Bridge Audit Report

Severity Classification

Severity	Count	Status
High	3	Resolve
Medium	2	Resolve
Low	2	Resolve
Informational	2	Resolve
Gas Optimization	2	Resolve

[H-01] Insufficient Signature Malleability Check in `SignatureUtils:recoverSigner`

Description The `recoverSigner` function attempts to recover the signer's address from a given message hash and signature. However, after extracting the `v`, `r`, and `s` values from the signature, it fails to enforce a crucial check on the `s` value. The `s` value in an ECDSA signature should always be in the lower half order to ensure signature uniqueness. Without this validation, the function allows for malleable (non-unique) signatures, leading to potential replay attacks. The EVM's `ecrecover` precompile permits multiple valid signatures for the same hash, meaning an attacker could generate two different signatures for the same message, potentially exploiting the system.

Impact Allows signature malleability, making the system vulnerable to replay attacks.

Recommended Mitigation Incorporate an additional check similar to OpenZeppelin's implementation to ensure `s` is within the lower half order:

```
In SignatureUtils:recoverSigner

assembly {
            r := mload(add(signature, 0x20))
            s := mload(add(signature, 0x40))
            v := byte(0, mload(add(signature, 0x60)))
        }

+         if (uint256(s) > 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE9
+             return (address(0));
+         }

        if (v < 27) {
            v += 27;
        }
```

Status - Fixed

[H-02] Insufficient Multi-Chain Replay Attack Protection in `SignatureUtils` Contract

Description The contract generates a signature hash using the `getEthSignedMessageHash` function. However, this function currently lacks additional contextual elements such as `chainId()` and `address(this)`. Without these elements, an attacker could reuse the same signature across multiple chains, leading to cross-chain replay attacks.

Impact An attacker could use a valid signature from one blockchain on another chain, executing unintended transactions.

Recommended Mitigation Enhance the signature hash creation by incorporating `block.chainId` and `address(this)`, ensuring uniqueness across different chains:

```
function getEthSignedMessageHash(
        bytes32 _messageHash
    ) internal pure returns (bytes32) {
        return
            keccak256(
                abi.encodePacked(
                    "\x19Ethereum Signed Message:\n32",
+                       block.chainId,
+                       address(this),
                       _messageHash
                   )
               );
    }
```

Status - Fixed

[H-03] Missing Constructor and `_disableInitializers()` to Prevent Unauthorized Initialization

Description Upgradeable contracts should explicitly disable their initializer function to prevent unauthorized initialization of the implementation contract. Failing to do so allows anyone to call the initializer function and take control of the contract.

Impact A malicious actor could initialize the contract, set themselves as an owner/admin, and gain unauthorized control.

Recommended Mitigation Include `_disableInitializers()` in the constructor:

```
+    constructor() {
+        _disableInitializers();
+    }
```

Status - Fixed

[M-01] Missing Call to `__UUPSUpgradeable_init()` in `Xenearaft:initialize`

Description The `Xenearaft` contract inherits `UUPSUpgradeable`, but fails to call its initializer function. This could cause unexpected behavior in upgradeable deployments.

Recommended Mitigation Add the missing initializer call:

```
function initialize() public initializer {
        __Ownable_init(msg.sender);
        __Pausable_init();
        __ReentrancyGuard_init();
+       __UUPSUpgradeable_init();

        adminLists[msg.sender] = true;
        signer = msg.sender;
    }
```

Status - Fixed

[M-02] Potential Returndata Gas Bomb Attack in `Xenearaft:unlock`

Description The `unlock` function executes a low-level call to `curUser_`:

```
(bool success, ) = address(curUser_).call{value: amount_}("");
require(success, "Unlock: Fail transfer native");
```

A malicious `curUser_` can exploit this by returning excessive data, causing a gas exhaustion attack, known as a “returndata bombing” attack. Although the user is responsible for covering the gas, they could exploit this vulnerability to force a revert with a dust amount.

Recommended Mitigation Use `ExcessivelySafeCall` with `maxCopy` set to 0 or 32 bytes to prevent excessive memory allocation.

Status - Acknowledge

[L-01] Missing Zero-Address Validation Leading to Potential State Corruption

Description Certain functions fail to validate addresses, potentially allowing incorrect state changes.

Examples:

```
function setSigner(address user_) external onlyAdmins {}
function setMinter(address minter_) external onlyOwner{}
```

Recommended Mitigation Add zero-address validation in relevant functions:

```
require(user_ != address(0), "Invalid address");
```

Status - Fixed

[L-02] Missing Explicit Function to Set and Validate Fees in `Xenearaft`

Description The contract allows zero-fee transactions if `fee == 0`, potentially undermining the lock and burn mechanism.

Impact Users may exploit fee-free transactions to bypass intended economic restrictions.

Recommended Mitigation Introduce a setter function with `minFee` and `maxFee` constraints.

Status - Acknowledge

[I-01] Use `Ownable2StepUpgradeable` Instead of `OwnableUpgradeable`

Description `OwnableUpgradeable` lacks a two-step ownership transfer process, making accidental or malicious transfers possible.

Recommended Mitigation Use `Ownable2StepUpgradeable` instead:

```diff
-    import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
+    import "@openzeppelin/contracts-upgradeable/access/Ownable2StepUpgradeable.
```

Status - Fixed

[I-02] Missing Event Emissions for Significant State Changes

Description State changes should emit events for better off-chain tracking.

Examples:

- `setAcceptedToken(..)`
- `setMinter(address minter_)`

Recommended Mitigation Add events to relevant functions.

Status - Fixed

[G-01] Use `external` Instead of `public` for Gas Optimization

Description Functions that are never called internally should be marked `external` for gas efficiency.

Examples:

```diff
- function initialize() public initializer {}
+ function initialize() external initializer {}
```

Status - Acknowledge

[G-02] Use Custom Errors Instead of `require` Statements to Save Gas

Description Using Solidity v0.8.4+ custom errors reduces gas consumption compared to string-based `require` statements.

Recommended Mitigation Replace `require` with custom errors:

```
error Unauthorized();
require(msg.sender == owner, Unauthorized());
```

Status - Fixed