Halborn CTF – Smart Contract Security Audit Summary

This document summarizes the vulnerabilities exploited through unit testing of the HalbornCTF NEAR contracts.

halborn-near-ctf (Main Contract)

Exploit 1: resume() Function Sets Incorrect Status

- **Test:** `test_resume_bug_contract_stays_paused`
- **Issue:** The `resume()` function incorrectly sets the contract status to `Paused` instead of `Working`.
- **Exploit:** After calling `pause()` and then `resume()`, the contract remains permanently paused because `resume()` sets the status to `Paused` rather than `Working`.
- **Impact:** Contract becomes permanently unusable after resume() is called. All functions that check `not_paused()` will continue to fail, resulting in complete denial of service.
- **Severity:** Critical

Exploit 2: mint_tokens() Loses Tokens for Unregistered Users

- **Test:** `test_mint_tokens_bug_unregistered_user`, `test_mint_tokens_bug_token_loss`
- **Issue:** The `mint_tokens()` function increases the total supply when minting to a user, but only adds tokens to the user's balance if the user is already registered in the accounts map.
- **Exploit:** When minting tokens to an unregistered user, the total supply increases but no tokens are added to the user's balance. The tokens are effectively lost as the supply is inflated without corresponding user balances.
- **Impact:** Permanent token loss when minting to unregistered users. Supply inflation without corresponding user balances creates accounting inconsistencies and economic attack vectors.
- **Severity:** Critical

Exploit 3: Metadata Functions Consume Metadata

- **Test:** `test_metadata_consumption_bug`, `test_metadata_consumption_bug_multiple_functions`, `test_metadata_consumption_bug_ft_metadata`
- **Issue:** The `get_symbol()`, `get_name()`, and `get_decimals()` functions use `token_metadata.take()` which consumes the metadata from storage.
- **Exploit:** After the first call to any of these functions, the metadata is removed from storage. Subsequent calls to these functions or `ft_metadata()` will fail because the metadata no longer exists.

- **Impact:** Metadata becomes inaccessible after first call. External integrations querying metadata will fail, and the contract appears broken to users after metadata consumption.
- **Severity:** High

halborn-near-ctf-associated-contract (Event Registration Contract)

Exploit 1: make_event_offline() Has No Effect

- **Test:** `test_make_event_offline_bug_no_effect`
- **Issue:** The `make_event_offline()` function attempts to modify the `is_live` field of an event, but it modifies a local copy returned by `get()` rather than the stored value.
- **Exploit:** The modified event is never stored back to the LookupMap, so the function has no effect. Events remain live even after being "taken offline", allowing users to continue registering for events that should be offline.
- **Impact:** Events cannot be properly taken offline. Access control bypass allows users to register for "offline" events. Function appears to work but has no actual effect.
- **Severity:** Critical

halborn-near-ctf-staking (Staking Contract)

Exploit 1: unstake() Logic Error and Accounting Inconsistency

- **Test:** `test_unstake_bug_when_balance_reaches_zero`, `test_unstake_bug_logic_inconsistency`, `test_unstake_edge_case_saturating_sub`
- **Issue:** The `unstake()` function has inconsistent refund logic and does not validate that the unstake amount is less than or equal to the user's balance.
- **Exploit:** When unstaking brings the balance to zero, it refunds the old balance instead of the unstaked amount. When attempting to unstake more than the user's balance, `saturating_sub` causes `total_staked` to be reduced by the requested amount rather than the actual balance, creating accounting inconsistencies.
- **Impact:** Accounting inconsistency in `total_staked` when unstaking more than balance. Logic error in refund amounts when balance reaches zero. Potential for incorrect accounting if other functions rely on `total_staked`.
- **Severity:** Critical

Overall Observations

- State management errors are present across multiple contracts, particularly in functions that modify contract state.

- Functions that modify storage must ensure changes are persisted back to storage rather than modifying local copies.
- Token minting functions must register users before adding tokens to prevent permanent token loss.
- Metadata access patterns should use read-only operations to prevent accidental consumption of stored data.
- Unstaking and withdrawal functions must validate amounts and maintain consistent accounting across all state variables.

Test Coverage

Five vulnerabilities were identified across the three contracts. Multiple test cases were created for several findings to demonstrate different attack vectors, edge cases, and impact scenarios. This comprehensive testing approach ensures each vulnerability is thoroughly validated and its full scope is understood.

All Test Cases:

Main Contract (halborn-near-ctf): - test_resume_bug_contract_stays_paused - test_resume_bug_cannot_use_contract - test_mint_tokens_bug_unregistered_user - test_mint_tokens_bug_token_loss - test_metadata_consumption_bug - test_metadata_consumption_bug_multiple_functions - test_metadata_consumption_bug_ft_metadata

Associated Contract (halborn-near-ctf-associated-contract): - test_make_event_offline_bug_no_effect

Staking Contract (halborn-near-ctf-staking): - test_stake_and_unstake - test_unstake_bug_when_balance_reaches_zero - test_unstake_bug_logic_inconsistency - test_unstake_edge_case_saturating_sub

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