
BELLUM
GALAXY

Table of Contents

▶ Details

See table

- Table of Contents
- About Barba
- Disclaimer
- Risk Classification
- Protocol Summary
- Audit Details
  - Scope
  - Roles
- Executive Summary
  - Issues found
- Audit Findings
  - High Severity Vulnerabilities
    - `KittyBridge.sol::bridgeNftWithData` doesn't implement access control mechanisms, leading to free mint cross-chain and waste of link funds.
    - `KittyConnect.sol::mintBridgedNFT` increments the counter every time one NFT is bridged in, inflating the totalSupply with copies and creating collection ID collisions
    - `KittyConnect.sol::mintCatToNewOwner` didn't check for input values, leading to the creation of NFTs without information or even manipulation.
  - Medium Severity Vulnerabilities
    - `KittyConnect.sol::safeTransferFrom` didn't update the owner registry, accumulating all the cats that he has, and the ones he already transferred.
    - `KittyConnect.sol::bridgeNftToAnotherChain` is missing the `idx` data in the bridge message, breaking the protocol's purpose.
  - Low Severity Vulnerabilities
    - `KittyConnect.sol::mintCatToNewOwner` doesn't follow the CEI pattern.
    - Several `KittyBridge.sol` functions don't emit events after storage update.
  - Gas Observations
    - Consider implementing `custom errors` instead of `require` statements
  - Informational Observations
    - The `KittyConnect.sol::TokensRedeemedForVetVisit` event is not being used.

About Barba

Solidity Developer, Security Researcher, Founder of Bellum Galaxy and Chainlink Advocate. With three months of programming experience I developed a Top Quality Project at Chainlink Constellation Hackathon. In my first competitive audit, I achieved a Top 5 position. I am a competitive person who daily fights for improvement. Driven by this way of thinking I founded Bellum Galaxy, a educacional community focused on science and technology to help people face life challeges, and grow personally and professionally.

Disclaimer

The Bellum Galaxy team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the 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.

Protocol Summary

Audit Details

- **Project Name:**
  - Kitty Connect
- **Smart Contract Address:**
  - None
- **Audit Date:**
  - 29/03/2024
- **The findings described in this document correspond the following commit hash:**
  - c0a6f2bb5c853d7a470eb684e1954dba261fb167

Scope

```
#-- src
|   #-- KittyBridge.sol
|   #-- KittyConnect.sol
```

Roles

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

Executive Summary

Issues found

Severity	Number of issues found
High	3
Medium	2
Low	2
Info	1
Total	8

Audit Findings

High Severity Vulnerabilities

`KittyBridge.sol::bridgeNftWithData` doesn't implement access control mechanisms, leading to free mint cross-chain and waste of link funds.

- **Description:**
  - `KittyConnect.sol::bridgeNftToAnotherChain` provides a way to transfer NFTs cross-chain through the `KittyBridge.sol::bridgeNftWithData` function. However, the `KittyBridge.sol::bridgeNftWithData` doesn't control access, allowing a malicious user to call it directly.
- **Impact:**
  - This missimplementation can lead to uncontrolled NFT emission on destination chains and use of all link funds.
- **Proof of Concept:**

▶ Add the code below to `KittyTest.t.sol`

```
    function testPoCAnyoneCanBridgeNfts() public {
        address sender = makeAddr("sender");
        bytes memory data = abi.encode(makeAddr("catOwner"), "meowdy",
"ragdoll", "ipfs://QmbxwGgBGrNdXPm84kqYskmcMT3jrzBN8LzQjixvkz4c62",
block.timestamp, partnerA);

        vm.prank(kittyConnectOwner);
        kittyBridge.allowlistSender(networkConfig.router, true);

        vm.prank(sender);
        kittyBridge.bridgeNftWithData(networkConfig.otherChainSelector,
sender, data);
    }
```

- **Recommendation:**
  - Implement the `onlyKittyConnect` modifier to control access.

    ▶ Adjust the code as follows

```
    function bridgeNftWithData(uint64 _destinationChainSelector,
                               address _receiver,
                               bytes memory _data) external
+                                              onlyKittyConnect

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;
    }
```

`KittyConnect.sol::mintBridgedNFT` increments the counter every time one NFT is bridged in, inflating the totalSupply with copies and creating collection ID collisions

- **Description:**
  - `KittyConnect.sol::mintBridgedNFT` increments the counter every time one NFT is bridged in. However, the ID is never subtracted, once the NFT is bridged out.
  - Once an NFT is bridged, it is transferred. Transference means the particular NFT, with those specifications, goes to another blockchain. It doesn't become something else, different.
- **Impact:**
  - With this approach, several duplicates will be created, inflating the total supply every time an NFT is bridged out and comes back in. It also creates collection ID collisions through all the blockchains that receive these NFTs.
- **Proof of Concept:**

▶ Add the following code to `KittyTest.t.sol`

```
    function testPoCNFTIdIncrementedOnNFTReturn() public {
        address DogsGuy = makeAddr("BARBA");
        bytes memory data = abi.encode(DogsGuy, "Athena", "AmericanBull",
"ipfs://QmbxwGgBGrNdXPm84kqYskmcMT3jrzBN8LzQjixvkz4c62", block.timestamp,
partnerA);

        uint256 catId = kittyConnect.getTokenCounter();

        vm.prank(address(kittyBridge));
        kittyConnect.mintBridgedNFT(data);

        uint256 catIdAfterBridginIn = kittyConnect.getTokenCounter();

        assertEq(catId, 0);
        assertEq(catIdAfterBridginIn, 1);
    }
```

- **Recommendation:**
    - Consider adjusting the protocol logic to allow the NFT to maintain its original ID in any blockchain.

        ▶ Adjust the code of `KittyConnect.sol::mintBridgedNFT` as follows

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

        emit NFTBridged(block.chainid, tokenId);
        _safeMint(catOwner, tokenId);
    }
```

`KittyConnect.sol::mintCatToNewOwner` didn't check for input values, leading to the creation of NFTs without information or even manipulation.

- **Description:**
    - The protocol's purpose is to allow users to mint an NFT that will store the information of a cat and track all related data. However, the `KittyConnect.sol::mintCatToNewOwner` is not validating inputs before emitting the NFT.
- **Impact:**

- Skipping the validation process can incur on NFTs with empty data. Or even manipulated data by `ShopPartners`.

- **Proof of Concept:**

▶ Add the following PoC to `KittyTest.t.sol`

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

        vm.prank(partnerA);
        kittyConnect.mintCatToNewOwner(user, catImageIpfsHash, "", "", 1);

        uint32 timeNow = 1711738682;
        vm.warp(timeNow);

        uint256 catAge = kittyConnect.getCatAge(tokenId);

        assertTrue(catAge > 1_000_000_000);
    }
```

- **Recommendation:**

  - Always check for input values. Especially if it's not updatable.

▶ Implement the following code

```
    function mintCatToNewOwner(address catOwner, string memory
catIpfsHash, string memory catName, string memory breed, uint256 dob)
external onlyShopPartner {
        require(!s_isKittyShop[catOwner],
"KittyConnect__CatOwnerCantBeShopPartner");
+       require(catOwner != address(0),
"KittyConnect__InvalidOwnerAddress");
+       require(bytes(catIpfsHash).length > 0,
"KittyConnect__EmptyIpfsHash");
+       require(bytes(catName).length > 0,
"KittyConnect__EmptyCatName");
+       require(bytes(breed).length > 0, "KittyConnect__EmptyBreed");
+       require(dob > 0 && dob <= block.timestamp,
"KittyConnect__InvalidDOB");

        uint256 tokenId = kittyTokenCounter;
        kittyTokenCounter++;

        s_catInfo[tokenId] = CatInfo({
            catName: catName,
            breed: breed,
            image: catIpfsHash,
```

```
                dob: dob,
                prevOwner: new address[](0),
                shopPartner: msg.sender,
                idx: s_ownerToCatsTokenId[catOwner].length
            });

            s_ownerToCatsTokenId[catOwner].push(tokenId);

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

Medium Severity Vulnerabilities

`KittyConnect.sol::safeTransferFrom` didn't update the owner registry, accumulating all the cats that he has, and the ones he already transferred.

- **Description:**
  - `KittyConnect.sol` allows owners to transfer their cats to a third party. The `KittyConnect.sol::safeTransferFrom` function was implemented and is operational to enable it. However, the storage is not updated correctly. Although the owner transfers his cat, his registers will always store the registry about it.
- **Impact:**
  - The previous owner will always have all the registers of cats that he had once. It will also keep increasing his registers. For example:
    - If he bought four cats and, for some reason, he sold these four cats.
    - The next buy will be number five. And not the number one, as it should be, considering that he is no longer the owner of the first four.
- **Proof of Concept:**

▶ Add the following code to `KittyTest.t.sol`

```
    function testPoCPreviousOwnerInfoItsNotUpdated() 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 firstOwner =
kittyConnect.getCatsTokenIdOwnedBy(user);
        uint256[] memory newOwnerTokenIds =
kittyConnect.getCatsTokenIdOwnedBy(newOwner);

        KittyConnect.CatInfo memory catInfo =
kittyConnect.getCatInfo(tokenId);
        string memory tokenUri = kittyConnect.tokenURI(tokenId);

        assertEq(firstOwner.length, 1);
    }
```

- **Recommendation:**
  - Always remember to update all storage dependencies related to the element being utilized.

    ▶ Implement the following adjustments

```
    function _updateOwnershipInfo(address currCatOwner, address
newOwner, uint256 tokenId) internal {
        //Push the current owner as a previous one.
        s_catInfo[tokenId].prevOwner.push(currCatOwner);
        //Update cat info, ading the number os cats of the next owner.
        s_catInfo[tokenId].idx = s_ownerToCatsTokenId[newOwner].length;
        //Push the cat into the newOwner ownership track.
        s_ownerToCatsTokenId[newOwner].push(tokenId);

+       _removeCatFromOwner(currCatOwner, tokenId);
    }

+   function _removeCatFromOwner(address owner, uint256 tokenId)
internal {
+       uint256 lastCatIndex = s_ownerToCatsTokenId[owner].length - 1;
+       uint256 catIndex;

+       for(uint256 i = 0; i <= lastCatIndex; i++) {
+           if(s_ownerToCatsTokenId[owner][i] == tokenId) {
+               catIndex = i;
+               break;
+           }
```

```
+       }

+       if (catIndex < lastCatIndex) {
+           s_ownerToCatsTokenId[owner][catIndex] =
s_ownerToCatsTokenId[owner][lastCatIndex];
+       }
+       s_ownerToCatsTokenId[owner].pop();
+   }
```

`KittyConnect.sol::bridgeNftToAnotherChain` is missing the `idx` data in the bridge message, breaking the protocol's purpose.

- **Description:**
  - Store Cat information is the core of the Kitty protocol. Although, the `KittyConnect.sol::bridgeNftToAnotherChain` function fails to deliver it by excluding the parameter `idx` of the `KittyConnect.sol::CatInfo` struct.
- **Impact:**
  - This not only breaks the protocol core but also private potential new owners and the current owner itself from clear information about your own asset.
  - As follows the function implementation, this info is already deleted from the storage. So, the information is also lost forever.
- **Proof of Concept:**

▶ See the miss implementation below

```
    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;
        //@audit-high missing information. The position on the cat ownership
array is missing.
@>      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];

        //@audit-high pop the last registry without check if it's the right
```

```
cat.
        s_ownerToCatsTokenId[msg.sender].pop();

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

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

- **Recommendation:**

▶ Adjust the code as follow

```
    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;
        //@audit-high missing information. The position on the cat ownership
array is missing.
-       bytes memory data = abi.encode(catOwner, catInfo.catName,
catInfo.breed, catInfo.image, catInfo.dob, catInfo.shopPartner);
+       bytes memory data = abi.encode(catOwner, catInfo.catName,
catInfo.breed, catInfo.image, catInfo.dob, catInfo.shopPartner, idx);

        _burn(tokenId);
        delete s_catInfo[tokenId];

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

        //@audit-high pop the last registry without check if it's the right
cat.
        s_ownerToCatsTokenId[msg.sender].pop();

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

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

```
    }
```

Low Severity Vulnerabilities

`KittyConnect.sol::mintCatToNewOwner` doesn't follow the CEI pattern.

- **Description:**
  - `KittyConnect.sol::mintCatToNewOwner` emits the `KittyConnect.sol::CatMinted` after a function call.
- **Impact:**
  - Event emission after calls can open opportunities for the manipulation of events. If this event is being monitored by another mechanism it can lead to unexpected outcomes.
- **Recommendation:**
  - Always follow the CEI pattern.

Several `KittyBridge.sol` functions don't emit events after storage update.

- **Description:**
  - The following functions don't implement events after storage update:

    ▶ Verify instances below

```
    function allowlistDestinationChain(uint64
_destinationChainSelector, bool allowed) external onlyOwner {
        allowlistedDestinationChains[_destinationChainSelector] =
allowed;
    }
```

```
    function allowlistSourceChain(uint64 _sourceChainSelector, bool
allowed) external onlyOwner {
        allowlistedSourceChains[_sourceChainSelector] = allowed;
    }
```

```
    function allowlistSender(address _sender, bool allowed) external
onlyOwner {
        allowlistedSenders[_sender] = allowed;
    }
```

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

- **Impact:**
  - Loss of transparency on protocol process dificulting track of critical information.
- **Recommendation:**
  - Follow best practices and always emit an event after storage update.

Gas Observations

Consider implementing `custom errors` instead of `require` statements

- **Description:**
  - Update the following instances with `custom errors`

    ▶ Check the instances bellow

```
    modifier onlyKittyConnectOwner() {
        //@audit-gas consider use custom errors
        require(msg.sender == i_kittyConnectOwner,
"KittyConnect__NotKittyConnectOwner");
        _;
    }
```

```
    modifier onlyShopPartner() {
        //@audit-gas consider use custom errors
        require(s_isKittyShop[msg.sender],
"KittyConnect__NotAPartner");
        _;
    }
```

```
    modifier onlyKittyBridge() {
        //@audit-gas consider use custom errors
        require(msg.sender == address(i_kittyBridge),
"KittyConnect__NotKittyBridge");
        _;
    }
```

- **Recommendation:**
  - Implement `custom errors` instead of `require` statments

Informational Observations

The `KittyConnect.sol::TokensRedeemedForVetVisit` event is not being used.

- **Description:**
  - The `KittyConnect.sol::TokensRedeemedForVetVisit` event is not being used, consider removing it.
- **Recommendation:**
  - Follow the best practices and remove the deadcode.