisdiction, no designation date. The answer is under-typed for the legal question (deeper than A4/M6's ignored modes).*

"Trustlined ⇒ compliant" is ***unsatisfiable by an address-boolean oracle****; the integrator's legal obligation is strictly broader than what the SDK can attest. See the Scope-of-Assurance note in the Executive Summary.*

Recommendations:

State the scope of assurance explicitly in product docs: the SDK attests address-membership against the configured oracle, not legal compliance. Integrators remain responsible for ownership/control derivation (50% rule), entity/identity resolution, jurisdiction/list selection, and designation freshness — at the application layer or via a richer attestation model. Do not market "compliant"; market "address-list screening."

Integrator-Surface Info

I11: On-Chain Compliance Record Is an Integrator Privacy / Data-Protection Liability

Vulnerability severity:	Info	Vulnerability probability:	Medium	Status:	Active

*By routing every screen on-chain, the integrator creates a **permanent, public record of compliance decisions about its users**, potentially violating (e.g., GDPR right-to-erasure).*

Recommendations:

Minimize on-chain personal-data exposure: avoid emitting per-user screening outcomes; prefer off-chain/commitment-based records where the regulatory regime allows; document the immutability/erasure tension for integrators subject to GDPR/CCPA.

I12: Adopt Path Emits No `ValidationEngineDeployed` Event (Indexer Completeness)

Vulnerability severity:	Info	Vulnerability probability:	High	Status:	Resolved

Resolved at commit `d7e1693` by emitting a distinct `ValidationEngineAdopted(address indexed client, address indexed engineProxy)` on the reuse branch, closing the indexer under-count for adopted proxies.

`Trustlined` emits `ValidationEngineDeployed(client, engineProxy, logic, initialOwner)` only in the deploy-fresh branch. The adopt-existing branch (`proxy != address(0)`) is silent, so an indexer keyed on topic `keccak256("ValidationEngineDeployed(address,address,address,address)")` undercounts every integrator that adopts an existing proxy.

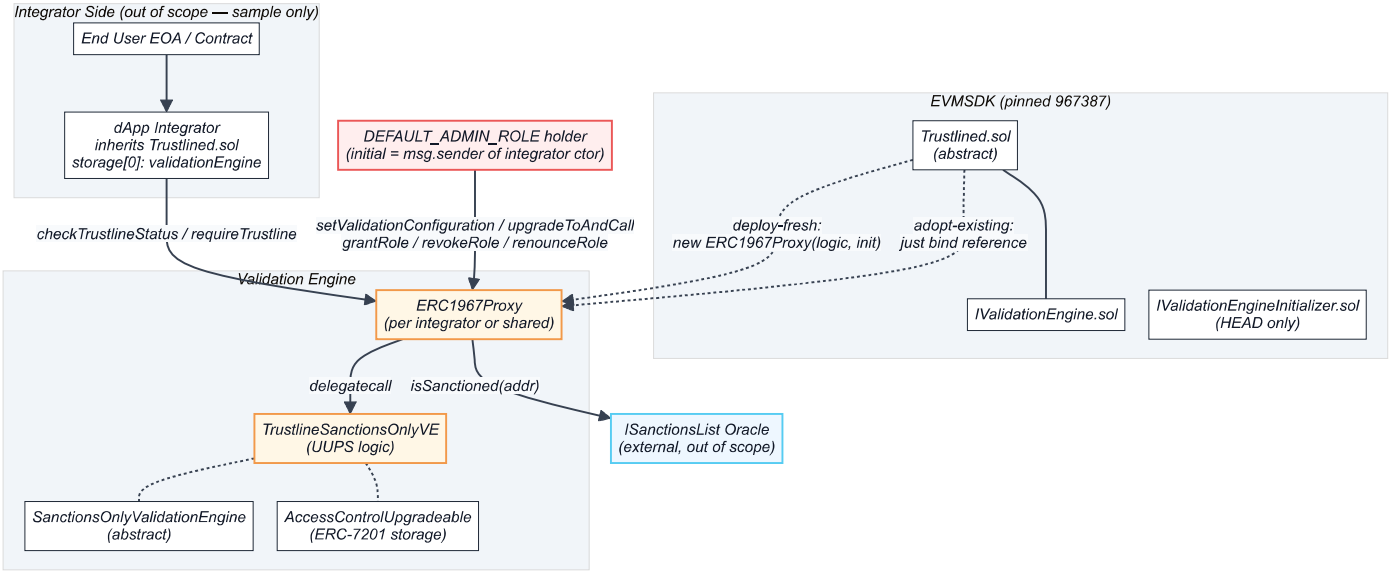
Recommendations:

Add a distinct event for the adopt path and emit in both branches:

```
event ValidationEngineAdopted(
    address indexed client, address indexed engineProxy
);

if (proxy != address(0)) {
    require(proxy.code.length > 0, "...");
    validationEngine = IValidationEngine(proxy);
    emit ValidationEngineAdopted(address(this), proxy);
} else { ... }
```

APPENDIX A: Architecture Diagram



*Two cooperating components: an **integrator-side** abstract (`Trustlined`) that the dApp inherits, and a **VE-side** UUPS-upgradeable engine accessible through an `ERC1967Proxy`. The integrator either auto-deploys a fresh proxy in its constructor (making the deployer admin) or adopts a pre-existing one. The engine consults an external `ISanctionsList` oracle.*

APPENDIX B: Authority Structure

Role / Actor	Where granted	What they control	Off-chain trust requirement
`DEFAULT_ADMIN_ROLE`	Granted at init to `msg.sender` of integrator ctor (= `initialOwner`)	setValidationConfiguration, upgradeToAndCall, grantRole/revokeRole on both roles, renounceRole	Must be multisig + timelock in production. Single point of failure (C1).
`AUDITOR_ROLE`	Granted at init to `initialOwner` (same address as admin)	**Nothing in scope.** Reserved/dormant — see L1.	Currently inert. Future upgrades will retroactively empower whoever holds it.
Sanctions oracle operator	Out of scope (external contract)	The truth function `isSanctioned(addr) returns (bool)`	Must be honest, live, and gas-bounded. Hostile oracle = H3. **Second independent admin domain**: the oracle is itself admin-owned/upgradeable, with its own keys/timelock — a separate compromise event from the VE admin. Compliance = both domains honest (compounds H3/M7/M8).
Trustline logic deployer	Off-chain (runs `DeploySanctionsOnlyValidationEngine.s.sol`)	Publishes canonical logic address. `_disableInitializers()` in logic constructor prevents direct init.	Address must be cryptographically attested to integrators (out-of-band).
Integrator dApp developer	n/a (off-chain code author)	Chooses deploy-fresh vs. adopt-existing; composes `addresses[]`; chooses `require` vs. `check`	Trusted by their own users only. Adversarial integrator can deploy compliance theater (Appendix E.C).
End user	n/a	Only their own transaction. Untrusted.	Bypasses possible (M3). Architectural limitation.
Direct VE caller	n/a	Can call any view function on the proxy with arbitrary `sender` parameter	Untrusted. Acceptable for sanctions-only (answer depends only on the addresses, not on the caller's identity).

Role administration

Both `DEFAULT_ADMIN_ROLE` and `AUDITOR_ROLE` have `DEFAULT_ADMIN_ROLE` as their `roleAdmin` (OZ default; no `_setRoleAdmin` override in scope). A DAR holder can grant or revoke either role for any account. The DAR holder can also grant DAR to additional addresses without limit — there is no on-chain cap on the admin set size.

APPENDIX C: Public Entry Points

Integrator contract (inherits Trustlined)

Entry	Visibility	Caller	Notes
`constructor(logic, proxy)`	deploy-time	anyone deploying	captures msg.sender as admin
`validationEngine()`	external view	anyone	auto-getter

VE proxy (delegates to `TrustlineSanctionsOnlyVE`)

State-mutating, permissioned

Selector	Entry	Restriction
`0xc4d66de8`	`initialize(address)`	one-shot (initializer modifier)
`0x7a7d8b25`	`setValidationConfiguration(bool,address)`	DEFAULT_ADMIN_ROLE
`0x4f1ef286`	`upgradeToAndCall(address,bytes)`	_authorizeUpgrade → DAR
`0x2f2ff15d`	`grantRole(bytes32,address)`	role admin (= DAR)
`0xd547741f`	`revokeRole(bytes32,address)`	role admin (= DAR)
`0x36568abe`	`renounceRole(bytes32,address)`	self only

View, permissionless

Selector	Entry	Notes
`0x1a27f395`	`checkTrustlineStatus(uint8,address,uint256,bytes,address[])`	"advanced" — ValidationMode variant
`0x8ad79ae3`	`checkTrustlineStatus(address,uint256,bytes,address[])`	main
`0xc05e9b62`	`checkTrustlineStatus(address,uint256,bytes)`	no extras
`0xff7c9eff`	`requireTrustline(uint8,address,uint256,bytes,address[])`	**declared view; interface says non-view (H6)**
`0x6cfd0c09`	`requireTrustline(address,uint256,bytes,address[])`	same
`0x1b93c32f`	`requireTrustline(address,uint256,bytes)`	same
various	getters: `validationRegistry`, `sanctionsOracle`, `sanctionsOracleEnabled`, `trustlineOracleEnabled`, `AUTO_APPROVED_VALIDITY_PERIOD`, `MANUAL_APPROVED_VALIDITY_PERIOD`, `AUDITOR_ROLE`, `DEFAULT_ADMIN_ROLE`, `hasRole`, `getRoleAdmin`, `supportsInterface`, `proxiableUUID`	—

APPENDIX C: Public Entry Points (cont.)

Event surface

Event	Source	Gap
`ValidationEngineDeployed`	Trustlined deploy path	Not emitted on adopt path (I12)
`RoleGranted` / `RoleRevoked`	OZ AccessControl	—
`Upgraded`	OZ ERC1967	—
`Initialized`	OZ Initializable	—
(none)	`setValidationConfiguration`	**Silent oracle / enable-flag change (M2)**

Logic contract called directly (not via proxy)

The deployed `TrustlineSanctionsOnlyVE` logic contract runs `_disableInitializers()` in its constructor. Calling `initialize` directly on the logic reverts with `InvalidInitialization`. Other state-mutating functions called directly on the logic revert with `AccessControlUnauthorizedAccount` because no role bitmap is set in logic-context storage.

APPENDIX D: Storage Layout

Output of `forge inspect TrustlineSanctionsOnlyVE storage-layout`:

Slot	Name	Type	Bytes	Note
0	`validationRegistry`	address	20	set at init, **never read**
1 (offset 0)	`sanctionsOracle`	ISanctionsList	20	settable
1 (offset 20)	`trustlineOracleEnabled`	bool	1	**no setter; dead**
1 (offset 21)	`sanctionsOracleEnabled`	bool	1	settable
2	`AUTO_APPROVED_VALIDITY_PERIOD`	uint256	32	set to 30 min at init, **never read**
3	`MANUAL_APPROVED_VALIDITY_PERIOD`	uint256	32	set to 5 days at init, **never read**
4	`_txMetadata`	mapping(bytes32 => TxMetadata)	32	**never written nor read**
5..54	`__gap[50]`	uint256[50]	1600	reserved

Notes

- *`AccessControlUpgradeable` (OZ 5.x) uses ERC-7201 namespaced storage. Its state does **not** appear in this linear layout — no collision risk with the engine's own state.*
- *Five of seven engine state variables are dormant. The available `__gap` is reduced by five slots compared to a cleaner layout (see I1).*
- *Slot 0 holds `validationRegistry`. Future engine variants that need a different "slot 0 contract" must repurpose this slot via upgrade migration logic.*

APPENDIX E: Adversarial Scenarios

This appendix enumerates the worst-case actions available to each privileged role if it becomes actively malicious. Each entry maps to the underlying finding and to a PoC.

E.A — Malicious `DEFAULT_ADMIN_ROLE` holder

Attack	Finding	Detectability
Silently disable sanctions (no event)	M2 + admin power	Off-chain storage diff only
Swap oracle to permissive (no event)	M2 + admin power	Off-chain storage diff only
Upgrade to permissive logic	C1b	Upgraded event fires
Upgrade to state-leech logic	C1c	RoleGranted spam (eventually noticed)
Renounce, bricking config + upgrade	C1	RoleRevoked event fires
Grant DAR to accomplice silently	—	RoleGranted event fires

E.B — Malicious sanctions oracle

Attack	Finding	Detectability
Lie (return false for sanctioned)	—	Off-chain reconciliation vs authoritative list
Revert on every call	H3	High — every integrator's `requireTrustline` reverts immediately
Consume all gas	H3	Medium — surfaces as out-of-gas at integrator entry point

E.C — Malicious integrator developer

Deploys their contract, becomes DAR + AR by default. Has full A-level powers over their own VE proxy — but only over their own. Worst realistic attack: deploy with sanctions disabled from day 1. End users have no on-chain signal to verify enforcement is on.

E.D — Malicious end user (bypasses)

Attack	Finding
Route via clean relay wallet	M3
Use a fresh, not-yet-listed EOA	M3
Omit dirty addresses from `addresses[]`	M3
Gas-amplify via large `addresses[]`	M4

APPENDIX E: Adversarial Scenarios (cont.)

E.E — Supply-chain attacker

Attack	Finding
Phishing the integrator with a fake VE proxy address	H2
Phishing with fake logic + matching `initialize(address)`	H1
Front-running initialize on a not-yet-initialized proxy	H4

Compound attack chains

Chain G.1 — Insider-then-renounce (permanent permissive subversion)

1. *Compromised admin deploys `PermissiveVE`.*
2. *Admin upgrades the proxy to it (C1b).*
3. *Admin `renounceRole(DEFAULT_ADMIN_ROLE, self)` (C1).*
4. *The proxy is now permanently permissive.* ***No recovery is possible.***

On-chain trace: one `Upgraded`, one `RoleRevoked`. M2 ensures no event fires from any configuration change leading up to this.

Chain G.4 — State-leech via shared proxies

1. *Admin upgrades a shared proxy to `HijackVE` (C1c).*
2. *Every honest `requireTrustline` call by any integrator on that proxy silently grants admin to a hidden recipient.*
3. *By the time anyone notices the RoleGranted event flood, the recipient holds admin on every shared proxy.*

Chain G.5 — Adopt-pre-init hijack at scale

1. *Adversary deploys an `ERC1967Proxy(logic, "")` at a deterministic CREATE2 address.*
2. *Adversary calls `initialize(self)` first (H4).*
3. *Adversary publishes the proxy address as a "Trustline-deployed engine" via a copycat docs site.*
4. *Integrators adopt it via `Trustlined(address(0), proxyAddr)`. Adversary controls every dApp that bound to this proxy.*

APPENDIX F: Differential Review vs Upstream

The pinned `Trustlined.sol` (`967387`) is materially older than EVMSDK HEAD (`22e3e9d`). Several findings have been partially or fully addressed upstream. This annex catalogs what's already fixed and what remains.

Diff summary (pinned → HEAD)

```
+ import {IValidationEngineInitializer} from "./interfaces/IValidationEngineInitializer.sol";
  ...
  constructor(address logic, address proxy) {
-     __Trustlined_init(logic, proxy);
-     ... (3-layer init pattern removed)
+     if (proxy != address(0)) {
+         require(proxy.code.length > 0, "Proxy is not a contract");
+         _assertValidationEngine(proxy);
+         validationEngine = IValidationEngine(proxy);
+     } else {
+         require(logic.code.length > 0, "Logic is not a contract");
+         address initialOwner = msg.sender;
+         bytes memory data = abi.encodeCall(
+             IValidationEngineInitializer.initialize, (initialOwner)
+         );
+         address proxy_ = address(new ERC1967Proxy(logic, data));
+         _assertValidationEngine(proxy_);
+         validationEngine = IValidationEngine(proxy_);
+         emit ValidationEngineDeployed(...);
+     }
  }

+ function _assertValidationEngine(address candidate) private view {
+     IValidationEngine engine = IValidationEngine(candidate);
+     try engine.checkTrustlineStatus(address(this), uint256(0), bytes(""))
+         returns (bool) {}
+     catch { revert("Invalid validation engine"); }
+ }
```

APPENDIX F: Differential Review vs Upstream (cont.)

Status per finding

Finding	Pinned	HEAD	Notes
H1 (selector-string init)	`abi.encodeWithSignature`	`abi.encodeCall`	**Fixed at HEAD** by `596c425`. Recommend integrators upgrade to a release containing it (npm-vendored).
H2 (weak proxy validation)	`code.length > 0`	`code.length > 0` + `_assertValidationEngine`	**Partially mitigated** at `e6e7147`. Fallback-truthy contracts still bypass.
I12 (adopt-path event)	Absent	Still absent	Not fixed. Recommend `ValidationEngineAdopted` event.
H4 (adopt-uninit hijack)	No init check	Conformance check only	Not fully fixed. `_assertValidationEngine` doesn't verify admin state.
L2 (floating pragma)	`^0.8`	Unchanged	Not fixed.
C1, H6, M1, M2, M4, L1, L3, I1–I6	n/a	n/a	VE repo is at HEAD; no upstream remediation has occurred for any of these.

Other post-pin EVMSDK commits

Commit	Subject	Relevance
`ef5aee0`	Split upgradable and non-upgradable patterns	Adds `TrustlinedUpgradeable` (out of scope); flattens 3-layer init (I3).
`e6e7147`	Add additional checks to ensure passed addresses are actual VE	H2 partial mitigation.
`596c425`	Switch to stronger, type-checked ABI encoding	H1 fix; introduces `IValidationEngineInitializer.sol`.
`71430e7`	Add tests	Coverage; out of scope for code review.
`9a6d1c8`, `25afb2a`, `4e8c18f`	Example contract hardenings	Affect `PaymentFirewall.sol` only — out of scope.

Recommendation: *if the client is willing, re-pin the audit to commit `596c425` or later. This eliminates H1 entirely, downgrades H2 to a partial-mitigation finding, and lets the audit focus on residual unfixed issues (C1, H6, M2/I12, M4, etc.) which dominate the risk picture.*

Out-of-scope advisory — `TrustlinedUpgradeable` initializer hardening

Out of scope *The upgradeable integrator base `TrustlinedUpgradeable.sol` (introduced post-pin at `ef5aee0`) is **not in the contracted scope** and was not formally reviewed. During scoping, however, a concrete hardening gap was noted that we surface here so it is not lost; it warrants a dedicated scoped review (see below).*

Observation. *`TrustlinedUpgradeable` inherits `Initializable` and exposes `__Trustlined_init` (`onlyInitializing`), but its constructor does **not** call `_disableInitializers()`. Responsibility for locking the implementation is therefore pushed onto every integrator. An integrator that inherits `TrustlinedUpgradeable` and forgets to add `constructor() { _disableInitializers(); }` to its own implementation leaves that implementation directly initializable. An attacker can then call the integrator's `initialize` on the implementation address; for any UUPS-based integrator this enables an `upgradeToAndCall` into a `selfdestruct` payload, bricking every proxy that delegates to the implementation (and, depending on the integrator's own design, seizing implementation-context privileges). This is the canonical uninitialized-implementation hijack.*

Evidence it is real and known. *Trustline's own `TestTrustlinedUpgradeableClient` mock does call `_disableInitializers()` in its constructor — so the pattern is understood, but the library does not enforce it. Note this is a **different attack surface** from in-scope H4 (uninitialized VE proxy) and M1 (`initialOwner = msg.sender` capture): it lives in the integrator's own implementation contract.*

Compounding init-race (M1 × H4 on the upgradeable path). *`TrustlinedUpgradeable.__Trustlined_init` deploys the nested VE proxy inside the integrator's own initializer. On a classically front-runnable integrator proxy, whoever wins the init race becomes `msg.sender` and is captured as the VE `DEFAULT_ADMIN_ROLE` — M1 (deployer-becomes-admin) × H4 (init front-run) on the upgradeable client path, which the in-scope report covers only for the VE-proxy adopt path.*

Recommendation. *Either (a) bake `constructor() { _disableInitializers(); }` directly into `TrustlinedUpgradeable` so integrators inherit the protection by default, or (b) at minimum, place a prominent warning in the README and integration docs that every integrator implementation must call `_disableInitializers()`. Option (a) is strongly preferred.*

Scope note. *Per client direction, this is recorded as an advisory only; a full scoped review of `TrustlinedUpgradeable.sol` (with a takeover/brick PoC) is recommended as a follow-on engagement and can be priced separately.*

Out-of-scope advisory — `PaymentFirewall` example contract

Out of scope *`evmsdk/contracts/examples/PaymentFirewall.sol` is a shipped example integrator, not in the contracted scope. It is reviewed informally here only because both stakeholder threat models (Appendix J/K) cite it and integrators are likely to copy it.*

Verified *`PaymentFirewall` inherits `ReentrancyGuard` and marks both `payEthers`/`payTokens` `nonReentrant`, uses `SafeERC20.safeTransferFrom` (checked transfer), and screens both `msg.sender` and the `destination`.*

One gap (instance of M3). *`payTokens(destination, token, value)` screens `destination` but **not the `token` address**. Whether a token contract should itself be sanctions-screened is an edge case (OFAC does designate contract addresses, e.g. Tornado Cash), but this is a concrete illustration of M3 — the integrator chooses the screened set and here omits a value-relevant address. An integrator copying this template inherits that choice silently.*

Recommendation. *If the integrator's obligations extend to designated token/contract addresses, add `token` to the screened set. Treat `PaymentFirewall` as illustrative, not a complete compliance template.*

Out-of-scope advisory — "wrong base class" storage / binding

Out of scope Complements the `TrustlinedUpgradeable` advisory: the cost of picking the wrong base. The non-upgradeable `Trustlined` binds `validationEngine` in its **constructor** and has **no storage gap**. An integrator who deploys `Trustlined` (instead of `TrustlinedUpgradeable`) behind a proxy gets: (a) the constructor never runs in proxy context, so `validationEngine` silently stays `address(0)` and every guarded call reverts / mis-behaves; and (b) no gap, so a future SDK version adding storage to `Trustlined` collides with integrator state. Behind a proxy, `validationEngine()==address(0)` and the guarded call reverts. **Recommendation:** document base-class selection explicitly — `Trustlined` for constructor-based (non-proxy) deployments, `TrustlinedUpgradeable` (initializer + gap) for proxies; consider a deploy-time guard that asserts `validationEngine != address(0)`.

Out-of-scope advisory — adopt-path probe is a constructor-time call to untrusted code

Out of scope Applies to HEAD / `TrustlinedUpgradeable`, **not** the pinned in-scope `Trustlined` (whose adopt path makes no external call — it only checks `proxy.code.length > 0` then stores). On HEAD, `_assertValidationEngine` makes an external call to an attacker-supplied address during the integrator's construction. It is a `staticcall` today (so mutation/reentrancy are bounded), but it is a deploy-time gas-griefing vector, and any future change making the probe non-view — or any integrator constructor with observable side effects — turns it exploitable. Worth flagging that the SDK hands control flow to untrusted code mid-construction.

Out-of-scope advisory — frontend bundle (`dist/bundle.js`) is an unaudited trust surface

Out of scope `build.js` emits a browserify bundle (`dist/bundle.js`, `standalone: 'Trustlined'`) for frontend consumption — the product is a Web-SDK-adjacent JS package. The bundle is what a dApp UI uses to **encode calls and render "compliant" status**, so it is a genuine trust surface entirely outside the contract threat model: a tampered or skewed bundle can mis-encode calls (selector skew → M11d) or fake a compliant UI, with users none the wiser. It is JavaScript and out of scope for this Solidity audit, but it ships in the same npm package as the audited contracts and inherits M11's provenance gap (gitignored, publish-time-generated, no SRI). **Recommendation:** treat it as its own review target — JS SAST, Subresource-Integrity hashes + CSP in the consuming UI, pinned/verified bundle hash, JS dependency scanning, and confirm the bundle's embedded ABI/selectors match the audited interface. Needs a JS-side review, not a Solidity audit.

APPENDIX G: Cross-Configuration Differential Testing

To support and prove findings, Fidesium built 88 Foundry PoCs

Configuration	solc	EVM	Result	Key gas (bomb / grief / M4×200)
Audited reproducer	`0.8.34`	osaka	88 / 88	160,318 / 608,937 / 767,991
EVM-version cross-check	`0.8.34`	cancun	88 / 88	identical
Shipped-SDK build (per M11)	`0.8.28`	cancun	88 / 88	identical

Result. *All three configurations pass 88/88 with* ***byte-identical gas*** *for every gas-sensitive finding (H3 return-data bomb 9,443→160,318; the EIP-150 63/64 gas-grief at 608,937; M4 unbounded-array linear cost 9,964→767,991). The Fusaka/`osaka` changes (EIP-7702, blob parameters, new precompiles) do not touch the opcodes these vectors exercise (memory-expansion / returndata copy, loop `SLOAD`) so the gas-grief thresholds are* ***not version-sensitive*** *between these forks, and the findings reproduce identically on the shipped build.*

APPENDIX H: Positive Observations

Items where the codebase is doing the right thing, called out so they remain protected through future changes.

P1 — Atomic initialization on deploy-fresh path

`Trustlined` *embeds the* `initialize(initialOwner)` *bytes in the* `ERC1967Proxy` *constructor, so init runs atomically with deployment. A front-runner cannot seize admin during the deployment transaction.*

P2 — Logic contract locked

`TrustlineSanctionsOnlyVE`*'s constructor calls* `_disableInitializers()`*. Direct calls to* `initialize` *on the logic address revert.*

P3 — Correct use of ERC-7201 for AccessControl

`AccessControlUpgradeable` *(OZ 5.x) uses namespaced storage. The engine's linear storage layout (slots 0–4) does not collide with AccessControl. Storage layout inspection in Appendix D corroborates.*

P4 — Storage gap declared

`uint256[50] private __gap;` *reserves 50 slots after the engine's state for future descendants.*

P5 — Clean separation of concerns

`Trustlined` *(integrator-side) is intentionally minimal — it only manages the engine reference and forwards* `msg.sender`/`msg.value`/`msg.data`*. Heavy logic lives in the VE side, where it can be upgraded without redeploying integrators.*

P6 — UUPS over Transparent proxy

The choice of UUPS (vs. Transparent) avoids the well-known function-selector collision class. `_authorizeUpgrade` *is correctly defined.*

P7 — Both proxy-deploy paths are covered

Integrators have the choice of fresh deployment (per-integrator engine) or adopting a shared one. Both paths compile and run; the safety properties differ and are documented as findings (H2/H4).

P8 — Compiles cleanly under `solc 0.8.34` with `evm_version = "osaka"`

No warnings from the in-scope files at the pinned configuration. The audit reproducer uses `evm_version = "osaka"`*, matching the in-scope VE's own foundry config; the SDK half, by contrast, builds* `cancun` *(see M11).*

P9 — Oracle call is correctly bounded by STATICCALL

`_isSanctionsApproved` *is* `view`*, so the external* `sanctionsOracle.isSanctioned(...)` *call compiles to* `STATICCALL`*. A malicious oracle therefore cannot mutate state or re-enter — the oracle-adversary findings (H3/M7/M8/M10) are correctly bounded to revert / gas / lie / return-bomb / staleness, with no reentrancy or state-corruption vector. The report's oracle threat model relies on this and it holds.*

P10 — VE self-reentrancy is moot (mutators are admin-gated)

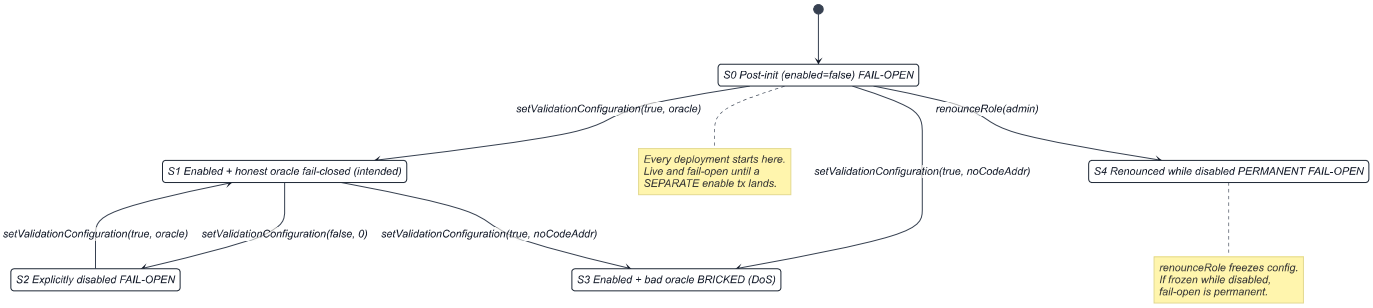
Every state-mutating VE entry point (`setValidationConfiguration`*,* `upgradeTo*`*,* `grant/revoke/renounceRole`*) is permissioned to* `DEFAULT_ADMIN_ROLE` *(or self, for renounce). There is no unprivileged state-changing path on the VE for an external party to re-enter, so VE-side reentrancy is not a concern (the reentrancy surface that does matter, H6, is on the integrator side through* `requireTrustline`*, not here).*

P11 — No selector collision across the proxy surface

The IValidationEngine overloads, AccessControl/UUPS functions, and auto-getters have pairwise-distinct 4-byte selectors. UUPS also rules out the Transparent-proxy admin/selector-shadowing class (P6); the "no two reachable selectors collide" property is therefore explicit.

APPENDIX I: Configuration State Machine

*Systematic enumeration of the reachable configuration triple `(sanctionsOracleEnabled, sanctionsOracle, admin)` from deploy → init → configure → upgrade, classifying each state as fail-closed (intended), **fail-open**, or bricked.*
***Conclusion: there is no on-chain invariant forcing "screening enabled before the engine is usable."** The engine is fully usable while screening is off, and the default state (S0) is fail-open.*



State table

State	enabled	oracle	admin	Sanctioned actor	Class
S0 Post-init	false	0	set	**passes**	FAIL-OPEN
S1 Enabled + honest oracle	true	valid	set	blocked	fail-closed
S2 Explicitly disabled	false	0	set	**passes**	FAIL-OPEN
S3 Enabled + bad oracle	true	no-code	set	all revert	BRICKED (M7)
S4 Renounced while disabled	false	0	none	**passes**	PERMANENT FAIL-OPEN

How long does a real deployment sit fail-open?

- ***Minimum one block.*** *Enabling is never atomic with init: the integrator's `Trustlined` constructor runs `new ERC1967Proxy(...)` → `initialize` (which sets `enabled = false`), and `setValidationConfiguration` is a strictly later call. The proxy is live and fail-open in between.*
- ***In practice, longer or indefinite.*** *The reference `DeploySanctionsOnlyValidationEngine.s.sol` deploys the **logic contract only** — no proxy, no `initialize`, no `setValidationConfiguration` — so the supplied tooling provides integrators with no enabling step or guidance at all; nothing moves them off S0. Enabling also requires an oracle in the same call (`setValidationConfiguration(true, address(0))` reverts), so reaching S1 presupposes a deployed oracle.*

Recommendation

Make S0 unreachable as a usable state: require a non-zero oracle and `enabled = true` at `initialize` (fail-closed construction), or have `requireTrustline` revert until a configuration has been explicitly set. Ship a deploy/bootstrap script that creates the proxy, enables screening with a real oracle, and asserts `sanctionsOracleEnabled()` post-deploy. (Ties to H5, M7, M2.)

APPENDIX J: Threat Model — Maintainer (Trustline)

*Stakeholder-oriented threat model for the party that owns the in-scope code and its upgrade/admin authority. Every threat maps to an in-scope finding. Components that cross a trust boundary into **out-of-scope** code (the oracle backend, the npm release pipeline, `TrustlinedUpgradeable`, the example contracts) are noted here but detailed under the **Out-of-Scope Advisories** (Appendix F).*

***In-scope assets:** integrity of the validation decision (`SanctionsOnlyValidationEngine`); admin/upgrade authority over `TrustlineSanctionsOnlyVE`; the source build configuration (pragma, `evm_version`); the hardcoded registry constant. **In-scope trust boundaries:** proxy↔logic upgrade path, on-chain admin-role gating, storage layout.*

***Non-attacker stakeholder — the regulator / compliance auditor.** Not an adversary, but the party whose inspection determines whether a fail-open or incompletely-screened deployment becomes a liability event. The maintainer's obligation is to make screening state on-chain-verifiable so an auditor can confirm it.*

#	Threat	Vector	In-scope finding(s)	Mitigation direction
A1	Fail-open ships as the default — auto-deployed engines approve everything	`sanctionsOracleEnabled=false` post-init; no forcing function	**H5** (Appendix I / S0)	Make the safe state the default: require an oracle at init, or revert guarded calls while unconfigured; emit a loud `ScreeningDisabled` signal.
A2	Admin/upgrade key is a single point of total failure	UUPS `_authorizeUpgrade` gated only by a role; key theft / insider	**C1**	Multisig + timelock in deployment guidance; ideally an in-code minimum-delay upgrade; publish the admin custody model.
A3	Engine-address validation gives false assurance	Weak proxy/logic check (`code.length` at the pinned commit); misconfig / social-engineered address	**H2**	Strengthen to codehash / registry-membership / EIP-165 + signed attestation. (The HEAD `_assertValidationEngine` conformance variant lives in out-of-scope `TrustlinedUpgradeable` — see Appendix F.)
A4	Ignored interface params imply guarantees not delivered	Spec drift: `mode`/`value`/`data`, validity periods, registry all dead in this engine	**M6, I1, I2**	Implement or remove from this engine's surface; document "sanctions-only" scope explicitly per method.
A5	Oracle is an unbounded external call with no isolation	Malicious / reverting / censoring oracle	**H3, M7, M8, M4**	Decide and document fail-open vs fail-closed; `try/catch` with a defined policy; bound `addresses[]`; conformance-check the oracle. (Oracle backend itself is out-of-scope — Appendix F.)
A5b	Oracle answer has no freshness gate	Stale / frozen feed; update-lag; cross-chain divergence	**M10**	Add `lastUpdated()`/heartbeat to `ISanctionsList`; enforce max-staleness, fail-closed past the threshold. Off-chain feed poisoning is OOS (Appendix F).
A6	Supply-chain blast radius	Caret deps + floating pragma (in-scope build config); prebuilt `dist/`+`artifacts/` + npm pipeline (out-of-scope ops)	**M11, L2**	Pin solc pragma; pin deps exactly; publish with provenance/SLSA; reproducible builds. Pipeline/artifact specifics under Appendix F.
A7	Cross-chain assumptions	`evm_version="osaka"` (build config); registry constant assumed identical across chains	**M11, I2**	Per-chain compatibility matrix; verify the registry address per chain; lower the EVM target to the floor actually needed.
A8	Storage-layout regression on upgrade	Future engine reorders storage; dead vars + `__gap` must stay stable	**I1** (Appendix D/I)	Layout-snapshot test in CI (`slither-check-upgradeability`); document the canonical layout; keep the abstract/concrete split disciplined.

APPENDIX K: Threat Model — Integrator

Stakeholder-oriented threat model for a team adopting the in-scope SDK. The integrator's own contract is out of scope, but each threat below is driven by in-scope SDK behavior and maps to an in-scope finding. Out-of-scope referents (the `PaymentFirewall` example, `TrustlinedUpgradeable`) are detailed under Appendix F.

Integrator assets: *their users' funds and the integrator's own compliance/legal standing; their contract's availability; their admin keys (if they auto-deployed an engine they now operate).* ***What they implicitly — and wrongly — trust:*** *that "wired in Trustlined" means transactions are screened; that the engine is the real Trustline one; that the oracle is honest and live; that `requireTrustline` is a harmless internal check.*

#	Threat	Vector	Finding(s)	What the integrator must do
B1	They ship an inert guard	Fail-open default (A1); shared engine whose config they don't control	**H5**	Assert on-chain that `sanctionsOracleEnabled()==true` and `sanctionsOracle()!=0` before trusting; monitor config; for shared engines get observable/contractual guarantees.
B2	They unknowingly became the engine admin	`Trustlined(logic, address(0))` auto-deploy	**M1**	Treat the engine admin as a critical key: transfer to multisig+timelock immediately; never deploy from a hot key.
B3	Incomplete `addresses[]` lets a sanctioned counterparty through	`requireTrustline()` (sender-only) on a fund flow to an arbitrary recipient; omitting recipients/tokens/intermediaries	**M3**	Enumerate every value-bearing address per fund-flow and pass them all; codify per-function.
B4	`msg.sender` evasion	Meta-tx, ERC-4337, router/multicall — screened address ≠ economic beneficiary	**M3**	Decide whether to screen the beneficiary, not just the caller; pass the real user into `addresses[]`; restrict entry via untrusted routers.
B5	Wrong/malicious engine accepted at config time	Bad constructor arg, social engineering, typo	**H2, H4**	Independently verify the engine address against a trusted Trustline publication; pin it; review on deploy.
B6	Availability — a reverting oracle bricks guarded functions	Oracle outage/attack/swap; no `try/catch`	**H3**	Understand this is fail-closed-by-revert (all guarded paths die); have an incident plan; for shared engines, know who can swap the oracle.
B6b	Stale oracle passes the now-listed	Frozen feed / update lag	**M10**	You cannot self-check freshness today (interface exposes no timestamp); monitor the oracle's update heartbeat off-chain until the interface gains one.
B7	Reentrancy / hostile callback via `requireTrustline`	It is a CALL to an upgradeable contract from inside a state-changing function; future/compromised logic reenters	**H6**	Treat the VE call as untrusted external code: CEI + `nonReentrant`; don't call `requireTrustline` then make unguarded external calls. (The out-of-scope `PaymentFirewall` example demonstrates the guarded pattern — see Appendix F.)
B8	Upgrade rug / silent policy change on a shared engine	Shared-engine governance; someone else's admin upgrades or flips config	**C1, M2, M12**	Prefer a dedicated engine you govern, or accept and disclose the third-party trust; monitor `Upgraded`/config events.
B9	`checkTrustlineStatus` used without enforcing the bool	Using the view variant for UI only, or discarding the return	**M5**	Use `requireTrustline` for enforcement paths; never let a `false` be advisory on a fund path.
B10	Gas-griefing via large `addresses[]`	Per-address external call in a loop; user-supplied unbounded array	**M4**	Bound array length; don't let untrusted callers inflate it.

Integrator self-audit method (per guarded function)

A concrete procedure an integrator should run against its own entrypoints before mainnet — each step maps to the threats above:

1. ***Draw the fund flow** for the guarded function and **list every address that can receive value** (recipients, payees, token contracts, intermediaries); confirm all are passed in `addresses[]`. (B3 / M3)*
2. ***Confirm the engine address** is the verified Trustline one, and is **configured and enabled** — assert `sanctionsOracleEnabled()==true` and `sanctionsOracle()!=0` on-chain. (B1/B5 / H5, H2)*
3. ***Confirm engine-admin custody** — who holds `DEFAULT_ADMIN_ROLE`? It must be a multisig+timelock you control, not the deploying hot key. (B2 / M1)*
4. ***Run the relayer/AA bypass test** against your own entrypoints — route a known-listed actor through a router/4337 wallet and confirm your screening still catches the beneficiary. (B4 / M3)*
5. ***Confirm reentrancy protection** around every `requireTrustline` that precedes an external call — CEI + `nonReentrant`. (B7 / H6)*

The same procedure appears in `05-integrator-risk.md §5` (pre-launch checklist).

APPENDIX L: Finding × Surface Coverage Matrix

Surface	Findings	What to watch
Admin / governance & keys	C1, M1, M2, L1, M16	Single-key DAR, no in-code timelock, silent config changes; deployer and oracle as separate privileged domains (M1)
Proxy / upgrade safety	C1, H6, I1, I4	UUPS upgrade authority; storage-layout stability; `newImpl` unvalidated; non-view `requireTrustline`
Initialization / deployment	H1, H2, H4, H5, M1, I3, I6, L4, M13, M14	Selector encoding, adopt-path validation, fail-open default; wrong-base-class behind a proxy (OOS advisory)
Oracle (external dependency)	H3, M7, M8, M10, M15	DoS / no-conformance / censorship / staleness; the oracle is a second admin domain; off-chain feed poisoning (OOS)
Screening semantics & completeness	M3, M4, M6, M9, L5	Who/what is screened — evasion *and* the mirror, forced-reversion injection DoS (M3f); ignored mode/value/data; relay/AA/composition
Integrator API usage	H6, M5, M9	Ignored `checkTrustlineStatus` return; the gate as a reentrancy vector; no VE rotation; screen-vs-settle TOCTOU; mutability mismatch
Surface	Findings	What to watch
Observability & events	M2, I12, L3, I8, I9	Silent config/adopt changes; untyped revert reasons; no typed signal for monitors or fail-open-vs-closed branching
Supply chain / build / distribution	M11, L2	Audited source ≠ shipped artifact; no lockfile/provenance; mismatched toolchains; floating pragma; frontend bundle (OOS)
Privacy / recon	I7, M12, M15, I11	Permissionless enumeration of the sanctions list via the VE's unauthenticated view functions
Dormant / latent code	I1, I2, L1	Dead state variables; hardcoded cross-chain registry constant; unused `AUDITOR_ROLE` (latent privilege on upgrade)
Process / scope	I5, I10	Scope inconsistency (`IValidationEngineInitializer.sol` absent at the pinned commit)

Risk-bearer concentration: *Admin/governance and Proxy/upgrade surfaces are Operator → all integrators (a single compromise subverts everyone on a shared proxy). Screening-semantics and Integrator-API surfaces are Integrator / End-user (each integrator owns its own exposure). Oracle is a shared second domain. Supply-chain spans Operator (publish) and Integrator (consume). See per-finding Risk-Bearer tags.*

Disclaimer

Disclaimer

This report is governed by the Fidesium terms and conditions.

This report does not constitute an endorsement or disapproval of any project or team, nor does it reflect the economic value or potential of any related product or asset. It is not investment advice and should not be used as the basis for investment decisions. Instead, this report provides an assessment intended to improve code quality and mitigate risks inherent in cryptographic tokens and blockchain technology.

Fidesium does not guarantee the absence of bugs or vulnerabilities in the technology assessed, nor does it comment on the business practices, models, or regulatory compliance of its creators. All services, reports, and materials are provided "as is" and "as available," without warranties of any kind, including but not limited to merchantability, fitness for a particular purpose, or non-infringement.

Cryptographic assets and blockchain technologies are novel and carry inherent technical risks, uncertainties, and the possibility of unpredictable outcomes. Assessment results may contain inaccuracies or depend on third-party systems, and reliance on them is solely at the Customer's risk.

Fidesium assumes no liability for content inaccuracies, personal injuries, property damages, or losses related to the use of its services, reports, or materials. Third-party components are provided "as is," and any warranties are strictly between the Customer and the third-party provider.

These services and materials are intended solely for the Customer's use and benefit. No third party or their representatives may claim rights to or rely on these services, reports, or materials under any circumstances.