Kitty Connect Security Audit Report by MK (16th DEC 2024)

High Risk Findings

[H-1] Missed adding `tokenId` to the `s_ownerToCatsTokenId` when `mintBridgedNFT()`

Description: Missed adding `tokenId` to the `s_ownerToCatsTokenId` when `mintBridgedNFT()`.

Impact: High, since this token won't be tracked as part of the entries of the owner, resulting to inconsistencies when bridging.

Proof of Concept: When `mintBridgedNFT()` function is called it is forgotten to add the `tokenId` to the list that keeps track of the nfts of the owner `s_ownerToCatsTokenId` resulting to inconsistencies when bridging.

Recommended Mitigation:

1. Line such as this to be added just before emitting the "NFTBridged" event:

```
s_ownerToCatsTokenId[catOwner].push(tokenId);
```

[H-2] Improper token ownership update in `_updateOwnershipInfo` leads to double ownership of the same Kitty

Description: `_updateOwnershipInfo` properly updates new owner's ownership of the Kitty, but fails to remove old owner's ownership of it.

Impact: This vulnerability undermines the security and integrity of the token ownership model within the `KittyConnect` ecosystem. Exploiting this flaw can lead to multiple parties claiming ownership of the same token, resulting in confusion, loss of trust, and potential financial implications for the involved parties.

Proof of Concept:

1. Add the following code to the `KittyTest.t.sol` file:

```
function test_transferCatToNewOwner_DoesntUpdateOwnership() public {
    // ******************** Mint kitty ********************
    uint256 tokenId = kittyConnect.getTokenCounter();

    vm.prank(partnerA);
    kittyConnect.mintCatToNewOwner(
        user,
        "ipfs://QmbxwGgBGrNdXPm84kqYskmcMT3jrzBN8LzQjixvkz4c62",
        "Meowdy",
        "Ragdoll",
        block.timestamp
    );
    // ****************************************************

    // ******************* Transfer kitty to a new owner *******************
    address newOwner = makeAddr("newOwner");

    vm.prank(user);
    kittyConnect.approve(newOwner, tokenId);
    vm.expectEmit(false, false, false, true, address(kittyConnect));
    emit CatTransferredToNewOwner(user, newOwner, tokenId);

    // Shop partner transfers to the new owner
    vm.prank(partnerA);
    kittyConnect.safeTransferFrom(user, newOwner, tokenId);

    uint256[] memory oldOwnerTokenIds = kittyConnect.getCatsTokenIdOwnedBy(
        user
    );
    uint256[] memory newOwnerTokenIds = kittyConnect.getCatsTokenIdOwnedBy(
        newOwner
    );

    // Confirm that they both own the same Kitty
    assertEq(oldOwnerTokenIds.length, 1);
    assertEq(newOwnerTokenIds.length, 1);

    assertEq(newOwnerTokenIds[0], tokenId);
    assertEq(oldOwnerTokenIds[0], tokenId);
    // **********************************************************************
}
```

Recommended Mitigation:

1. Implement logic within the `_updateOwnershipInfo` function to remove the token ID from the previous owner's `s_ownerToCatsTokenId` array. This update ensures the contract's state accurately reflects the current ownership of tokens and prevents the previous owner from interacting with tokens they no longer own.

```
function _updateOwnershipInfo(
    address currCatOwner,
    address newOwner,
    uint256 tokenId
) internal {
    s_catInfo[tokenId].prevOwner.push(currCatOwner);
    s_catInfo[tokenId].idx = s_ownerToCatsTokenId[newOwner].length;
    s_ownerToCatsTokenId[newOwner].push(tokenId);

+   // Remove the token from the previous owner's array
+   uint256 tokenIndex = s_catInfo[tokenId].idx;
+   uint256 lastTokenIndex = s_ownerToCatsTokenId[currCatOwner].length - 1;
+   uint256 lastTokenId = s_ownerToCatsTokenId[currCatOwner][lastTokenIndex];

+   // Swap and pop
+   s_ownerToCatsTokenId[currCatOwner][tokenIndex] = lastTokenId;
+   s_catInfo[lastTokenId].idx = tokenIndex;
+   s_ownerToCatsTokenId[currCatOwner].pop();
}
```

[H-3] Missing fee token approval for `IRouterClient` in `bridgeNftWithData`

Description: The `bridgeNftWithData` function in the `KittyBridge` contract attempts to bridge Kitties across chains using Chainlink's CCIP and requires a fee in LINK tokens to be paid to the `IRouterClient` contract. However, it does not include a step to approve the router contract to spend the required LINK fee on behalf of the `KittyBridge` contract.

Impact: This vulnerability renders the `bridgeNftWithData` function inoperative, as it will always fail when attempting to bridge NFTs due to the inability to transfer LINK tokens as fees.

Proof of Concept: ERC-20 tokens, such as LINK, require an owner to approve a spender to transfer tokens up to a specified allowance. The `bridgeNftWithData` function calculates the necessary fees in LINK tokens for the bridging operation and checks if the contract has sufficient LINK token balance. However, it overlooks the need to set an allowance for the router contract, assuming the contract's balance is sufficient for the fee. This assumption fails when the router contract attempts to run transfer LINK tokens from the `KittyBridge` contract to itself without the necessary approval, causing the transaction to revert.

```
function test_bridgeNftWithData_FailsIfGasTokenNotApproved() public {
    vm.prank(kittyConnectOwner);
    kittyBridge.allowlistSender(networkConfig.router, true);

    // ******************** Bridge NFT with Data ********************
    string
        memory _catImage =
"ipfs://QmbxwGgBGrNdXPm84kqYskmcMT3jrzBN8LzQjixvkz4c62";
    string memory _catName = "Meowdy";
    string memory _catBreed = "Ragdoll";
    uint256 _catDob = block.timestamp;

    uint256 _tokenId = kittyConnect.getTokenCounter();

    vm.prank(partnerA);
    kittyConnect.mintCatToNewOwner(
        user,
        _catImage,
        _catName,
        _catBreed,
        _catDob
    );
    // Before bridging, confirm that the Bridge does not have enough LINK
allowance for fees
    uint256 _kittyBridgeLinkAllowanceToRouter = IERC20(
        kittyBridge.getLinkToken()
    ).allowance(address(kittyBridge), address(networkConfig.router));
    assertEq(_kittyBridgeLinkAllowanceToRouter, 0);

    vm.prank(user);
    vm.expectRevert("ERC20: insufficient allowance");
    kittyConnect.bridgeNftToAnotherChain(
        networkConfig.otherChainSelector,
        address(kittyBridge),
        _tokenId
    );
    // *******************************************************
}
```

Recommended Mitigation:

1. Implement a LINK token approval step within the `bridgeNftWithData` function or as part of the initial setup/configuration of the `KittyBridge` contract to grant the router contract an allowance to spend LINK tokens on its behalf. This approval should be for an amount at least equal to the anticipated fee for the operation.

```
function bridgeNftWithData(
        uint64 _destinationChainSelector,
        address _receiver,
        bytes memory _data
    )
        external
        onlyAllowlistedDestinationChain(_destinationChainSelector)
        validateReceiver(_receiver)
        returns (bytes32 messageId)
    {
        // Create an EVM2AnyMessage struct in memory with necessary information
for sending a cross-chain message
        Client.EVM2AnyMessage memory evm2AnyMessage = _buildCCIPMessage(
            _receiver,
            _data,
            address(s_linkToken)
        );

        // Initialize a router client instance to interact with cross-chain router
        IRouterClient router = IRouterClient(this.getRouter());

        // Get the fee required to send the CCIP message
        uint256 fees = router.getFee(_destinationChainSelector, evm2AnyMessage);

        if (fees > s_linkToken.balanceOf(address(this))) {
            revert KittyBridge__NotEnoughBalance(
                s_linkToken.balanceOf(address(this)),
```

```diff
                fees
            );
        }

+       s_linkToken.approve(address(router), fees);

        // Code below stays the same
    }
```

[H-4] Potential for Malicious Drainage of LINK Tokens in KittyBridge

Description: The `KittyConnect::bridgeNftToAnotherChain` and `KittyBridge::bridgeNftWithData` functions within the NFT bridge protocol are designed to facilitate the transfer of NFTs across different blockchain networks. However, a malicious user could potentially exploit these functions to repeatedly drain the bridge contract's LINK tokens. This could be achieved by repeatedly calling these functions, which would cause the bridge contract to repeatedly attempt to deduct LINK tokens as fees for the transfers. If not properly managed, this could lead to the bridge contract running out of LINK tokens, causing the project to lose money and potentially leading to a denial of service (DoS) situation where the bridge function becomes unavailable due to the depletion of LINK tokens.

Impact: The potential for malicious drainage of LINK tokens can have several significant impacts on the project:

1. Financial Loss: The project could incur significant financial losses as the bridge contract's LINK tokens are depleted. This could affect the project's ability to cover operational costs and maintain the service.
2. Service Availability: The depletion of LINK tokens could lead to a denial of service (DoS) situation, where the bridge function becomes unavailable. This would prevent users from transferring NFTs across different blockchain networks, severely impacting the project's utility and user experience.

Proof of Concept:

1. To demonstrate this vulnerability, a malicious user could repeatedly call the `KittyConnect::bridgeNftToAnotherChai`n or `KittyBridge::bridgeNftWithData` functions, causing the bridge contract to repeatedly attempt to deduct LINK tokens as fees for the transfers. Here is an example test which can be added to the protocol test suite, showing how the attack works:

```
 function test_userCanEmptyBridgeOfFunds() public {
        ////fund the kittyBridge with link
        uint256 InitialLink = 10_000_000_000_000_000_000;
        uint256 minfeeNeededToBridge = 48_381_912_000_000_000; //min fee needed to bridge data once
        address linkHolder = 0x61E5E1ea8fF9Dc840e0A549c752FA7BDe9224e99; //address of linkHolder on sepolia testnet
        vm.prank(linkHolder);
        linkToken.approve(address(kittyBridge), InitialLink + 1);
        vm.prank(address(kittyBridge));

        linkToken.transferFrom(linkHolder, address(kittyBridge), InitialLink);
        ////Malicious user empties the bridge by calling bridgeNftWithData
```

```
repeatedly
        uint256 bridgeAttemptCounter = 0;
        vm.startPrank(user);
        while (linkToken.balanceOf(address(kittyBridge)) >= minfeeNeededToBridge)
{
            uint64 _destinationChainSelector = 14767482510784806043;

            kittyBridge.bridgeNftWithData(
                _destinationChainSelector,
                address(kittyBridge),
                abi.encode("0x0")
            );
            bridgeAttemptCounter++;
        }
        uint256 linkBalanceAfterAttack = linkToken.balanceOf(
            address(kittyBridge)
        );
        console.log(bridgeAttemptCounter);
        assert(linkBalanceAfterAttack < minfeeNeededToBridge);
    }
```

2. To run the test, use a Sepolia testnet forked URL:

```
forge test --fork-url $SEPOLIA_RPC_URL --mt test_userCanEmptyBridgeOfFunds -vv
```

3. Also, the gas cost for this test function is approximately 14,500,000, which is quite high and will drop the probability of the attack.

Recommended Mitigation: To mitigate this vulnerability, several measures can be implemented:

1. Add an `onlyKittyConnect` Modifier: Implement an `onlyKittyConnect` modifier to the `kittyBridge::bridgeNftWithData` function. This will significantly decrease the attack probability because it costs more gas to transfer NFTs instead of just text as data, and a project user is less likely to attack the project than any other person.
2. Require Additional Approvals: For high-value transfers, repeated transfers, or for any bridge transfer, require additional approvals or confirmations from shop owners. This could involve a multi-signature mechanism or additional user confirmation.
3. Do Not Charge the Bridge Contract: Add `onlyKittyConnect` modifier and Instead of charging the bridge contract for the LINK tokens, reduce the fee amount from the user when they call for the bridge process. This is the main recommendation as it directly addresses the vulnerability by shifting the cost to the user, thereby reducing the risk of the bridge contract running out of LINK tokens. By implementing these mitigation strategies, the project can better protect against the potential for malicious drainage of LINK tokens and ensure the reliability and security of the NFT bridge service. Here is how the 3rd recommendation can be implemented:

```
function bridgeNftWithData(
        uint64 _destinationChainSelector,
```

```diff
        address _receiver,
        bytes memory _data
    )
        external
        onlyAllowlistedDestinationChain(_destinationChainSelector)
        validateReceiver(_receiver)
+       onlyKittyConnect
        returns (bytes32 messageId, bytes memory)
    {
    .
    .
    .
```

```diff
function bridgeNftToAnotherChain(uint64 destChainSelector, address destChainBridge, uint256 tokenId) external {
        address catOwner = _ownerOf(tokenId);

        require(msg.sender == catOwner);

        CatInfo memory catInfo = s_catInfo[tokenId];
        uint256 idx = catInfo.idx;
        bytes memory data = abi.encode(catOwner, catInfo.catName, catInfo.breed, catInfo.image, catInfo.dob, catInfo.shopPartner);

        _burn(tokenId);
        delete s_catInfo[tokenId];

        uint256[] memory userTokenIds = s_ownerToCatsTokenId[msg.sender];
        uint256 lastItem = userTokenIds[userTokenIds.length - 1];

        s_ownerToCatsTokenId[msg.sender].pop();

        if (idx < (userTokenIds.length - 1)) {
            s_ownerToCatsTokenId[msg.sender][idx] = lastItem;
        }
+       uint256 fee = i_kittyBridge.getFee(destChainSelector, destChainBridge, data);
+       // this function does not currently exist in kitty bridge and should be added if protocol wants to use this method
+       IERC20 linkToken = IERC20(i_kittyBridge.getLinkToken());
+       linkToken.transferFrom(msg.sender,i_kittyBridge,fee);
        emit NFTBridgeRequestSent(block.chainid, destChainSelector, destChainBridge, tokenId);
        i_kittyBridge.bridgeNftWithData(destChainSelector, destChainBridge, data);
    }
```

[H-5] Incorrect access control in `KittyBridge::_ccipReceive` allows anyone to get free kitties

Description: `KittyBridge::_ccipReceive` has the purpose of gatekeeping cross-chain messages by validating the sender and the chain id of the source chain. The problem in this process is who validates the function as the sender, instead of validating the originator the message, the function validates the caller of `KittyBridge::ccipReceive` which is the router of the destination chain - this means that as long the message comes from a valid source chain and the `msg.sender` is the router, that message will be accepted. This will allow anyone to create a message to instruct the bridge to mint a kitty for free.

Impact: Anyone can get free kitties for free, breaking the intended behaviour of the protocol.

Proof of Concept:

1. To prove the attack I've deployed the protocol on Polygon Mumbai and Ethereum Sepolia. Also I've deployed an Attacker contract at this address on Polygon Mumbai for faster testing. The attacker contract is already funded with LINK to pay for the fees.

```
# addresses

MUMBAI_ATTACKER = 0x284FB07c3c51629Bc48479d85AFA0BD123c8C5B8
SEPOLIA_KITTY_BRIDGE = 0xFd829a345fd28a0f6eeD6851822ebbCe6b923702
SEPOLIA_KITTY_CONNECT = 0xC55e8Fd249dAb84E563249340dA769FE8B7378cF
```

Follow the next steps:

1. Call `KittyBridgeAttacker::computePayload`on `MUMBAI_ATTACKER` with the address of the account to be owner of the kitty to mint on the destination chain
2. Call `KittyBridgeAttacker::sendMessagePayLINK` with the chain id of Ethereum Sepolia, the receiver of the message (`SEPOLIA_KITTY_BRIDGE`) and the computed payload. The chain id of sepolia and address of the receiver are `16015286601757825753` and `0xFd829a345fd28a0f6eeD6851822ebbCe6b923702`, respectively
3. Use the transaction hash to find the transaction on CCIP Explorer and wait for finality
4. After reaching block finality, verify the specified owner got a free NFT by calling`KittyConnect::ownerOf`, I recommend testing from three onward as token id.

Recommended Mitigation:

1. To properly gatekeep messages I recommend validating the originator of the message instead of the `msg.sender`.

```
    function _ccipReceive(Client.Any2EVMMessage memory any2EvmMessage)
        internal
        override
+       onlyAllowlisted(any2EvmMessage.sourceChainSelector,
abi.decode(any2EvmMessage.sender, (address)))
-       onlyAllowlisted(any2EvmMessage.sourceChainSelector, msg.sender)
    {
        // rest of the function..
    }
```

[H-6] The bridged NFT will be lost for the users using account abstraction

Description: All users using account abstraction wallets will not be able to bridge their NFT from one chain to another. This because they have different addresses across chain for the same account and therefore, all the NFTs will be bridged to a wrong address and lost forever

Impact: For all account abstraction wallet users, all the NFTs will be bridged to a wrong address and lost forever

Proof of Concept:

1. The modifier `onlyAllowlisted` uses `msg.sender` as the address of the `sender` but this is not true when using account abstraction wallets

```
    function _ccipReceive(Client.Any2EVMMessage memory any2EvmMessage)
        internal
        override
        onlyAllowlisted(any2EvmMessage.sourceChainSelector, msg.sender)
    {
        KittyConnect(kittyConnect).mintBridgedNFT(any2EvmMessage.data);

        emit MessageReceived(
            any2EvmMessage.messageId,
            any2EvmMessage.sourceChainSelector,
            abi.decode(any2EvmMessage.sender, (address)),
            any2EvmMessage.data
        );
    }
```

Recommended Mitigation:

1. Give the users the option to pass in the `sender` address

```diff
-   function _ccipReceive(Client.Any2EVMMessage memory any2EvmMessage)
+  function _ccipReceive(Client.Any2EVMMessage memory any2EvmMessage, address sender)
        internal
        override
-        onlyAllowlisted(any2EvmMessage.sourceChainSelector, msg.sender)
+       onlyAllowlisted(any2EvmMessage.sourceChainSelector, sender)
   {
        KittyConnect(kittyConnect).mintBridgedNFT(any2EvmMessage.data);

        emit MessageReceived(
            any2EvmMessage.messageId,
            any2EvmMessage.sourceChainSelector,
            abi.decode(any2EvmMessage.sender, (address)),
            any2EvmMessage.data
        );
    }
```

Medium Risk Findings

[M-1] "Idx" of burned nft won't be switched with "lastItem" popped out of "s_ownerToCatsTokenId"

Description: "Idx" of burned nft won't be switched with "lastItem" popped out of "s_ownerToCatsTokenId" if idx itself is the last item inside of the array after the previous last item was popped out.

Impact: High, since the owner of the nft loses one of his nfts and remains with a reference to the id of the one that is burned.

Proof of Concept: Inside "KittyConnect.sol" inside "bridgeNftToAnotherChain()" the "Idx" of burned nft won't be switched with "lastItem" popped out of "s_ownerToCatsTokenId" if idx itself is the last item inside of the array after the previous last item was popped out. To make it more clear if we have s_ownerToCatsTokenIds filled with ids as such (1,2,3,4) and the idx of the nft owner wants to transfer to be equal to 3 (idx=3 with index=2, cause arrays in Solidity are 0 indexed) and we follow the business logic:

```
        uint256[] memory userTokenIds = s_ownerToCatsTokenId[msg.sender];
        uint256 lastItem = userTokenIds[userTokenIds.length - 1]; //Last item
represents last token Id of the owner, so lastItem = userTokenIds.length(which is
//of value 4 - 1, meaning lastItem will be at index 3 meaning value=4;

        s_ownerToCatsTokenId[msg.sender].pop(); //this removes the last element of
the array with "value=4 and index 3" of token Ids of the owner, and //leaves the
array with total of 3 elements (1,2,3)

      if (idx < (userTokenIds.length - 1)) { //This is where vulnerability occurs
            s_ownerToCatsTokenId[msg.sender][idx] = lastItem;
        }
```

to continue the example off-code our idx=3 which is not smaller than 2 (userTokenIds.length is 3-1) meaning we will not go in the if and value=4 (lastItem) will not get to the place of the idx, which leads to the idx staying in the "s_ownerToCatsTokenId" and lastItem being lost.

Recommended Mitigation: The equation should use equal sign as well in the if statement so:

```
if (idx <= (userTokenIds.length - 1))
```

[M-2] Bypassing Shop Authorization Access Control via `ERC721::transferFrom`

Description: The NFT bridge protocol utilizes an overridden safeTransferFrom function to ensure that the transfer of Kitty NFTs between users is authorized by shop partners. This mechanism is prbably designed to ensure cats security!! by ensuring that all transfers are approved by the designated shop partners. However, after the current owner approves the NFT transfer, the new owner can subsequently use the ERC721::transferFrom function to transfer the NFT without the required shop authorization.

Impact: It can enable unauthorized transfers, and also because the transfer is not done via `safeTransferFrom` the `KittyConnect::_updateOwnershipInfo` is not called and the storage variables and mappings saving cats owners data are not updated.

Proof of Concept:

1. currOwner (user) approves transfer to newOwner
2. newOwner transfers the NFT using transferFrom use the following test in the protocol test suit:

```
    function test_newOwnerCanTransferUsingTransferFrom()
        public
        partnerGivesCatToOwner
    {
        uint256 tokenId = kittyConnect.getTokenCounter() - 1;
        address newOwner = makeAddr("newOwner");
        // user approves the transfer
        vm.prank(user);
        kittyConnect.approve(newOwner, tokenId);
        // newOwner transfers using transferFrom instead of safeTransferFrom
        vm.prank(newOwner);
        kittyConnect.transferFrom(user, newOwner, tokenId);

        uint256[] memory user_tokens = kittyConnect.getCatsTokenIdOwnedBy(user);
        uint256[] memory newOwner_tokens = kittyConnect.getCatsTokenIdOwnedBy(
            newOwner
        );
        assertEq(kittyConnect.balanceOf(newOwner), 1); // newOwner has the NFT
        assertEq(user_tokens.length, 1); // the s_ownerToCatsTokenId of current
owner is not updated
        assertEq(newOwner_tokens.length, 0); // the s_ownerToCatsTokenId of new
owner is not updated
    }
```

Recommended Mitigation: To fix this issue we can also override the `transferFrom` function to use `onlyShopPartner` modifier and also update the state variables.

```
+function transferFrom(
+        address currCatOwner,
+        address newOwner,
+        uint256 tokenId,
+        bytes memory data
+    ) public override onlyShopPartner {
+        require(
+            _ownerOf(tokenId) == currCatOwner,
+            "KittyConnect__NotKittyOwner"
+        );
+        require(
+            getApproved(tokenId) == newOwner,
+            "KittyConnect__NewOwnerNotApproved"
+        );

+        _updateOwnershipInfo(currCatOwner, newOwner, tokenId);

+        emit CatTransferredToNewOwner(currCatOwner, newOwner, tokenId);
+        _safeTransfer(currCatOwner, newOwner, tokenId, data);
+    }
```

Low Risk Findings

[L-1] Date of birth of a cat should be equal to block.timestamp

Description: `CatInfo.dob` should not be an arbitrary parameter.

Impact: Cats' date of birth is not a reliable variable, calling function `KittyConnect::getCatAge()` will probably return an incorrect date of birth of the cat. If the set dob is higher than the current time, the function will revert.

Proof of Concept: `KittyConnect::CatInfo` struct has a `uint256 dob` variable which represents the date of birth of the cat. This variable can have the value desired by the shop partner minting the cat, which is a clear mistake, it should be as follows: `dob = block.timestamp;`. As we do not have a list of existing cats with their data, it is required to suppose that a new minted cat is born at the moment of minting.

It is even possible to set the date of birth at a time that has not come yet (e.g. year 2100).

Recommended Mitigation: When minting a new cat, `dob` should be equal to the current time, not a parameter chosen by the shop partner.

```
-function mintCatToNewOwner(address catOwner, string memory catIpfsHash, string
memory catName, string memory breed, uint256 dob) external
-onlyShopPartner {
+function mintCatToNewOwner(address catOwner, string memory catIpfsHash, string
memory catName, string memory breed) external onlyShopPartner {
        require(!s_isKittyShop[catOwner],
"KittyConnect__CatOwnerCantBeShopPartner");

        uint256 tokenId = kittyTokenCounter;
        kittyTokenCounter++;

        s_catInfo[tokenId] = CatInfo({
            catName: catName,
            breed: breed,
            image: catIpfsHash,
-           dob: dob,
+          dob: block.timestamp,
            prevOwner: new address[](0),
            shopPartner: msg.sender,
            idx: s_ownerToCatsTokenId[catOwner].length
        });

        s_ownerToCatsTokenId[catOwner].push(tokenId);

        _safeMint(catOwner, tokenId);
        emit CatMinted(tokenId, catIpfsHash);
    }
```

[L-2] Lack of a mechanism to transfer ownership and state variable `KittyConnect::i_kittyConnectOwner` declared as an immutable makes it impossible to transfer the ownership of `KittyConnect`

Description: The `kittyConnect.sol::i_kittyConnectOwner` variable is currently declared as immutable, preventing future ownership transfers.

Impact: Ownership of the contract cannot be transferred.

Proof of Concept: Immutable variables, once set during deployment, remain unchanged thereafter. If the owner's wallet is compromised or lost, critical functionalities may be impacted such as adding new shops, source and destinations chains or senders to the allow lists.

Recommended Mitigation: Consider declaring the variable as mutable. Additionally, implement a function allowing ownership transfers, restricted to only the current owner's invocation.

[L-3] Inconsistent behaviour of functions will lead to a user malpractice

Description: We need to look at 3 functions:

1. mintCatToNewOwner() => `require(!s_isKittyShop[catOwner], "KittyConnect__CatOwnerCantBeShopPartner");`
2. addShop() => No require
3. safeTransferFrom() => No require

So on the one hand it says "Cat Owner Cannot Be Shop Partner", but on the other hand it is possible to transfer an NFT to an account and then the addShop() function doesn't check if the shop has NFTs.

Any user can store NFTs first and then become a shop without the protocol checking the status.

Impact: Any protocol must have clear rules and reasons for behaving in a certain way. If there is a requirement that says that it is not possible to attach an NFT to a shop, all functions should have the same requirement.

Since this is not the case, it is a HIGH risk vulnerability.

Proof of Concept:

1. Check this foundry test showing the funcitons behaviour:

```
function test_differentBehaivourBetweenFunctions() external {
        string memory catImageIpfsHash =
"ipfs://QmbxwGgBGrNdXPm84kqYskmcMT3jrzBN8LzQjixvkz4c62";
        /**
         * mintCatToNewOwner() doesn´t allow to mint to a shop
         */
        vm.prank(shopOwner0);
        vm.expectRevert("KittyConnect__CatOwnerCantBeShopPartner");
        kittyConnect.mintCatToNewOwner(address(shopOwner1), catImageIpfsHash,
"Meowdy", "Ragdoll", block.timestamp);
        /**
         * It is allowed to transfer the NFT to a shop
         */
        // 1. Add NFT to user
        vm.prank(shopOwner0);
        kittyConnect.mintCatToNewOwner(address(user), catImageIpfsHash, "Meowdy",
"Ragdoll", block.timestamp);
        uint256 tokenId = kittyConnect.getTokenCounter() - 1;
        assertEq(kittyConnect.ownerOf(tokenId), address(user));

        // 2. User send the NFT to the shop. No problem!
        vm.prank(user);
        kittyConnect.approve(address(shopOwner0), tokenId);
        vm.prank(shopOwner0);
        kittyConnect.safeTransferFrom(user, shopOwner0, 0);
        assertEq(kittyConnect.ownerOf(tokenId), address(shopOwner0));

        /**
         * Users with NFTs can become shops
         */
        // 1. Add NFT to user
        vm.prank(shopOwner0);
        kittyConnect.mintCatToNewOwner(address(user), catImageIpfsHash, "Meowdy2",
"Ragdoll2", block.timestamp);
        tokenId = kittyConnect.getTokenCounter() - 1;
        assertEq(kittyConnect.ownerOf(tokenId), address(user));

        // 2. Convert user into a shop. No problem!
        vm.prank(KittyConnectOwner);
        kittyConnect.addShop(user);
        assertEq(kittyConnect.getKittyShopAtIdx(2), user);
    }
```

```
➜  2024-03-kitty-connect git:(main) ✗ forge test --mt
test_differentBehaivourBetweenFunctions
[⠊] Compiling...
[⠒] Compiling 1 files with 0.8.19
[⠑] Solc 0.8.19 finished in 1.64s
Compiler run successful!

Ran 1 test for test/AuditTest.t.sol:AuditTest
```

```
[PASS] test_differentBehaivourBetweenFunctions() (gas: 687515)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.36ms (227.08µs CPU
time)

Ran 1 test suite in 146.46ms (1.36ms CPU time): 1 tests passed, 0 failed, 0
skipped (1 total tests)
➜  2024-03-kitty-connect git:(main) ✗
```

As we can see, there is different ways to surpass the requirement `KittyConnect__CatOwnerCantBeShopPartner`

Recommended Mitigation: It is advisable to review the entire protocol to ensure that the requirement is met, by adding the appropriate requires. For example, for safeTransferFrom():

```
function safeTransferFrom(address currCatOwner, address newOwner, uint256 tokenId,
bytes memory data)
        public
        override
        onlyShopPartner
    {
        require(_ownerOf(tokenId) == currCatOwner, "KittyConnect__NotKittyOwner");

        require(getApproved(tokenId) == newOwner,
"KittyConnect__NewOwnerNotApproved");

        /**********  ADD THIS *****************/
        require(!s_isKittyShop[newOwner],
"KittyConnect__CatOwnerCantBeShopPartner");

        _updateOwnershipInfo(currCatOwner, newOwner, tokenId);

        emit CatTransferredToNewOwner(currCatOwner, newOwner, tokenId);
        _safeTransfer(currCatOwner, newOwner, tokenId, data);
    }
```