CANTINA

Sweep n' Flip Bridge

Security Review

Cantina Managed review by:
Slowfi, Security Researcher
Sujith Somraaj, Security Researcher

January 16, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Sweep n' Flip is an AMM for NFTs that provides fundamental NFT Liquidity for multiple chains.

From Nov 11th to Nov 24th the Cantina team conducted a review of sweep-n-flip-monorepo on commit hash c1038810. The team identified a total of **36** issues in the following risk categories:

- Critical Risk: 2
- High Risk: 2
- Medium Risk: 4
- Low Risk: 12
- Gas Optimizations: 4
- Informational: 12

3 Findings

3.1 Critical Risk

3.1.1 Permanent failure to bridge wrapped ERC721 using `Bridge::sendERC721UsingNative` function

Severity: Critical Risk

Context: Bridge.sol#L159

Description: The `sendERC721UsingNative` function of `Bridge.sol` bridges ERC721 NFTs between multiple supported networks using Layerzero's message bridge.

This function helps to bridge both the original asset (the unwrapped asset) and the wrapped asset. However, due to an implementation error, this function works only for the original asset and locks the wrapped asset on the bridged chain, unable to bridge it.

This failure to bridge would permanently lock the original NFT in the source bridge contract.

Proof of Concept: The vulnerability is demonstrated using the following foundry test code. To run it, copy it to the test file and run `foundry test --mt test_e2e`

```
function test_e2e() external {
    vm.selectFork(forkIds[ETH]);
    deal(user, 100 ether);
    uint256[] memory tokenIds = new uint256[](2);
    tokenIds[0] = 1;
    tokenIds[1] = 2;

    vm.startPrank(user);
    testNft.setApprovalForAll(address(bridge[ETH]), true);

    vm.recordLogs();
    bridge[ETH].sendERC721UsingNative{value: 1 ether}(137, address(testNft), tokenIds);
    vm.stopPrank();

    lzHelper.help(lzEndpoint, forkIds[POLY],  vm.getRecordedLogs());

    vm.selectFork(forkIds[POLY]);
    deal(user, 100 ether);

    address wrappedNft = bridge[POLY].getWERC721FromOriginERC721Address(address(testNft)).wrappedAddress;
    vm.prank(admin);
    bridge[POLY].setWrapperRoyalty(address(wrappedNft), treasure, 1, 1);

    vm.startPrank(user);
    ERC721(wrappedNft).setApprovalForAll(address(bridge[POLY]), true);

    vm.recordLogs();
    bridge[POLY].sendERC721UsingNative{value: 1 ether}(137, address(wrappedNft), tokenIds);
    vm.stopPrank();

    lzHelper.help(lzEndpoint, forkIds[ETH],  vm.getRecordedLogs());
}
```

While trying to bridge from Polygon to ETH using both the `originalNft` and `wrappedNft` addresses the following error occurred:

```
Bridge::sendERC721UsingNative{value: 1000000000000000000}(137, WERC721:
↪  [0x7731e5F9c2c4cD867510F83b54A3B93A9Ce16D5b], [1, 2])
    [246] ReentrancyGuard::_nonReentrantBefore()
      ←
    [980] Bridge::getEvmChainSettings(137, 0)
      ← <unknown>
    [981] Bridge::getEvmChainSettings(137, 0)
      ← <unknown>
    [981] Bridge::getEvmChainSettings(137, 0)
      ← <unknown>
    [982] Bridge::getEvmChainSettings(137, 1)
      ← <unknown>
    [603] Bridge::getWERC721FromOriginERC721Address(0xe0dBab467aaea6B2e33EbaD8fdAe07235242D566)
      ← <unknown>
    [0] TestNft::tokenURI(1) [staticcall]
      ← [Stop]
    ← [Revert] EvmError: Revert
  ← [Revert] Contract 0xe0dBab467aaea6B2e33EbaD8fdAe07235242D566 does not exist on active fork with id `1`
    But exists on non active forks: `[0]`
```

Further analysis shows that the function reverts as it tries to access the token URI from the original asset on the bridged chain rather than the wrapped asset.

Recommendation: To fix the above mentioned error, the two internal functions `_getPayload` and `_getPayloadMessage` has to be updated as follows:

```
function getPayload(
    uint256 evmChainId,
    address ERC721Address_,
    uint256[] memory tokensIds_
) internal view returns (IBaseAdapter.MessageSend memory) {
    // ...

    /// @dev if ERC721Address not wrapped, use the ERC721 address
    address currChainAddress_ = ERC721Wrapped.originAddress == address(0)
        ? ERC721Address
        : ERC721Wrapped.wrappedAddress;

    address originERC721Address = ERC721Wrapped.originAddress == address(0)
        ? ERC721Address
        : ERC721Wrapped.originAddress;

    // ...

    return getPayloadMessage(
        tokensIds_,
        currChainAddress_,
        originERC721Address_,
        originEvmChainId_,
        targetEvmChainSettings
    );
}

function getPayloadMessage(
    uint256[] memory tokensIds_,
    address currChainAddress_,
    address originERC721Address_,
    uint256 originEvmChainId_,
    IBridge.EvmChainSettings memory evmChainSettings_
) internal view returns (IBaseAdapter.MessageSend memory) {
    IERC721Metadata metadata = IERC721Metadata(currChainAddress);

    // ...

    uint256 royaltyPercentage = getRoyaltyPercentage(currChainAddress_);

    IBridge.ERC721Token memory tokenData = IBridge.ERC721Token({
        evmChainId: originEvmChainId,
        tokenAddress: originERC721Address_,
        name: metadata.name(),
        symbol: metadata.symbol(),
        royaltyPercentage: royaltyPercentage
    });
```

```
    // ...
}
```

Sweep n' Flip: Fixed in snf-bridge-contracts-v1 PR 10.

Cantina Managed: The security team has been unable to compile the codebase after introducing the fixes, so the fixes remain unverified.

3.1.2 `LayerZeroAdapter::executeMessages` could lead to irrecoverable state

Severity: Critical Risk

Context: LayerZeroAdapter.sol#L122

Description: The `executeMessages` function executes stored payloads in `LayerZeroAdapter.sol` Upon receiving a cross-chain message, the `LayerZeroAdapter.sol` tries to deliver it to the `Bridge.sol` contract. Any failure would result in storing the payload in the intermediary storage variable named `s_pendingMessagesToExecute`.

After storage, the `executeMessages` function tries to redeliver them to the `Bridge.sol` contract later. This function accepts a `limitToExecute_` parameter specifying the number of pending messages to execute. For example, if there are two messages stored in the `s_pendingMessagesToExecute` variable, specifying one will execute one of them.

However, the implementation has a flaw. The function executes a different message but pops a different one from the `s_pendingMessagesToExecute` storage variable, leading to an irrecoverable state. The impact of this issue will lead to permanent locking of NFTs in the bridge contract, which in turn would lead to loss of user funds.

Proof of Concept: The vulnerability mentioned above is demonstrated using the following proof of concept:

```
function test_e2e_execute() external {
    vm.selector(forkIds[ETH]);
    deal(user, 100 ether);

    uint256[] memory tokenIds = new uint256[](2);
    tokenIds[0] = 1;
    tokenIds[1] = 2;

    vm.startPrank(user);
    testNft.setApprovalForAll(address(bridge[ETH]), true);
    vm.recordLogs();
    bridge[ETH].sendERC721UsingNative{ value: 1 ether }(137, address(testNft), tokenIds);
    vm.stopPrank();

    lzHelper.help(lzEndpoint, forkIds[POLY], vm.getRecordedLogs());

    vm.selectFork(forkIds[POLY]);
    deal(user, 400 ether);

    address wrappedNft = bridge[POLY].getWERC721FromOriginERC721Address(address(testNft)).wrappedAddress;

    vm.prank(admin);
    bridge[POLY].setWrapperRoyalty(address(wrappedNft), treasure, 1, 1);

    vm.startPrank(user);
    ERC721(wrappedNft).setApprovalForAll(address(bridge[POLY]), true);
    vm.recordLogs();

    uint256[] memory tokenIdsPoly = new uint256[](1);
    tokenIdsPoly[0] = 1;
    bridge[POLY].sendERC721UsingNative{ value: 200 ether }(1, address(wrappedNft), tokenIdsPoly);

    tokenIdsPoly[0] = 2;
    bridge[POLY].sendERC721UsingNative{ value: 200 ether }(1, address(wrappedNft), tokenIdsPoly);
    vm.stopPrank();

    lzHelper.help(lzEndpoint, forkIds[ETH], vm.getRecordedLogs());

    vm.selectFork(forkIds[ETH]);

    bytes4[] memory selector = new bytes4[](1);
```

```
    selector[0] = IBridge.receiveERC721.selector;

    vm.prank(admin);
    accessManagement[ETH].setTargetFunctionRole(address(bridge[ETH]), selector, 2);

    lzAdapter[ETH].executeMessages(1);
    lzAdapter[ETH].getPendingMessagesToExecuteCount();
    lzAdapter[ETH].executeMessages(1);
}
```

Upon investigation, it was found that the first `executeMessages` call executed the first stored message and transferred NFT id: 1 to the user. Later, on the second call, it reverted as it again tried to transfer NFT ID: 1 to the user instead of 2. This happened because the initial `executeMessages` call popped the unused message from the pending queue.

Recommendation: The current way of storing and executing messages is ambiguous, and a new approach of allocating individual nonces to unique pending messages and processing them later is suggested. However, another simple fix to the issue is to execute the message from the last index, as only the `pop()` operation on the array is natively supported by solidity.

Sweep n' Flip: Fixed on `snf-bridge-contracts-v1` PR 9 and `snf-bridge-contracts-v1` PR 13.

Cantina Managed: The fixes do not fully mitigate the issue. Moreover, the fixes introduced a new function (LayerZeroAdapter.sol#L53) leading to another critical vulnerability.

3.2 High Risk

3.2.1 Incorrect casting in `Bridge::_getRoyaltyPercentage` leads to miscalculated royalty fees

Severity: High Risk

Context: Bridge.sol#L248

Description: The `_getRoyaltyPercentage` function in the `Bridge` contract attempts to get royalty information using the `ERC2981::royaltyInfo` function but incorrectly treats the returned royalty amount as a royalty percentage.

Moreover, this function passes in an arbitrary and small sale price (10000) for royalty calculations, which doesn't provide accurate percentage representation.

For example, if an NFT contract specifies a 1% royalty, `royaltyInfo(0, 1000)` returns 10 (1% of 1000). The returned ten value is returned as the `royaltyPercentage`. However, in the callback case, the function returns 1e15 (0.001 ether) for the same 1%.

This discrepancy will result in unexpected behavior, where either the user will overpay royalties (or) the projects might lose royalties.

Recommendation: Consider refactoring the `_getRoyaltyPercentage` function to return the proper `royaltyPercentage` based on the `salePrice` passed in.

Sweep n' Flip: Fixed in `snf-bridge-contracts-v1` PR 12.

Cantina Managed: Verified fix. Although the function operates correctly, the hardcoded tokenID: 0 may revert in some situations if the NFT IDs start from one instead of zero.

3.2.2 Inaccurate royalty calculation in `WERC721::royaltyInfo` function

Severity: High Risk

Context: WERC721.sol#L70

Description: A unit mismatch exists between how the `Bridge` and `WERC721` contracts handle royalty percentages. The Bridge contract uses 0.001 ether to represent 1%, while `WERC721`'s `royaltyInfo` calculates with the assumption that 0.01 ether represents 1%. This results in royalties being multiple times lower than intended.

The `Bridge` sets the default percentage:

```
function _getRoyaltyPercentage(address nftContract) internal view returns (uint256) {
    try IERC2981(nftContract).royaltyInfo(0, 10000) returns (address, uint256 royaltyAmount) {
        return royaltyAmount > 0 ? royaltyAmount : 0.001 ether; // 1% if undefined
    } catch {
        return 0.001 ether; // 1% default
    }
}
```

While `WERC721` calculates royalties:

```
// In WERC721.sol
function royaltyInfo(uint256 tokenId_, uint256 salePrice_) public view override returns (address receiver,
↪  uint256 royaltyAmount) {
    uint256 percentage = s_royaltyInfo.originalPercentage + s_royaltyInfo.wrapperPercentage;
    console.log("percentage", percentage);
    return (s_royaltyInfo.receiver, salePrice_ * percentage / 1 ether);
}
```

Impact example:

```
// In Bridge:
defaultRoyalty = 0.001 ether  // Intended to be 1%

// In WERC721:
salePrice = 1 ether
royaltyAmount = 1 ether * 0.001 ether / 1 ether = 0.001 ether // Actually 0.1%

// Expected:
royaltyAmount = 1 ether * 1 / 100 = 0.01 ether // Should be 1%
```

Recommendation: Consider aligning percentage representations between contracts to ensure the royalty calculations are accurate.

Sweep n' Flip: Fixed in `snf-bridge-contracts-v1` PR 11.

Cantina Managed: Verified fix. The constant for 1% is now updated to be 100 and the overall calculation works as expected in the `royaltyInfo` function.

3.3 Medium Risk

3.3.1 RefundAddress forwarded to LayerZero endpoint is incorrect

Severity: Medium Risk

Context: LayerZeroAdapter.sol#L172, LayerZeroAdapter.sol#L175

Description: The `sendMessageUsingNative` function from the `LayerZeroAdapter.sol` contract is used by the `Bridge.sol` contract to send a message using LayerZero. This function, in turn, calls the `_sendMessage` function, which again calls `_lzSend` to call the LayerZero endpoint with the right parameters.

The `sendMessageUsingNative` function accepts `msg.value` equal to / greater than the fee quoted by LayerZero to cover LayerZero's messaging costs. It forwards them to the endpoint contract, from which any unused value is refunded to the `refundAddress` specified.

However, these refunds don't work as expected as the refund address is always specified as the `Bridge.sol` contract, which is the `msg.sender` in the context of the `LayerZeroAdapter.sol` contract. Thus, any refunds received from LayerZero's endpoint are locked indefinitely in the `Bridge.sol` contract.

Recommendation: The above-mentioned vulnerability could be fixed in multiple ways, including:

- Forwarding a user-specified refund address to LayerZero's endpoint to process refunds properly.
- Collecting all refunds from endpoint to Bridge and refunding all the `msg.value` stored in `Bridge.sol` at the end of a bridging transaction back to the `msg.sender`.
- Make sure `msg.value` is strictly equal to the quoted fee, making sure no refunds are possible.

Sweep n' Flip: Fixed in snf-bridge-contracts-v1 PR 16.

Cantina Managed: Verified fix. A new variable called `refundAddress` is forwarded from `Bridge.sol` to the LayerZero adapter, encoding `msg.sender` as the refund address.

3.3.2 `debug_forceReceiveMessage` is dangerous and could be abused

Severity: Medium Risk

Context: LayerZeroAdapter.sol#L77

Description: The `debug_forceReceiveMessage` function in `LayerZeroAdapter` is a restricted function that allows privileged actors to deliver arbitrary messages to the Bridge contract. For example, a privileged actor could deliver a random payload to the Bridge contract without sending it through LayerZero.

This ability could lead to stealing NFTs from the Bridge contract and minting them without constraints. This also undermines the security offered by LayerZero as a cross-chain communication protocol.

Recommendation: Consider removing the `debug_forceReceiveMessage` function.

Sweep n' Flip: Fixed in snf-bridge-contracts-v1 PR 15.

Cantina Managed: Verified fix.

3.3.3 Improper execution options generated in the `_sendMessage` function

Severity: Medium Risk

Context: LayerZeroAdapter.sol#L174

Description: The `_sendMessage` function in `LayerZeroAdapter.sol` is a helper function utilized by both the `sendMessageUsingERC20` and `sendMessageUsingNative` functions to dispatch a message using LayerZero.

This function has two branches based on the type of fee: payment in native tokens and payment in ERC20 tokens. The function operates correctly for the native token branch, but there is an issue generating execution options for the ERC20 token payment branch.

The execution options are generated using the `buildOptions` function from the `LayerZeroUtils` library contract. It accepts two parameters _gas and _tokenValue representing the gas limit of the receiving transaction and the native token to be airdropped alongside the `lzReceive` call on the destination chain.

```
function buildOptions(uint128 _gas, uint128 _tokenValue) public pure returns (bytes memory) {
  bytes memory options = OptionsBuilder.newOptions().addExecutorLzReceiveOption(_gas, _tokenValue);
  return options;
}
```

In the ERC20 token payment branch of `_sendMessage`, the function incorrectly passes in the gas limit to the `_tokenValue` parameter instead of the `_gas` parameter, leading to unexpected behavior and possible inability to use the `sendMessageUsingERC20` function.

```
 function _sendMessage(
   IBaseAdapter.MessageSend memory payload_,
   uint256 quotedFee_,
   address currencyFee
) internal override {
    if (currencyFee == address(0)) {
      // ...
    } else {
       bytes memory options = LayerZeroUtils.buildOptions(0, uint128(payload_.gasLimit));
        // ...
    }
    // ...
}
```

Recommendation: Passing the correct values for the `_gas` and `_tokenValue` parameters can fix the message execution option generation issue mentioned above.

```
- bytes memory options = LayerZeroUtils.buildOptions(0, uint128(payload_.gasLimit));
+ bytes memory options = LayerZeroUtils.buildOptions(uint128(payload_.gasLimit, 0));
```

Sweep n' Flip: Fixed in `snf-bridge-contracts-v1` PR 14.

Cantina Managed: Verified fix. The option to use `lzToken` for paying fees has been completely removed, so the reported issue is no longer present.

3.3.4 Privileged role and actions lead to centralization risks for users

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: Several restricted/access-controlled functions across bridging logic affect critical protocol state and semantics, leading to centralization risk for users. Some examples are highlighted below:

- `setTargetFunctionRole()` can potentially allow an EOA or a malicious contract acting as `LayerZeroAdapter` to steal user NFTs.
- Abuse of `sendMessageUsingERC20` / `sendMessageUsingNative` can occur by adding an EOA to the Bridge role, allowing messages to be sent without transferring NFTs to the Bridge.sol contract.
- Updating the `s_bridge` address through the `updateBridge` function can result in the loss of NFTs if a message is still in transit during the update.
- `setEvmChainSetting` can be used to disable an existing route while a message is in transit, which can lead to a temporary or permanent inability to deliver a message in the queue.

Recommendation: Consider:

- Making certain configurations immutable and considering deploying newer versions if a change has to be effected.
- Enforce deterministic access-control measures and remove the `AccessManager` library from OpenZeppelin, which introduces opaque access controls.
- Documenting the privileged role and actions for protocol user awareness.
- Privilege actions affecting critical protocol semantics should be locked behind timelocks so users can decide to exit or engage.
- Following the strictest opsec guidelines for privileged keys, e.g., use of reasonable multi-sig and hardware wallets.

Sweep n' Flip: Acknowledged.

Cantina Managed: Acknowledged.

3.4 Low Risk

3.4.1 `WERC721` contract is not fully ERC2981 compliant

Severity: Low Risk

Context: WERC721.sol#L7

Description: The `WERC721` contract implements the `ERC721` from the OpenZeppelin library. This contract primarily represents bridged NFTs on a chain that is different from the NFT's original chain.

This function implements a few additional functions from the standard ERC721 implementation to support royalties. One such function is the `royaltyInfo` function, which is compliant with the royalty standard, ERC2981. This function establishes a universal method to signal marketplaces and other platforms about the royalties of trading these NFTs.

Per the ERC-2981, the contracts should return the `royaltyAmount` on a call to the `royaltyInfo` function and also should register the interface id: `0x2a55205a`.

However, `WERC721` implements the `royaltyInfo` function without registering the interface, resulting in a broken royalty implementation.

```
WERC721::supportsInterface(0x2a55205a) [staticcall]
        ← [Return] false
```

Recommendation: Consider registering the interfaceId: `0x2a55205a` in the `WERC721` contract to make it fully ERC2981 compliant (or) use the ERC2981.sol contract from OpenZeppelin library.

Sweep n' Flip: Fixed on commit ba5c6495.

Cantina Managed: Fix verified.

3.4.2 Resolve all TODOs for production readiness

Severity: Low Risk

Context: BaseAdapter.sol#L47, BaseAdapter.sol#L56, LayerZeroAdapter.sol#L35, LayerZeroAdapter.sol#L178, Bridge.sol#L21

Description: Multiple TODOs across the entire repository under review must be addressed, and all TODO comments must be removed/fixed. This will improve code quality, reduce technical debt, and implement all pending tasks correctly.

Recommendation: Please ensure all pending tasks are appropriately tracked and implemented.

Sweep n' Flip: Fixed on commit ba5c6495. The only TODO implemented was the possibility of update the bridge address. The rest of the things will be implemented if necessary

Cantina Managed: Fix verified.

3.4.3 Use `call` instead of `transfer` for native token transfers

Severity: Low Risk

Context: Bridge.sol#L175

Description: The current implementation uses the transfer function to send a native token from the `Bridge.sol` contract to an external address. The transfer function is limited to 2300 gas, which may not be enough for all contract interactions (if the treasureAddress is a smart wallet / multi-sig).

This can lead to potential issues, such as the transfer failing if the receiving contract has a fallback function that requires more than 2300 gas to execute.

Also, additional use of `safeTransferETH` from Solady is suggested.

Recommendation: Instead of using the `transfer` function, consider using the `call` method to send native tokens from the contract to an external address. The call function does not have the 2300 gas limit and allows for more complex contract interactions.

```
(bool success, ) = treasureAddress.call{value: systemFee}("");
if(!success) revert();
```

Sweep n' Flip: Fixed in `snf-bridge-contracts-v1` PR 17.

Cantina Managed: Verified fix.

3.4.4 Emit events if a function alters storage variables

Severity: Low Risk

Context: LayerZeroAdapter.sol#L106, UniswapV2Factory.sol#L86, UniswapV2Factory.sol#L91, UniswapV2Factory.sol#L96, UniswapV2Factory.sol#L101

Description: Multiple functions across different files within the audit scope modify state variables without emitting events. Events are essential for transparency and enabling off-chain monitoring of on-chain state changes.

Recommendation: Implement event emissions whenever critical state variables are updated. This will:

- Provide real-time notifications of state changes.
- Enable external systems for monitoring and front-end applications to track contract state.
- Improve overall transparency of the smart contracts.

Sweep n' Flip: Acknowledged.

Cantina Managed: Acknowledged.

3.4.5 Unsafe casting of `uint256` to `uint32` in `LayerZeroAdapter.sol`

Severity: Low Risk

Context: LayerZeroAdapter.sol#L51, LayerZeroAdapter.sol#L57, LayerZeroAdapter.sol#L172, LayerZeroAdapter.sol#L175

Description: LayerZero represents chains using `uint32`, while the sweep-n-flip protocol uses chain IDs as `uint256`.

Multiple instances exist in the `LayerZeroAdapter.sol` contract where `uint256` values are directly casted to `uint32`. This casting type is risky and could lead to unexpected behavior within the protocol.

Proof of Concept: For example, an attempt to call the `getFeeNative` function using a value greater than the maximum for `uint32` resulted in the chain ID being cast to an arbitrary number.

Generally, this situation will cause a revert, as the chances of adding a peer leading to a collision are minimal. However, this scenario should be avoided.

```
BaseTest::test_getFeeNative()
    [8368] LayerZeroAdapter::getFeeNative(MessageSend({ toChain: 18446744073709551615 [1.844e19], receiver:
    ↪  0x0000000000000000000000000000000DeaDBeef, data: 0x, gasLimit: 400000 [4e5] })) [staticcall]
       [0] console::log("chainId:", 4294967295 [4.294e9]) [staticcall]
         ← [Stop]
       ← [Revert] NoPeer(4294967295 [4.294e9])
     ← [Revert] NoPeer(4294967295 [4.294e9])
```

Recommendation:

- Consider bounding the incoming `uint256` chain ID to a value of `type(uint32).max` as follows:

```
function getFeeNative(IBaseAdapter.MessageSend memory payload_) public view returns (uint256) {
+     if(payload_.toChain > type(uint32).max) revert();
    // ...
}
```

- Or consider using the OpenZeppelin SafeCast library.

Sweep n' Flip: Acknowledged.

Cantina Managed: Acknowledged.

3.4.6 NFT bridging from smart contract wallets will result in permanent loss

Severity: Low Risk

Context: Bridge.sol#L159

Description: By default, the `sendERC721UsingNative` function uses `msg.sender` as the final recipient of the bridged NFTs.

This approach works well for Externally Owned Accounts (EOAs). However, for some account abstraction wallets and smart contract wallets, such as Gnosis, the likelihood of deploying the smart contract to the same address across different chains is minimal.

Hence, if the NFTs are bridged from such wallets, there might be a permanent loss since the receiver address will not be controlled by the `msg.sender`

Recommendation: Consider adding a new parameter called `receiverAddress` in the `sendERC721UsingNative` function to ensure compatibility with both smart contract wallets and account abstraction (AA) wallets.

Sweep n' Flip: Fixed on commit ba5c6495.

Cantina Managed: Fix verified. The `sendERC721UsingNative` function implements a `receiver_` address parameter that allows to specify the receiver without relying on the `msg.sender`.

3.4.7 Incorrect tracking of executed messages by `s_messagesExecutedCount` variable

Severity: Low Risk

Context: LayerZeroAdapter.sol#L138

Description: The variable `s_messagesExecutedCount` in `LayerZeroAdapter.sol` tracks the number of pending messages executed (i.e. delivered to the bridge) by the adapter through the `executeMessage` function.

However, this function double counts if the pending messages fail to be delivered during the `executeMessage` call and are stored again in the pending queue.

Recommendation: Consider updating the `_receiveMessage` function to return an execution status and only increment the `s_messagesExecutedCount` if the bridge delivery call is successful.

```
  function executeMessages(uint256 limitToExecute_) public nonReentrant {
      // ...
      while (limit > 0) {
          IBaseAdapter.MessageReceive memory payload = s_pendingMessagesToExecute[lastIndex];
-         _receiveMessage(payload);
-         s_messagesExecutedCount++;
+        bool status = _receiveMessage(payload);
+        if(status) s_messagesExecutedCount++;
      // ...
  }

  function _receiveMessage(IBaseAdapter.MessageReceive memory payload_)
-     internal virtual override {
+     internal virtual override returns(bool success) {
      /// @dev try before set as peding
      try IBridge(getBridge()).receiveERC721(payload_) {
          /** @dev ignore */
+         success = true;
      } catch {
          _setPendingMessage(payload_);
      }

      emit IBaseAdapter.MessageReceived(payload_.fromChain, payload_.sender, payload_.data);
  }
```

Sweep n' Flip: Fixed on commit ba5c6495.

Cantina Managed: Fix verified. The new implemented code catches if the delivery fails and does not count it as a success and ads it to the pending queue.

3.4.8 Use `safeTransferFrom` instead of `transferFrom` to transfer NFTs

Severity: Low Risk

Context: Bridge.sol#L289, WERC721.sol#L54, WERC721.sol#L66

Description:

- There is a general discrepancy in transferring NFTs to and from the `Bridge.sol` contract. The function `sendERC721UsingNative` uses the `safeTransferFrom` method to transfer NFTs from the user's wallet to the `Bridge.sol` contract; the `receiveERC721` function does not.
- The `_mint` and `_burn` functions in `WERC721.sol` contract also use the `transferFrom` method instead of the `safeTransferFrom` method.

The use of the `transferFrom` function is discouraged by OpenZeppelin. If the arbitrary receiver address is a contract and is not aware of the incoming ERC721 token, the sent token could be lost forever.

Recommendation: Consider using the `safeTransferFrom` method for transferring NFTs. Additionally, if the `safeTransferFrom` method is used, ensure that these functions are non-reentrant. For example:

```
  function receiveERC721(
      IBaseAdapter.MessageReceive memory payload_
  )
      external
      restricted
      nonReentrant
      checkEvmChainSettingsAdapter(IBridge.AdapterType.Source,
      ↪  getEvmChainSettings(s_nonEvmChainToEvmChain[payload_.fromChain], IBridge.AdapterType.Source))
      checkEvmChainSettingsIsEnabled(getEvmChainSettings(s_nonEvmChainToEvmChain[payload_.fromChain],
      ↪  IBridge.AdapterType.Source))
  {
      // ...
-     IERC721(_WERC721Address).transferFrom(address(this), _receiverAddress, _tokenId);
+     IERC721(_WERC721Address).safeTransferFrom(address(this), _receiverAddress, _tokenId);
      // ...
  }
```

To ensure `safeTransfer`s, the receiving contracts should also import the ERC721Holder from OpenZeppelin.

Sweep n' Flip: Acknowledged.

Cantina Managed: Acknowledged.

3.4.9 Use `_safeMint` instead of `_mint` in WERC721 contract

Severity: Low Risk

Context: WERC721.sol#L29

Description: The `bridgeMint` function is used by the `Bridge.sol` contract to mint Wrapped ERC721 tokens on the destination chain to the recipient to complete the bridging process. However, this function uses the `_mint` method from OpenZeppelin instead of the `_safeMint` method, which is discouraged by the library itself.

The `_mint` method lacks the `onERC721Received` checks, thus leading to the loss of NFTs while bridging to a contract that does not support ERC721s.

Recommendation: Consider replacing the `_mint` method with `_safeMint` while minting new NFTs from the `WERC721` contract. Additionally, add a re-entrancy guard as a malicious `onERC721Received` callback in `_safeMint` could be exploited.

Sweep n' Flip: Acknowledged.

Cantina Managed: Acknowledged.

3.4.10 `Bridge.sol` is incompatible with multi-chain ERC-721 contracts

Severity: Low Risk

Context: Bridge.sol#L293

Description: Bridging multi-chain NFTs, such as LayerZero ONFTs or other related NFTs, will not be compatible with the current implementation of the Sweep n' Flip bridge.

The current implementation of `Bridge.sol` maps a contract address to a wrapped address based on the source chain address as follows:

```
/// @dev from origin ERC721 address to wERC721 address
mapping(address => IBridge.ERC721Wrapped) private s_WERC721FromOriginERC721Address;
```

However, it is possible to deploy an NFT contract to the same address across multiple chains. If this is done, the bridge will not recognize it and will always fail during the `bridgeMint` process, as identical IDs can be minted from another chain.

In summary, while deploying an NFT contract to the same address on different chains is technically feasible, this creates a compatibility issue with the bridge.

Proof of Concept: The following code is a proof of concept where the same NFT contract is deployed to the same address on Ethereum and Optimism. Post-deployment, they're bridged to Polygon, where the

first bridging action from Optimism succeeds, and the latter from Ethereum fails as they share a common wrapper address.

```
function test_e2e() external {
    vm.selectFork(forkIds[OP]);
    deal(user, 100 ether);
    uint256[] memory tokenIds = new uint256[](2);
    tokenIds[0] = 1;
    tokenIds[1] = 2;

    vm.startPrank(user);
    testNft[OP].setApprovalForAll(address(bridge[OP]), true);

    vm.recordLogs();
    bridge[OP].sendERC721UsingNative{value: 1 ether}(137, address(testNft[OP]), tokenIds);
    vm.stopPrank();

    lzHelper[OP].help(lzEndpoint, forkIds[POLY],  vm.getRecordedLogs());

    /// ETHEREUM BRIDGING

    vm.selectFork(forkIds[ETH]);
    deal(user, 100 ether);

    vm.startPrank(user);
    testNft[ETH].setApprovalForAll(address(bridge[ETH]), true);

    vm.recordLogs();
    bridge[ETH].sendERC721UsingNative{value: 1 ether}(137, address(testNft[ETH]), tokenIds);
    vm.stopPrank();

    lzHelper[ETH].help(lzEndpoint, forkIds[POLY],  vm.getRecordedLogs());
}
```

Recommendation: Consider updating the mapping to be chain aware and deploy individual wrappers for multi-chain ERC721 contracts:

```
mapping(uint256 => mapping(address => IBridge.ERC721Wrapped)) private s_WERC721FromOriginERC721Address;
```

Sweep n' Flip: Acknowledged.

Cantina Managed: Acknowledged.

3.4.11 Bridge contract does not compile due to syntax error

Severity: Low Risk

Context: Bridge.sol#L250, Bridge.sol#L253

Description: In Solidity, using the `ether` keyword directly after a number without a space is invalid syntax. The compiler interprets it as an invalid token because `ether` is a keyword for specifying Ether denominations, not part of a numeric literal.

Bridge.sol#L250 and Bridge.sol#L253 cause a compilation error.

Recommendation: To fix the issue, add a space between the number and the `ether` keyword to ensure proper syntax. Solidity requires the number and the `ether` keyword to be clearly separated.

This syntax is consistent with how Solidity parses numeric literals with denomination keywords like `wei`, `gwei`, or `ether`. Always ensure spacing when using Ether denominations to avoid similar issues.

Sweep n' Flip: Fixed on commit ba5c6495.

Cantina Managed: Fix verified. However on the incoming fixed commits there were other syntax errors that made the contracts to not compile again. Sweep n' Flip team is still actively developing the bridge.

3.4.12 Missing events and sanity checks in `Bridge.sol` state changing functions

Severity: Low Risk

Context: Bridge.sol#L72-L74, Bridge.sol#L80-L82

Description: The `updateTreasureAddress` and `updateSystemFeePercent` functions in the `Bridge.sol` contract change state variables without emitting an event to log the change and performing any sanity checks on the input.

Recommendation:

1. Consider emitting events:
    - Add events for both functions to log state changes, making changes auditable and easier to trace.
2. Consider adding sanity check:
    - Validate inputs to ensure they are within acceptable range or values.

Sweep n' Flip: Acknowledged.

Cantina Managed: Acknowledged.

3.5 Gas Optimization

3.5.1 Unnecessary system fee transfer if fee is zero

Severity: Gas Optimization

Context: Bridge.sol#L175

Description: In the `sendERC721UsingNative` function, the contract transfers the system fee to the treasure address without validation. If the fee is zero, this can lead to unnecessary gas consumption and potentially an unwanted extra transaction.

```
/// @dev send system fee to treasure
payable(treasureAddress).transfer(systemFee);
```

Recommendation: Consider modifying the code to transfer the system fee only if it exceeds zero.

```
/// @dev send system fee to treasure only if fee is greater than zero
if (systemFee > 0) {
    payable(treasureAddress).transfer(systemFee);
}
```

Sweep n' Flip: Fixed on commit ba5c6495.

Cantina Managed: Fix verified.

3.5.2 Combine `checkEvmChainSettingsAdapter` and `checkEvmChainSettingsIsEnabled` to improve gas efficiency

Severity: Gas Optimization

Context: Bridge.sol#L51, Bridge.sol#L61

Description: The `Bridge.sol` contract currently uses two separate modifiers, **checkEvmChainSettingsAdapter** and **checkEvmChainSettingsIsEnabled**, to validate EVM chain settings.

These modifiers validate the same input parameters, and hence, combining them will save:

- 1 `SLOAD` operation.
- 1 function jump.
- And associated stack operations.

Combining the two modifiers results in a gas reduction of ~1000 GAS, which is significant on expensive networks like Ethereum.

Recommendation: Consider combining the two modifiers into a single, more efficient modifier:

```
modifier validateEvmChainSettings(AdapterType adapterType_, IBridge.EvmChainSettings memory evmChainSettings_) {
    if (adapterType_ == IBridge.AdapterType.Destination && evmChainSettings_.adapter == address(0)) {
        revert IBridge.AdapterNotFound();
    }
    if (!evmChainSettings_.isEnabled) {
        revert IBridge.AdapterNotEnabled();
    }
    _;
}
```

Sweep n' Flip: Acknowledged.

Cantina Managed: Acknowledged.

3.5.3 `memory` can be changed to `calldata` to save gas

Severity: Gas Optimization

Context: Bridge.sol#L130, Bridge.sol#L162, Bridge.sol#L210, Bridge.sol#L225, Bridge.sol#L259

Description: In Solidity, variables declared as `memory` are copied into memory during execution, which increases gas costs. When dealing with external function inputs, the `calldata` keyword is more gas-efficient since it avoids copying and operates directly on the call data.

The following instances in the `Bridge` contract use `memory` unnecessarily, where `calldata` can be used instead:

1. Bridge.sol#L130: `tokenIds_`
2. Bridge.sol#L162: `tokenIds_`
3. Bridge.sol#L210: `tokenIds_`
4. Bridge.sol#L225: `tokenIds_`
5. Bridge.sol#L259: `payload_`

Recommendation: Consider changing the variables from `memory` to `calldata` in the function signatures for these lines. This reduces gas costs by eliminating the need to copy data into memory.

This reduces gas costs by directly referencing the input data without copying it improving overall efficiency, especially for large arrays or payloads.

Sweep n' Flip: Acknowledged.

Cantina Managed: Acknowledged.

3.5.4 Unnecessary use of `tryDiv` for division with a constant

Severity: Gas Optimization

Context: Bridge.sol#L148

Description: The `_calculateBridgeFee` function from the`Bridge.sol` uses the `tryDiv` function to perform division, even though `b` (the divisor) is always `100`. The function checks for division by zero, but since the divisor is constant and non-zero, this check is redundant. Additionally:

1. The `tryDiv` function unnecessarily returns a `bool`, which is not required here since the divisor cannot be zero.
2. This adds complexity and gas overhead.
3. The subsequent `require` statement is also redundant, as division by a non-zero constant is guaranteed to succeed.

Recommendation: Consider simplifying the division to:

```
additionalFee = additionalFee / 100;
```

This avoids the unnecessary `tryDiv` function call and removes the need for handling a `bool`. It is also possible to consider wrapping the division in an `unchecked` block to save additional gas, as overflow and underflow are not possible with Solidity 0.8's default safety checks and the constant divisor.

The updated code optimizes gas usage by removing unnecessary function calls and checks, simplifies the logic by directly performing the division, and ensures safer, cleaner code without redundant require statements.

Sweep n' Flip: Acknowledged.

Cantina Managed: Acknowledged.

3.6 Informational

3.6.1 Code quality issues: unused imports (or) code

Severity: Informational

Context: LayerZeroAdapter.sol#L4, LayerZeroAdapter.sol#L7, LayerZeroAdapter.sol#L18, Bridge.sol#L9, UniswapV2Router01.sol#L6

Description: Many files throughout the repository contain unused import statements or code not utilized anywhere in the contract.

Depending on the compiler settings, importing unnecessary libraries or retaining unused code can raise the gas costs for contract deployment and execution. Additionally, this practice can make the codebase less readable and harder to maintain, ultimately affecting code quality.

Recommendation: Consider removing the unused file imports (or) code to improve code quality

Sweep n' Flip: Acknowledged.

Cantina Managed: Acknowledged.

3.6.2 Implement getters to read royalty information from `WERC721` contract

Severity: Informational

Context: WERC721.sol#L10

Description: The `WERC721.sol` contract intends to store the `royaltyInfo` struct as one of its state variables. The `royaltyInfo` contains information about the royalty receiver, the original royalty percentage from the NFT contract, and the wrapper royalty percentage to be charged by the Sweep n' Flip protocol.

However, since the variable is declared `private`, external integrators/users cannot read this information from the contract.

Recommendation: Consider changing the variable visibility from `private` to `public`.

Sweep n' Flip: Fixed in `snf-bridge-contracts-v1` PR 11.

Cantina Managed: Verified fix. A new getter function `getRoyaltyInfo` is introduced.

3.6.3 Code quality issues: unused `ExcessivelySafeCall` library

Severity: Informational

Context: ExcessivelySafeCall.sol#L4

Description: The project repository contains an unused `ExcessivelySafeCall.sol` library file, which is not used/needed.

Recommendation: Consider removing the unused `ExcessivelySafeCall.sol` library.

Sweep n' Flip: Acknowledged.

Cantina Managed: Acknowledged.

3.6.4 Unnecessary local contract duplication of LayerZero OApp contracts

Severity: Informational

Context: OAppCore.sol#L12, OAppReceiver.sol#L13, OAppSender.sol#L13, OApp.sol#L17

Description: Multiple files created inside the `contracts/layer-zero` folder in the `snf-bridge-contracts-v1` are direct copies of LayerZero's official library. This can lead to maintenance overhead, increased project complexity, and potential security risks.

Recommendation: Consider importing the LayerZero packages from the library. For further information, please refer to LayerZero's official documentation.

Sweep n' Flip: Acknowledged.

Cantina Managed: Acknowledged.

3.6.5 `s_bridgeAddress` could be made immutable

Severity: Informational

Context: WERC721.sol#L8

Description: The `WERC721.sol` contract declares the `s_bridgeAddress` variable private but mutable. However, this variable is set only in the constructor and is not changed/updated after that.

As this variable is not updated, making it `immutable` is more gas-efficient.

Recommendation: Consider declaring `s_bridgeAddress` immutable (or) if mutable, consider implementing a function to update it post deployment.

Sweep n' Flip: Acknowledged.

Cantina Managed: Acknowledged.

3.6.6 Code quality issues: replace magic numbers with constants

Severity: Informational

Context: Bridge.sol#L250, Bridge.sol#L253, RoyaltyHelper.sol#L40, RoyaltyHelper.sol#L45, RoyaltyHelper.sol#L51

Description: Magic numbers are often used in key contracts under review. They can lead to unexpected issues, reduce code quality, and make it difficult to understand and maintain the code over time.

Recommendation: To improve readability and maintainability, a constant can be used instead of the magic number.

```
+ uint256 public constant DEFAULT_ROYALTY_PERCENT = 0.001 ether;

  function _getRoyaltyPercentage(address nftContract) internal view returns (uint256) {
      try IERC2981(nftContract).royaltyInfo(0, 10000) returns (address, uint256 royaltyAmount) {
        // Calculate percentage (royaltyAmount is in basis points, so divide by 100)
-       return royaltyAmount > 0 ? royaltyAmount : 0.001 ether; // 1% if undefined
+       return royaltyAmount > 0 ? royaltyAmount : DEFAULT_ROYALTY_PERCENT;
      } catch {
        // If the contract does not support IERC2981 or fails, default to 1%
-       return 0.001 ether;
+       return DEFAULT_ROYALTY_PERCENT;
      }
  }
```

Sweep n' Flip: Partially fixed on commit ba5c6495.

Cantina Managed: Partial fix verified.

3.6.7 Code quality issues: remove unused function parameter

Severity: Informational

Context: WERC721.sol#L68

Description: The `royaltyInfo` function in `WERC721.sol` has an unused parameter `tokenId` which could be removed (or) commented out. Unused function parameters could be confusing and decrease code quality.

Recommendation: Consider removing the unused parameters to improve code quality.

Sweep n' Flip: Acknowledged.

Cantina Managed: Acknowledged.

3.6.8 `name()`, `symbol()` and `tokenURI()` are optional functions, per the ERC-721 standard

Severity: Informational

Context: Bridge.sol#L234, WERC721.sol#L22

Description: Per EIP-721, the methods, `name()`, `symbol()` and `tokenURI()` are optional extension functions. As such, some valid ERC-721 tokens do not support this interface and are incompatible with the Sweep n' Flip protocol.

Recommendation: Consider using the try/catch mechanism to query these values and handle them accordingly.

Sweep n' Flip: For the `WERC721.sol` file, name()/symbol() calls will only fail if they also fail for the wrapped collection. That is the intended behavior.

Cantina Managed: Acknowledged. It should be added in the documentation that only contracts to implement the three methods: `name()`, `symbol()`, and `tokenURI()` are compatible with the protocol.

3.6.9 Increase test coverage

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: Key contracts under review have less than complete test coverage. Adequate test coverage and regular reporting ensure the codebase works as intended. Insufficient code coverage may lead to unexpected issues and regressions due to underlying smart contract implementation changes.

Recommendation: Add to test coverage, ensuring all execution paths are covered. See also Paul R Berg's BTT thread and talk.

Sweep n' Flip: Acknowledged.

Cantina Managed: Acknowledged.

3.6.10 Improve inline documentation

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: Key contracts under review have limited comments (inline documentation) within the code, and critical functions lack adequate explanations. This makes understanding the protocol's operations difficult when reviewing the code.

Recommendation: Ensure every function within the protocol has accompanying comments. This will provide clear documentation throughout the codebase.

Sweep n' Flip: Acknowledged.

Cantina Managed: Acknowledged.

3.6.11 ERC20 swaps are allowed on `SwipNFlip` pools without NFT retrieval

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The `SwipNFlip` system wraps NFTs into a wrapper contract that issues `1e18` ERC20 tokens per token ID. These tokens are then used in liquidity pools to facilitate swaps with other currencies. The router extends functionalities with methods like `swapTokensForExactTokensCollection` and `swapExactTokensForTokensCollection`, which manage NFT retrieval by burning ERC20 tokens.

However, standard UniswapV2 functions (e.g., `swapTokensForExactTokens` and `swapExactTokensForTokens`) remain functional, allowing direct ERC20 token swaps without invoking the NFT retrieval process. Also pools allow for direct swaps like in UniswapV2. This leads to:

1. ERC20 token fragmentation: Direct swaps bypass the burning mechanism, distributing ERC20 tokens that cannot be redeemed for NFTs since only extended router functions have burn permissions.
2. Potential NFT locking: Fragmented ERC20 tokens left in circulation increase the likelihood of NFTs remaining locked in the wrapper contract, potentially reducing access to original NFTs.

While this behaviour might be considered a feature, it introduces risks of fragmentation and decreased recoverability of NFTs.

Recommendation: To mitigate these risks while preserving functionality, consider the following measures:

1. Restrict standard swap functions: Disable standard UniswapV2 swap functions in the router to ensure all interactions with ERC20 tokens and NFTs pass through the extended functions.
2. Add a redemption mechanism: Implement a fallback mechanism to allow users to consolidate fragmented ERC20 tokens and redeem NFTs if sufficient tokens are available. This can include a forced fee payment to avoid users bypassing the original fee.
3. Restrict pool access: Add a modifier to the pool contracts to ensure only the router can initiate swaps or interactions.
4. Enhance documentation: Clearly document the implications of ERC20 fragmentation and encourage users to rely on the extended router functions for NFT retrieval. Include warnings about potential risks of using standard swap functions.

These recommendations will help balance the system's flexibility with safeguards, reducing the risks of fragmentation and inaccessible NFTs while maintaining user trust in the `SwipNFlip` ecosystem.

Sweep n' Flip: The described behavior is intended. The Sweep'n'Flip team wanted this fractional functionality available even when it provides poor experience to users. In earlier versions we would prevent fractions from being traded, and that morphed towards the current implementation.

Cantina Managed: Acknowledged.

3.6.12 Code quality issues: function ordering

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: Key contracts under review do not follow the standard Solidity style guide recommendations for function ordering. Per the solidity style guide, grouping and ordering functions in a specific manner to improve code readability and maintainability.

Functions should be grouped according to their visibility and ordered:

- `constructor`.
- `receive` function (if it exists).
- `fallback` function (if it exists).
- `external`.
- `public`.
- `internal`.
- `private`.

Within each visibility group, functions should be ordered:

- `view` and `pure` functions last.

- Alphabetically where possible (optional but recommended).

Recommendation: Consider reviewing all contract files and reorganizing functions according to the style guide. Additionally, automated tools should be used during development to enforce function ordering.

Sweep n' Flip: Acknowledged.

Cantina Managed: Acknowledged.