/solana-vulnerability-scanner

Source: `~/.claude/skills/tob-building-secure-contracts/skills/solana-vulnerability-scanner/SKILL.md`

---

name: solana-vulnerability-scanner
description: Scans Solana programs for 6 critical vulnerabilities including arbitrary CPI, improper PDA validation, missing signer/ownership checks, and sysvar spoofing. Use when auditing Solana/Anchor programs.

Solana Vulnerability Scanner

1. Purpose

Systematically scan Solana programs (native and Anchor framework) for platform-specific security vulnerabilities related to cross-program invocations, account validation, and program-derived

addresses. This skill encodes 6 critical vulnerability patterns unique to Solana's account model.

2. When to Use This Skill

- Auditing Solana programs (native Rust or Anchor)
- Reviewing cross-program invocation (CPI) logic
- Validating program-derived address (PDA) implementations
- Pre-launch security assessment of Solana protocols
- Reviewing account validation patterns
- Assessing instruction introspection logic

3. Platform Detection

File Extensions & Indicators

- **Rust files**: `.rs`

Language/Framework Markers

```
// Native Solana program indicators
use solana_program::{
    account_info::AccountInfo,
    entrypoint,
    entrypoint::ProgramResult,
    pubkey::Pubkey,
    program::invoke,
    program::invoke_signed,
};

entrypoint!(process_instruction);

// Anchor framework indicators
use anchor_lang::prelude::*;

#[program]
pub mod my_program {
    pub fn initialize(ctx: Context<Initialize>) -> Result<()> {
```

```
        // Program logic
    }
}

#[derive(Accounts)]
pub struct Initialize<'info> {
    #[account(mut)]
    pub authority: Signer<'info>,
}

// Common patterns
AccountInfo, Pubkey
invoke(), invoke_signed()
Signer<'info>, Account<'info>
#[account(...)] with constraints
seeds, bump
```

Project Structure

- `programs/*/src/lib.rs` - Program implementation
- `Anchor.toml` - Anchor configuration
- `Cargo.toml` with `solana-program` or `anchor-lang`
- `tests/` - Program tests

Tool Support

- **Trail of Bits Solana Lints**: Rust linters for Solana
- Installation: Add to Cargo.toml
- **anchor test**: Built-in testing framework
- **Solana Test Validator**: Local testing environment

---

4. How This Skill Works

When invoked, I will:

1. **Search your codebase** for Solana/Anchor programs
2. **Analyze each program** for the 6 vulnerability patterns
3. **Report findings** with file references and severity
4. **Provide fixes** for each identified issue

5. **Check account validation** and CPI security

---

5. Example Output

---

5. Vulnerability Patterns (6 Patterns)

I check for 6 critical vulnerability patterns unique to Solana. For detailed detection patterns, code examples, mitigations, and testing strategies, see VULNERABILITY_PATTERNS.md.

Pattern Summary:

1. **Arbitrary CPI** ⚠ CRITICAL - User-controlled program IDs in CPI calls
2. **Improper PDA Validation** ⚠ CRITICAL - Using create_program_address without canonical bump
3. **Missing Ownership Check** ⚠ HIGH - Deserializing accounts without owner validation
4. **Missing Signer Check** ⚠ CRITICAL - Authority operations without is_signer check
5. **Sysvar Account Check** ⚠ HIGH - Spoofed sysvar accounts (pre-Solana 1.8.1)
6. **Improper Instruction Introspection** ⚠ MEDIUM - Absolute indexes allowing reuse

For complete vulnerability patterns with code examples, see VULNERABILITY_PATTERNS.md.

5. Scanning Workflow

Step 1: Platform Identification

1. Verify Solana program (native or Anchor)
2. Check Solana version (1.8.1+ for sysvar security)
3. Locate program source (`programs/*/src/lib.rs`)
4. Identify framework (native vs Anchor)

Step 2: CPI Security Review

```
# Find all CPI calls
rg "invoke\(|invoke_signed\(" programs/
```

```
# Check for program ID validation before each
# Should see program ID checks immediately before invoke
```

For each CPI:

- [ ] Program ID validated before invocation
- [ ] Cannot pass user-controlled program accounts
- [ ] Anchor: Uses `Program<'info, T>` type

Step 3: PDA Validation Check

```
# Find PDA usage
rg "find_program_address|create_program_address" programs/
rg "seeds.*bump" programs/

# Anchor: Check for seeds constraints
rg "#\[account.*seeds" programs/
```

For each PDA:

- [ ] Uses `find_program_address()` or Anchor `seeds` constraint
- [ ] Bump seed stored and reused
- [ ] Not using user-provided bump

Step 4: Account Validation Sweep

```
# Find account deserialization
rg "try_from_slice|try_deserialize" programs/

# Should see owner checks before deserialization
rg "\.owner\s*==|\.owner\s*!=" programs/
```

For each account used:

- [ ] Owner validated before deserialization
- [ ] Signer check for authority accounts
- [ ] Anchor: Uses `Account<'info, T>` and `Signer<'info>`

Step 5: Instruction Introspection Review

```
# Find instruction introspection usage
rg "load_instruction_at|load_current_index|get_instruction_relative" programs/

# Check for checked versions
rg "load_instruction_at_checked|load_current_index_checked" programs/
```

- [ ] Using checked functions (Solana 1.8.1+)
- [ ] Using relative indexing
- [ ] Proper correlation validation

Step 6: Trail of Bits Solana Lints

```
# Add to Cargo.toml
[dependencies]
solana-program = "1.17"  # Use latest version

[lints.clippy]
# Enable Solana-specific lints
# (Trail of Bits solana-lints if available)
```

---

6. Reporting Format

Finding Template

```
## [CRITICAL] Arbitrary CPI - Unchecked Program ID

**Location**: `programs/vault/src/lib.rs:145-160` (withdraw function)

**Description**:
The `withdraw` function performs a CPI to transfer SPL tokens without validating that the provided `token_program` account is actually the SPL Token program. An attacker can provide a malicious program that appears to perform a transfer but actually steals tokens or performs unauthorized actions.

**Vulnerable Code**:
```rust
```
```

```
// lib.rs, line 145
pub fn withdraw(ctx: Context<Withdraw>, amount: u64) -> Result<()> {
    let token_program = &ctx.accounts.token_program;

    // WRONG: No validation of token_program.key()!
    invoke(
        &spl_token::instruction::transfer(...),
        &[
            ctx.accounts.vault.to_account_info(),
            ctx.accounts.destination.to_account_info(),
            ctx.accounts.authority.to_account_info(),
            token_program.to_account_info(),  // UNVALIDATED
        ],
    )?;
    Ok(())
}
```

Attack Scenario:

1. Attacker deploys malicious "token program" that logs transfer instruction but doesn't execute it
2. Attacker calls withdraw() providing malicious program as token_program
3. Vault's authority signs the transaction
4. Malicious program receives CPI with vault's signature
5. Malicious program can now impersonate vault and drain real tokens

Recommendation: Use Anchor's `Program<'info, Token>` type:

```
use anchor_spl::token::{Token, Transfer};

#[derive(Accounts)]
pub struct Withdraw<'info> {
    #[account(mut)]
    pub vault: Account<'info, TokenAccount>,
    #[account(mut)]
    pub destination: Account<'info, TokenAccount>,
    pub authority: Signer<'info>,
    pub token_program: Program<'info, Token>,  // Validates program ID automatically
}

pub fn withdraw(ctx: Context<Withdraw>, amount: u64) -> Result<()> {
```

```
    let cpi_accounts = Transfer {
        from: ctx.accounts.vault.to_account_info(),
        to: ctx.accounts.destination.to_account_info(),
        authority: ctx.accounts.authority.to_account_info(),
    };

    let cpi_ctx = CpiContext::new(
        ctx.accounts.token_program.to_account_info(),
        cpi_accounts,
    );

    anchor_spl::token::transfer(cpi_ctx, amount)?;
    Ok(())
}
```

References:

- building-secure-contracts/not-so-smart-contracts/solana/arbitrary_cpi
- Trail of Bits lint: `unchecked-cpi-program-id`

```
---

## 7. Priority Guidelines

### Critical (Immediate Fix Required)
- Arbitrary CPI (attacker-controlled program execution)
- Improper PDA validation (account spoofing)
- Missing signer check (unauthorized access)

### High (Fix Before Launch)
- Missing ownership check (fake account data)
- Sysvar account check (authentication bypass, pre-1.8.1)

### Medium (Address in Audit)
- Improper instruction introspection (logic bypass)

---

## 8. Testing Recommendations
```

```
### Unit Tests
```rust
#[cfg(test)]
mod tests {
    use super::*;

    #[test]
    #[should_panic]
    fn test_rejects_wrong_program_id() {
        // Provide wrong program ID, should fail
    }

    #[test]
    #[should_panic]
    fn test_rejects_non_canonical_pda() {
        // Provide non-canonical bump, should fail
    }

    #[test]
    #[should_panic]
    fn test_requires_signer() {
        // Call without signature, should fail
    }
}
```

Integration Tests (Anchor)

```
import * as anchor from "@coral-xyz/anchor";

describe("security tests", () => {
  it("rejects arbitrary CPI", async () => {
    const fakeTokenProgram = anchor.web3.Keypair.generate();

    try {
      await program.methods
        .withdraw(amount)
        .accounts({
          tokenProgram: fakeTokenProgram.publicKey, // Wrong program
```
```

```
        })
        .rpc();

      assert.fail("Should have rejected fake program");
    } catch (err) {
      // Expected to fail
    }
  });
});
```

Solana Test Validator

```
# Run local validator for testing
solana-test-validator

# Deploy and test program
anchor test
```

---

9. Additional Resources

- **Building Secure Contracts**: `building-secure-contracts/not-so-smart-contracts/solana/`
- **Trail of Bits Solana Lints**: https://github.com/trailofbits/solana-lints
- **Anchor Documentation**: https://www.anchor-lang.com/
- **Solana Program Library**: https://github.com/solana-labs/solana-program-library
- **Solana Cookbook**: https://solanacookbook.com/

---

10. Quick Reference Checklist

Before completing Solana program audit:

CPI Security (CRITICAL):

- ☐ ALL CPI calls validate program ID before `invoke()`
- ☐ Cannot use user-provided program accounts
- ☐ Anchor: Uses `Program<'info, T>` type

PDA Security (CRITICAL):

- [ ] PDAs use `find_program_address()` or Anchor `seeds` constraint
- [ ] Bump seed stored and reused (not user-provided)
- [ ] PDA accounts validated against canonical address

Account Validation (HIGH):

- [ ] ALL accounts check owner before deserialization
- [ ] Native: Validates `account.owner == expected_program_id`
- [ ] Anchor: Uses `Account<'info, T>` type

Signer Validation (CRITICAL):

- [ ] ALL authority accounts check `is_signer`
- [ ] Native: Validates `account.is_signer == true`
- [ ] Anchor: Uses `Signer<'info>` type

Sysvar Security (HIGH):

- [ ] Using Solana 1.8.1+
- [ ] Using checked functions: `load_instruction_at_checked()`
- [ ] Sysvar addresses validated

Instruction Introspection (MEDIUM):

- [ ] Using relative indexes for correlation
- [ ] Proper validation between related instructions
- [ ] Cannot reuse same instruction across multiple calls

Testing:

- [ ] Unit tests cover all account validation
- [ ] Integration tests with malicious inputs
- [ ] Local validator testing completed
- [ ] Trail of Bits lints enabled and passing

Revision #4
Created 2026-02-18 08:40:04 UTC by John
Updated 2026-05-31 20:01:42 UTC by John