PocketCows Farm

Smart Contract Audit Report

December 10, 2021

Introduction

1. About PocketCows Farm

Pocketcows is a collection of 10,000 one of kind Non-Fungible Tokens or NFTs.These tokens take the form of cows with a huge variety of themes, and each cow is randomly generated and unique. We've worked hard to provide more than 250 different attributes with varying rarity so that you know that your cow is unlike any other.

Visit https://pocketcows.farm/ to know more about.

2. About ImmuneBytes

ImmuneBytes is a security start-up to provide professional services in the blockchain space. The team has hands-on experience in conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and have a great understanding of DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, dydx.

The team has been able to secure 105+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-up with a detailed analysis of the system ensuring security and managing the overall project.

Visit http://immunebytes.com/ to know more about the services.

Documentation Details

The PocketCows team has provided the following doc for the purpose of audit:

1. https://github.com/Censored-Studios/PocketCows/blob/main/pocketcow.contract/README.md

Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract in order to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer in order to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

- Project Name: PocketCows Farm
- Contracts Name: ERC721Tradable.sol, PocketsCow.sol
- Languages: Solidity(Smart contract)
- Github commit/Smart Contract Address for audit: 0579566b1d3dff3a76e55edf28e898a7acac33fc
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:
    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Security Level References

Every issue in this report was assigned a severity level from the following:

High severity issues will bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Issues	High	Medium	Low
Open	-	2	2
Closed	-	-	-

High Severity Issues

No issues were found.

Medium Severity Issues

1. **_Potential Reentrancy attack**

   A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

   - The function ERC721Tradable.mintTokens(uint16) has external calls before state updates, so it is vulnerable to reentrancy attacks.

   - The function ERC721Tradable.reserveTokens(uint16,address) has external calls before state updates, so it is vulnerable to reentrancy attacks.

   Recommendation:

   - We recommend moving the statement `m_bLock = false;` before the external call `(bool bFinalSuccess, ) = m_adrSaleReceiver.call{value: msg.value}("");` for the aforementioned function to prevent reentrancy attacks.

   - We recommend moving the statement `incrementTokenId();` before the external call `_safeMint(adrReserveDest, newTokenId);` for the aforementioned function to prevent reentrancy attacks.

2. **Multiplication performed on the result of a division**

   The contract ERC721Tradable (Line#139-161) multiplication is done on the result of division.

   Method: `doAllBurnPayouts()`

```
uint256 valueForBurns = address(this).balance / 10.0;
uint256 unValuePer = valueForBurns / m_unBurntTokenTotal;

BurnPayoutTotal += unValuePer * m_unBurntTokenTotal;
```

`unValuePer` is the division operation between `valueForBurns` and `m_unBurntTokenTotal`. In line 156, `unValuePer` is multiplied with `m_unBurntTokenTotal` to calculate the `BurnPayoutTotal` which can cause precision issues due to lack of float-based operations in solidity.

Let us consider this example in solidity:

// Multiplication on a result of division
*uint c = (a / b) * 100000;*

// Division on the result of the multiplication
*uint d = (a * 100000) / b;*

// if a = 523 and b = 10
c = 5200000
d = 5230000

As seen in the above example, solidity integer division might truncate. As a result, performing division before multiplication can sometimes cause a loss of precision and value.

Similar for lines #149-151

Recommendation:
Consider ordering multiplication before division.

Low Severity Issues

1. **Undeclared identifier**
   The methods ERC721Tradable.burnToken(uint26) and ERC721Tradable.doAllBurnPayouts() has the undeclared variable (`m_unBurntTokenTotal`) which will prevent the smart contracts from compiling

   Recommendation:
   We recommend declaring the `m_unBurntTokenTotal` variable in the smart contract.

2. **Unnecessary require check**

```
    require(msg.sender != address(0), "Burn caller must not be null.");
```

   The `msg.sender` can never be `0x0` address.

   Recommendation:
   We recommend removing this check to save gas cost

Recommendations / Informational

1. **Unnecessary bool checks in require condition**
   `m_bSalesEnabled` variable will return a bool value, checking the equality will not be helpful

```
    require(m_bSalesEnabled == true, "Minting is currently unavailable.");
```

   Recommendation:
   We recommend removing the bool equality check

```
    require(m_bSalesEnabled, "Minting is currently unavailable.");
```

2. **Lack of Event Emissions for Significant Transaction**
   The functions affect the status of sensitive state variables and should be able to emit events as notifications

   Recommendation:
   Consider adding events for sensitive actions and emit them.

3. **Typo in the filename of contract ContextMixin**
   The filename of contract ContextMixin has a typo.

   Consider updating the name to ContextMixin instead of ContentMixin

4. **Unnecessary use of SafeMath**
   Solidity version 0.8 was released with SafeMath checks inbuilt, we can avoid using an explicit safe math library

5. **Public methods are only being used externally**
   ‘public’ functions that are never inherited by the contract should be declared ‘external’ to save gas.

   Recommendation:
   Make this method `contractURI` external

6. **Naming Conventions**
   The contract follows consistent naming conventions for public variables but not for private variables.

   Examples:

   bool public m_bSalesEnabled = true;

```
uint16 private m_unCurrTokenId = 0;

uint16 public m_TotalCowsMinted = 0;
```

Recommendation:
Consider adding "_" before the private variable name
Example:

```
uint16 private _m_unCurrTokenId = 0;
```

7. **Access visibility for variables and mappings**
   Add access visibility to the state variables and mappings which are private in smart contract

   Recommendation:
   Consider adding the access visibility "private" for private variables
   Example: uint16 private BurntTokenTotal;

8. **Missing method and state variable documentation**

   State variables and methods have no documentation and or comments.

   Recommendation:
   Consider adding comments for public variables and methods.

9. **Missing implementation of the method (Payable)**

   The ERC721Tradable.receive() method has no implementation, and that is payable to which can lead to some vulnerabililties

   Recommendation:
   We recommend adding `revert()` method with "Not implemented" so that users can save gas.

10. **Inconsistent Natspec**
    According to the solidity documentation, the smart contract should follow the Natspec.

    Recommendation:
    Consider moving all the getters at the end of the smart contract

11. Regulate the order of require checks

The order of require check can be updated to check the setting of payment collection

Recommendation:
We recommend moving the,

```
      require(m_adrSaleReceiver != address(0), "Payment collection is not 
setup.");
```

after the reentrancy check to save the gas cost.

12. Inconsistent error messages
The require checks in the contract have error messages without the contract name.

Recommendation:
Consider following the error message as follows,

```
      require(!m_bLock, "ERC721Tradable: Contract locked to prevent 
reentrancy");
```

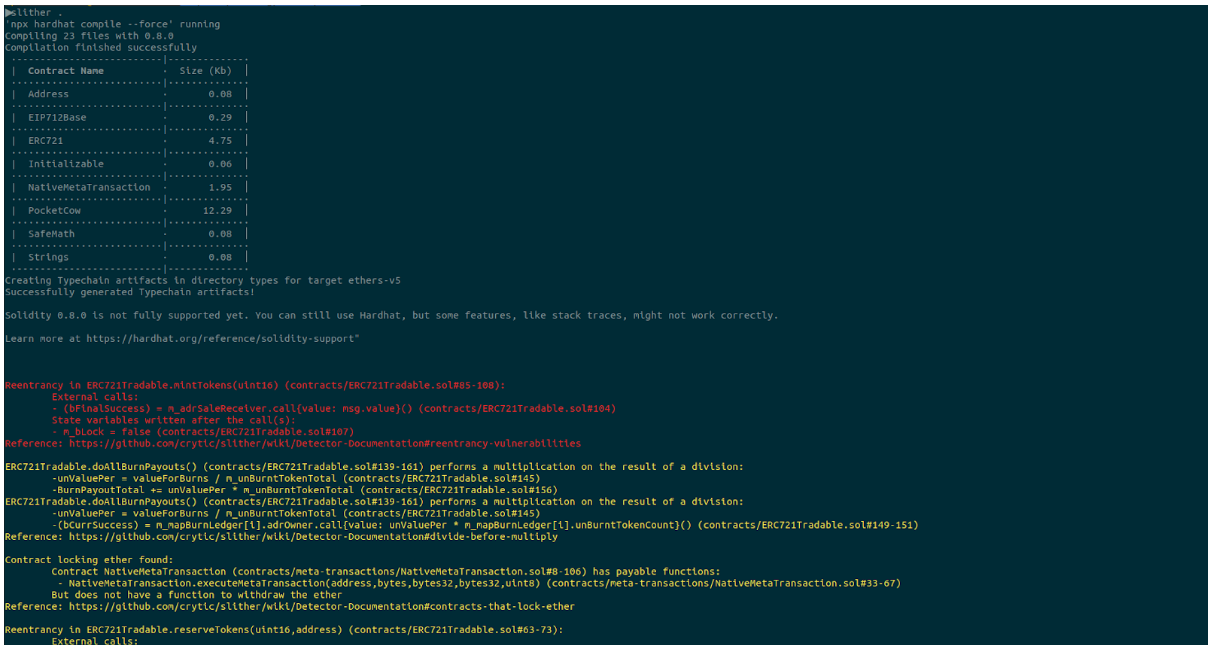
Goerli Test Contracts

PocketCow: 0x7E9d8c2C8ef6d18fC74CE4C73B557FbCb1a8C188

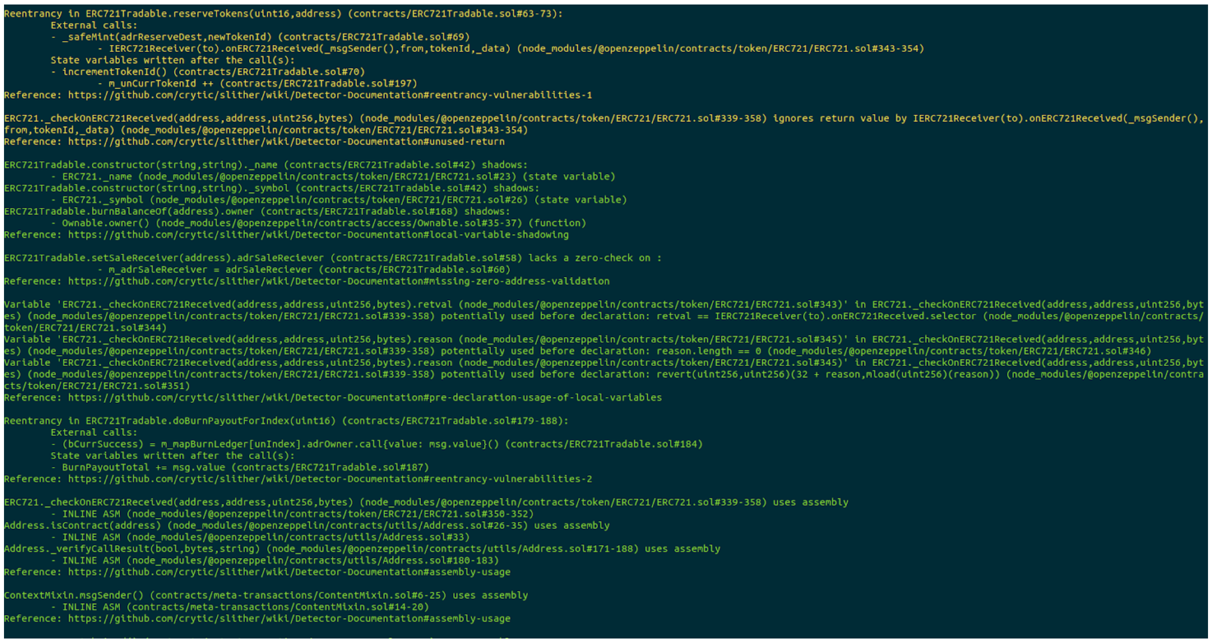
Functional Tests

1. setSaleReceiver
2. mintTokens
3. reserveTokens
4. setSalesEnabled
5. burnToken
6. mintTokens
7. mintTokens
8. burnToken
9. doAllBurnPayouts

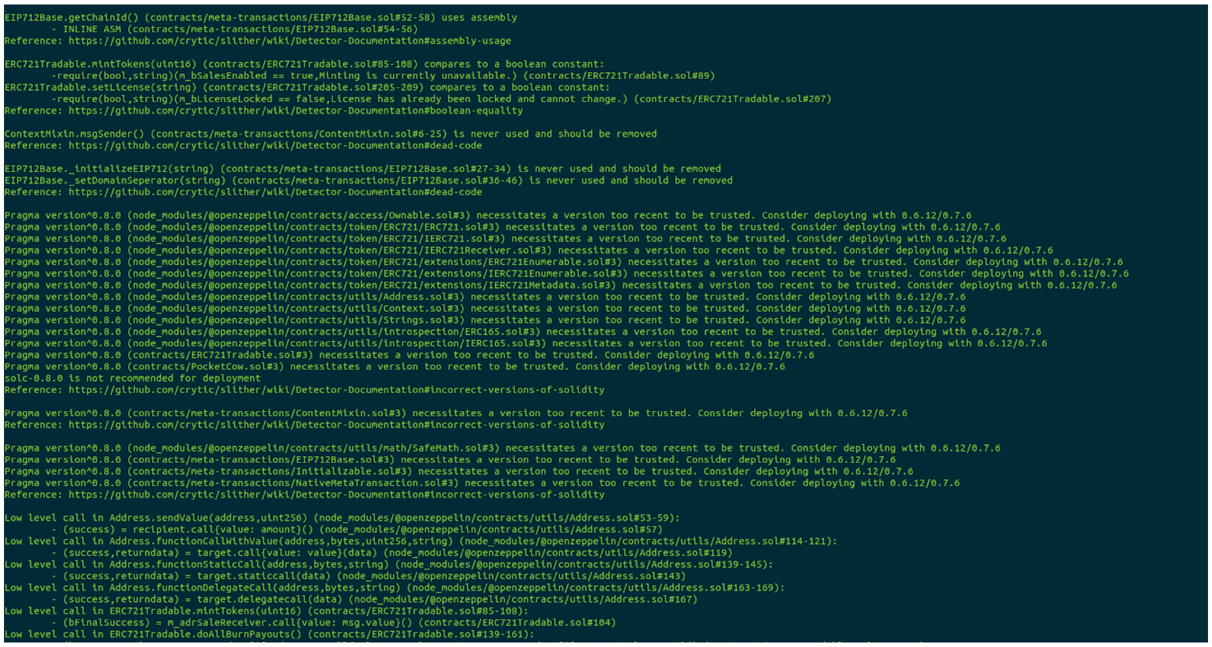
Automated Audit Result



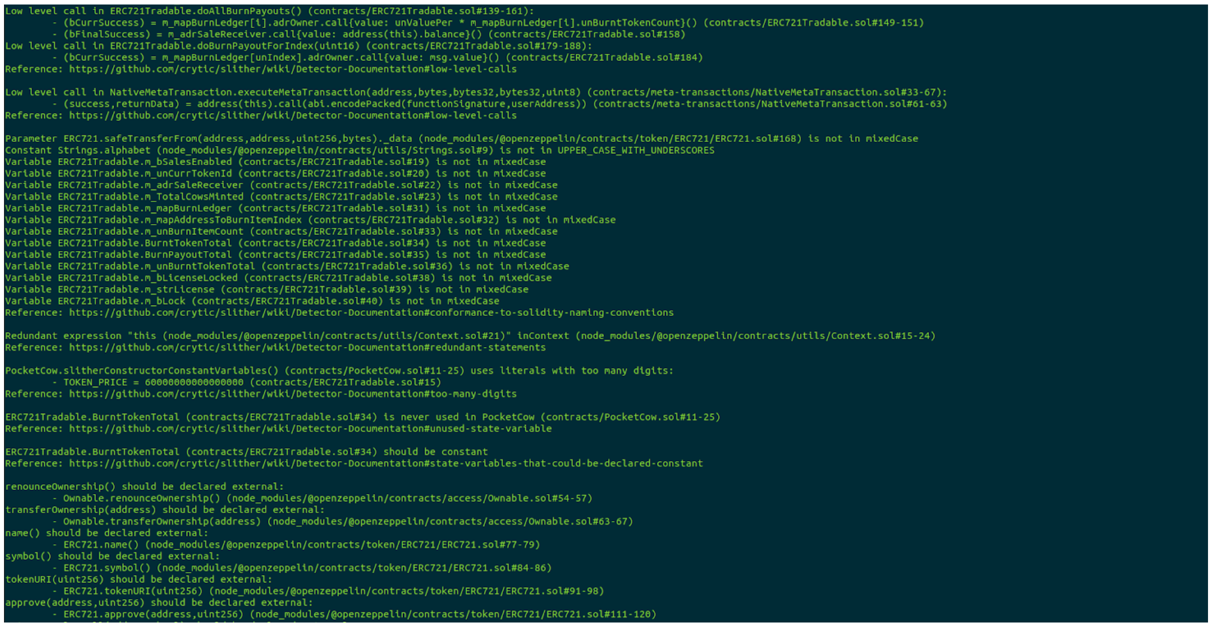
```
▶slither .
'npx hardhat compile --force' running
Compiling 23 files with 0.8.0
Compilation finished successfully
 ·························|··············
 |  Contract Name          ·  Size (Kb)  |
 ·························|··············
 |  Address                ·       0.08  |
 ·························|··············
 |  EIP712Base             ·       0.29  |
 ·························|··············
 |  ERC721                 ·       4.75  |
 ·························|··············
 |  Initializable          ·       0.06  |
 ·························|··············
 |  NativeMetaTransaction  ·       1.95  |
 ·························|··············
 |  PocketCow              ·      12.29  |
 ·························|··············
 |  SafeMath               ·       0.08  |
 ·························|··············
 |  Strings                ·       0.08  |
 ·························|··············
Creating Typechain artifacts in directory types for target ethers-v5
Successfully generated Typechain artifacts!

Solidity 0.8.0 is not fully supported yet. You can still use Hardhat, but some features, like stack traces, might not work correctly.

Learn more at https://hardhat.org/reference/solidity-support"



Reentrancy in ERC721Tradable.mintTokens(uint16) (contracts/ERC721Tradable.sol#85-108):
        External calls:
        - (bFinalSuccess) = m_adrSaleReceiver.call{value: msg.value}() (contracts/ERC721Tradable.sol#104)
        State variables written after the call(s):
        - m_bLock = false (contracts/ERC721Tradable.sol#107)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities

ERC721Tradable.doAllBurnPayouts() (contracts/ERC721Tradable.sol#139-161) performs a multiplication on the result of a division:
        -unValuePer = valueForBurns / m_unBurntTokenTotal (contracts/ERC721Tradable.sol#145)
        -BurnPayoutTotal += unValuePer * m_unBurntTokenTotal (contracts/ERC721Tradable.sol#156)
ERC721Tradable.doAllBurnPayouts() (contracts/ERC721Tradable.sol#139-161) performs a multiplication on the result of a division:
        -unValuePer = valueForBurns / m_unBurntTokenTotal (contracts/ERC721Tradable.sol#145)
        -(bCurrSuccess) = m_mapBurnLedger[i].adrOwner.call{value: unValuePer * m_mapBurnLedger[i].unBurntTokenCount}() (contracts/ERC721Tradable.sol#149-151)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

Contract locking ether found:
        Contract NativeMetaTransaction (contracts/meta-transactions/NativeMetaTransaction.sol#8-106) has payable functions:
         - NativeMetaTransaction.executeMetaTransaction(address,bytes,bytes32,bytes32,uint8) (contracts/meta-transactions/NativeMetaTransaction.sol#33-67)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether

Reentrancy in ERC721Tradable.reserveTokens(uint16,address) (contracts/ERC721Tradable.sol#63-73):
        External calls:
```



```
Reentrancy in ERC721Tradable.reserveTokens(uint16,address) (contracts/ERC721Tradable.sol#63-73):
        External calls:
        - _safeMint(adrReserveDest,newTokenId) (contracts/ERC721Tradable.sol#69)
                - IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,_data) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#343-354)
        State variables written after the call(s):
        - incrementTokenId() (contracts/ERC721Tradable.sol#70)
                - m_unCurrTokenId ++ (contracts/ERC721Tradable.sol#197)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#339-358) ignores return value by IERC721Receiver(to).onERC721Received(_msgSender(),
from,tokenId,_data) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#343-354)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

ERC721Tradable.constructor(string,string)._name (contracts/ERC721Tradable.sol#42) shadows:
        - ERC721._name (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#23) (state variable)
ERC721Tradable.constructor(string,string)._symbol (contracts/ERC721Tradable.sol#42) shadows:
        - ERC721._symbol (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#26) (state variable)
ERC721Tradable.burnBalanceOf(address).owner (contracts/ERC721Tradable.sol#168) shadows:
        - Ownable.owner() (node_modules/@openzeppelin/contracts/access/Ownable.sol#35-37) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

ERC721Tradable.setSaleReceiver(address).adrSaleReciever (contracts/ERC721Tradable.sol#58) lacks a zero-check on :
                - m_adrSaleReceiver = adrSaleReciever (contracts/ERC721Tradable.sol#60)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).retval (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#343)' in ERC721._checkOnERC721Received(address,address,uint256,byt
es) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#339-358) potentially used before declaration: retval == IERC721Receiver(to).onERC721Received.selector (node_modules/@openzeppelin/contracts/
token/ERC721/ERC721.sol#344)
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).reason (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#345)' in ERC721._checkOnERC721Received(address,address,uint256,byt
es) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#339-358) potentially used before declaration: reason.length == 0 (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#346)
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).reason (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#345)' in ERC721._checkOnERC721Received(address,address,uint256,byt
es) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#339-358) potentially used before declaration: revert(uint256,uint256)(32 + reason,mload(uint256)(reason)) (node_modules/@openzeppelin/contra
cts/token/ERC721/ERC721.sol#351)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables

Reentrancy in ERC721Tradable.doBurnPayoutForIndex(uint16) (contracts/ERC721Tradable.sol#179-188):
        External calls:
        - (bCurrSuccess) = m_mapBurnLedger[unIndex].adrOwner.call{value: msg.value}() (contracts/ERC721Tradable.sol#184)
        State variables written after the call(s):
        - BurnPayoutTotal += msg.value (contracts/ERC721Tradable.sol#187)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#339-358) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#350-352)
Address.isContract(address) (node_modules/@openzeppelin/contracts/utils/Address.sol#26-35) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#33)
Address._verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#171-188) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#180-183)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

ContextMixin.msgSender() (contracts/meta-transactions/ContentMixin.sol#6-25) uses assembly
        - INLINE ASM (contracts/meta-transactions/ContentMixin.sol#14-20)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```



```
EIP712Base.getChainId() (contracts/meta-transactions/EIP712Base.sol#52-58) uses assembly
        - INLINE ASM (contracts/meta-transactions/EIP712Base.sol#54-56)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

ERC721Tradable.mintTokens(uint16) (contracts/ERC721Tradable.sol#85-108) compares to a boolean constant:
        -require(bool,string)(m_bSalesEnabled == true,Minting is currently unavailable.) (contracts/ERC721Tradable.sol#89)
ERC721Tradable.setLicense(string) (contracts/ERC721Tradable.sol#205-209) compares to a boolean constant:
        -require(bool,string)(m_bLicenseLocked == false,License has already been locked and cannot change.) (contracts/ERC721Tradable.sol#207)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality

ContextMixin.msgSender() (contracts/meta-transactions/ContentMixin.sol#6-25) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

EIP712Base._initializeEIP712(string) (contracts/meta-transactions/EIP712Base.sol#27-34) is never used and should be removed
EIP712Base._setDomainSeperator(string) (contracts/meta-transactions/EIP712Base.sol#36-46) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Enumerable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Address.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/ERC165.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/IERC165.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/ERC721Tradable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/PocketCow.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Pragma version^0.8.0 (contracts/meta-transactions/ContentMixin.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/math/SafeMath.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/meta-transactions/EIP712Base.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/meta-transactions/Initializable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/meta-transactions/NativeMetaTransaction.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#53-59):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts/utils/Address.sol#57)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#114-121):
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#119)
Low level call in Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#139-145):
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#143)
Low level call in Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#163-169):
        - (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#167)
Low level call in ERC721Tradable.mintTokens(uint16) (contracts/ERC721Tradable.sol#85-108):
        - (bFinalSuccess) = m_adrSaleReceiver.call{value: msg.value}() (contracts/ERC721Tradable.sol#104)
Low level call in ERC721Tradable.doAllBurnPayouts() (contracts/ERC721Tradable.sol#139-161):
```



```
Low level call in ERC721Tradable.doAllBurnPayouts() (contracts/ERC721Tradable.sol#139-161):
        - (bCurrSuccess) = m_mapBurnLedger[i].adrOwner.call{value: unValuePer * m_mapBurnLedger[i].unBurntTokenCount}() (contracts/ERC721Tradable.sol#149-151)
        - (bFinalSuccess) = m_adrSaleReceiver.call{value: address(this).balance}() (contracts/ERC721Tradable.sol#158)
Low level call in ERC721Tradable.doBurnPayoutForIndex(uint16) (contracts/ERC721Tradable.sol#179-188):
        - (bCurrSuccess) = m_mapBurnLedger[unIndex].adrOwner.call{value: msg.value}() (contracts/ERC721Tradable.sol#184)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Low level call in NativeMetaTransaction.executeMetaTransaction(address,bytes,bytes32,bytes32,uint8) (contracts/meta-transactions/NativeMetaTransaction.sol#33-67):
        - (success,returnData) = address(this).call(abi.encodePacked(functionSignature,userAddress)) (contracts/meta-transactions/NativeMetaTransaction.sol#61-63)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Parameter ERC721.safeTransferFrom(address,address,uint256,bytes)._data (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#168) is not in mixedCase
Constant Strings.alphabet (node_modules/@openzeppelin/contracts/utils/Strings.sol#9) is not in UPPER_CASE_WITH_UNDERSCORES
Variable ERC721Tradable.m_bSalesEnabled (contracts/ERC721Tradable.sol#19) is not in mixedCase
Variable ERC721Tradable.m_unCurrTokenId (contracts/ERC721Tradable.sol#20) is not in mixedCase
Variable ERC721Tradable.m_adrSaleReceiver (contracts/ERC721Tradable.sol#22) is not in mixedCase
Variable ERC721Tradable.m_TotalCowsMinted (contracts/ERC721Tradable.sol#23) is not in mixedCase
Variable ERC721Tradable.m_mapBurnLedger (contracts/ERC721Tradable.sol#31) is not in mixedCase
Variable ERC721Tradable.m_mapAddressToBurnItemIndex (contracts/ERC721Tradable.sol#32) is not in mixedCase
Variable ERC721Tradable.m_unBurnItemCount (contracts/ERC721Tradable.sol#33) is not in mixedCase
Variable ERC721Tradable.BurntTokenTotal (contracts/ERC721Tradable.sol#34) is not in mixedCase
Variable ERC721Tradable.BurnPayoutTotal (contracts/ERC721Tradable.sol#35) is not in mixedCase
Variable ERC721Tradable.m_unBurntTokenTotal (contracts/ERC721Tradable.sol#36) is not in mixedCase
Variable ERC721Tradable.m_bLicenseLocked (contracts/ERC721Tradable.sol#38) is not in mixedCase
Variable ERC721Tradable.m_strLicense (contracts/ERC721Tradable.sol#39) is not in mixedCase
Variable ERC721Tradable.m_bLock (contracts/ERC721Tradable.sol#40) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (node_modules/@openzeppelin/contracts/utils/Context.sol#21)" inContext (node_modules/@openzeppelin/contracts/utils/Context.sol#15-24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

PocketCow.slitherConstructorConstantVariables() (contracts/PocketCow.sol#11-25) uses literals with too many digits:
        - TOKEN_PRICE = 60000000000000000 (contracts/ERC721Tradable.sol#15)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

ERC721Tradable.BurntTokenTotal (contracts/ERC721Tradable.sol#34) is never used in PocketCow (contracts/PocketCow.sol#11-25)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

ERC721Tradable.BurntTokenTotal (contracts/ERC721Tradable.sol#34) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (node_modules/@openzeppelin/contracts/access/Ownable.sol#54-57)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (node_modules/@openzeppelin/contracts/access/Ownable.sol#63-67)
name() should be declared external:
        - ERC721.name() (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#77-79)
symbol() should be declared external:
        - ERC721.symbol() (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#84-86)
tokenURI(uint256) should be declared external:
        - ERC721.tokenURI(uint256) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#91-98)
approve(address,uint256) should be declared external:
        - ERC721.approve(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#111-120)
```

Results:

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

Concluding Remarks

While conducting the audits of the PocketCows Farm smart contract, it was observed that the contracts contain Medium and Low severity issues.

Our auditors suggest that Medium and Low severity issues should be resolved by the developers. The recommendations given will improve the operations of the smart contract.

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the PocketCows Farm platform or its product nor this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for the code refactor by the team on critical issues.

ImmuneBytes