Halborn CTF – Smart Contract Security Audit Summary

This document summarizes the vulnerabilities identified and demonstrated through unit testing of the HalbornCTF Substrate pallets.

pallet-pause (Pause Pallet)

Exploit 1: toggle() Function Doesn't Actually Toggle State

- **Test:** `toggle_bug_does_not_toggle()`
- **Issue:** The `toggle()` function on line 44 sets the paused state to `Self::paused()` (the current state) instead of `!Self::paused()` (the toggled state).
- **Code Location:** `pallets/pause/src/lib.rs:44`
- **Exploit:** When `toggle()` is called, it reads the current paused state and sets it to the same value, effectively doing nothing. The function should toggle between `true` and `false`, but instead it maintains the current state.
- **Impact:** The toggle functionality is completely broken. Administrators cannot reliably toggle the pause state, leading to potential denial of service or inability to resume operations after pausing.
- **Severity:** Critical

Exploit 2: unpause() Function Doesn't Actually Unpause

- **Test:** `unpause_bug_does_not_unpause()`
- **Issue:** The `unpause()` function on line 70 sets the paused state to `Self::paused()` (the current state, likely `true`) instead of `false`.
- **Code Location:** `pallets/pause/src/lib.rs:70`
- **Exploit:** When `unpause()` is called, it reads the current paused state (which is `true` when paused) and sets it to the same value, keeping the system paused. The function emits an event indicating `false` (unpaused), but the actual state remains `true` (paused).
- **Impact:** Once the system is paused, it cannot be unpaused using the `unpause()` function. The system remains permanently paused, causing complete denial of service. The event emission is misleading as it indicates unpause while the state remains paused.
- **Severity:** Critical

pallet-allocations (Allocation Pallet)

Exploit 1: Allocations Bypass Pause Check

- **Test:** `allocations_bypass_pause_check()`
- **Issue:** The `allocate_coins()` function does not check if the system is paused before allowing allocations, even though the pallet depends on `pallet_pause::Config`.
- **Code Location:** `pallets/allocations/src/lib.rs:76-104`
- **Exploit:** When the pause pallet sets the system to paused state, the allocations pallet continues to function normally. An oracle can continue allocating coins even when the system should be paused, bypassing the emergency shutdown mechanism.
- **Impact:** The emergency pause mechanism is ineffective for the allocations pallet. During an emergency shutdown, allocations should be halted, but they continue to operate, potentially allowing unauthorized or unwanted allocations during a critical situation.
- **Severity:** High

Overall Observations

The audit revealed critical logic errors in the pause pallet that completely break its intended functionality. Both `toggle()` and `unpause()` functions fail to modify the pause state correctly, making the emergency shutdown mechanism unreliable.

Additionally, the allocations pallet does not respect the pause state, allowing critical operations to continue even when the system should be paused. This defeats the purpose of having a pause mechanism in the first place.

These issues demonstrate the importance of thorough testing of state management logic and ensuring that dependent pallets properly check pause states before executing critical operations.

Test Coverage

Three vulnerabilities were identified across the two pallets. Test cases were created to demonstrate each vulnerability:

Pause Pallet (pallet-pause) - 6 tests total:

- `root_pause` (functional test)
- `pause_origin_unpause` (functional test)
- `bad_origin_fails` (functional test)
- `toggle_bug_does_not_toggle` (vulnerability test)
- `unpause_bug_does_not_unpause` (vulnerability test)
- `__construct_runtime_integrity_test::runtime_integrity_tests` (framework test)

Allocation Pallet (pallet-allocations) - 5 tests total:

- `non_oracle_can_not_trigger_allocation` (functional test)
- `oracle_triggers_allocation` (functional test)
- `allocate_the_right_amount_of_coins_to_everyone` (functional test)

- `allocations_bypass_pause_check` (vulnerability test)
- `__construct_runtime_integrity_test::runtime_integrity_tests` (framework test)

Status: All issues demonstrated successfully via unit tests using Cargo. All tests passing.

For instructions on running the tests, see run.md (./run.md).