KittyConnect Audit Report

Version 1.0

Ardeshir Gholami

March 31, 2024

KittyConnect Audit Report

Ardeshir Gholami

27 feb 2024

Prepared by: Ardeshir Lead Auditors: - Ardeshir Gholami

Table of Contents

    * [H-3] `KittyConnect::mintBridgedNFT` Does Not Update `KittyConnect::s_ownerToCatsTokenId`, Preventing Owners from Bridging NFTs Back to the Original Chain
- Medium
    * [M-1] Pointless Access Control in `KittyBridge::_ccipReceive` Preventing Router Contract Access
    * [M-2] Bypassing Shop Authorization Access Control via `ERC721::transferFrom`
- Low
    * [L-1] `KittyConnect::_updateOwnershipInfo` Does Not Update `s_ownerToCatsTokenId` of the Previous Owner
    * [L-2] Missing Event Emission When Changing Storage Variable `KittyBridge::gasLimit` in `KittyBridge::updateGaslimit` Function
- Informational
    * [I-1] Unused Error Definitions in `KittyBridge`
    * [I-2] Unused Event Definition in `KittyBridge`

Protocol Summary

This project allows users to buy a cute cat from our branches and mint NFT for buying a cat. The NFT will be used to track the cat info and all related data for a particular cat corresponding to their token ids. Kitty Owner can also Bridge their NFT from one chain to another chain via Chainlink CCIP.

Disclaimer

I make 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 team 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
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

The findings described in this document correspond the following commit hash:

`c0a6f2bb5c853d7a470eb684e1954dba261fb167`

Scope

The codebase is broken up into 2 contracts (In Scope):

1. KittyConnect.sol
2. KittyBridge.sol

Roles

1. Cat Owner User who buy the cat from our branches and mint NFT for buying a cat.
2. Shop Partner Shop partner provide services to the cat owner to buy cat.
3. KittyConnect Owner Owner of the contract who can transfer the ownership of the contract to another address.

Executive Summary

How this Project Works

Buying a Cat

A user is required to visit our shop partner to buy a cat. The shop partner will call the function from KittyConnect contract to mint NFT for buying a cat. (This NFT will track all the data related to the cat)

Bridge Kitty NFT from one chain to another chain

User can bridge Kitty NFT from one chain to another chain by calling this function from KittyConnect contract. This involves burning of the kitty NFT on the source chain and minting on the destination chain. Bridging is powered by chainlink CCIP.

Transferring Ownership of cat to new owner

Sometimes a user wants to transfer their cat to a new owner, this can be easily done by transferring the Kitty NFT to that desired owner. A user is first required to approve the kitty NFT to the new owner, and is then required to visit our shop partner to finally facilitate transfer the ownership of the cat to the new owner. ## Issues found

Severity	number of issues found
High	3
Medium	2
Low	2
Info	2
Total	9

Findings

High

[H-1] Missing Link Token Approval in `KittyBridge::bridgeNftWithData` Making Transaction to Revert

Description: The `KittyBridge::bridgeNftWithData` function is designed to facilitate the transfer of NFTs across different blockchain networks. However, it does not explicitly approve the Chainlink CCIP Router to take fees in the form of LINK tokens.

The ChainLink CCIP Example:

```
    .
    .
    .
    function sendMessage(
        uint64 destinationChainSelector,
        address receiver,
        string calldata text
    ) external onlyOwner returns (bytes32 messageId) {
        .
        .
        .
        if (fees > s_linkToken.balanceOf(address(this)))
            revert NotEnoughBalance(s_linkToken.balanceOf(address(this)), fees);

        // approve the Router to transfer LINK tokens on contract's behalf. It will spend the fees in LINK
@>      s_linkToken.approve(address(s_router), fees);

        // Send the message through the router and store the returned message ID
        messageId = s_router.ccipSend(destinationChainSelector, evm2AnyMessage);
        .
        .
        .
    }
```

Kitty Bridge Function:

```
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)),
                fees
            );
        }
@>
        messageId = router.ccipSend(_destinationChainSelector, evm2AnyMessage);

        emit MessageSent(
            messageId,
            _destinationChainSelector,
            _receiver,
            _data,
            address(s_linkToken),
            fees
        );

        return messageId;
    }
```

Impact: This oversight leads to the transaction reverting when the CCIP Router attempts to deduct fees, as it lacks the necessary approval to do so.

Proof of Concept: To demonstrate this issue, one can attempt to execute the bridgeNftWithData function without first approving the CCIP Router to take LINK tokens. Add the following test to project test suit.

```
function test_callingBridgeNftWithDataFailsWithoutApproval()
        public
        partnerGivesCatToOwner
    {
        ////fund the kittyBridge with link
        uint256 InitialLink = 10_000_000_000_000_000_000;

        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);
        vm.startPrank(user);
        uint256 tokenId = kittyConnect.getTokenCounter();
        uint64 destChainSelector = 14767482510784806043;
        vm.expectRevert("ERC20: transfer amount exceeds allowance");
        kittyConnect.bridgeNftToAnotherChain(
            destChainSelector,
            address(kittyBridge),
            tokenId - 1 //tokenId 1
        );
    }
```

Recommended Mitigation: To mitigate this issue, the KittyBridge::bridgeNftWithData function should be modified to include a step where it approves the CCIP Router to take a certain amount of LINK tokens as fees.

```
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)),
                fees
            );
        }
+       s_linkToken.approve(address(router), fees);
        messageId = router.ccipSend(_destinationChainSelector,
           evm2AnyMessage);

        emit MessageSent(
            messageId,
            _destinationChainSelector,
            _receiver,
            _data,
            address(s_linkToken),
            fees
        );

        return messageId;
    }
```

[H-2] 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: To demonstrate this vulnerability, a malicious user could repeatedly call the `KittyConnect::bridgeNftToAnotherChain` 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);
    }
```

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

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

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

```
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-3] `KittyConnect::mintBridgedNFT` Does Not Update `KittyConnect::s_ownerToCatsTokenId`, Preventing Owners from Bridging NFTs Back to the Original Chain

Description: The `mintBridgedNFT` function within the NFT bridge protocol is designed to mint a new NFT on the target blockchain, effectively bridging an NFT from the original chain to the target chain. However, it appears that this function does not update the `s_ownerToCatsTokenId` mapping, which is crucial for tracking the ownership of NFTs across chains. This mapping is used in the `KittyConnect::bridgeNftToAnotherChain` function, and if it's empty, this function will revert if the user tries to send the NFT back to the original or any other chain.

```
function mintBridgedNFT(bytes memory data) external onlyKittyBridge {
        (
            address catOwner,
            string memory catName,
            string memory breed,
            string memory imageIpfsHash,
            uint256 dob,
            address shopPartner
        ) = abi.decode(data, (address, string, string, string, uint256,
            address));

        uint256 tokenId = kittyTokenCounter;
        kittyTokenCounter++;

        s_catInfo[tokenId] = CatInfo({
            catName: catName,
            breed: breed,
            image: imageIpfsHash,
            dob: dob,
            prevOwner: new address[](0),
            shopPartner: shopPartner,
            idx: s_ownerToCatsTokenId[catOwner].length
        });
@>      // s_ownerToCatsTokenId should be updated here
        emit NFTBridged(block.chainid, tokenId);
        _safeMint(catOwner, tokenId);
    }
```

```
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(); // this will revert if
    user has no other nfts in this chain or worth will even remove the
    wrong nft from the mapping

        if (idx < (userTokenIds.length - 1)) {
            s_ownerToCatsTokenId[msg.sender][idx] = lastItem;
        }

        emit NFTBridgeRequestSent(block.chainid, destChainSelector,
            destChainBridge, tokenId);
        i_kittyBridge.bridgeNftWithData(destChainSelector,
            destChainBridge, data);
    }
```

Impact: This issue can significantly impact the functionality of the NFT bridge, as it prevents owners from transferring their NFTs back to the original chain. This could lead to a loss of access to NFTs for users who have transferred them to another chain and wish to return them.

Proof of Concept: To demonstrate this issue, one can attempt to bridge an NFT back to the original chain after it has been minted on the target chain. If the `mintBridgedNFT` function does not update the `s_ownerToCatsTokenId` mapping, the system will not recognize the owner of the NFT, and the bridge operation will fail. This can be verified by attempting to call the `KittyConnect::bridgeNftToAnotherChain` function and observing that the operation reverts due to the lack of an updated `s_ownerToCatsTokenId` entry. To test this add following test function to protocol test suit, and run it using a forked speolia testnet url.

PoC

```
    function test_bridgeToOriginalChainWillFail()
        public
        partnerGivesCatToOwner
    {
        ////fund the kittyBridge with link
        uint256 InitialLink = 10_000_000_000_000_000_000;
        address linkHolder = 0x61E5E1ea8fF9Dc840e0A549c752FA7BDe9224e99
            ;
        vm.prank(linkHolder);
        linkToken.approve(address(kittyBridge), InitialLink + 1);
        vm.prank(address(kittyBridge));
        linkToken.transferFrom(linkHolder, address(kittyBridge),
            InitialLink);

        ////user bridges the nft to target chain ->
        vm.startPrank(user);
        uint64 destChainSelector = 14767482510784806043;
        bytes32 messageId;
        bytes memory _data;
        // i slightly changed the function to return messageId and
            _data to help me mock router message
        (messageId, _data) = kittyConnect.bridgeNftToAnotherChain(
            destChainSelector,
            address(kittyBridge),
            0 //tokenId 0
        );
        assertEq(kittyConnect.balanceOf(user), 0); // assert balanceOf
            user == 1 to make sure nft is indeed bridged and burned
        assertEq(kittyConnect.getCatsTokenIdOwnedBy(user).length, 0);
            // assert length of mapping == 0 to make sure mapping is
            updated when burend
        vm.stopPrank();

        //@notice note that in test we do not really bridge the nft and
            we will just manually call the receive function by pranking
            ccip router
        ////router calls the ccipReceive function on the kittyBridge in
            dstChain
        Client.Any2EVMMessage memory message;
        message = Client.Any2EVMMessage({
            messageId: messageId,
            sourceChainSelector: 14767482510784806043,
            sender: abi.encode(address(kittyBridge)),
            data: _data,
            destTokenAmounts: new Client.EVMTokenAmount[](0)
        });
        vm.prank(kittyBridge.getRouter()); //calling ccipReceive as
           router
        kittyBridge.ccipReceive(message);
        assertEq(kittyConnect.balanceOf(user), 1); // assert balanceOf
           user ==1 to make sure NFT is received by target contract

        //trying to bridge the nft agian
        vm.startPrank(user);
        vm.expectRevert();
        kittyConnect.bridgeNftToAnotherChain(
            destChainSelector,
            address(kittyBridge),
            1
        );
        assertEq(kittyConnect.getCatsTokenIdOwnedBy(user).length, 0);
           // assert length of mapping == 0 to make sure mapping is NOT
            updated when bridged and minted
    }
```

```
forge test --fork-url $SEPOLIA_RPC_URL3 --mt
    test_bridgeToOriginalChainWillFail
```

Recommended Mitigation: o mitigate this issue, the mintBridgedNFT function should be modified to include a step where it updates the s_ownerToCatsTokenId mapping with the new owner's address and the token ID of the newly minted NFT. Here's a simplified example of how the mintBridgedNFT function might be modified to include the necessary update to the s_ownerToCatsTokenId mapping:

```
    function mintBridgedNFT(bytes memory data) external onlyKittyBridge
        {
        (
            address catOwner,
            string memory catName,
            string memory breed,
            string memory imageIpfsHash,
            uint256 dob,
            address shopPartner
        ) = abi.decode(data, (address, string, string, string, uint256,
                address));

        uint256 tokenId = kittyTokenCounter;
        kittyTokenCounter++;

        s_catInfo[tokenId] = CatInfo({
            catName: catName,
            breed: breed,
            image: imageIpfsHash,
            dob: dob,
            prevOwner: new address[](0),
            shopPartner: shopPartner,
            idx: s_ownerToCatsTokenId[catOwner].length
        });

+       s_ownerToCatsTokenId[catOwner].push(tokenId);
        emit NFTBridged(block.chainid, tokenId);
        _safeMint(catOwner, tokenId);
    }
```

Medium

[M-1] Pointless Access Control in `KittyBridge::_ccipReceive` Preventing Router Contract Access

Description: The `_ccipReceive` function within the `KittyBridge` contract is designed to handle the reception of cross-chain asset transfers. However, it includes an access control mechanism that restricts the function's execution to only those addresses listed in `KittyBridgeBase::allowlistedSenders`. This restriction is unnecessary because `_ccipReceive` is a internal function and is called by `CCIPReceiver::ccipReceive` only which is checked by `CCIPReceiver::onlyRouter` modifier which already checks if the `msg.sender` is the Router contract. Here is the `onlyAllowListed` modifer which is used on `_ccipReceive` internal function:

```
    modifier onlyAllowlisted(uint64 _sourceChainSelector, address
        _sender) {
        if (!allowlistedSourceChains[_sourceChainSelector]) {
            revert KittyBridge__SourceChainNotAllowlisted(
                _sourceChainSelector);
        }
@>      if (!allowlistedSenders[_sender])
@>          revert KittyBridge__SenderNotAllowlisted(_sender);
        _;
    }
```

And here is the `onlyRouter` modifier used on `ccipReceive` external function, which essentially checks the same thing.

```
    modifier onlyRouter() {
@>     if (msg.sender != address(i_router)) revert InvalidRouter(msg.
    sender);
        _;
    }
```

Impact: This access control issue can severely impact the functionality of the NFT bridge, especially in scenarios where the CCIP Router is not included in the allowlist.(whcih already is the case because its not included in allowlist in deploy script `DeployKittyConnect.s.sol`) It would prevent the bridge from receiving assets from other blockchains, effectively breaking the cross-chain functionality. This could lead to a significant loss of functionality for users and developers relying on the bridge for cross-chain NFT transfers, negatively affecting the overall user experience and the bridge's utility.

Proof of Concept: Use the following code in your test suit, Note that the message sent by Router is just random strings as it does not matter here:

```
function test_routerFailsToCallRecieverFunctionExtraAccessControl()
    public {
        Client.Any2EVMMessage memory message;
        //   struct Any2EVMMessage {
        //     bytes32 messageId; // MessageId corresponding to
            ccipSend on source.
        //     uint64 sourceChainSelector; // Source chain selector.
        //     bytes sender; // abi.decode(sender) if coming from an
            EVM chain.
        //     bytes data; // payload sent in original message.
        //     EVMTokenAmount[] destTokenAmounts; // Tokens and their
            amounts in their destination chain representation.
        //   }

        message = Client.Any2EVMMessage({
            messageId: bytes32(abi.encode("some random message Id")),
            sourceChainSelector: 14767482510784806043,
            sender: abi.encode(address(kittyBridge)),
            data: abi.encode("Not important data"),
            destTokenAmounts: new Client.EVMTokenAmount[](0)
        });
        // router tries to call ccipReceive by some random message
        address router = kittyBridge.getRouter();
        vm.prank(router);
        vm.expectRevert(
            abi.encodeWithSelector(
                KittyBridge__SenderNotAllowlisted.selector,
                router
            )
        );
```

```
            kittyBridge.ccipReceive(message);
        }
```

Recommended Mitigation: To mitigate this issue, the access control mechanism in the `_ccipReceive` function should be reviewed and modified to ensure that it does not inadvertently restrict legitimate contracts, such as the CCIP Router, from interacting with the bridge. This could be done via two ways:

1. Completely removing the access control check for msg.sender, as it's already done by the `onlyRouter` modifier, and just keeping the source chain check. This approach simplifies the function by removing unnecessary restrictions, ensuring that only the intended router can call `_ccipReceive`.
2. Adding the `i_router` to the allowlist in the constructor, if there are specific reasons to keep the address check. This method maintains the current access control mechanism but ensures that the CCIP Router is explicitly allowed to call `_ccipReceive`, thus maintaining the bridge's cross-chain functionality without compromising security. Here's a simplified example of how the access control might be modified to allow the CCIP Router to call `_ccipReceive` without being on the allowlist:

```diff
    modifier onlyAllowlisted(uint64 _sourceChainSelector, address
        _sender) {
        if (!allowlistedSourceChains[_sourceChainSelector]) {
            revert KittyBridge__SourceChainNotAllowlisted(
                _sourceChainSelector);
        }
-       if (!allowlistedSenders[_sender])
-           revert KittyBridge__SenderNotAllowlisted(_sender);
        _;
    }
```

[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: Exploit Steps: 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

[L-1] `KittyConnect::_updateOwnershipInfo` Does Not Update `s_ownerToCatsTokenId` of the Previous Owner

Description: The KittyConnect::_updateOwnershipInfo function is intended to update the ownership information of an NFT within the NFT bridge protocol. However, it appears that this function does not update the s_ownerToCatsTokenId mapping for the previous owner of the NFT. This mapping is crucial for tracking the ownership of NFTs across chains, as it links an owner's address to the token ID of their NFT.

```
 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);
@>      //s_ownerToCatsTokenId[currCatOwner] is not removed
    }
```

Impact: While this issue might not have a major impact on the overall functionality of the bridge, as other functions use multiple checks before bridging or transferring an NFT, it is still important to ensure that the s_ownerToCatsTokenId mapping does not contain incorrect data. Incorrect data in this mapping could cause confusion for users, potentially leading to misinterpretations of ownership and incorrect actions, such as attempting to transfer an NFT to the wrong owner or failing to recognize the current owner of an NFT.

Proof of Concept: There is no need to proof of concept because the test suit already has a test function which fails because of this issue.

```
function test_transferCatToNewOwner() public {
        string
            memory catImageIpfsHash = "ipfs://
                QmbxwGgBGrNdXPm84kqYskmcMT3jrzBN8LzQjixvkz4c62";
        uint256 tokenId = kittyConnect.getTokenCounter();
        address newOwner = makeAddr("newOwner");

        // Shop Partner gives Cat to user
        vm.prank(partnerA);
        kittyConnect.mintCatToNewOwner(
            user,
            catImageIpfsHash,
            "Meowdy",
            "Ragdoll",
            block.timestamp
        );

        // Now user wants to transfer the cat to a new owner
        // first user approves the cat's token id to new owner
        vm.prank(user);
        kittyConnect.approve(newOwner, tokenId);

        // then the shop owner checks up with the new owner and confirms the transfer
        vm.expectEmit(false, false, false, true, address(kittyConnect));
        emit CatTransferredToNewOwner(user, newOwner, tokenId);
        vm.prank(partnerA);
        kittyConnect.safeTransferFrom(user, newOwner, tokenId);

        uint256[] memory newOwnerTokenIds = kittyConnect.getCatsTokenIdOwnedBy(
            newOwner
        );
        KittyConnect.CatInfo memory catInfo = kittyConnect.getCatInfo(tokenId);
        string memory tokenUri = kittyConnect.tokenURI(tokenId);
        console.log(tokenUri);
@>      assert(kittyConnect.getCatsTokenIdOwnedBy(user).length == 0);    // this fails because the length will stay 1 after transfer,
        assert(newOwnerTokenIds.length == 1);
        assert(newOwnerTokenIds[0] == tokenId);
        assert(catInfo.prevOwner[0] == user);
    }
```

Recommended Mitigation: Add the following line to `_updateOwnershipInfo` function:

```
 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);
+       s_ownerToCatsTokenId[currCatOwner].pop();
    }
```

[L-2] Missing Event Emission When Changing Storage Variable `KittyBridge::gasLimit` in `KittyBridge::updateGaslimit` Function

Description: The KittyBridge::updateGaslimit function is designed to update the gasLimit storage variable within the NFT bridge protocol. However, it appears that there is no event emission when this variable is changed.

```
    function updateGaslimit(uint256 gasLimit) external onlyOwner {
        gaslimit = gasLimit;
@>
    }
```

Impact: This omission can make it difficult for external observers, such as front-end applications or other smart contracts, to track changes to the gas limit, which is crucial for monitoring and debugging purposes.

Proof of Concept:

Recommended Mitigation: To mitigate this issue, an event should be emitted whenever the `gasLimit` variable is changed within the `updateGaslimit` function. This event should include the new gas limit value, allowing external observers to easily track changes to the gas limit. Here's an example of how you might define and emit an event for this purpose within the `updateGaslimit` function:

1. Add event declaration to `KittyBridgeBase.sol`:

```
+     event GaslimitChanged(
+         uint256 indexed GasLimit,
+     );
```

2. Emit event in `updateGaslimit` function:

```
    function updateGaslimit(uint256 gasLimit) external onlyOwner {
        gaslimit = gasLimit;
+       emit GaslimitChanged(gasLimit);
    }
```

Informational

[I-1] Unused Error Definitions in `KittyBridge`

Description: Within the `KittyBridge` contract, there are two error definitions: `KittyBridge__NothingToWith` and `KittyBridge__FailedToWithdrawEth`. However, these errors are defined but not used anywhere in the contract. This situation can lead to confusion for developers and auditors, as it suggests that the contract might be missing functionality or that the errors are intended for future use but have not been implemented yet.

[I-2] Unused Event Definition in `KittyBridge`

Description: Within the contract, there is an event definition for `TokensRedeemedForVetVisit`. However, this event is defined but not used anywhere in the contract. This situation can lead to confusion for developers and auditors, as it suggests that the contract might be missing functionality or that the event is intended for future use but has not been implemented yet.