Halborn CTF – Smart Contract Security Audit Summary

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

Test Configuration - Version Alignment

The OpenZeppelin contracts versions were aligned to match the reference implementation: - `openzeppelin-contracts`: v5.0.0 (for ERC1967Utils support in tests) - `openzeppelin-contracts-upgradeable`: v4.9.5 (for compatibility with test contracts)

This version combination ensures that all required libraries and interfaces are available.

HalbornToken.sol

Exploit 1: UUPS Upgrade Bypass

- **Test:** `test_vulnerableUUPSupgrade()`
- **Issue:** `_authorizeUpgrade(address)` is empty.
- **Exploit:** Any address can call `upgradeTo(...)` and take over the contract.
- **Impact:** Full contract compromise.
- **Severity:** Critical

Exploit 2: Loans Address Manipulation Attack

- **Test:** `test_setLoansAddress()`
- **Issue:** `setLoans(address)` lacks access control and can be called by any address after a malicious upgrade.
- **Exploit:** A malicious implementation is upgraded in; attacker reinitializes and calls `setLoans()` to assign themselves as the loan authority.
- **Impact:** Attacker gains permanent mint/burn privileges while locking out the original owner & can no longer change loans address.
- **Severity:** Critical

Exploit 3: Unrestricted Minting (`mintToken`)

- **Test:** `test_unlimitedMint()`
- **Issue:** The `mintToken(address, uint256)` function can be called by any address previously set via `setLoans()`, without validation.
- **Exploit:** A malicious contract registers itself and mints tokens arbitrarily.
- **Impact:** Infinite token supply, economic breakdown.
- **Severity:** Critical

Exploit 4: Unrestricted Burning (`burnToken`)

- **Test:** `test_unlimitedBurn()`
- **Issue:** `burnToken` can be used to destroy tokens from any address.
- **Exploit:** A fake loan contract burns a user's tokens without permission.
- **Impact:** Token holder funds loss.
- **Severity:** Critical

HalbornLoans.sol

Exploit 1: UUPS Upgrade Bypass

- **Test:** `test_vulnerableUUPSupgrade()`
- **Issue:** `_authorizeUpgrade(address)` is empty.
- **Exploit:** Any address can call `upgradeTo(...)` and take over the contract.
- **Impact:** Full contract compromise and loan logic hijacking.
- **Severity:** Critical

Exploit 2: Infinite Token Minting via Malicious Loan Contract

- **Test:** `test_vulnerableLoanContractReksTokenMint()`
- **Issue:** Token contract trusts `loans` address for minting; no validation after upgrade.
- **Exploit:** Attacker upgrades to a malicious contract and mints unlimited tokens via `token.mintToken(...)`.
- **Impact:** Infinite token inflation, economic collapse.
- **Severity:** Critical

Exploit 3: Arbitrary Token Burning via Loan Contract

- **Test:** `test_vulnerableLoanContractReksTokenBurn()`
- **Issue:** Token contract allows `loans` address to burn tokens from any user.
- **Exploit:** Malicious loan contract calls `token.burnToken(...)` on users like Alice.
- **Impact:** Irreversible user fund destruction.
- **Severity:** Critical

Exploit 4: Reentrancy in NFT Collateral Withdrawal

- **Test:** `test_Reentrancy()`
- **Issue:** `withdrawCollateral()` lacks reentrancy protection.
- **Exploit:** Re-enter during `onERC721Received` callback to withdraw multiple NFTs and call `getLoan(...)` before state updates.
- **Impact:** Double NFT withdrawal and max loan drain.
- **Severity:** Critical

Exploit 5: Insecure Loan Collateralization

- **Test:** Implicit in `test_Reentrancy()`, confirmed in reentrant logic
- **Issue:** Loan amount is based on collateral count without lock mechanism or atomicity.
- **Exploit:** Reentrancy alters collateral count mid-calculation, inflating borrowable tokens.
- **Impact:** Collateral fraud, overdrawing loans.
- **Severity:** Medium

HalbornNFT.sol

Exploit 1: Merkle Root Manipulation

- **Test:** `test_setMerkelRoot()`
- **Issue:** `setMerkleRoot()` has no access control.
- **Exploit:** Any address can replace the Merkle root, bypassing the whitelist mechanism entirely.
- **Impact:** Whitelist bypass, unauthorized users gain airdrop minting access.
- **Severity:** Critical

Exploit 2: Unlimited Airdrop Minting

- **Test:** `test_setMintUnlimited()`
- **Issue:** Once the Merkle root is manipulated, crafted proofs can be used repeatedly.
- **Exploit:** Attacker mints unlimited NFTs using a custom Merkle tree and valid proofs.
- **Impact:** NFT supply inflation, ecosystem collapse.
- **Severity:** Critical

Exploit 3: UUPS Upgrade Bypass

- **Test:** `test_vulnerableUUPSupgrade()`
- **Issue:** `_authorizeUpgrade()` is left empty, allowing anyone to upgrade the contract.
- **Exploit:** Attacker upgrades to a malicious implementation, reinitializes, and gains control.
- **Impact:** Full protocol takeover — including minting logic, pricing, and ETH withdrawal.
- **Severity:** Critical

Exploit 4: Price Manipulation

- **Test:** `test_setPrice()`
- **Issue:** NFT price is settable via `initialize()` after malicious upgrade.
- **Exploit:** Attacker sets custom NFT price via reinitialization.

- **Impact:** Undermines fair pricing model, opens door to abuse or griefing.
- **Severity:** Critical

Exploit 5: ETH Drainage via Malicious Upgrade

- **Test:** `test_stealETH()`
- **Issue:** ETH stored in contract can be drained post-upgrade through a malicious `withdrawETH()` function.
- **Exploit:** Attacker upgrades to a version with `withdrawETH()` and drains the full contract balance.
- **Impact:** Complete ETH theft, user losses, contract bankruptcy.
- **Severity:** Critical

Code Fixes Applied

During the audit process, several critical bugs were identified and fixed to ensure the test suite runs correctly:

HalbornNFT.sol - Fixed Inverted Logic in mintAirdrops

The `mintAirdrops` function had inverted logic in the token existence check. The original code used `require(_exists(id), "Token already minted")` which would only allow minting if the token already existed, preventing any new mints. This was corrected to `require(!_exists(id), "Token already minted")` to properly check that the token does not exist before minting.

HalbornLoans.sol - Added Missing ERC721 Receiver Implementation

The `HalbornLoans` contract was missing the `onERC721Received` function, which is required for contracts that receive NFTs via `safeTransferFrom`. Without this implementation, the `depositNFTCollateral` function would fail when attempting to transfer NFTs to the contract. The function was added to match the reference implementation, allowing the contract to properly receive NFTs during collateral deposits.

Overall Observations

- UUPS vulnerabilities affect every contract.
- Token mint/burn control must not be externally assigned without proper validation.
- Reentrancy and withdrawal logic should follow best practices.
- Whitelist minting must include per-address + per-ID limitations.
- Contracts receiving NFTs must implement the ERC721Receiver interface.

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