HorseStore Audit Report

Version 1.0

César Escribano

January 18, 2024

HorseStore Audit Report

César Escribano

January 15, 2024

Prepared by: César Escribano (@ceseshi)

- Low
  - [L-1] (Huff) Incorrect control flow makes the contract return incorrect values for undefined functions
- Informational
  - [I-1] (Solidity, Huff) isHappyHorse() does not verify horse id, so the contract will return incorrect information
  - [I-2] (Huff) Implementation of ERC721 standard is incomplete, so some functions of the standard interface are not usable
  - [I-3] (Huff) Incorrect call to MINT_HORSE()
- Gas

Protocol Summary

This is a security review of First Flight #7: Horse Store, a public contest from CodeHawks.

The contract is a collection of NFT horses. The horses can be minted, fed and checked if they are happy. Anyone can feed any horse, and if a horse is not fed in a period of 24 hours, it becomes unhappy.

Two implementations are provided: a reference implementation in Solidity and a optimized implementation in Huff.

Disclaimer

The auditor makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the auditor is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M

		Impact		
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

Repository

https://github.com/Cyfrin/2024-01-horse-store

Commit Hash

```
01bce4f0a2271c4105ee7c9121b27fe7973b0eaf
```

Scope

```
#-- HorseStore.huff
#-- HorseStore.sol
#-- IHorseStore.sol
```

Roles

Minter: A user that mints a horse

Executive Summary

This audit took place in January 2024, over 5 days, totalling 25 hours. The tools used were Visual Studio Code and Foundry for Linux.

Issues found

Severity	Number of issues
High	4
Medium	2
Low	1
Info	3
Total	10

Findings

High

[H-1] (Huff) Incorrect loading of totalSupply from storage makes it impossible to mint more than one horse

Description: The MINT_HORSE Huff macro is passing TOTAL_SUPPLY as the value for tokenId (which is zero), instead of the stored value of totalSupply, so the tokenId to mint is always zero.

Vulnerability Details

https://github.com/Cyfrin/2024-01-horse-store/blob/01bce4f0a2271c4105ee7c9121b27fe7973b0eaf/src/HorseStore.hu

In line 75 of HorseStore.huff, the value of the TOTAL_SUPPLY pointer is being passed as total supply

```
1 #define macro MINT_HORSE() = takes (0) returns (0) {
2     [TOTAL_SUPPLY] // [TOTAL_SUPPLY]
3 @>  caller         // [msg.sender, TOTAL_SUPPLY]
4     _MINT()        // []
5     stop           // []
6 }
```

Impact: Only one horse can be minted, subsequent mints will revert.

Proof of Concept: Make a test to mint two horses and expect a revert in the second mint.

```
1 function testMultipleMintFailsHuff() public {
2     vm.startPrank(user);
3     horseStore.mintHorse();
4     vm.expectRevert();
5     horseStore.mintHorse();
6 }
```

Run Huff test

```
forge test --mc HorseStoreHuff --mt testFeedingMakesHappyHorse
```

The test will pass, confirming that the second mint reverts.

```
Running 1 test for test/HorseStoreHuff.t.sol:HorseStoreHuff
[PASS] testMultipleMintFailsHuff() (gas: 60297)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.40s
```

Recommended Mitigation: Correctly load value of totalSupply

Load the value from the storage location

```
#define macro MINT_HORSE() = takes (0) returns (0) {
    [TOTAL_SUPPLY] // [TOTAL_SUPPLY]
-   caller         // [msg.sender, TOTAL_SUPPLY]
+   sload          // [totalSupply]
+   caller         // [msg.sender, totalSupply]
    _MINT()        // []
    stop           // []
}
```

[H-2] (Huff) Minting does not increment totalSupply, so only one horse can be minted

Description: The _MINT() Huff macro is not incrementing TOTAL_SUPPLY after transferring the new NFT, so it will always remain zero.

Vulnerability Details

https://github.com/Cyfrin/2024-01-horse-store/blob/01bce4f0a2271c4105ee7c9121b27fe7973b0eaf/src/HorseStore.hu

The _MINT() macro should increment TOTAL_SUPPLY after the transfer.

```
#define macro _MINT() = takes (2) returns (0) {
...
    // Emit the transfer event.
    __EVENT_HASH(Transfer)                          // [sig, from (0x00
       ), to, tokenId]
    0x00 0x00 log4                                  // []
@>
    // Continue Executing
    cont jump
...
}
```

Impact: Only one horse can be minted, subsequent mints will revert.

Proof of Concept: Make a test to mint two horses and expect a revert in the second mint.

```
function testMultipleMintFailsHuff() public {
    vm.startPrank(user);
    horseStore.mintHorse();
    vm.expectRevert();
    horseStore.mintHorse();
}
```

Run Huff test

```
forge test --mc HorseStoreHuff --mt testFeedingMakesHappyHorse
```

The test will pass, confirming that the second mint reverts.

```
Running 1 test for test/HorseStoreHuff.t.sol:HorseStoreHuff
[PASS] testMultipleMintFailsHuff() (gas: 60297)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.40s
```

Recommended Mitigation: Increment TOTAL_SUPPLY after mint.

Modify _MINT() macro

```
    // Emit the transfer event.
    __EVENT_HASH(Transfer)                          // [sig, from (0x00
        ), to, tokenId]
    0x00 0x00 log4                                  // []
+
+   // Increment TOTAL_SUPPLY
+   [TOTAL_SUPPLY]                                  // [TOTAL_SUPPLY,
    from (0x00), to, tokenId]
+   sload                                           // [totalSupply,
    from (0x00), to, tokenId]
+   0x01 add                                        // [totalSupply+1,
    from (0x00), to, tokenId]
+   [TOTAL_SUPPLY]                                  // [TOTAL_SUPPLY,
    totalSupply+1, from (0x00), to, tokenId]
+   sstore                                          // [from (0x00), to
    , tokenId]
+
    // Continue Executing
    cont jump
```

[H-3] (Huff) Improper logic in feedHorse() will make it fail at random times

Description: The FEED_HORSE macro has a condition to check if the block timestamp is a multiple of 17, and reverts in that case. This is improper behaviour and should be removed.

Vulnerability Details

https://github.com/Cyfrin/2024-01-horse-store/blob/01bce4f0a2271c4105ee7c9121b27fe7973b0eaf/src/HorseStore.hu

This code is incorrect and should be removed.

```
#define macro FEED_HORSE() = takes (0) returns (0) {
    timestamp               // [timestamp]
    0x04 calldataload       // [horseId, timestamp]
    STORE_ELEMENT(0x00)     // []

    // End execution
@>  0x11 timestamp mod
@>  endFeed jumpi
@>  revert
@>  endFeed:
    stop
}
```

Impact: The feedHorse function will fail randomly, breaking the rule that horses must be able to be fed at all times.

Proof of Concept: Make a test to feed a horse with a block.timestamp that will pass, and another block.timestamp that will fail.

```
function testFeedOnInvalidTimestamp() public {
    vm.warp(horseStore.HORSE_HAPPY_IF_FED_WITHIN());
    horseStore.mintHorse();

    // This will pass
    vm.warp(block.timestamp - block.timestamp % 0x10);
    horseStore.feedHorse(0);

    // This will fail
    vm.warp(block.timestamp - block.timestamp % 0x11);
    vm.expectRevert();
    horseStore.feedHorse(0);
}
```

Run Huff test

```
forge test --mc HorseStoreHuff --mt testFeedOnInvalidTimestamp
```

Test passes, confirming that the second feed reverts.

```
Running 1 test for test/HorseStoreHuff.t.sol:HorseStoreHuff
[PASS] testFeedOnInvalidTimestamp() (gas: 104974)
```

Recommended Mitigation: Remove the incorrect condition in FEED_HORSE.

```
    timestamp               // [timestamp]
    0x04 calldataload       // [horseId, timestamp]
    STORE_ELEMENT(0x00)     // []
```

```
4
5       // End execution
6   -   0x11 timestamp mod
7   -   endFeed jumpi
8   -   revert
9   -   endFeed:
10      stop
```

[H-4] (Huff) Incorrect time limit checking in IS_HAPPY_HORSE() will make a feeded horse return as unhappy

Description: The IS_HAPPY_HORSE macro is incorrectly comparing the elapsed time since feeding a horse and the the time limit for the horse to be happy, so it returns that the horse is unhappy.

Vulnerability Details

https://github.com/Cyfrin/2024-01-horse-store/blob/01bce4f0a2271c4105ee7c9121b27fe7973b0eaf/src/HorseStore.hu

This opcode is comparing if the time limit is lower than the time elapsed since feeding, which is incorrect.

```
1  #define macro IS_HAPPY_HORSE() = takes (0) returns (0) {
2      0x04 calldataload                    // [horseId]
3      LOAD_ELEMENT(0x00)                   // [horseFedTimestamp]
4      timestamp                            // [timestamp,
           horseFedTimestamp]
5      dup2 dup2                            // [timestamp,
           horseFedTimestamp, timestamp, horseFedTimestamp]
6      sub                                  // [timestamp -
           horseFedTimestamp, timestamp, horseFedTimestamp]
7      [HORSE_HAPPY_IF_FED_WITHIN_CONST]    // [HORSE_HAPPY_IF_FED_WITHIN,
           timestamp - horseFedTimestamp, timestamp, horseFedTimestamp]
8  @>  lt                                   // [HORSE_HAPPY_IF_FED_WITHIN <
       timestamp - horseFedTimestamp, timestamp, horseFedTimestamp]
9      start_return_true jumpi              // [timestamp,
           horseFedTimestamp]
10     eq                                   // [timestamp ==
           horseFedTimestamp]
11     start_return
12     jump
```

Impact: Horses that were fed are returned as unhappy, breaking the rule that a horse is happy when it is fed.

Proof of Concept: Make a test to feed a horse, warp time and check if it is happy.

```
1  function testHorseIsUnhappyAfterFeeding() public {
2      vm.warp(horseStore.HORSE_HAPPY_IF_FED_WITHIN());
```

```
    vm.prank(user);
    horseStore.mintHorse();

    vm.warp(block.timestamp + 3600);
    horseStore.feedHorse(0);

    vm.warp(block.timestamp + 3600);
    bool isHappyHorse = horseStore.isHappyHorse(0);

    assertEq(isHappyHorse, false);
}
```

Run Huff test

```
forge test --mc HorseStoreHuff --mt testFeedOnInvalidTimestamp
```

Test passes, confirming that the horse is unhappy after feeding.

```
Running 1 test for test/HorseStoreHuff.t.sol:HorseStoreHuff
[PASS] testHorseIsUnhappyAfterFeeding() (gas: 107151)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.50s
```

Recommended Mitigation: Correct the conditionals.

```
    0x04 calldataload                   // [horseId]
    LOAD_ELEMENT(0x00)                  // [horseFedTimestamp]
    timestamp                           // [timestamp,
        horseFedTimestamp]
    dup2 dup2                           // [timestamp,
        horseFedTimestamp, timestamp, horseFedTimestamp]
    sub                                 // [timestamp -
        horseFedTimestamp, timestamp, horseFedTimestamp]
    [HORSE_HAPPY_IF_FED_WITHIN_CONST]   // [HORSE_HAPPY_IF_FED_WITHIN,
        timestamp - horseFedTimestamp, timestamp, horseFedTimestamp]
-   lt                                  // [HORSE_HAPPY_IF_FED_WITHIN <
    timestamp - horseFedTimestamp, timestamp, horseFedTimestamp]
+   gt                                  // [HORSE_HAPPY_IF_FED_WITHIN >
    timestamp - horseFedTimestamp, timestamp, horseFedTimestamp]
    start_return_true jumpi             // [timestamp,
        horseFedTimestamp]
    eq                                  // [timestamp ==
        horseFedTimestamp]
    start_return
    jump
```

Medium

[M-1] (Solidity) No verification of horse id in feedHorse(), so any horse can be fed before mint

Description: The feedHorse() function does not verify if the horseId exists, so any user can feed any horse even if not minted yet.

Vulnerability Details

https://github.com/Cyfrin/2024-01-horse-store/blob/01bce4f0a2271c4105ee7c9121b27fe7973b0eaf/src/HorseStore.so

You should verify that the horse exists before feeding it.

```
    function feedHorse(uint256 horseId) external {
@>      horseIdToFedTimeStamp[horseId] = block.timestamp;
    }
```

Impact: Any new horse may already be fed at its creation, breaking the rule that only horse NFTs can be fed, and that a horse is happy only when it is fed.

Proof of Concept: Make a test to feed a horse with an arbitrary Id and check if it is happy.

```
function testCanFeedHorseBeforeMint() public {
    vm.warp(horseStore.HORSE_HAPPY_IF_FED_WITHIN());
    horseStore.feedHorse(0);

    vm.warp(block.timestamp + 3600);
    vm.prank(user);
    horseStore.mintHorse();
    bool isHappyHorse = horseStore.isHappyHorse(0);

    assertEq(isHappyHorse, true);
}
```

Run Solidity tests

```
forge test --mc HorseStoreSolidity --mt testCanFeedNonexistentHorse
```

Test passes, even though it receives a non-existent horse Id.

```
Running 1 test for test/HorseStoreSolidity.t.sol:HorseStoreSolidity
[PASS] testCanFeedNonexistentHorse() (gas: 27664)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 4.87ms
```

Recommended Mitigation: Verify that the horse exists, and throw a custom error if it does not. Do it also in isHappyHorse(), to avoid confusion for users.

```
     mapping(uint256 id => uint256 lastFedTimeStamp) public
        horseIdToFedTimeStamp;
```

```
+    error HorseDoesNotExist();
+
     constructor() ERC721(NFT_NAME, NFT_SYMBOL) {}
```

```
     function feedHorse(uint256 horseId) external {
+        if (_ownerOf(horseId) == address(0)) {
+            revert HorseDoesNotExist();
+        }
+
         horseIdToFedTimeStamp[horseId] = block.timestamp;
```

[M-2] (Huff) No verification of horse id in feedHorse(), so any horse can be fed before mint

Description: The FEED_HORSE() Huff macro does not verify if the horseId exists, so any user can feed any horse even if not minted.

Vulnerability Details

https://github.com/Cyfrin/2024-01-horse-store/blob/01bce4f0a2271c4105ee7c9121b27fe7973b0eaf/src/HorseStore.hu

You should verify that the horse exists before feeding it.

```
#define macro FEED_HORSE() = takes (0) returns (0) {
@>  timestamp               // [timestamp]
    0x04 calldataload       // [horseId, timestamp]
    STORE_ELEMENT(0x00)     // []
```

Impact: Any new horse may already be fed at its creation, breaking the rule that only horse NFTs can be fed, and that a horse is happy only when it is fed.

Proof of Concept: Make a test to feed a horse and mint it afterwards, then check if it is happy.

```
function testCanFeedHorseBeforeMint() public {
    vm.warp(horseStore.HORSE_HAPPY_IF_FED_WITHIN());
    horseStore.feedHorse(0);

    vm.warp(block.timestamp + 3600);
    vm.prank(user);
    horseStore.mintHorse();
    bool isHappyHorse = horseStore.isHappyHorse(0);

    assertEq(isHappyHorse, true);
}
```

Run Huff test

```
forge test --mc HorseStoreHuff --mt testCanFeedNonexistentHorse
```

Test passes, confirming that the horse was successfuly feeded before mint.

```
Running 1 test for test/HorseStoreHuff.t.sol:HorseStoreHuff
[PASS] testCanFeedHorseBeforeMint() (gas: 27457)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.52s
```

Recommended Mitigation: Verify that the horse exists, and throw a custom error if it does not.

```
    #define macro FEED_HORSE() = takes (0) returns (0) {
+   0x04 calldataload                                  // [tokenId]
+   [OWNER_LOCATION] LOAD_ELEMENT_FROM_KEYS(0x00)      // [owner]
+   // revert if owner is zero address/not minted
+   continue jumpi
+   NOT_MINTED(0x00)
+   continue:

    timestamp                 // [timestamp]
```

Low

[L-1] (Huff) Incorrect control flow makes the contract return incorrect values for undefined functions

Description: The control flow in MAIN macro is incorrectly implemented, so any function whose selector is not defined will be passed to GET_TOTAL_SUPPLY().

Vulnerability Details

https://github.com/Cyfrin/2024-01-horse-store/blob/01bce4f0a2271c4105ee7c9121b27fe7973b0eaf/src/HorseStore.hu

If no declared function matches the function selector called, it will end up in totalSupply:

```
    dup1 __FUNC_SIG(getApproved) eq getApproved jumpi
    dup1 __FUNC_SIG(isApprovedForAll) eq isApprovedForAll jumpi

    dup1 __FUNC_SIG(balanceOf) eq balanceOf jumpi
    dup1 __FUNC_SIG(ownerOf)eq ownerOf jumpi

@>  totalSupply:
        GET_TOTAL_SUPPLY()
    feedHorse:
        FEED_HORSE()
    isHappyHorse:
        IS_HAPPY_HORSE()
```

Impact: Calling any undefined function will always return totalSupply, which may cause other protocols to receive confusing information from the NFT collection.

Proof of Concept: Make a test for tokenByIndex() function, which is defined in the standard interface but not implemented in Huff.

This test mints one token and gets the token index, which should be zero but will be equal to totalSupply.

```
function testUnimplementedFunction() public {
    vm.prank(user);
    horseStore.mintHorse();
    uint256 token0index = horseStore.tokenByIndex(0);
    assertEq(token0index, horseStore.totalSupply());
}
```

Run Huff test

```
forge test --mc HorseStoreHuff --mt testUnimplementedFunction
```

The test passes confirming that tokenByIndex() is returning totalSupply()

```
Running 1 test for test/HorseStoreHuff.t.sol:HorseStoreHuff
[PASS] testUnimplementedFunction() (gas: 62709)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 2.63s
```

Recommended Mitigation: Revert if the called function is not implemented

```
    dup1 __FUNC_SIG(balanceOf) eq balanceOf jumpi
    dup1 __FUNC_SIG(ownerOf)eq ownerOf jumpi

+   // no match
+   0x00 dup1 revert
+
    totalSupply:
        GET_TOTAL_SUPPLY()
```

Informational

[I-1] (Solidity, Huff) isHappyHorse() does not verify horse id, so the contract will return incorrect information

Description: isHappyHorse() does not verify the horseId, so it will return false for any non-existent horse id.

Impact: External protocols can receive incorrect information from the NFT collection.

Proof of Concept: Make a test to feed a horse with an arbitrary Id.

```
function testAnyHorseIdCanBeChecked() public {
```

```
    vm.warp(horseStore.HORSE_HAPPY_IF_FED_WITHIN());
    assertEq(horseStore.isHappyHorse(1337), false);
}
```

Run Solidity and Huff tests

```
forge test --mt testCanFeedNonexistentHorse
```

Both test pass, even though they receive a non-existent horse Id.

```
Running 1 test for test/HorseStoreSolidity.t.sol:HorseStoreSolidity
[PASS] testAnyHorseIdCanBeChecked() (gas: 27664)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 4.87ms

Running 1 test for test/HorseStoreHuff.t.sol:HorseStoreHuff
[PASS] testAnyHorseIdCanBeChecked() (gas: 27457)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.52s
```

Recommended Mitigation: Verify that the horse exists, and throw a custom error if it does not.

Corrections for Solidity

```
    mapping(uint256 id => uint256 lastFedTimeStamp) public
        horseIdToFedTimeStamp;

+   error HorseDoesNotExist();
+
    constructor() ERC721(NFT_NAME, NFT_SYMBOL) {}
```

```
    function isHappyHorse(uint256 horseId) external view returns (bool
        ) {
+       if (_ownerOf(horseId) == address(0)) {
+           revert HorseDoesNotExist();
+       }
+
        if (horseIdToFedTimeStamp[horseId] <= block.timestamp -
            HORSE_HAPPY_IF_FED_WITHIN) {
```

Corrections for Huff

```
    #define macro IS_HAPPY_HORSE() = takes (0) returns (0) {
+   0x04 calldataload                               // [tokenId]
+   [OWNER_LOCATION] LOAD_ELEMENT_FROM_KEYS(0x00)   // [owner]
+   // revert if owner is zero address/not minted
+   continue jumpi
+   NOT_MINTED(0x00)
+   continue:
+
    0x04 calldataload                   // [horseId]
```

[I-2] (Huff) Implementation of ERC721 standard is incomplete, so some functions of the standard interface are not usable

Description: The Huff implementation lacks the safeTransferFrom(), tokenByIndex() and tokenOfOwnerByIndex() functions from the ERC721Enumerable interface, so they cannot be used.

Impact: External protocols may have issues interacting with the NFT collection.

Proof of Concept: The Huff implementation does not include the standard functions.

Recommended Mitigation: Add Huff implementations for safeTransferFrom(), tokenByIndex() and tokenOfOwnerByIndex(), or explicitly state that this collection doesn't support them.

[I-3] (Huff) Incorrect call to MINT_HORSE()

Description: The MAIN macro has an incorrect call to MINT_HORSE() at the end, which is never called, but should not exist there.

Impact: No impact on the protocol.

Recommended Mitigation: Remove the incorrect call.

```
    ownerOf:
        OWNER_OF()
-   MINT_HORSE()
}
```

Gas