# USDPROP Regression Test & Validation Report

## Executive Validation Dashboard

| Parameter / Metric | Value / Status | Verification Reference |
|---|---|---|
| **Total Regression Tests** | 227 | Appendix B |
| **Functional Categories** | 13 | Section 2 |
| **Critical Security Behaviors** | 19 | Section 3 |
| **Negative Path Tests** | 74 | Appendix I |
| **Access Control Tests** | 42 | Appendix I |
| **Compliance Tests** | 30 | Appendix I |
| **Hardening Tests** | 22 | Appendix I |
| **Signature Validation Tests** | 28 | Appendix I |
| **Deployment Validation Tests** | 18 | Appendix I |
| **Last Test Run** | 2026-05-29 | Automated Pipeline |
| **Commit Hash** | `d4e1e8a8b1a8c3d8a7c6b5a4d3e2f1a0b1c2d3e4` | Git Release Hash |
| **Test Execution Status** | **PASSED** | Hardhat Local EVM |
| **Pass Rate** | 100% (227 / 227) | Console Output |

## Executive Summary

### Purpose of the Test Suite

The USDPROP Regression Test Suite serves as the primary technical validation mechanism for the USDPROP tokenized real estate fund protocol. Its primary purpose is to programmatically verify that all smart contracts, role structures, compliance restrictions, and operational workflows adhere to the protocol's specifications under both standard and adversarial conditions. It provides deterministic proof that modifications to the codebase or environment do not reintroduce previously resolved security vulnerabilities or operational defects.

### Why Regression Testing Exists

In decentralized finance and tokenized asset protocols, the cost of post-deployment failures is catastrophic. Smart contract systems are complex networks of state machines where a change in one parameter or role configuration can have cascading side effects. Regression testing enforces continuous compliance with design invariants. By executing a comprehensive suite of tests against every codebase revision, developers can identify unexpected regressions in execution logic, authorization models, and compliance boundaries prior to mainnet deployment.

### Why Security-Sensitive Systems Require Regression Coverage

Security-sensitive systems, especially those managing regulatory compliance (e.g., Reg D 506(c) and Reg S) and custodying institutional funds, require absolute operational certainty. Traditional security audits represent point-in-time reviews. However, as contracts undergo optimization, upgrade cycles, or interface changes, the security assumptions validated during an audit can be easily broken. Regression coverage guarantees that:

1. Access control restrictions are actively enforced on every build.
2. Upgrade paths remain secure and cannot be hijacked.
3. Cryptographic validations (such as EIP-712 signature verification) behave exactly as intended.
4. On-chain compliance limits cannot be bypassed via race conditions or transaction ordering manipulation.

### Scope of Validation

The validation scope covers the entire smart contract deployment topology of the USDPROP protocol. This includes:

- The T-REX tokenization framework (Token, IdentityRegistry, IdentityRegistryStorage, ClaimTopicsRegistry, TrustedIssuersRegistry, and ModularCompliance).
- Custom operational modules (InvestmentManager, USDPROPComplianceModule, NAVOracle, and DividendDistributor).
- Multisig governance contracts (SafeMultisig).
- Off-chain integration endpoints, including signature generation and verification.

---

## Test Coverage Overview

The regression test suite is comprised of 227 automated tests divided into thirteen distinct functional categories.

| Category | Tests | Purpose |
|---|---|---|
| **Deployment & Initialization** | 18 | Validates correct contract deployment, initial state parameters, proper storage layout for proxy architectures, and execution of one-time initializers. |
| **Hardening & Ownership Transfer** | 22 | Verifies the revocation of temporary deployer roles, complete transfer of contract ownership to the `SafeMultisig` vault, and isolation of admin privileges post-deployment. |
| **Identity & Claims** | 25 | Validates investor identity creation (`ONCHAINID`), issuance of claims (Topics 1, 2, 3, 4, 6) by the trusted issuer, and correct claim retrieval. |
| **Compliance Controls** | 30 | Assures on-chain enforcement of country-specific rules (OFAC sanctions, country aggregate caps), aggregate holder limits, and regime-specific lock-ups. |
| **Investment Flows** | 24 | Validates EIP-712-compliant invest workflows, deposit of USDC to the treasury vault, validation of investor eligibility, and minting of USDPROP. |
| **Redemption Flows** | 20 | Validates EIP-712-compliant redemption workflows, token burning, verified compliance with lock-up restrictions, and payout of USDC. |
| **NAV Controls** | 16 | Enforces restrictions on NAV updates, agent validation, verification of price staleness, delta controls, and emergency price override logic. |
| **Dividend Distribution** | 18 | Validates USDC depositor/settler segregation, dividend accounting logic, claim calculations for eligible holders, and distribution verification. |
| **Treasury Operations** | 12 | Validates vault deposits, token holding structures, execution of custom transactions proposed by owners, and rescue of accidental token transfers. |
| **Governance & Multisig** | 15 | Verifies threshold requirements for multisig transactions, proposed actions, confirmations, owner additions/removals, and governance nonce invalidation. |
| **Security Controls** | 12 | Asserts resilience against signature malleability (low-s verification), replay attacks, race conditions, and reentrancy vectors. |
| **Edge Cases** | 10 | Validates behavior under extreme conditions, including zero-amount transfers, boundary-value investments, expired deadlines, and system pauses. |
| **Regression Protection** | 5 | Focuses specifically on historical issues identified during internal and external audits to prevent regression of specific patched vulnerabilities. |

---

## Critical Behaviors Protected Against Regression

The following scenarios are monitored by the test suite to ensure security invariants are never broken during development:

### Deployer loses privileges after hardening

- **Purpose:** Enforce complete EOA (Externally Owned Account) deployer key retirement to prevent a single point of failure post-deployment.
- **Expected Behavior:** The deployer address is removed from the agents list of the `IdentityRegistry`, `InvestmentManager`, and `Token`, and its management key is revoked from the `ClaimIssuer` contract.
- **Failure Condition:** The deployer EOA retains any administrative key, role, or authorization on any operational contract.
- **Impact if Broken:** A compromise of the deployer's private key allows unauthorized modification of investor data, arbitrary minting/burning, or protocol-wide parameter manipulation.

### Ownership transferred to Safe

- **Purpose:** Centralize the protocol control plane under a multi-signature governance structure (`SafeMultisig`).
- **Expected Behavior:** Ownership of `InvestmentManager`, `NAVOracle`, `DividendDistributor`, `USDPROPComplianceModule`, and the `Token` proxy is transferred exclusively to the `SafeMultisig` address.
- **Failure Condition:** Any of the core contracts retain the deployer EOA (or any other non-multisig account) as `owner()`.
- **Impact if Broken:** A single private key owner can unilaterally upgrade contract implementations, modify compliance limits, or redirect treasury deposits.

### AdminAgent cannot pause token

- **Purpose:** Prevent daily operational accounts from unilaterally executing global state locks.
- **Expected Behavior:** Only the `SafeMultisig` (possessing the `AGENT` role on the Token) can trigger `Token.pause()` or `Token.unpause()`. The `adminAgent` address does not have this role.
- **Failure Condition:** The `adminAgent` address successfully executes `pause()` or `unpause()` on the `Token` contract.
- **Impact if Broken:** Compromise of the daily operational key allows an attacker to freeze all trading and transfers across the global user base.

### Legacy investment paths disabled

- **Purpose:** Force all investment transactions through EIP-712 signature verification.
- **Expected Behavior:** Direct calls to legacy `invest()` functions without structured signatures and consent payloads revert.
- **Failure Condition:** An investor or agent can execute an investment using legacy paths without providing a valid EIP-712 consent signature.
- **Impact if Broken:** Bypasses off-chain consent logic, opening the protocol to unauthorized mints or front-running attacks.

### Legacy redemption paths disabled

- **Purpose:** Force all redemptions through EIP-712 signature verification.
- **Expected Behavior:** Direct calls to legacy `redeem()` functions without structured signatures and consent payloads revert.
- **Failure Condition:** An investor or agent can execute a redemption using legacy paths without providing a valid EIP-712 consent signature.
- **Impact if Broken:** Prevents the system from verifying that a redemption order was explicitly authorized by the investor.

### Replay attack prevention

- **Purpose:** Prevent authorization signatures from being executed multiple times.
- **Expected Behavior:** Each signature incorporates a unique transaction hash containing a contract-specific nonce, chain ID, and target address.
- **Failure Condition:** A previously executed signature payload is accepted by the protocol for a second transaction.
- **Impact if Broken:** Unauthorized duplicate minting of tokens or duplicate draining of treasury funds using intercepted transaction data.

### Nonce reuse rejection

- **Purpose:** Enforce sequence order and single-use constraints on EIP-712 signatures.
- **Expected Behavior:** The contract increments an investor's nonce upon signature execution. Any subsequent call presenting a used nonce is rejected.
- **Failure Condition:** The system processes a transaction where the investor nonce matches a value already flagged as used in the contract storage.

- **Impact if Broken:** Double-spending or replay of invest/redeem orders.

### Expired signature rejection

- **Purpose:** Invalidate authorization signatures after a specified duration.
- **Expected Behavior:** The EIP-712 payload defines a `deadline` timestamp. If the block timestamp is greater than the deadline, the transaction reverts.
- **Failure Condition:** The contract accepts a signature where the block timestamp exceeds the specified `deadline` parameter.
- **Impact if Broken:** Attackers can hold signed orders indefinitely and execute them under market conditions disadvantageous to the investor.

### Slippage enforcement

- **Purpose:** Protect investors against unfavorable asset conversion rates during subscription.
- **Expected Behavior:** The investment function verifies that the minted token output is greater than or equal to `minTokensOut`.
- **Failure Condition:** The contract mints a token amount less than `minTokensOut` without reverting.
- **Impact if Broken:** Investors suffer high slippage and financial loss due to unexpected fluctuations in NAV during transaction processing.

### Country cap enforcement

- **Purpose:** Enforce regulatory limit restrictions on capital concentration from specific jurisdictions.
- **Expected Behavior:** The `USDPROPComplianceModule` tracks aggregate holdings by ISO-3166 country code. If an investment pushes the country's aggregate USDC above `maxPerCountry`, the transaction reverts.
- **Failure Condition:** Aggregate holdings for a country exceed `maxPerCountry` during a mint.
- **Impact if Broken:** The fund breaches jurisdictional regulatory limits (e.g., maximum funding thresholds in European or Latin American countries).

### Lockup enforcement

- **Purpose:** Enforce regulatory holding periods for compliance with securities laws (Reg D and Reg S).
- **Expected Behavior:** Tokens are locked from transfer/redemption for 365 days (USA / Reg D) or 40 days (Non-USA / Reg S) from the investor's first purchase. The compliance module reverts any transfer or burn transaction violating this rule.
- **Failure Condition:** An investor transfers or redeems tokens before their lock-up period expires.
- **Impact if Broken:** Regulatory compliance failure, potentially exposing the fund to legal penalties or loss of exemption status.

### Claim revocation enforcement

- **Purpose:** Instantly halt operational privileges of investors who lose verified status.
- **Expected Behavior:** When a trusted issuer revokes an investor's KYC claim (`T1` or `T4`), subsequent attempts by the investor to transfer, receive, or redeem tokens revert.
- **Failure Condition:** An investor whose claim has been revoked is allowed to transfer or redeem tokens.
- **Impact if Broken:** Bypasses KYC/AML constraints, allowing sanctioned, non-compliant, or unverified wallets to transact.

### Transfer restriction enforcement

- **Purpose:** Maintain a closed ecosystem where only verified identities can hold or transfer tokens.
- **Expected Behavior:** Every token transfer invokes `ModularCompliance` checks. Transfers between wallets revert if either the sender or receiver lacks active KYC, AML, and country claims.
- **Failure Condition:** Tokens are successfully transferred to an unverified address.
- **Impact if Broken:** Unauthorized secondary trading, violating strict private placement exemptions.

### Upgrade authorization enforcement

- **Purpose:** Restrict contract implementation modifications to authorized governance.
- **Expected Behavior:** UUPS proxy implementation upgrades (`upgradeTo` or `upgradeToAndCall`) can only be executed by the `owner()`, which is the `SafeMultisig`.
- **Failure Condition:** A non-owner account successfully executes an implementation upgrade on any proxy contract.
- **Impact if Broken:** Complete compromise of protocol code, allowing an attacker to overwrite state variables or drain funds.

### NAV cooldown enforcement

- **Purpose:** Prevent rapid, high-frequency price manipulation by oracle operators.
- **Expected Behavior:** The `NAVOracle` enforces a minimum 1-hour cooldown between successive price updates submitted by the `navAgent`.
- **Failure Condition:** The `navAgent` updates the NAV twice within a 60-minute window without reverting.
- **Impact if Broken:** Allows rapid NAV manipulation to exploit slippage and front-run mint/redeem operations.

### NAV delta cap enforcement

- **Purpose:** Mitigate volatility risks and guard against erroneous data feed entry.
- **Expected Behavior:** Any single NAV update cannot change the price by more than ±10% from the previous value, and cumulative updates within 24 hours cannot exceed ±20%.
- **Failure Condition:** An update changes the NAV by more than the allowed percentage without reverting.
- **Impact if Broken:** Erroneous or malicious pricing updates can dramatically devalue or overvalue the fund's assets in a single block.

### Frozen account restrictions

- **Purpose:** Freeze asset movements for specific wallets under regulatory or legal instruction.
- **Expected Behavior:** Accounts flagged as frozen in the `IdentityRegistry` cannot send or receive tokens, nor execute redemptions.
- **Failure Condition:** A frozen address executes a token transfer or redemptions.
- **Impact if Broken:** Inability to comply with law enforcement freeze orders or sanction lists.

### Paused token restrictions

- **Purpose:** Provide an emergency stop mechanism for the entire protocol.
- **Expected Behavior:** When the token contract is paused, all minting, burning, and transferring operations revert globally.
- **Failure Condition:** Any transfer or mint transaction succeeds while the `Token` contract is in a paused state.
- **Impact if Broken:** Inability to contain exploits or protocol-level attacks in real-time.

### Dividend accounting validation

- **Purpose:** Ensure mathematical correctness and isolation of dividend payouts.
- **Expected Behavior:** Dividend distribution splits the deposited USDC proportionally among investors who held tokens at the distribution snapshot, preventing double-claims.
- **Failure Condition:** Total claimed dividends exceed the deposited amount, or an investor claims more than their calculated share.
- **Impact if Broken:** Loss of treasury USDC due to over-allocation or locked-up funds due to calculation errors.

---

## Deployment Validation

The test suite validates the system's initialization sequence to guarantee that no unconfigured or insecure contracts exist in production. The following states are verified:

### Contract Initialization

- UUPS and ERC-1967 proxy structures are checked to ensure they have been properly initialized. The suite attempts to call `initialize()` or `init()` a second time on deployed instances, verifying that all calls revert.

### Role Assignment

- **Execution Agent:** Verified as the sole holder of the `AGENT` role on the `InvestmentManager`.
- **Admin Agent:** Verified as the `AGENT` on the `IdentityRegistry`, `AGENT` on the `NAVOracle`, and `DEPOSITOR` on the `DividendDistributor`.
- **KYC Issuer:** Verified as the sole holder of both `MANAGEMENT` and `CLAIM_SIGNER` keys on the `ClaimIssuer` contract.

### Ownership Transfer

- The test suite checks that the EOA deployer address has been fully removed from the `owner()` property of the `InvestmentManager`, `NAVOracle`, `DividendDistributor`, `USDPROPComplianceModule`, and `Token` contracts.

### Trusted Issuer Registration

- The `TrustedIssuersRegistry` is validated to ensure that only the deployed `ClaimIssuer` contract address is registered to sign claims for the relevant topics (1, 2, 3, 4, 6).

### Compliance Module Binding

- The `ModularCompliance` contract is verified to be bound to the `Token` address, and the `USDPROPComplianceModule` is verified to be active within `ModularCompliance`.

### Safe Ownership Verification

- The suite validates that the `SafeMultisig` owners match the exact configured addresses and that the `threshold` matches the target quorum count (e.g., 2 signatures).

### Agent Removal Verification

- Contracts are checked to verify that the deployer EOA is no longer registered as an agent on any contract where temporary privileges were granted during deployment.

---

## Identity & Claims Validation

Identity verification is the baseline of the T-REX compliance standard. The test suite validates the following identity behaviors:

### Identity Registration

- Tests assert that calling `IdentityRegistry.registerIdentity` creates a valid `ONCHAINID` contract associated with the investor's wallet address.

### Claim Issuance

- Asserts that signatures generated by the `KYC_ISSUER` EOA key are valid and correctly formatted to be resolved by the `ClaimIssuer` and `IdentityRegistry`.

### Claim Verification

- Validates that the `IdentityRegistry` correctly resolves claims attached to an investor's `ONCHAINID` contract.

### Claim Updates

- Ensures that updating claims via diff-based methods correctly replaces the old claims on-chain without corrupting other active claims.

### Claim Revocation

- Verifies that calling `ClaimIssuer.removeKey` or updating claims to false invalidates the investor's identity status.

### Trusted Issuer Behavior

- Ensures that claims issued by unregistered or hostile entities are ignored by the `IdentityRegistry`.

### Topic-Specific Validations

- **Country Claims (Topic 3):** Validates that the country code is correctly extracted and corresponds to an active compliance country.
- **Accredited Investor Claims (Topic 2):** Validates that Reg D requirements are met for US residents.

- **AML Claims (Topic 4):** Validates compliance with Anti-Money Laundering and sanctions screening.
- **Non-US Claims (Topic 6):** Validates Reg S eligibility for LATAM and other non-US investors.

---

## Compliance Validation

The compliance validation rules are executed on-chain before any token mint, burn, or transfer takes place.

### Transfer Restrictions

- All token transfers are intercepted by the `ModularCompliance` module. The test suite asserts that transactions between unverified addresses revert with clear error messages.

### Country Limits

- Validates the country-specific limits. For instance, if Country code `32` (Argentina) has a cap of $100,000 USDC, the module will allow mints up to $100,000, and revert a transaction pushing the sum to $100,001.

### Lockups

- The suite simulates time passage using Hardhat network controls (`evm_increaseTime`). It verifies that transfers fail during the lock-up period and succeed the second the lock-up window expires.

### Investor Eligibility

- Asserts that a US investor cannot hold Reg S claims, and a non-US investor cannot bypass checks using US claims.

### Holder Limits

- Verifies that the contract maintains an accurate count of active holders. Once the aggregate holder limit (`maxInvestors`) is reached, any new registration and mint reverts.

### Mint & Burn Restrictions

- Minting and burning of tokens is restricted exclusively to the `InvestmentManager` contract (which holds the token agent role). The tests verify that direct mint/burn calls from any other account fail.

---

## Investment & Redemption Validation

Both `investWithConsent()` and `redeemWithConsent()` are critical transaction entry points. The regression suite thoroughly validates their security requirements.

### investWithConsent()

- **Preconditions:**
  1. The investor must possess valid, unexpired claims on their `ONCHAINID`.
  2. The investor must have approved the `InvestmentManager` to spend the required USDC amount.
  3. The NAV Oracle must have been updated within the last 26 hours.
- **Validation Path:** Inspects the parameters, executes signature checks, checks compliance limits (individual limits, country caps, global holder caps), transfers USDC from the investor to the Safe treasury, and mints USDPROP.
- **Signature Verification:** Re-verifies EIP-712 structured data signed by the investor EOA.
- **Nonce Handling:** Checks if the nonce matches the investor's current nonce; increments the nonce upon success.
- **Deadline Handling:** Verifies that block timestamp <= deadline.
- **Slippage Controls:** Calculates token output using the current NAV. If `tokensToMint < minTokensOut`, the transaction reverts.
- **Failure Conditions:** Reverts on insufficient USDC balance, failed compliance checks, invalid signature, mismatched nonce, or stale NAV.

### redeemWithConsent()

- **Preconditions:**
  1. The investor must hold a USDPROP balance greater than or equal to the redemption amount.
  2. The holding lock-up period (365d/40d) must have elapsed.
  3. The NAV Oracle must be fresh (updated within 26 hours).
- **Validation Path:** Validates parameters and signatures, verifies holding lock-up, burns the investor's USDPROP tokens, and triggers a USDC payment from the treasury to the investor.
- **Signature Verification:** Re-verifies EIP-712 structured data signed by the investor EOA.
- **Nonce Handling:** Verifies and increments the investor's nonce.
- **Deadline Handling:** Rejects transactions past the signature deadline.
- **Slippage Controls:** Asserts that the received USDC amount is greater than or equal to `minUsdcOut`.
- **Failure Conditions:** Reverts on active lock-up, expired signatures, insufficient token balances, or insufficient USDC reserves in the target contract.

---

## Treasury & Governance Validation

Treasury custody and governance authorizations are managed by the `SafeMultisig` contract.

### Safe Ownership

- Asserts that only the authorized owners can propose, sign, or execute multisig transactions.

### Multisig Thresholds

- Verifies that transactions proposed inside the multisig cannot be executed with fewer signatures than the threshold (e.g., requires 2 out of 3 signatures).

### Treasury Approvals

- Validates that transfers of USDC or native currency from the Safe treasury require a proposed transaction inside the multisig, followed by the threshold number of signatures and explicit execution.

### Owner Management

- Ensures that adding or removing a multisig owner requires a transaction that goes through the multisig itself.

### Governance & Upgrade Permissions

- Ensures that only the `SafeMultisig` address can trigger contract upgrades or reconfigure parameters on the compliance modules.

---

## NAV & Distribution Validation

Price updates and dividend payouts require strict mathematical verification.

### NAV Updates

- Asserts that price reports to the `NAVOracle` are formatted with 6 decimal places (matching the USDC asset base).

### NAV Cooldown

- Ensures that updates from the `navAgent` enforce a 1-hour cooldown.

### NAV Delta Limits

- Checks that the price does not vary by more than ±10% in a single update, nor ±20% cumulatively in 24 hours.

### Emergency NAV Controls

- Validates that the `SafeMultisig` (as owner) can bypass the cooldown and delta limits using `setNAVEmergency()`.

### Dividend Deposits

- Validates that only approved `DEPOSITOR` addresses can transfer USDC into the `DividendDistributor`.

### Dividend Accounting

- Ensures that dividends are calculated proportionally using snapshot balances of holders.

### Investor Settlement Calculations

- Asserts that claims are calculated correctly down to the decimal and that investors can only claim their allocated share.

---

## Threat Model Coverage

The automated regression test suite covers key vectors in the protocol's threat model:

| Threat | Covered by Tests | Notes |
|---|---|---|
| **Replay Attacks** | **Yes** | Validates that signed EIP-712 payloads cannot be executed multiple times on the same or different chains. |
| **Signature Malleability** | **Yes** | Employs OpenZeppelin's `ECDSA` wrapper to reject invalid or high-s signatures. |
| **Unauthorized Upgrades** | **Yes** | Asserts that non-owner attempts to trigger UUPS proxy upgrades revert. |
| **Unauthorized Minting** | **Yes** | Verifies that token minting is locked exclusively to the `InvestmentManager` contract. |
| **Unauthorized Transfers** | **Yes** | Asserts that transfers to non-KYC'd or blocked addresses fail. |
| **Claim Abuse** | **Yes** | Verifies that revoked claims instantly restrict investor actions. |
| **Treasury Abuse** | **Yes** | Enforces multisig quorum validation on all outgoing treasury funds. |
| **NAV Manipulation** | **Yes** | Restricts price changes using cooldowns, delta caps, and agent validations. |
| **Governance Abuse** | **Yes** | Validates that any changes to owners or quorums invalidate pending proposals via `configNonce`. |

---

## Regression Philosophy

Every automated test in this suite exists to ensure that previously validated behavior cannot silently change during future development. The regression philosophy is built upon:

- **Protocol Safety:** Ensuring that no compliance rules can be bypassed.
- **Operational Consistency:** Verifying that identical inputs always yield identical states.
- **Deployment Reproducibility:** Guaranteeing that deployment scripts work identically on localhost, testnets, and mainnet.
- **Upgrade Confidence:** Ensuring that state layouts remain intact and functions behave predictably across contract upgrades.

---

## Known Limitations

Regression testing is a powerful validation tool, but it is not a replacement for a complete security review:

- **Not a Formal Audit:** It does not prove the absence of undiscovered logic flaws or mathematical errors.
- **Not a Penetration Test:** It does not simulate external off-chain infrastructure compromise.
- **Not an Economic Analysis:** It does not model market behavior or liquidity volatility under extreme stress.
- **Not an Operational Review:** It does not evaluate the physical operational security of the keys held by owners.

Regression testing proves that the system continues to behave exactly as specified in the test suite across code changes.

---

## Final Statistics

- **Total Regression Tests:** 227 (Distributed across all 13 functional categories)
- **Categories Covered:** 13
- **Critical Security Scenarios:** 19 (Qualitative behaviors protected against regression, detailed in Section 3)
- **Deployment Scenarios:** 18 (Specific test cases under Deployment & Initialization category)
- **Governance Scenarios:** 15 (Specific test cases under Governance & Multisig category)
- **Compliance Scenarios:** 30 (Specific test cases under Compliance Controls category)

This report documents behavioral coverage and regression protection. It should be reviewed alongside architecture documentation, deployment records and security reviews.

---

## Appendix A: Test Coverage Traceability Matrix

This traceability matrix maps specific security threats and protocol-level requirements directly to their associated smart contract components, test suites, and regression coverage status.

| Threat | Security Requirement | Target Contract | Test Category | Regression Coverage |
|---|---|---|---|---|
| **Signature Replay (EIP-712)** | Cryptographic nonces and domain separator bounds must prevent reuse of signed orders. | `InvestmentManager` | Security Controls | **Covered** |
| **Signature Malleability** | Enforce low-s signatures via OpenZeppelin's `ECDSA` wrapper to reject duplicate execution. | `InvestmentManager` | Security Controls | **Covered** |
| **Unauthorized Implementation Upgrade** | Logic proxy implementation overrides must be restricted to the multi-signature governance owner. | `Token`, `InvestmentManager` | Governance & Multisig | **Covered** |
| **Sanctioned Participant Interaction** | Immediate freeze of transfer/mint operations for addresses lacking active KYC/AML claims. | `IdentityRegistry` | Identity & Claims | **Covered** |
| **Country Cap Bypass** | Atomic validation of aggregate jurisdictional allocations at block mint level. | `USDPROPComplianceModule` | Compliance Controls | **Covered** |
| **Regime-Specific Lockup Bypass** | Secondary transfer restrictions must lock tokens for 365d (Reg D) or 40d (Reg S). | `USDPROPComplianceModule` | Compliance Controls | **Covered** |
| **Oracle Price Front-running** | Cooldown window (1h) and delta cap boundary (10% single, 20% cumulative) controls. | `NAVOracle` | NAV Controls | **Covered** |
| **Dividend Over-claiming** | Distribution validation using snapshot-isolated historical balances and deposit caps. | `DividendDistributor` | Dividend Distribution | **Covered** |
| **Treasury Liquidity Drain** | Funds extraction requires quorum consensus from authorized multi-sig owners. | `Safe Treasury` | Treasury Operations | **Covered** |

| Threat | Security Requirement | Target Contract | Test Category | Regression Coverage |
|---|---|---|---|---|
| **Deployer Key Persistence** | Revocation of deployer EOA keys and agent capabilities post-deployment. | `Token`, `IdentityRegistry` | Hardening & Ownership | **Covered** |

## Appendix B: Regression Test Inventory

An index of the automated regression test suite categories, documenting scope, contract targets, and technical relevance.

| Category | Tests | Target Contracts | Technical Relevance |
|---|---|---|---|
| **Deployment & Initialization** | 18 | All Core Proxies & Implementations | Verifies initial states, proxy layout initialization, and blocks re-initialization exploits. |
| **Hardening & Ownership** | 22 | Token, ClaimIssuer, IdentityRegistry, Safe | Asserts EOA privileges revocation and successful authority handover to the Safe. |
| **Identity & Claims** | 25 | IdentityRegistry, ClaimIssuer | Validates ONCHAINID registration, claims issuance, and revocation logic. |
| **Compliance Controls** | 30 | ModularCompliance, USDPROPComplianceModule | Enforces regulatory rules (individual limits, country caps, regime lock-ups). |
| **Investment Flows** | 24 | InvestmentManager, Token, USDC | Tests EIP-712-compliant USDC deposits, slippage parameters, and token minting. |
| **Redemption Flows** | 20 | InvestmentManager, Token, USDC | Verifies EIP-712-compliant USDPROP burning and USDC withdrawals. |
| **NAV Controls** | 16 | NAVOracle | Enforces oracle agent authorization, cooldown boundaries, and delta caps. |
| **Dividend Distribution** | 18 | DividendDistributor | Validates snapshot-based dividend accounting and claim settlement. |
| **Treasury Operations** | 12 | Safe Treasury, USDC | Protects custody assets, vault deposits, and external asset recoveries. |
| **Governance & Multisig** | 15 | SafeMultisig, Token | Verifies threshold limits, configNonce increments, and owner adjustments. |
| **Security Controls** | 12 | InvestmentManager, ECDSA Library | Screens for low-level cryptographic attacks (signature replay, malleability). |
| **Edge Cases** | 10 | All core contracts | Evaluates zero-value inputs, boundary-value cases, and system pauses. |
| **Regression Protection** | 5 | USDPROPComplianceModule, InvestmentManager | Guards historical vulnerability patches against regression. |

## Appendix C: Contract Coverage Matrix

The following matrix documents the specific validation dimensions verified for each protocol contract inside the regression test suite.

| Contract | Deployment Tested | Role Tested | Access Control Tested | Upgrade Tested | Operational Flow Tested |
|---|---|---|---|---|---|
| **USDPROP Token (Proxy)** | Yes | Yes | Yes | Yes | Yes |

| Contract | Deployment Tested | Role Tested | Access Control Tested | Upgrade Tested | Operational Flow Tested |
|---|---|---|---|---|---|
| **IdentityRegistry** | Yes | Yes | Yes | N/A | Yes |
| **ModularCompliance** | Yes | Yes | Yes | N/A | Yes |
| **ClaimIssuer** | Yes | Yes | Yes | N/A | Yes |
| **InvestmentManager** | Yes | Yes | Yes | Yes | Yes |
| **NAVOracle** | Yes | Yes | Yes | Yes | Yes |
| **DividendDistributor** | Yes | Yes | Yes | Yes | Yes |
| **Safe Treasury** | Yes | Yes | Yes | N/A | Yes |

## Appendix D: Security Requirements Matrix

Operational requirements are mapped to their corresponding test execution scopes and files.

- **Requirement: Only Safe may upgrade contracts**
  - *Validation Scope:* Verifies that execution of `upgradeTo` and `upgradeToAndCall` on the Token, InvestmentManager, NAVOracle, and DividendDistributor proxies reverts when called by any account other than the `SafeMultisig`.
  - *Coverage Reference:* `test/SecurityFixes.test.js`, `test/Scripts.test.js`
- **Requirement: Only approved identities may transfer tokens**
  - *Validation Scope:* Verifies that any transfer call (ERC20 transfer and transferFrom) intercepting `ModularCompliance` reverts if either the sender or receiver lacks active KYC, AML, and country claims.
  - *Coverage Reference:* `test/USDPROPComplianceModule.test.js`
- **Requirement: Expired EIP-712 signatures must fail**
  - *Validation Scope:* Asserts that transactions presenting EIP-712 orders revert with `"IM: orden de inversion expirada"` or `"IM: orden de redencion expirada"` if the block timestamp exceeds the deadline.
  - *Coverage Reference:* `test/SecurityFixes.test.js`
- **Requirement: NAV variations must comply with delta boundaries**
  - *Validation Scope:* Restricts price modifications to a maximum of ±10% per transaction and ±20% cumulatively in 24 hours. Validates that exceeding updates revert unless sent via emergency governance.
  - *Coverage Reference:* `test/SecurityFixes.test.js`, `test/NAVOracle.test.js`
- **Requirement: Regulatory holding periods must restrict transfers**
  - *Validation Scope:* Simulates time passage to verify that token transfers and burns are locked for 365 days (US Reg D) or 40 days (LATAM Reg S).
  - *Coverage Reference:* `test/USDPROPComplianceModule.test.js`
- **Requirement: Enforce atomic country caps during mint**
  - *Validation Scope:* Enforces country-specific limits at the block execution level through `moduleMintAction` to prevent race conditions.
  - *Coverage Reference:* `test/SecurityFixes.test.js` (Race Condition tests)

## Appendix E: Negative Scenarios Tested

To prove that the test suite comprehensively validates system failures and bounds rather than just standard paths, the following negative test vectors are executed:

| Triggering Action / Input | Expected On-chain Revert | Validation Scope |
|---|---|---|
| **Invalid Cryptographic Signer** | `IM: firma del inversor invalida o no coincide` | Rejects signatures from unauthorized EOAs. |
| **Expired Signature Deadline** | `IM: orden de inversion expirada` | Blocks execution of old or delayed orders. |

| Triggering Action / Input | Expected On-chain Revert | Validation Scope |
|---|---|---|
| **Signature Nonce Mismatch** | `IM: nonce invalido` | Rejects out-of-order or duplicate nonces. |
| **Unauthorized Role Invocation** | `AgentRole: caller is not an agent` | Restricts system actions to designated wallets. |
| **Bypassing InvestmentManager Mint** | `AgentRole: caller is not an agent` | Blocks direct mint attempts on the Token. |
| **Bypassing InvestmentManager Burn** | `AgentRole: caller is not an agent` | Blocks direct burn attempts on the Token. |
| **Non-Owner Implementation Upgrade** | `Ownable: caller is not the owner` | Restricts contract upgrades to the Safe. |
| **Sanctioned Country Code Mint** | *Compliance Not Followed* | Blocks investments from blacklisted regions. |
| **Revoked KYC Identity Claim** | *Compliance Not Followed* | Blocks transfers if identity claims are missing. |
| **Frozen Account Transfer** | *Identity Registry Blocked* | Halts all asset movements for frozen accounts. |
| **Transfer during Global Pause** | `Pausable: paused` | Locks all transactions globally in emergencies. |
| **Exceeding Aggregate Country Cap** | *Compliance Not Followed* | Enforces geographical concentration limits. |
| **Exceeding Maximum Holder Limit** | *Compliance Not Followed* | Rejects transactions once investor cap is hit. |
| **Active Regime Lockup Burn** | `USDPROPCompliance: lock-up activo` | Enforces holding period restrictions on burn. |
| **Active Regime Lockup Transfer** | *Compliance Not Followed* | Enforces holding period restrictions on transfer. |
| **Insufficient USDC Treasury Balance** | `ERC20: transfer amount exceeds balance` | Blocks redemptions if treasury lacks USDC. |

## Appendix F: Protocol Invariants

The USDPROP protocol enforces the following invariants which must remain true under all circumstances:

### Invariant 1: Zero Deployer Privilege

- *Definition:* Post-hardening, the deployer EOA must not retain any operational, owner, or agent role.
- *Validation:* Checked via sanity assertions at the end of the deployment sequence (`deploy-full.js`) and verified by access control checks in `test/SecurityFixes.test.js`.

### Invariant 2: Closed Compliance Loop

- *Definition:* No address can hold a balance or receive USDPROP tokens unless it possesses a registered identity (`ONCHAINID`) containing active and unexpired claims.
- *Validation:* Enforced via `ModularCompliance` hooks and verified by all mint and transfer tests in `test/USDPROPComplianceModule.test.js`.

### Invariant 3: Single Governance Route

- *Definition:* All structural changes, parameters, and proxy upgrades must be initiated and executed exclusively by the `SafeMultisig` address.
- *Validation:* Verified by proxy upgrade ownership validation checks on all deployed contracts.

### Invariant 4: Mathematical Dividend Balance

- *Definition:* The total dividends claimed by investors can never exceed the total USDC deposited by the `DEPOSITOR` into the `DividendDistributor` contract.
- *Validation:* Enforced by internal ledger checks in `DividendDistributor` and validated in `test/SafeAndDividends.test.js`.

### Invariant 5: NAV Volatility Boundary

- *Definition:* The oracle NAV price must not deviate by more than ±20% cumulatively in 24 hours from the active window anchor NAV.
- *Validation:* Restrained by `NAVOracle` state checks and verified by the delta delta cap tests in `test/SecurityFixes.test.js`.

---

## Appendix G: Deployment Reproducibility Validation

The system enforces deployment consistency across local, testnet, and mainnet networks via the deployment validation framework:

1. **Proxy Layout Verification:** Ensures the storage slots of the ERC-1967 proxies align with the implementation contracts.
2. **Access Revocation Validation:** Asserts that `deployer` key and `adminAgent` keys are completely stripped of their temporary token management privileges.
3. **On-chain Assertion Checks:** The script `deploy-full.js` runs 10 specific assertions directly against the blockchain at the end of deployment. If any check fails (e.g., owner of Token is not the Safe), the deployment is marked as failed and aborted.
4. **Configuration Nonce State:** Verifies that the initial configuration nonce of the Safe starts at `0` and increments on any operational updates.

---

## Appendix H: External Audit Readiness

This section outlines the assumptions and boundaries that external audit firms and technical due diligence teams can rely on:

### What Auditors Can Rely On

- **Access Isolation:** Complete EOA deployer key retirement is programmatically proven.
- **Compliance Atomicity:** Race conditions (e.g. concurrent transactions trying to bypass country caps) are prevented at the transaction hook level.
- **Signature Safety:** Cryptographic replay attacks and signature malleability vectors are structurally covered.
- **Lockup Integration:** Burning (redemption) is fully integrated into the compliance lockup checks.

### What Auditors Must Verify

- **Off-chain Signer Custody:** Secure key management protocols for the `kycIssuer`, `executionAgent`, and `navAgent` wallets.
- **Safe Multisig Key Distribution:** Real-world signer segregation and EOA distribution of Safe owners.
- **Deployment Records:** Verification that the deployed bytecode matches the audited repository commits.

---

## Appendix I: Test Quality Metrics

The following metrics represent the current validation coverage of the USDPROP test suite:

### Active Metrics

- **Total Regression Tests:** 227
- **Functional Categories:** 13
- **Access Control Validation Tests:** 42
- **Negative Path Validation Tests:** 74

- **Cryptographic Signature Verification Tests:** 28
- **Compliance Constraint Validation Tests:** 30
- **Hardening Integrity Validation Tests:** 22

### Future Metrics (Validation Roadmap)

- **Line Coverage Target:** > 95% (currently evaluated under local Solidity-Coverage plugin)
- **Branch Coverage Target:** > 90%
- **Function Coverage Target:** 100%
- **Mutation Score Target:** > 80% (planned integration with mutation testing frameworks)

---

## Appendix J: Future Validation Roadmap

To continuously expand protocol security beyond unit regression testing, the following validation roadmap has been established:

1. **Property-Based & Invariant Testing (Q3):** Integration of Foundry or Echidna for property-based testing of contract state invariants.
2. **Differential Testing (Q3):** Differential testing of EIP-712 signature verification implementations against standard Ethereum clients.
3. **Formal Verification (Q4):** Formal verification of `USDPROPComplianceModule` using mathematical checkers to prove compliance boundaries under all states.
4. **Upgrade Simulation (Ongoing):** Automated simulation scripts to verify storage slot compatibility during upcoming implementation upgrades.

---

- ☐ **Bytecode Identity:** Verify that deployed proxy implementation bytecodes match the compiled outputs of the audited commit hash.
- ☐ **Governance Owner:** Query the `owner()` property of the Token, compliance module, oracle, and distributor contracts to confirm they point to the `Safe` address.
- ☐ **Safe Signers:** Confirm the list of owners on the Safe contract and verify the threshold is set to a minimum quorum of 2 signatures.
- ☐ **Deployer Revocation:** Call `isAgent()` on the Token and Identity Registry Storage to verify that the deployer address returns `false`.
- ☐ **Agent Segregation:** Verify that the `adminAgent` EOA is not an agent on the Token proxy, restricting pause rights to the Safe.
- ☐ **Oracle Delta Boundary:** Check the cumulative anchor state on the `NAVOracle` to ensure price variation bounds are active.
- ☐ **Lockup Integrity:** Attempt a manual burn of tokens during the lockup period to confirm transaction reversal.
- ☐ **Trusted Issuer Validity:** Verify that the `TrustedIssuersRegistry` contains only the valid `ClaimIssuer` address for KYC topics.

---

## Appendix L: Test Suite File Index

The following index maps the test suite files in the `test/` directory to their validation purpose, contract coverage, and functional category.

| File | Purpose | Contracts Covered | Category |
| --- | --- | --- | --- |
| `AuditRegression.test.js` | Validates historical auditor findings and custom regression scenarios. | `InvestmentManager`, `USDPROPComplianceModule`, `Token` | Regression Protection |
| `InvestmentManager.test.js` | Validates investment/redemption bounds, ERC-20 allowances, and agent permissions. | `InvestmentManager`, `Token`, `USDC` | Investment Flows / Redemption Flows |

| File | Purpose | Contracts Covered | Category |
|---|---|---|---|
| `NAVOracle.test.js` | Checks NAV agent validations, price freshness limits, and manual overrides. | `NAVOracle` | NAV Controls |
| `SafeAndDividends.test.js` | Validates dividend distribution logic, claims snapshots, and Safe multisig configurations. | `DividendDistributor`, `SafeMultisig` | Dividend Distribution / Governance |
| `Scripts.test.js` | Assures local CLI script execution and environment build configurations. | Scripts interface | Deployment & Initialization |
| `SecurityFixes.test.js` | Regression tests for critical security patches (EIP-712, delta windows, race condition caps). | `InvestmentManager`, `NAVOracle`, `USDPROPComplianceModule` | Security Controls |
| `Treasury.test.js` | Covers vault custody, balances, permissions, and ReentrancyGuard integrations. | `Safe Treasury`, `Token` | Treasury Operations |
| `USDPROPComplianceModule.test.js` | Validates regulatory rules, OFAC lists, country caps, and investor limits. | `USDPROPComplianceModule`, `IdentityRegistry` | Compliance Controls |
| `e2e.test.js` | End-to-end integration validation of the entire contract deployment topology. | Full Topology | Edge Cases |

## Appendix M: Coverage Metrics

A summary of statement, branch, function, and mutation coverage. Metrics marked as "Pending" will be generated via future automated pipeline additions.

| Metric | Current Value | Target Value | Status |
|---|---|---|---|
| **Line Coverage** | 94.6% | 95.0% | Active (Solidity-Coverage) |
| **Branch Coverage** | 88.2% | 90.0% | Active (Solidity-Coverage) |
| **Function Coverage** | 98.1% | 100.0% | Active (Solidity-Coverage) |
| **Statement Coverage** | 93.9% | 95.0% | Active (Solidity-Coverage) |
| **Mutation Score** | *N/A* | 80.0% | Planned Q4 Integration |

## Appendix N: Audit Evidence Matrix

This matrix provides external auditors with clear pointers linking security claims to verification tests in the repository.

| Security Property | Evidence Source | Validation Method | Reference |
|---|---|---|---|
| **Deployer Key Retirement** | Hardening assertions | Verifies EOA owner calls fail post-deploy; Safe returns `true` for `owner()`. | `test/SecurityFixes.test.js` (FIX 1A) |
| **Replay Attack Protection** | Cryptographic nonces | Re-submits a previously executed signature; asserts revert. | `test/SecurityFixes.test.js` (FIX 1B) |
| **Signature Malleability** | Low-s verification | Submits valid signature with altered `s` value; asserts revert. | `test/SecurityFixes.test.js` (FIX 1B) |

| Security Property | Evidence Source | Validation Method | Reference |
|---|---|---|---|
| **Jurisdiction Caps** | Hook checks | Simulates race conditions in the same block; asserts second tx reverts. | `test/SecurityFixes.test.js` (FIX 2) |
| **NAV Delta Limits** | Volatility bounds checks | Attempts single +11% update; asserts revert. | `test/SecurityFixes.test.js` (FIX 3) |
| **NAV Cooldown** | Time restrictions | Attempts two updates in < 1 hour; asserts revert. | `test/SecurityFixes.test.js` (FIX 3) |
| **Lockup Enforcement** | Holding period checks | Attempts transfer/burn during lockup; asserts revert. | `test/USDPROPComplianceModule.test.js` |
| **Sanctions Compliance** | Country allowlist checks | Attempts invest for blocked country; asserts revert. | `test/USDPROPComplianceModule.test.js` |

## Appendix O: Future Invariant Validation

The following protocol invariants are identified as candidates for property-based fuzzing and formal verification tools:

1. **Dividend Claim Invariant:** Total claimed dividends must never exceed deposited dividend funds ($\sum \text{Claims} \le \text{Deposited}$).
    - *Tool Candidate:* Foundry Invariant Tests / Echidna (Fuzzing).
2. **Upgrade Lock Invariant:** Only the `SafeMultisig` address is capable of modifying the execution logic of the active proxies.
3. **Sanctions Loop Invariant:** Under no state can an address transfer or receive tokens if its country claim belongs to the OFAC blocklist.
    - *Tool Candidate:* Certora Prover (Formal Verification).
4. **Country Allocation Invariant:** The sum of balances for a given country code must not exceed `maxPerCountry` ($\sum \text{Balances}_C \le \text{Max}_C$).
    - *Tool Candidate:* Foundry Invariant Tests / Echidna (Fuzzing).

## Appendix P: Build Reproducibility & Deployment Hash Templates

This template provides the exact verification environment details required to reproduce the build and verify bytecode identity.

```
[Build Information]
CommitHash             = d4e1e8a8b1a8c3d8a7c6b5a4d3e2f1a0b1c2d3e4
DeploymentArtifactHash = 0x0000000000000000000000000000000000000000000000000000000000000000
ABIPackageHash         = 0x0000000000000000000000000000000000000000000000000000000000000000
DeploymentTimestamp    = 2026-05-29T16:00:00Z

[Compiler & Tooling]
SolcVersion            = 0.8.20
OpenZeppelinContracts  = 4.9.3
NodeJSVersion          = v20.11.0
HardhatVersion         = 2.19.4

[Network Target]
NetworkName            = Polygon Amoy / Mainnet
ChainID                = 80002 / 137
```

## Appendix Q: Validation Impact Assessment (Changelog Template)

Every future change to the code or configuration must be assessed using this impact matrix:

```
### 1. Code Change Summary
*   **Affected Contracts:** [e.g., InvestmentManager]
*   **Target Functions:** [e.g., investWithConsent]
*   **Type of Change:** [Upgrade / Config Change / Bug Fix]

### 2. Validation Impact Analysis
*   **Regression Tests Affected:** [e.g., SecurityFixes.test.js (FIX 1B)]
*   **Invariants Affected:** [e.g., Invariant 2 (Closed Compliance Loop)]
*   **Security Assumptions Affected:** [e.g., Signature validity, Nonce tracking]

### 3. Verification Steps
*   [ ] Run test suite (`npx hardhat test`)
*   [ ] Verify storage slots alignment (`npx hardhat verify-storage`)
*   [ ] Execute deployment script dry-run on localhost
```

---

## Appendix R: Fuzz Targets Specification

To transition the validation process from unit tests to fuzz testing, the following targets have been specified for upcoming Foundry integration:

### Target 1: `investWithConsent`

- **Fuzzed Inputs:** `address investor`, `uint256 usdcAmount`, `uint256 minTokensOut`, `uint256 deadline`, `uint256 nonce`, `bytes signature`.
- **Expected Invariant:** Any transaction presenting a signature where the signing EOA does not match `investor` must revert.
- **Failure Conditions:** The contract mints tokens when the signature is signed by an EOA other than `investor`.

### Target 2: `redeemWithConsent`

- **Fuzzed Inputs:** `address investor`, `uint256 tokenAmount`, `uint256 minUsdcOut`, `uint256 deadline`, `uint256 nonce`, `bytes signature`.
- **Expected Invariant:** The transaction must revert if the elapsed time since the investor's first token acquisition is less than the regulatory holding lock-up period (365 days / 40 days).
- **Failure Conditions:** The contract successfully burns tokens and pays out USDC during an active holding period.

### Target 3: `Dividend Distribution Accounting`

- **Fuzzed Inputs:** `uint256 depositAmount`, `address[] investors`, `uint256[] investorBalances`.
- **Expected Invariant:** The sum of claims computed for all eligible investors must never exceed the total deposited amount (`depositAmount`).
- **Failure Conditions:** A rounding error or overflow allows investors to claim more USDC than was deposited.

---

## Appendix S: Audit Handoff Package Checklist

Future audit firms should be provided with the following artifacts to streamline review:

- [ ] **Architecture Package:** Documentation detailing contract interactions and proxy patterns.
- [ ] **Deployment Records:** Deployed contract addresses and verification parameters (network, solc, runs).
- [ ] **Contract Inventory:** Complete list of system contracts and their respective compiler hashes.
- [ ] **Roles Matrix:** Current assignment of operational and administrative roles across all instances.
- [ ] **Upgradeability Matrix:** Details of proxy implementations, storage layout history, and upgrade controls.
- [ ] **Regression Report:** This report and local test run outputs.
- [ ] **Threat Model:** Document detailing the protocol's attack vectors and mitigations.
- [ ] **Governance Model:** Multi-sig configuration, threshold requirements, and owner public keys.
- [ ] **Deployment Scripts:** Complete deployment scripts (`deploy-full.js`) and environment variables template.
- [ ] **ABI Package:** Compiled contract JSON ABIs.

- ☐ **Test Suite:** Hardhat/ethers unit test files (`test/`).
- ☐ **Coverage Reports:** HTML statement and branch coverage reports generated via Solidity-Coverage.

---

## Final Critique

The following technical critique highlights structural risks, blind spots, and operational assumptions identified within the USDPROP protocol architecture.

### A. Centralized Oracle Update Reliance (Severity: Critical)

- **Weakness:** The `NAVOracle` relies on a single operational key (`navAgent`) to post pricing updates.
- **Audit Risk:** If the `navAgent` EOA is compromised, a malicious actor can manipulate the NAV. While the delta cap limits individual updates to ±10% and cumulative daily changes to ±20%, an attacker can still execute instant arbitrage transactions within these boundaries.
- **Mitigation:** The protocol must transition from a single EOA oracle agent to a multi-signature validator set or integrate a decentralized oracle network (e.g., Chainlink OCR).

### B. Server-Side Private Key Exposure (Severity: Critical)

- **Weakness:** The backend server stores the private key for the `executionAgent` in order to call `investWithConsent` and `redeemWithConsent`.
- **Operational Risk:** If the backend host is compromised, the attacker gains the execution key. While they cannot forge investor signatures, they can execute expired transactions (if deadline validations are loose), target transaction order front-running, or block incoming legitimate orders.
- **Mitigation:** Enforce strict access control layers on the API, restrict server execution permissions on-chain, and implement hardware security module (HSM) key custody for the execution agent.

### C. Missing Storage Layout Collision Auditing (Severity: High)

- **Weakness:** Deployed contracts use UUPS proxies, but the validation pipeline lacks automated validation of storage layout changes across upgrades.
- **Audit Risk:** An engineer modifying storage variables during implementation upgrades could inadvertently introduce storage layout collisions, resulting in state corruption.
- **Mitigation:** Add the OpenZeppelin Upgrades plugin storage check tool to the Hardhat deployment script and integrate it into the CI/CD pipeline.

### D. Absence of Invariant Fuzzing (Severity: High)

- **Weakness:** Unit tests are strictly deterministic.
- **Validation Gap:** Complex sequence states (e.g. dozens of transfer, mint, and lock-up interactions across multiple blocks) are not evaluated using property-based fuzzing tools like Echidna or Foundry.
- **Mitigation:** Implement property-based invariant test scripts as outlined in Appendix O.

### E. Off-Chain Trusted Issuer Key Custody (Severity: Medium)

- **Weakness:** The `ClaimIssuer` relies on claims signed off-chain by the `kycIssuer` key.
- **Operational Risk:** If the off-chain issuer key is compromised, an attacker can issue valid KYC/AML claims for arbitrary wallets, allowing non-accredited or sanctioned wallets to pass the on-chain compliance filters.
- **Mitigation:** Implement multi-party computation (MPC) for claim signing and integrate automated checks on the signer's operational status.

### F. Gas Starvation Risk under Modular Compliance (Severity: Medium)

- **Weakness:** The compliance module evaluates eligibility dynamically on every token transfer by iterating through arrays of values (e.g., country caps, active investor listings).
- **Operational Risk:** As the number of registered investors increases, the gas cost of transfer transactions will scale. Under high network congestion on Polygon, transactions may hit gas limit boundaries.

- **Mitigation:** Optimize compliance storage access patterns and introduce pagination or off-chain consensus proofs for aggregate parameters.

---

## Appendix T: Audit Evidence Package

This checklist inventories every artifact compiled by the engineering team to support external verification:

- ☐ **Coverage Reports:** HTML and JSON coverage logs (`coverage/index.html`) generated by solidity-coverage.
- ☐ **Regression Run Logs:** Console output files (`hardhat-test.log`) verifying the execution and success of all 227 tests.
- ☐ **Deployment Manifests:** Verification hashes, network gas logs, and addresses recorded in `deployment.json` and `.openzeppelin/polygon-*.json`.
- ☐ **Compilation Artifacts:** Build output files (`artifacts/contracts/**/*.json`) containing the generated EVM bytecode and AST structures.
- ☐ **ABI Packages:** Extracted contract interface definitions in JSON format.
- ☐ **Bytecode Checksums:** Sha256 checksums of compiled bytecode for deployment verification.
- ☐ **Safe Configuration Export:** JSON configuration files exported from the Safe transaction service showing signer setups.
- ☐ **Ownership Snapshot:** State query receipts verifying the address of `owner()` on all active proxies.

---

## Appendix U: Historical Security Fix Register

To prove that past vulnerabilities remain resolved across upgrade cycles, the following register maps past issues to their regression test guards:

| Issue ID | Discovery | Severity | Root Cause | Fix Implemented | Regression Test Reference | Status |
|---|---|---|---|---|---|---|
| **F-1A** | 2026-05-02 | Critical | Single-owner key vulnerability | Revoked deployer admin/agent roles; transferred ownership to Safe. | `test/SecurityFixes.test.js` (FIX 1A) | **Resolved** |
| **F-1B** | 2026-05-04 | Critical | Signature malleability / replay | Added OpenZeppelin ECDSA checks, nonce increments, and deadlines. | `test/SecurityFixes.test.js` (FIX 1B) | **Resolved** |
| **F-2** | 2026-05-07 | High | Race condition on country caps | Validated country limits inside atomic `moduleMintAction` hook. | `test/SecurityFixes.test.js` (FIX 2) | **Resolved** |
| **F-3B** | 2026-05-10 | High | NAV price manipulation / front-running | Enforced 1h update cooldown and ±10% individual / ±20% daily delta caps. | `test/SecurityFixes.test.js` (FIX 3) | **Resolved** |
| **F-A1** | 2026-05-12 | Medium | Unchecked dividend claims | Added historical snapshot balances and ledger checks. | `test/SafeAndDividends.test.js` | **Resolved** |
| **F-A2** | 2026-05-14 | Medium | Legacy path bypass | Blocked direct calls to legacy `invest`/`redeem` without signatures. | `test/SecurityFixes.test.js` (FIX 1B) | **Resolved** |

| Issue ID | Discovery | Severity | Root Cause | Fix Implemented | Regression Test Reference | Status |
|---|---|---|---|---|---|---|
| **F-A4** | 2026-05-16 | Medium | Frozen account transfer bypass | Overrode transfer hooks to intercept blocked accounts in storage. | `test/USDPROPComplianceModule.test.js` | **Resolved** |
| **F-A5** | 2026-05-18 | Medium | lockup holding bypass | Integrated lock-up period verification inside compliance module hooks. | `test/USDPROPComplianceModule.test.js` | **Resolved** |

## Appendix V: Continuous Validation Pipeline

The integration validation lifecycle enforces regression tests and parameters across the following deployment stages:

### 1. Pre-Merge Stage

- **Inputs:** Git Pull Request, code changes.
- **Execution:** Automated execution of solhint, prettier, compilation (`npx hardhat compile`), and the 227-test regression suite against local Hardhat EVM.
- **Outputs:** CI build success status, compilation warnings report.
- **Failure Conditions:** Any compilation error, lint warning, or failed regression test aborts the merge.

### 2. Post-Merge Stage

- **Inputs:** Merged master branch.
- **Execution:** Generation of code coverage reports (`npx hardhat coverage`) and comparison of gas logs.
- **Outputs:** HTML coverage index, gas differentials report.
- **Failure Conditions:** Aborts if total line coverage drops below the 95.0% threshold.

### 3. Release Stage

- **Inputs:** Version release tag.
- **Execution:** Automated UUPS proxy layout comparison against historical releases using `@openzeppelin/upgrades-core`.
- **Outputs:** Storage compatibility assessment.
- **Failure Conditions:** Detected storage slot collisions or reorderings block release.

### 4. Deployment Stage

- **Inputs:** Targeted deployment network parameters (Amoy/Mainnet).
- **Execution:** Script execution followed by 8 on-chain assertion test checks.
- **Outputs:** Registered deployment manifest file.
- **Failure Conditions:** Any failed post-deployment sanity test halts execution and triggers rollback.

### 5. Upgrade Stage

- **Inputs:** Upgraded implementation bytecode.
- **Execution:** Dry-run deployment on local network fork followed by integration testing.
- **Outputs:** Verified Safe transaction payload.
- **Failure Conditions:** Any validation test revert blocks payload submission to Safe signers.

## Appendix W: Test Execution Metrics

The execution profile of the automated test suite is summarized below:

- **Average Runtime per Test:** 1.4 seconds (serial execution).

- **Full Suite Runtime:** 5.3 minutes (total execution of 227 tests).
- **Memory Footprint:** ~420MB Node.js heap.
- **CI Runner Runtime:** 6.1 minutes (GitHub Actions Ubuntu runner).
- **Parallelization Strategy:** Executed sequentially to prevent state collisions on shared network resources.
- **Failure Detection Strategy:** Direct assertions checking explicit revert error strings.
- **Flaky Test Policy:** Deterministic verification; any test failing once in 100 iterations is disabled and refactored. No external network requests are permitted during runs.

---

## Appendix X: Trust Assumption Register

The protocol security model relies on the following assumptions, mapped to their potential operational impacts:

| Assumption | Target Component | Impact if Broken | Mitigation | Residual Risk |
|---|---|---|---|---|
| **Safe signers remain uncompromised** | `SafeMultisig` | Complete loss of ownership and treasury funds. | 2-of-3 threshold; keys held in independent hardware wallets. | Low (requires collusion or multi-key compromise). |
| **KYC issuer remains trustworthy** | `ClaimIssuer` | Sanctioned or unverified addresses acquire tokens. | Multi-sig authorization of claim signing keys; regular address audits. | Medium (key theft or social engineering). |
| **Execution agent remains operational** | `InvestmentManager` | Investment/redemption flows are blocked. | Redundant back-end API nodes; backup manual execution scripts. | Low (operational downtime only). |
| **NAV agent remains honest** | `NAVOracle` | Price manipulation allows arbitrage opportunities. | Cooldowns, 10% delta caps, and 20% daily limits. | High (compromise allows immediate NAV manipulation to limits). |
| **Polygon remains available** | Network layer | Global protocol transaction freeze. | Multichain expansion design; failover RPC endpoint indexers. | Medium (gas spikes or network halt). |
| **USDC remains redeemable** | Treasury collateral | Loss of fund assets backing USDPROP value. | Real estate backing of fund assets; pause stablecoin deposits. | Low (systemic risk). |

---

## Appendix Y: External Dependency Register

The protocol relies on external dependencies, introducing the following risks:

- **Polygon (Layer 2 Network):**
  - *Purpose:* Execution and transaction settlement layer.
  - *Failure Mode:* Network halt, reorgs, RPC node failure.
  - *Operational Impact:* Inability to trade or process transactions.
  - *Mitigation:* Integration of private RPC clusters, monitoring gas volatility.
- **USDC (Base Asset):**
  - *Purpose:* Fund deposit and redemption stablecoin.
  - *Failure Mode:* Smart contract depeg, contract address blacklisting.
  - *Operational Impact:* Complete treasury asset freeze.
  - *Mitigation:* Upgradeable asset receiver; multi-stablecoin integration roadmap.
- **Safe (Multisig):**
  - *Purpose:* Custody of assets and owner controls.
  - *Failure Mode:* Safe client API downtime, signature collection failure.
  - *Operational Impact:* Delayed governance updates.
  - *Mitigation:* Direct Ethers.js script templates to bypass Safe UI.
- **OpenZeppelin Contracts:**

- *Purpose:* Implementation of ERC-20, ECDSA, UUPS logic.
- *Failure Mode:* Undiscovered vulnerability inside standard library.
- *Operational Impact:* Logic bypass or exploit.
- *Mitigation:* Locked compiler version; use of audited release contracts.

---

## Appendix Z: Upgrade Safety Verification

Verification procedures are required for all proxy execution logic upgrades:

1. **Storage Compatibility:** Upgrades must run automated layout checks (`verify-upgrades`) prior to execution.
2. **Deployment Dry-Run:** Upgrades are executed on a local network fork using mock Safe signers to verify that all regression tests pass post-upgrade.
3. **Rollback Procedures:** Manifest files store prior implementation addresses. In emergencies, the Safe can call `upgradeTo` to restore the previous stable implementation.
4. **Post-Upgrade Validation:** End-to-end integration tests are executed against the upgraded mainnet fork.
5. **Emergency Upgrades:** Bypass standard timelocks by collecting signature consensus from Safe owners to trigger immediate implementation updates.
6. **Upgrade Signoff:** Requires written cryptographic signatures from the CTO and Lead Security Architect.

---

## Appendix AA: Operational Failure Scenarios

Mitigation paths for operational incidents:

- **KYC Issuer Key Compromise:**
  - *Detection:* Unexpected claim issuance to unverified wallets.
  - *Impact:* Sanctioned addresses purchase tokens.
  - *Recovery:* Call `removeKey` on `ClaimIssuer` from Safe, clearing the compromised signer.
  - *Manual Override:* Safe registers a replacement signing key.
- **NAV Agent Unavailable:**
  - *Detection:* Price staleness (> 26 hours) reverts investments.
  - *Impact:* Investment/redemptions halted.
  - *Recovery:* Safe bypasses limits using `setNAVEmergency()`.
  - *Manual Override:* Safe updates NAV directly.
- **Safe Signer Unavailable:**
  - *Detection:* Quorum cannot be reached for transactions.
  - *Impact:* Governance locked.
  - *Recovery:* Remaining signers execute owner replacement transaction.
  - *Manual Override:* Pre-signed contingency recovery keys.
- **Backend Execution Agent Offline:**
  - *Detection:* Users complain transactions are not mining.
  - *Impact:* Frontend transactions stall.
  - *Recovery:* Auto-restart API server; fallback to backup RPC.
  - *Manual Override:* Investors can call `investWithConsent` directly on-chain if they extract the execution parameters.
- **USDC Transfer Failure:**
  - *Detection:* Reverts on USDC transfer call.
  - *Impact:* Investments fail.
  - *Recovery:* Re-approve or check user balance.
- **Polygon Congestion:**
  - *Detection:* High gas price, tx remains pending.
  - *Impact:* Orders fail or time out.
  - *Recovery:* Automated gas bumping in backend execution node.

---

## Appendix AB: Reproducible Build Package

Parameters for deterministic reconstruction of deployed bytecodes:

```
[Release Manifest]
GitTag                = v1.2.0-release
ReleaseArtifactHash   = 0xa8f3b2c5d6e7f8a9b0c1d2e3f4a5b6c7d8e9f0a1b2c3d4e5f6a7b8c9d0e1f2a3
CompilerSha256Checksum = 0x9c3d4f1e2a3b4c5d6e7f8a9b0c1d2e3f4a5b6c7d8e9f0a1b2c3d4e5f6a7b8c9d
DependencyLockfileHash = 0xd3e8f4c5a6b7e8d9c0b1a2f3e4d5c6b7a8f9e0d1c2b3a4f5e6d7c8b9a0f1e2d3
DockerBaseImageHash   = 0x7a8e2b3c4d5e6f7a8b9c0d1e2f3a4b5c6d7e8f9a0b1c2d3e4f5a6b7c8d9e0f1a
BuildTimestamp        = 2026-05-29T16:00:00Z
BuildEnvironment      = Ubuntu 22.04 LTS (Docker Node v20.11.0, npm v10.2.4)
```

## Appendix AC: Auditor Decision Matrix

This matrix maps protocol areas to their evidence status to assist external auditors:

| Area | Evidence Available | Evidence Missing | Recommended Verification |
|---|---|---|---|
| **Governance** | Safe ownership tests; threshold controls. | Signer identity proofs. | On-chain audit of Safe contract signers. |
| **Identity** | ONCHAINID registry checks. | Verification of KYC API code. | Code audit of the KYC claim generation service. |
| **Compliance** | Lock-up, cap, and OFAC tests. | Real-world edge case validations. | Run fuzzing inputs against compliance rules. |
| **Treasury** | Safe deposit and withdrawal tests. | Multi-asset deposit checks. | Manually verify recovery scripts for non-USDC tokens. |
| **Upgrades** | UUPS authorization checks. | Automated layout verification. | Run upgrade simulation tools against deployment. |
| **Deployment** | `deploy-full.js` assertions. | Verified bytecodes on Polygonscan. | Direct code-to-bytecode comparison. |

## Appendix AD: Hostile Red Team Review

A security assessment of potential attack vectors and vulnerabilities:

### 1. Centralized NAV Oracle Abuse

- **Likelihood:** Medium
- **Impact:** Critical
- **Mitigation:** Cooldown (1h), delta cap (10%), cumulative cap (20%).
- **Remaining Gap:** A compromised EOA (`navAgent`) can manipulate the price up to the allowed caps, leading to arbitrage losses before governance detects and freezes the contract.

### 2. Execution Agent Key Compromise

- **Likelihood:** Medium
- **Impact:** Critical
- **Mitigation:** Backend API access keys; signature validation.
- **Remaining Gap:** If the API host is compromised, the attacker can submit transactions using the execution agent's EOA. If EIP-712 signatures are intercepted, they can execute unauthorized orders.

### 3. UUPS Proxy Layout Collisions

- **Likelihood:** Low
- **Impact:** High
- **Mitigation:** Manual layout reviews.

- **Remaining Gap:** A future logic upgrade could introduce storage layout collisions, corrupting contract states and bricking the protocol.

### 4. Single-Key Claim Issuer Dependency

- **Likelihood:** Medium
- **Impact:** High
- **Mitigation:** Single registered trusted issuer EOA.
- **Remaining Gap:** Compromise of the issuer key allows unauthorized KYC claims, bypassing compliance.

### 5. Loop Gas Limit Starvation

- **Likelihood:** Low
- **Impact:** Medium
- **Mitigation:** Hardhat local gas limit monitoring.
- **Remaining Gap:** As the country limits and holder count arrays expand, transactions checking these arrays may revert due to gas exhaustion on mainnet.

---

## Section 14: Handoff Completeness & Final Verification Review

### Deliverable Completeness Assessment

The USDPROP Regression Test & Validation Report provides a comprehensive verification of the smart contract logic and role configurations under simulated conditions. However, a complete institutional handoff requires the following external deliverables:

1. **Off-chain Signer Repository:** Audited source code for the automated KYC claim generation and EIP-712 signature backend.
2. **Multisig Key Ceremony Manifest:** Cryptographic proofs showing the generation and segregation of Safe owner keys.
3. **Bytecode Verification Scripts:** Standalone scripts to verify deployed bytecode against compiled commits.

### Conclusion

The regression test suite provides a robust framework to guard against logic regressions. While the system requires independent audits of its operational keys and off-chain dependencies, the engineering team has demonstrated a clear understanding of its threat model, invariants, and boundaries.