/token-integration-analyzer

Source: `~/.claude/skills/tob-building-secure-contracts/skills/token-integration-analyzer/SKILL.md`

---

name: token-integration-analyzer
description: Token integration and implementation analyzer based on Trail of Bits' token integration checklist. Analyzes token implementations for ERC20/ERC721 conformity, checks for 20+ weird token patterns, assesses contract composition and owner privileges, performs on-chain scarcity analysis, and evaluates how protocols handle non-standard tokens. Context-aware for both token implementations

and token integrations.

Token Integration Analyzer

Purpose

Systematically analyzes the codebase for token-related security concerns using Trail of Bits' token integration checklist:

1. **Token Implementations**: Analyze if your token follows ERC20/ERC721 standards or has non-standard behavior
2. **Token Integrations**: Analyze how your protocol handles arbitrary tokens, including weird/non-standard tokens
3. **On-chain Analysis**: Query deployed contracts for scarcity, distribution, and configuration
4. **Security Assessment**: Identify risks from 20+ known weird token patterns

Framework: Building Secure Contracts - Token Integration Checklist + Weird ERC20 Database

---

How This Works

Phase 1: Context Discovery

Determines analysis context:

- **Token implementation**: Are you building a token contract?
- **Token integration**: Does your protocol interact with external tokens?
- **Platform**: Ethereum, other EVM chains, or different platform?
- **Token types**: ERC20, ERC721, or both?

Phase 2: Slither Analysis (if Solidity)

For Solidity projects, I'll help run:

- `slither-check-erc` - ERC conformity checks
- `slither --print human-summary` - Complexity and upgrade analysis
- `slither --print contract-summary` - Function analysis
- `slither-prop` - Property generation for testing

Phase 3: Code Analysis

Analyzes:

- Contract composition and complexity
- Owner privileges and centralization risks
- ERC20/ERC721 conformity
- Known weird token patterns
- Integration safety patterns

Phase 4: On-chain Analysis (if deployed)

If you provide a contract address, I'll query:

- Token scarcity and distribution
- Total supply and holder concentration
- Exchange listings
- On-chain configuration

Phase 5: Risk Assessment

Provides:

- Identified vulnerabilities
- Non-standard behaviors
- Integration risks
- Prioritized recommendations

Assessment Categories

I check 10 comprehensive categories covering all aspects of token security. For detailed criteria, patterns, and checklists, see ASSESSMENT_CATEGORIES.md.

Quick Reference:

1. **General Considerations** - Security reviews, team transparency, security contacts
2. **Contract Composition** - Complexity analysis, SafeMath usage, function count, entry points
3. **Owner Privileges** - Upgradeability, minting, pausability, blacklisting, team accountability
4. **ERC20 Conformity** - Return values, metadata, decimals, race conditions, Slither checks

5. **ERC20 Extension Risks** - External calls/hooks, transfer fees, rebasing/yield-bearing tokens
6. **Token Scarcity Analysis** - Supply distribution, holder concentration, exchange distribution, flash loan/mint risks
7. **Weird ERC20 Patterns** (24 patterns including):
   - Reentrant calls (ERC777 hooks)
   - Missing return values (USDT, BNB, OMG)
   - Fee on transfer (STA, PAXG)
   - Balance modifications outside transfers (Ampleforth, Compound)
   - Upgradable tokens (USDC, USDT)
   - Flash mintable (DAI)
   - Blocklists (USDC, USDT)
   - Pausable tokens (BNB, ZIL)
   - Approval race protections (USDT, KNC)
   - Revert on approval/transfer to zero address
   - Revert on zero value approvals/transfers
   - Multiple token addresses
   - Low decimals (USDC: 6, Gemini: 2)
   - High decimals (YAM-V2: 24)
   - transferFrom with src == msg.sender
   - Non-string metadata (MKR)
   - No revert on failure (ZRX, EURS)
   - Revert on large approvals (UNI, COMP)
   - Code injection via token name
   - Unusual permit function (DAI, RAI, GLM)
   - Transfer less than amount (cUSDCv3)
   - ERC-20 native currency representation (Celo, Polygon, zkSync)
   - And more...
8. **Token Integration Safety** - Safe transfer patterns, balance verification, allowlists, wrappers, defensive patterns
9. **ERC721 Conformity** - Transfer to 0x0, safeTransferFrom, metadata, ownerOf, approval clearing, token ID immutability
10. **ERC721 Common Risks** - onERC721Received reentrancy, safe minting, burning approval clearing

---

Example Output

When analysis is complete, you'll receive a comprehensive report structured as follows:

```
=== TOKEN INTEGRATION ANALYSIS REPORT ===

Project: MultiToken DEX
```

```
Token Analyzed: Custom Reward Token + Integration Safety
Platform: Solidity 0.8.20
Analysis Date: March 15, 2024

---

## EXECUTIVE SUMMARY

Token Type: ERC20 Implementation + Protocol Integrating External Tokens
Overall Risk Level: MEDIUM
Critical Issues: 2
High Issues: 3
Medium Issues: 4

**Top Concerns:**
⚠ Fee-on-transfer tokens not handled correctly
⚠ No validation for missing return values (USDT compatibility)
⚠ Owner can mint unlimited tokens without cap

**Recommendation:** Address critical/high issues before mainnet launch.

---

## 1. GENERAL CONSIDERATIONS

✓ Contract audited by CertiK (June 2023)
✓ Team contactable via security@project.com
✗ No security mailing list for critical announcements

**Risk:** Users won't be notified of critical issues
**Action:** Set up security@project.com mailing list

---

## 2. CONTRACT COMPOSITION

### Complexity Analysis

**Slither human-summary Results:**
- 456 lines of code
```

- Cyclomatic complexity: Average 6, Max 14 (transferWithFee())
- 12 functions, 8 state variables
- Inheritance depth: 3 (moderate)

✓ Contract complexity is reasonable
⚠ transferWithFee() complexity high (14) - consider splitting

SafeMath Usage

✓ Using Solidity 0.8.20 (built-in overflow protection)
✓ No unchecked blocks found
✓ All arithmetic operations protected

Non-Token Functions

Functions Beyond ERC20:
- setFeeCollector() - Admin function ✓
- setTransferFee() - Admin function ✓
- withdrawFees() - Admin function ✓
- pause()/unpause() - Emergency functions ✓

⚠ 4 non-token functions (acceptable but adds complexity)

Address Entry Points

✓ Single contract address
✓ No proxy with multiple entry points
✓ No token migration creating address confusion

Status: PASS

---

3. OWNER PRIVILEGES

Upgradeability

⚠ Contract uses TransparentUpgradeableProxy
Risk: Owner can change contract logic at any time

```
**Current Implementation:**
- ProxyAdmin: 0x1234... (2/3 multisig) ✓
- Timelock: None ✗

**Recommendation:** Add 48-hour timelock to all upgrades

### Minting Capabilities

 CRITICAL: Unlimited minting
File: contracts/RewardToken.sol:89
```solidity
function mint(address to, uint256 amount) external onlyOwner {
    _mint(to, amount);  // No cap!
}
```

Risk: Owner can inflate supply arbitrarily **Fix:** Add maximum supply cap or rate-limited minting

Pausability

✓ Pausable pattern implemented (OpenZeppelin) ✓ Only owner can pause ⚠ Paused state affects all transfers (including existing holders)

Risk: Owner can trap all user funds **Mitigation:** Use multi-sig for pause function (already implemented ✓)

Blacklisting

✗ No blacklist functionality **Assessment:** Good - no centralized censorship risk

Team Transparency

✓ Team members public (team.md) ✓ Company registered in Switzerland ✓ Accountable and contactable

Status: ACCEPTABLE

---

4. ERC20 CONFORMITY
```

Slither-check-erc Results

Command: slither-check-erc . RewardToken --erc erc20

✓ transfer returns bool ✓ transferFrom returns bool ✓ name, decimals, symbol present ✓ decimals returns uint8 (value: 18) ✓ Race condition mitigated (increaseAllowance/decreaseAllowance)

Status: FULLY COMPLIANT

slither-prop Test Results

Command: slither-prop . --contract RewardToken

Generated 12 properties, all passed: ✓ Transfer doesn't change total supply ✓ Allowance correctly updates ✓ Balance updates match transfer amounts ✓ No balance manipulation possible [... 8 more properties ...]

Echidna fuzzing: 50,000 runs, no violations ✓

Status: EXCELLENT

---

5. WEIRD TOKEN PATTERN ANALYSIS

Integration Safety Check

Your Protocol Integrates 5 External Tokens:

1. USDT (0xdac17f9...)
2. USDC (0xa0b86991...)
3. DAI (0x6b175474...)
4. WETH (0xc02aaa39...)
5. UNI (0x1f9840a8...)

Critical Issues Found

 Pattern 7.2: Missing Return Values Found in: USDT integration File: contracts/Vault.sol:156

```
IERC20(usdt).transferFrom(msg.sender, address(this), amount);
// No return value check! USDT doesn't return bool
```

Risk: Silent failures on USDT transfers **Exploit:** User appears to deposit, but no tokens moved **Fix:** Use OpenZeppelin SafeERC20 wrapper

---

 Pattern 7.3: Fee on Transfer Risk for: Any token with transfer fees File: contracts/Vault.sol:170

```
uint256 balanceBefore = IERC20(token).balanceOf(address(this));
token.transferFrom(msg.sender, address(this), amount);
shares = amount * exchangeRate;  // WRONG! Should use actual received amount
```

Risk: Accounting mismatch if token takes fees **Exploit:** User credited more shares than tokens deposited **Fix:** Calculate shares from `balanceAfter - balanceBefore`

Known Non-Standard Token Handling

✓ **USDC:** Properly handled (SafeERC20, 6 decimals accounted for) ⚠ **DAI:** permit() function not used (opportunity for gas savings) ✗ **USDT:** Missing return value not handled (CRITICAL) ✓ **WETH:** Standard wrapper, properly handled ⚠ **UNI:** Large approval handling not checked (reverts >= 2^96)

[... Additional sections for remaining analysis categories ...]

```
For complete report template and deliverables format, see
[REPORT_TEMPLATES.md](resources/REPORT_TEMPLATES.md).

---

## Rationalizations (Do Not Skip)

| Rationalization | Why It's Wrong | Required Action |
|-----------------|----------------|-----------------|
| "Token looks standard, ERC20 checks pass" | 20+ weird token patterns exist beyond ERC20
compliance | Check ALL weird token patterns from database (missing return, revert on zero,
hooks, etc.) |
| "Slither shows no issues, integration is safe" | Slither detects some patterns, misses
integration logic | Complete manual analysis of all 5 token integration criteria |
```

"No fee-on-transfer detected, skip that check"	Fee-on-transfer can be owner-controlled or conditional	Test all transfer scenarios, check for conditional fee logic
"Balance checks exist, handling is safe"	Balance checks alone don't protect against all weird tokens	Verify safe transfer wrappers, revert handling, approval patterns
"Token is deployed by reputable team, assume standard"	Reputation doesn't guarantee standard behavior	Analyze actual code and on-chain behavior, don't trust assumptions
"Integration uses OpenZeppelin, must be safe"	OpenZeppelin libraries don't protect against weird external tokens	Verify defensive patterns around all external token calls
"Can't run Slither, skipping automated analysis"	Slither provides critical ERC conformance checks	Manually verify all slither-check-erc criteria or document why blocked
"This pattern seems fine"	Intuition misses subtle token integration bugs	Systematically check all 20+ weird token patterns with code evidence

---

Deliverables

When analysis is complete, I'll provide:

1. **Compliance Checklist** - Checkboxes for all assessment categories
2. **Weird Token Pattern Analysis** - Presence/absence of all 24 patterns with risk levels and evidence
3. **On-chain Analysis Report** (if applicable) - Holder distribution, exchange listings, configuration
4. **Integration Safety Assessment** (if applicable) - Safe transfer usage, defensive patterns, weird token handling
5. **Prioritized Recommendations** - CRITICAL/HIGH/MEDIUM/LOW issues with specific fixes

Complete deliverable templates available in [REPORT_TEMPLATES.md](resources/REPORT_TEMPLATES.md).

Ready to Begin

What I'll need:
- Your codebase
- Context: Token implementation or integration?
- Token type: ERC20, ERC721, or both?
- Contract address (if deployed and want on-chain analysis)
- RPC endpoint (if querying on-chain)

```
Let's analyze your token implementation or integration for security risks!
```

Revision #5
Created 2026-02-18 08:40:04 UTC by John
Updated 2026-07-05 20:01:41 UTC by John