OmniNFT

Second

Security Review

June 30, 2024

Conducted by:
Blckhv, Independent Security Researcher
Slavcheww, Independent Security Researcher

Contents

1. About SBSecurity

SBSecurity is a duo of skilled smart contract security researchers. Based on the audits conducted and numerous vulnerabilities reported, we strive to provide the absolute best security service and client satisfaction. While it's understood that 100% security and bug-free code cannot be guaranteed by anyone, we are committed to giving our utmost to provide the best possible outcome for you and your product.

Book a Security Review with us at sbsecurity.net or reach out on Twitter @Slavcheww.

2. Disclaimer

A smart contract security review can only show the presence of vulnerabilities **but not their absence**. Audits are a time, resource, and expertise-bound effort where skilled technicians evaluate the codebase and their dependencies using various techniques to find as many flaws as possible and suggest security-related improvements. We as a company stand behind our brand and the level of service that is provided but also recommend subsequent security reviews, on-chain monitoring, and high whitehat incentivization.

3. Risk classification

	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1. Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - leads to a moderate loss of assets in the protocol or some disruption of the protocol's functionality.
- **Low** - funds are not at risk.

3.2. Likelihood

- **High** - almost **certain** to happen, easy to perform, or highly incentivized.
- **Medium** - only **conditionally possible**, but still relatively likely.
- **Low** - requires specific state or **little-to-no incentive**.

3.3. Action required for severity levels

- High - **Must** fix (before deployment if not already deployed).
- Medium - **Should** fix.
- Low - **Could** fix.

4. Executive Summary

OmniNFT is a cross-chain NFT, that allows its users to mint, burn and transfer on any chain supported by the OmniCat token. The OmniCat token is locked into the Blast chain (source) when user mints NFT and then these OmniCat tokens are returned to the user when the NFT is burned, no matter on which chain. There are a limited number of NFTs in the collection, and the minting phase lasts as long as all the NFTs are not minted. Burns can only be initiated after the minting phase.

OmniNFT contracts have been audited through the Hyacinth platform.

Overview

Project	OmniNFT
Repository	https://github.com/omnicat-labs/omnicat-NFT
Commit Hash	a90efb3a12e078d21c99620c77066f1a9b110529
Resolution	9f4b243f14995c8b6d325757d90892647af56828
Timeline	June 25 - June 29, 2024

Scope

OmniNft.sol
OmniNFTA.sol
OmniNFTBase.sol

Issues Found

Critical Risk	1
High Risk	1
Medium Risk	3
Low/Info Risk	3

5. Findings

5.1. Critical severity

5.1.1. Payload for storing credits is not decoded properly

Severity: Critical Risk

Context: OmniNFT.sol#L191, OmniNFTA.sol#L113

Description: The payload passed to `_nonblockingLzReceive` will not be decoded properly in case gas is not enough and the payload has to be saved in `storedCredits`.

Due to the usage of `MessageType` and assembly, the payload will be forged and credits will be stored with the wrong payload hash making it impossible for the unminted `NFTs` to be taken.

Since `MessageType` is appended at the beggiging of the payload, then the actual payload is retrived in assembly with `add(_payload,1)`. however this one gives a big random hash which then it hardly to be re-computed.

Recommendation:

Approach that doesn't use assembly should be considered

Add slice() in OmniNFTBase and change payloadWithoutMessage retrieving with slice() in both _nonblockingLzReceive().

```
function slice(bytes calldata payload) public pure returns(bytes memory) {
    bytes memory payloadWithoutMessage = payload[1:];

    return payloadWithoutMessage;
}
```

```
 function _nonblockingLzReceive(
       uint16 _srcChainId,
       bytes memory _srcAddress,
       uint64, /*_nonce*/
       bytes memory _payload
   ) internal virtual override {
+      bytes memory payloadWithoutMessage = this.slice(_payload);
-      bytes memory payloadWithoutMessage;
-      assembly {
-          payloadWithoutMessage := add(_payload,1)
-      }
```

Resolution: Fixed

5.2. High severity

5.2.1. `UserMintedNumber` is not updated in `onOFTReceived`

Severity: High Risk

Context: OmniNFTA.sol#L191

Description: `OmniNFTA::onOFTReceived` doesn't increase the `UserMintedNumber` when NFTs are being minted, allowing users to mint more than `MAX_MINTS_PER_ACCOUNT`, gaining an unfair advantage in the system.

Recommendation: Apply the following changes to the code:

```
function onOFTReceived(uint16 _srcChainId, bytes calldata , uint64 , bytes32 _from, uint _amount, bytes calldata _payload) external override {
      require(msg.sender == address(omnicat), "not omnicat");

      address rootCaller = address(uint160(uint256(_from)));
      address trustedRemoteLookupAddress = retrieveTrustedRemote(_srcChainId);
      require(rootCaller == trustedRemoteLookupAddress, "not trusted caller");

      MessageType messageType = MessageType(uint8(_payload[0]));
      if(messageType == MessageType.MINT){
          (address userAddress, uint256 mintNumber) = abi.decode(_payload[1:], (address, uint256));
          if((_amount < mintNumber*MINT_COST) || (mintStartTimestamp > block.timestamp) || (nextTokenIdMint + mintNumber > COLLECTION_SIZE) || (UserMintedNumber[msg.sender] + min
              // create refund for user
              omniUserRefund[userAddress][_srcChainId] += _amount;
              emit SetUserOmniRefund(userAddress, _srcChainId, omniUserRefund[userAddress][_srcChainId]);
              return;
          }

+         UserMintedNumber[msg.sender] += mintNumber;

          for(uint256 i=0;i<mintNumber;){
              _mint(userAddress, nextTokenIdMint);
              nextTokenIdMint++;
              unchecked {
                  i++;
              }
          }
      }
  }
```

Resolution: Fixed

5.3. Medium severity

5.3.1. Passing `msg.sender` as interchain recipient can cause loss of NFTs

Severity: Medium Risk

Context: OmniNft.sol#L89

Description: `OmniNFT::mint` on the source chain uses `msg.sender` as the NFT recipient on the destination, although EOAs will always have identical addresses on both chains, this is not true for smart contracts and old versions of smart wallets that were being deployed at different addresses across the other chains.

Recommendation: Extend the mint function to allow minters to provide the address which to receive the minted `NFTs` on the Blast chain.

Resolution: Fixed

5.3.2. Arbitrary zroPaymentAddress can be passed

Severity: Medium Risk

Context: OmniNft.sol#L84

Description: `OmniNFT::mint` allows passing non-zero `zroPaymentAddress`, which means the callers want to pay in `layerZeroToken`, but `OmniNFT` system is not designed to work like that and it always requires passing enough from the native token.

Recommendation: Both `OmniNFT::mint`, `OmniNFT::burn` and their estimate functions should pass `address(0)` to `lzCallParams::zroPaymentAddress`

Resolution: Fixed

5.3.3. Usage of `_mint` instead of `_safeMint` can lock `NFTs`

Severity: Medium Risk

Context: OmniNFTA.sol#L193

Description: `OmniNFTA::onOFTReceived` uses `_mint` for the interchain mint, but using it instead of `_safeMint` can lock some `NFTs` if the recipient is a smart contract that cannot operate with `ERC721` tokens.

Recommendation: Use `_safeMint` instead.

Resolution: Fixed

5.4. Low/Info severity

5.4.1. `TokenURI` do not follow EIP-721

Severity: Low Risk

Context: OmniNFTBase.sol#L133

Description: In order to be `EIP721` compliant the `tokenURI` should check the existence of the `tokenId`, which is not being performed here:

```
function tokenURI(uint256 tokenId) public view override returns (string memory) {
      return bytes(baseURI).length > 0 ? string(abi.encodePacked(baseURI, tokenId.toString())) : "";
}
```

Recommendation: No need to override `tokenURI` - the implementation in `ERC721` contract which is inherited in `OmniNFT` will use your overriden `_baseURI`:

```
function tokenURI(uint256 tokenId) public view virtual override returns (string memory) {
   _requireMinted(tokenId);

   string memory baseURI = _baseURI();
   return bytes(baseURI).length > 0 ? string(abi.encodePacked(baseURI, tokenId.toString())) : "";
}
```

Resolution: Fixed

5.4.2. `OmniNFTA::sendNFTRefund` redundant gas configuration

Severity: Low Risk

Context: OmniNFTA.sol#L235

Description: `adapterParams` takes the length of the `refundObject` but since the only way to be NFT refunds to the destination chain is if burn fails. But the burn is always for a single token.

Recommendation: In `OmniNFT::sendNFTRefund` remove the `refundObject.tokens.length*uint256(dstGasReserve)` from `adapterParams` and pass only `dstGasReserve`.

Resolution: Fixed

5.4.3. Use `call` instead of `send`

Severity: Low Risk

Context: OmniNFT.sol#L116

Description: In `OmniNFT::mint` gas refunds are being performed with send which forwards only 2300 gas, in cases where the recipient has some logic in his `receive` function there won't be a way him to execute the interchain mint.

Recommendation: Replace `send` with `call` the same way as it is being done in the `OmniNFT::burn`.

Resolution: Fixed