Sweep n’ Flip NFT AMM

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 **34** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 1
- Medium Risk: 5
- Low Risk: 6
- Gas Optimizations: 6
- Informational: 16

3 Findings

3.1 High Risk

3.1.1 Premature `createPair` function call will result in the creation of unusable delegation pairs

Severity: High Risk

Context: UniswapV2Factory.sol#L40

Description: Users use the `createPair` function of `UniswapV2Factory.sol` to create new trading pairs. This function allows the users to create a pair in sweep-n-flip (or) delegate such creation to external DEXes if the two input tokens are not ERC721 wrappers created using the `createWrapper` function.

Hence, the ideal flow is, for any ERC721 collection, the users should call the `createWrapper` function in the `UniswapV2Factory.sol` contract to deploy a wrapper, later should call the `createPair` function for creating trading pairs.

The vulnerability here is that the wrapper address deployed by the `UniswapV2Factory.sol` for any ERC721 collection is predictable and can be precomputed. Thus, an attacker can leverage this to create stale pairs and block them from being traded on the platform.

Attack Workflow:

- Identify target NFT collection.
- Precompute wrapper address.
- Front-run wrapper creation by calling `createPair`.
- Create a pair on the delegated DEX and mark them as delegated in the factory contract.
- Prevent legitimate use of the pair.

Impact:

- Denial of service for legitimate token pairs. As there are only a handful of ERC721 tokens, even creating ten transactions for the top 10 NFTs can be disastrous to the protocol.

Proof of Concept: The proof of concept below explains how the vulnerability could lead to creating a pair on a delegated DEX (e.g., Sushiswap) instead of the protocol for a valid NFT contract and incorrectly updates the `delegates` mapping.

```
function computeWrapperAddress(address collection) internal view returns (address) {
   bytes32 salt = keccak256(abi.encodePacked(collection));
   bytes32 bytecodeHash = keccak256(type(WERC721).creationCode);
   return address(uint160(uint(keccak256(abi.encodePacked(
       bytes1(0xff),
       address(factory),
       salt,
       bytecodeHash
   )))));
}

function test_createCompulsoryDelegate() public {
    address precomputedWrapper = computeWrapperAddress(address(apes));
    address pair = factory.createPair(address(usdc), address(precomputedWrapper));
    address realWrapper = factory.createWrapper(address(apes));

    assertEq(factory.delegates(address(usdc), address(precomputedWrapper)), true );
    assertEq(precomputedWrapper, realWrapper);
}
```

Recommendation:

- The issue can be resolved by verifying if the pair address provided is associated bytecode.
- Alternatively, you can merge the two function calls into one. During the `createPair` function, use the `supportsInterface` method to determine the contract type. If it is identified as ERC721, create a wrapper and utilize that for generating pairs.

Sweep n' Flip: Fixed in `uniswap-v2-nft` PR 5

Cantina Managed: Verified fix.

3.2 Medium Risk

3.2.1 `DOMAIN_SEPARATOR` in `UniswapV2ERC20` will be invalid after chain forks

Severity: Medium Risk

Context: UniswapV2ERC20.sol#L20

Description: The current implementation of `DOMAIN_SEPARATOR` in the `UniswapV2ERC20.sol` contract is computed once during contract deployment and stored as an immutable value using the following code:

```
DOMAIN_SEPARATOR = keccak256(
    abi.encode(
        keccak256("EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)"),
        keccak256(bytes(name)),
        keccak256(bytes("1")),
        block.chainid,
        address(this)
    )
);
```

The `block.chainid` is read once during contract deployment. If the chain forks and the `block.chainid` changes, the stored `DOMAIN_SEPARATOR` will still contain the old chain ID, making all permit signatures invalid on the new chain.

Recommendation:

- Consider using the EIP712 library from OpenZeppelin.
- Or cache the **DOMAIN_SEPARTOR** and update it later if the `block.chainid` changes, but the above recommendation is more straight forward and simpler.

Sweep n' Flip: We do not attempt to automatically support forks of the underlying chain. Furthermore, any fixes will deviate to much from the original Uniswap V2 implementation which also does not properly handle chain forks. We will choose not to action on this one for now.

Cantina Managed: Acknowledged. We are still advocating for this to be communicated in the documentation as there will be a brief action plan after the forks, and they are not inherently supported.

3.2.2 Revert on swaps involving non-delegated ERC721 pairs through `RouterCollection`

Severity: Medium Risk

Context: UniswapV2Factory.sol#L66, UniswapV2Library.sol#L102-L122

Description: The current design of the router and factory system for delegated pairs (non-wrapped ERC721 tokens) introduces a significant limitation:

1. Pair Not Marked as Delegated: The `delegates` mapping for a pair is only updated to `true` via the `createPair` function, which is invoked on normal operations during `_addLiquidity`.
2. Revert Behavior: If a pair is not marked as delegated in the `delegates` mapping:
    - The `pairForWithDelegates` function in the `UniswapV2Library` defaults to generating a pair address as if it were a SwipNFlip factory-generated address.
    - Any swap operation involving such a pair will fail, as functions like `getAmountsOut` or `getAmountsIn` rely on the `getReserves` function, which will revert for non-existent pairs.
3. Operational Limitations: This design requires all delegated pairs to be explicitly marked in advance using the `createPair` function. Without this pre-setup, any attempt to swap through non-SwipNFlip pools will revert.

Recommendation: To address these limitations, consider the following solutions:

1. Automatically register delegated pairs during interaction. Modify the code to handle non-marked pairs gracefully. If the pair is not already in the `delegates` mapping, automatically register it by creating or validating the pair on the fly.
2. Proactively mark delegated pairs. Provide a utility function to batch-mark pairs as delegated. This function can be executed during deployment or by an admin to pre-register all potential pairs. *Note: this can be done also with `createPair` itself.*

3. Fallback Logic for Non-Delegated Pairs. Update the `getAmountsOut` and `getAmountsIn` functions to detect non-existent pairs and provide fallback logic or clear error messages.
4. Create specific delegate swap functions that handle the logic differently.

Ensure that developers and integrators are aware of the need to register delegated pairs in advance. Provide clear deployment scripts or documentation that include the steps to pre-register pairs or initialize the system with required pairs.

Consider implementing some of these recommendations to ensure smoother operation for swaps involving non-SwipNFlip pools while maintaining system integrity and reducing user friction.

Sweep n' Flip: The develompment team does not see this being different from the behavior users would get if there was no delegation. The same errors would happen. For all intents and purposes, delegation should be viewed as an implementation issue. The fact that the pool was created in a delegated AMM is oblivious to users. In other words, without delegation users need to add liquidity to use a pair, with delegation users also need to do that, but when they add liquidity suddenly a lot of liquidity "appears" along the operation. So not working on a fix for this one.

Cantina Managed: Acknowledged.

3.2.3 Broken protocol invariant: Decreasing `WERC721` `totalSupply` without unlocking different NFTs

Severity: Medium Risk

Context: WERC721.sol#L59-L69

Description: The `totalSupply` and `balanceOf` logic in the pool is susceptible to a scenario where users can repeatedly buy the same NFT, artificially decreasing the `totalSupply` while the actual NFTs remain locked in the wrapper contract. This creates a situation where:

1. Decreased `totalSupply`: The `totalSupply` metric decreases with each repeated transaction.
2. No change in NFT ownership: The remaining NFTs stay locked in the wrapper contract without being accessible for swaps.

This allows users to deplete the `totalSupply` without effectively draining the NFTs, leaving the wrapped contract in a broken state.

Proof of Concept:

```
function test_buySameIdSeveralTimes() public {
    usdc.mint(address(this), 10000e6);
    usdc.approve(address(routerCollection), type(uint256).max);
    uint256[] memory tokenIds = new uint256[](10);
    for(uint i; i < 10; ++i) {
        apes.mint(address(this), i);
        apes.approve(address(routerCollection), i);
        tokenIds[i] = i;
    }

    address wrapper = factory.createWrapper(address(apes));
    address pair = factory.createPair(address(wrapper), address(usdc));

    (,,uint256 liquidity) = routerCollection.addLiquidityCollection(
        address(usdc), address(apes), 1000e6, tokenIds, 1, address(this), 1 days
    );

    usdc.mint(address(this), 10000e6);
    tokenIds = new uint256[](1);
    tokenIds[0] = 0;

    address[] memory path = new address[](2);
    path[0] = address(usdc);
    path[1] = address(apes);

    uint256[] memory amounts;

    // Retrieve 5 times the same NFT
    for(uint i; i < 5; ++i) {
        amounts = routerCollection.swapTokensForExactTokensCollection(
            tokenIds, 5000e6, path, true, address(wrapper), 1 days
        );
```

```
    }

    // Total Supply decreased along with the balance of the pool
    assertEq(WERC721(wrapper).totalSupply(), 5e18);
    assertEq(WERC721(wrapper).balanceOf(pair), WERC721(wrapper).totalSupply());

    // The owner of all NFTs is still the wrapped contract
    for(uint i=1; i < 10; ++i){
        assertEq(apes.ownerOf(i), wrapper);
    }
}
```

Recommendation: Consider the next two potential improvements:

1. Track unique token IDs:
    - Implement logic to ensure that once a token is swapped out, it cannot be swapped back into the pool as a "new" NFT.
    - Maintain a mapping of token IDs currently in the pool to verify against when swapping.
2. Validate NFT transfers:
    - Ensure that when an NFT is transferred back to the wrapper contract, it doesn't become eligible for resale without proper state updates.
    - Add a check in the wrapper logic to prevent recycled NFTs from artificially affecting pool metrics.

To have a unique set of IDs consider using the EnumerableSet from Openzeppelin to track the IDs.

Sweep n' Flip: As discussed, sending any tokens to the wrapper contract is protocol misuse. AFAICT, it will simply lock the token(s) from the collection into the wrapper contract, like would happen when sending the tokens(s) to any other arbitrary address. The fact that the same token can be redeemed over and over is a unintended side effect of that unusual situation, as users can pick and chose that/those token/s. It does not impact the the proper operation of the AMM in terms of price, liquidity, user ownership, and so on. For simplicity we will not attempt to fix it.

Cantina Managed: Acknowledged.

3.2.4 Incorrect handling of Velodrome stable pools

Severity: Medium Risk

Context: UniswapV2Factory.sol#L58

Description: The `createPair` function in the `UniswapV2Factory` for SweepnFlip always sets the `stable` flag to `false` when interacting with Velodrome's factory. This hardcoding effectively prevents the creation or interaction with stable pools, as the system only supports volatile pools. As a result:

1. Stable pools excluded: Users cannot create or interact with stable pools in Velodrome, limiting functionality.
2. Reverts on stable pool paths: Any swap path requiring stable pools will fail due to the inability to query or create stable pool addresses correctly.

Recommendation: To address this limitation, update the `createPair` function to dynamically handle the `stable` flag based on user input or configuration.

Consider adding a `stable` parameter to the `createPair` function to allow flexibility in choosing between stable and volatile pools. Main approaches to solve this issue:

1. Dynamic `stable` flag: The `stable` flag is now passed as a parameter to the `createPair` function, allowing users to explicitly specify whether the pool should be stable or volatile.
2. Support for stable pools: The updated function queries and creates pools in Velodrome based on the provided `stable` flag, enabling support for stable pools.

These changes ensure the SweepnFlip implementation can fully interact with Velodrome's stable pools, enhancing compatibility and user experience while avoiding unnecessary reverts.

Sweep n' Flip: This will not be fixed now. Will remove the support for Velodrome in the future as it is not fully compatible with Uniswap V2. The issue can be circumvented if absolutely necessary by manually creating the pair on Velodrome.

Cantina Managed: Acknowledged.

3.2.5 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.

3.3 Low Risk

3.3.1 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.

3.3.2 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 `safeTransfers`, the receiving contracts should also import the ERC721Holder from OpenZeppelin.

Sweep n' Flip: Acknowledged.

Cantina Managed: Acknowledged.

3.3.3 Lack of marketplace fee sanity check on constructor

Severity: Low Risk

Context: UniswapV2Router01Collection.sol#L25

Description: The `constructor` of `UniswapV2Router01Collection.sol` initializes the `marketplaceFee` global state variable using supplied input parameters however it does not perform the sanity checks for appropriate values as it is done on the setter function `updateFeeConfig`.

Recommendation: Consider performing the same sanity check to avoid the contract from operating with improper values.

```
require(_marketplaceFee <= 100e16, "SweepnFlipRouter: INVALID_FEE");
```

Sweep n' Flip: This would be consistent and good practice. But we will not implement that at the moment for practical and convenience reasons. The contract is already very close to the size limitations imposed by the EVM. Those kind of checks in the constructor would likely be removed anyways to more easily accommodate other critical changes/adjustments that might need to happen during the development life-cycle.

Cantina Managed: Acknowledged.

3.3.4 Potential risks in `updateAdmin` function in `UniswapV2Router01Collection.sol`

Severity: Low Risk

Context: UniswapV2Router01Collection.sol#L28-L33

Description: The `updateAdmin` function in the `UniswapV2Router01Collection.sol` contract allows the current admin to update the `marketplaceAdmin` address. However, this function has two notable risks:

1. Renouncing control: The admin can inadvertently or intentionally set the `marketplaceAdmin` address to `address(0)`, effectively renouncing control over the router, leaving it in an ungovernable state.
2. Irreversible changes: If an incorrect or unintended address is set as the new admin, the change is non-reversible without external intervention or redeployment of the contract.

These issues could lead to operational disruptions.

Recommendation: To mitigate these risks, consider implementing a two-step ownership transfer pattern or leveraging OpenZeppelin's `Ownable` or `AccessControl` contracts, which provide secure patterns for ownership and role management. Benefits:

1. Prevent accidental renouncement: Ensures the admin address cannot be set to `address(0)` without deliberate action.
2. Reversible before completion: Allows the current admin to reconsider or update the pending admin before transfer finalization.
3. Improved governance: Provides a clear and auditable process for admin updates, reducing the likelihood of mismanagement.

Sweep n' Flip: This is a valid issue with a valuable recommendation. Deliberate ownership renouncement is not an unintended behavior. Mistakes on ownership transfer indeed could be a potential problem. However, we will not fix/modify to address this issue at the moment. It is important to note that admin will likely be a mulltisig in a real-world scenario and multiple people would be reviewing any changes to privileged controls.

Cantina Managed: Acknowledged.

3.3.5 Weak role management and lack of robust access control in `UniswapV2Factory.sol`

Severity: Low Risk

Context: UniswapV2Factory.sol#L91-L94, UniswapV2Factory.sol#L101-L104

Description: The `UniswapV2Factory.sol` contract defines key role-based setters (`feeToSetter` and `routerSetter`) responsible for managing critical contract configurations, such as fees and router settings. However, the current implementation has the following issues:

1. Role renouncement risk: Both `feeToSetter` and `routerSetter` can inadvertently or intentionally set their respective roles to `address(0)`, effectively renouncing their control and leaving the system in an unmanaged state.
2. Irreversible misconfiguration: If incorrect or unintended addresses are set for the roles, it could lead to irreversible mismanagement without external intervention or contract redeployment.
3. No robust access control: The current implementation relies solely on `require` statements for access control, which can lead to fragmentation and inconsistencies. A centralized, standardized access control mechanism, such as OpenZeppelin's `Ownable` or `AccessControl` modules, is not utilized.

Recommendation: To address these issues, consider using OpenZeppelin's `Ownable` or `AccessControl` for Role Management

- `Ownable`: Provides a tested and widely adopted ownership transfer pattern, ensuring safe and intentional transfers of ownership.
- `AccessControl`: Offers fine-grained role-based access control, allowing multiple roles (e.g., `feeToSetter` and `routerSetter`) to be managed securely.

Benefits:

1. Improved governance: Prevents accidental or intentional role renouncement and ensures proper role transfers.
2. Traceability: Events provide a clear audit trail for state changes, improving debugging and transparency.
3. Standardization: OpenZeppelin libraries are widely used and thoroughly tested, reducing the likelihood of errors or vulnerabilities in access control.

Implementing these changes ensures a more secure, maintainable, and auditable role management mechanism for the `UniswapV2Factory.sol` contract.

Sweep n' Flip: There are valuable suggestions and recommendations. But this will deviate from the original Uniswap V2 implementation. For now we will keep it as is.

Cantina Managed: Acknowledged.

3.3.6 Narrow redemption range and inflexible ID requirements in `removeLiquidityCollection`

Severity: Low Risk

Context: UniswapV2Router01Collection.sol#L141

Description: The `removeLiquidityCollection` function treats the `amountBMin` parameter as an **exact number of NFTs** to redeem. This approach introduces two key issues:

1. Precise and inflexible ID requirements: The user must specify exact and precise NFT IDs (`tokenIdsB`). If any ID changes (e.g., due to swaps or market conditions), the transaction will revert. This rigid requirement can hinder usability, especially for users who simply want to retrieve their liquidity without concerns over specific NFT IDs.
2. Limited flexibility: Users may only retrieve NFTs matching `tokenIdsB`. There is no mechanism to retrieve:
    - Any available NFTs from the pool.
    - A minimum number of NFTs, regardless of their IDs.

These limitations make the function less user-friendly and prone to transaction failures in active market conditions.

Recommendation: To enhance flexibility and user experience, consider the following improvements:

1. Allow flexible redemption: Modify the function to accept `tokenIdsB` as an optional parameter. If `tokenIdsB` is not provided, allow users to retrieve:
    - Any NFTs available in the pool.
    - A specified minimum number of NFTs.
2. Introduce a flexible function for minimum IDs: Add a separate function for retrieving a specified minimum number of NFTs without requiring specific IDs.
3. Enhance Documentation: Clearly describe the behavior of `removeLiquidityCollection` and the new functions, highlighting the differences and use cases for specific IDs versus flexible retrieval.

By implementing these changes, the system will provide users with more robust and user-friendly options for retrieving their liquidity, reducing transaction failures and improving overall usability.

Sweep n' Flip: Might consider these changes in the future. The team acknowledge the recommendations and highlight the trade-off between flexibility (both in user experience and concurrency handling) versus additional costs in gas, code complexity and contract size.

Cantina Managed: Acknowledged.

3.4 Gas Optimization

3.4.1 `initialize` function can be included on constructor

Severity: Gas Optimization

Context: UniswapV2Pair.sol#L57

Description: The `initialize` function on the `UniswapV2Pair.sol` contract can be included on the constructor without altering the contract functionality. Doing this it is also possible to make `token0` and `token1` `immutable`.

Recommendation: Consider including the `initialize` function on the constructor to save gas and making global state variables immutable.

Sweep n' Flip: This deviates from the original Uniswap V2 implementation. We will keep as is.

Cantina Managed: Acknowledged.

3.4.2 `factory` global state variable can be set as `immutable`

Severity: Gas Optimization

Context: UniswapV2Pair.sol#L18

Description: The `factory` global state variable from the `UniswapV2Pair.sol` contract can be set as immutable as it is only initialized on constructor with the `msg.sender` value and never modofied, nor have a setter.

Recommendation: Consider making the `factory` global state variable immutable to save gas.

Sweep n' Flip: We will consider this one. This deviates from the original Uniswap V2 implementation though.

Cantina Managed: Acknowledged.

3.4.3 Optimizations on `WERC721.sol` contract

Severity: Gas Optimization

Context: WERC721.sol#L14, WERC721.sol#L15, WERC721.sol#L42-L44

Description: The `factory` global state variable from the `WERC721.sol` contract can be set as immutable as it is only initialised on constructor with the `msg.sender` value and never modified, nor have a setter.

Also the `intialize` function can be included on the `constructor`, saving one function call and also allowing to set the `collection` global state variable as immutable.

Recommendation: Consider making the `factory` and `collection` global state variable immutable and including the `intialize` on the `constructor` to save gas.

Sweep n' Flip: We will consider this one.

Cantina Managed: Acknowledged.

3.4.4 Avoid unnecessary code on royalty fee calculations

Severity: Gas Optimization

Context: RoyaltyHelper.sol#L33-L47, UniswapV2Router01Collection.sol#L242

Description: The `RoyaltyHelper.sol` library is designed to calculate the fees that users need to pay when transfering NFTs. There are mainly two fees: the marketplace fee and the royalty fee, if supported for each NFT.

The protocol allows to bypass the payment of the royalty. However even the royalty fee is supposed to be zero, the library still executes the code to check the royalty fee values and performs the corresponding calculations per NFT.

Recommendation: Avoid RoyaltyHelper.sol#L33-L47 if the royalty fee is zero to save gas.

Sweep n' Flip: We will consider this one.

Cantina Managed: Acknowledged.

3.4.5 Several optimizations on `RoyaltyFeeHelper` library

Severity: Gas Optimization

Context: RoyaltyHelper.sol#L12, RoyaltyHelper.sol#L13, RoyaltyHelper.sol#L21

Description: The `getRoyaltyInfo` function contains multiple redundant operations that increase gas costs:

1. Unnecessary Reinitialization of `implementsIERC2981`:
    - In the `catch` block, the `implementsIERC2981` variable is explicitly re-assigned to `false`, even though it is `false` by default. This is redundant.
2. Redundant Initialization of `totalRoyaltyAmount`:
    - The `totalRoyaltyAmount` variable is initialized to `0` at its declaration and re-initialized later in the logic, which is unnecessary since it starts at `0` by default.
3. Reinitialization of Arrays in the First `else` Block:
    - In the branch where `implementsIERC2981` is `false`, the `royaltyReceivers` and `royaltyAmounts` arrays are unnecessarily re-initialized to zero-length arrays multiple times, which wastes gas.

Recommendation:

1. Remove the explicit re-initialization of `implementsIERC2981` to `false` in the `catch` block.
2. Avoid redundant reinitialization of `totalRoyaltyAmount` since it is already set to `0` by default.
3. Eliminate unnecessary array reinitializations in the `else` block when `implementsIERC2981` is `false`, as they are not required and contribute to gas inefficiency.

These changes streamline the function, reduce gas usage, and improve the overall efficiency of the `RoyaltyHelper` library.

Sweep n' Flip: These are optimizations are likely performed by the Solidity compiler, while subjectively sacrificing code readability. But we will consider the trade-off.

Cantina Managed: Acknowledged.

3.4.6 Unused state variables `discrete0` and `discrete1` in `UniswapV2Pair` contract

Severity: Gas Optimization

Context: UniswapV2Pair.sol#L61-L62

Description: The global state variables `discrete0` and `discrete1` in the `UniswapV2Pair` contract are declared but not utilized anywhere in the contract logic. This results in unnecessary storage consumption and increasing deployment costs.

Recommendation: Consider removing the unused `discrete0` and `discrete1` state variables to:

1. Reduce deployment gas costs by minimizing the contract's storage size.
2. Improve code clarity by eliminating redundant variables that serve no functional purpose.

Removing these unused variables decreases deployment gas costs, simplifies the contract by eliminating unnecessary elements, and enhances readability and maintainability.

Sweep n' Flip: These we kept due to earlier versions of the implementation. It should not be the case, but we have been modifying it back and forth by business requirement/demand. We will consider removing before final version.

Cantina Managed: Acknowledged.

3.5 Informational

3.5.1 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.

3.5.2 Inconsistent `uint` type usage instead of `uint256`

Severity: Informational

Context: UniswapV2ERC20.sol#L11, UniswapV2ERC20.sol#L12, UniswapV2ERC20.sol#L54, UniswapV2ERC20.sol#L59, UniswapV2ERC20.sol#L64

Description: The `UniswapV2ERC20.sol` and `WERC721.sol` contracts use `uint` instead of `uint256` for balance mapping, allowance mapping, and the total supply variable. The functions `transfer`, `transferFrom` and `approve` also accepts an amount parameter that is of type `uint` instead of `uint256`.

While `uint` is an alias for `uint256`, explicit type usage is preferred for clarity and consistency with the ERC-20 standard.

Recommendation: Use explicit uint256 type declarations:

```
uint256  public totalSupply
mapping(address => uint256) public balanceOf;
mapping(address => mapping(address => uint256)) public allowance;

function transfer(address to, uint256 value) external returns (bool) {
  // ...
}

function transferFrom(address from, address to, uint256 value) external returns (bool) {
  // ...
}

function approve(address spender, uint256 value) external returns (bool) {
  // ...
}
```

Sweep n' Flip: It should use uint to be consistent with the original Uniswap V2 code. I always use uint256 as suggested, hence the source of the unintended discrepancies.

Cantina Managed: Acknowledged.

3.5.3 Avoid wrapping `sqrt` function in `unchecked` block

Severity: Informational

Context: Math.sol#L13

Description: The `sqrt` function in the `Math.sol` library calculates the square roots of an input number using the Babylonian method. The code is forked from Uniswap's v2 and is wrapped inside an `unchecked` block.

The implementation looks safe for extreme values in the order of `type(uint256).max`, it is not advised to wrap them inside `unchecked` blocks.

Recommendation: Consider using the exact version of the library from Uniswap v2 to avoid any unexpected protocol behavior.

Sweep n' Flip: We upgraded the code from the original Solidity version to a newer version that supports checked arithmetic operations. So, including the unchecked block aims to maintain the original behavior.

Cantina Managed: Acknowledged. As mentioned in the report, it looks safe, but the gain here is minimal by changing the original Uniswap-v2 code.

3.5.4 Consider using the `ReentrancyGuard` from OpenZeppelin

Severity: Informational

Context: WERC721.sol#L26

Description: The `WERC721.sol` contract has a custom modifier (`lock()`) to prevent re-entrancy. While this implementation is straightforward and fulfills its purpose, it may not be as robust as standard implementations found in libraries like OpenZeppelin. Additionally, it could lead to potential edge cases in more complex interaction scenarios.

Recommendation: Consider using the `ReentrancyGuard` from OpenZeppelin to replace the custom re-entrancy implementation.

Sweep n' Flip: We would have done that, but we tried to maintain consistency with the original Uniswap V2 implementation that uses this lock idiom.

Cantina Managed: Acknowledged

3.5.5 Code quality issues: naming convention

Severity: Informational

Context: UniswapV2Factory.sol#L106

Description: - The `UniswapV2Factory.sol` uses an internal naming convention for the external function **_initCodeHash()**, violating solidity naming best practices. Despite being marked as external, the function uses an underscore prefix typically reserved for internal functions.

- The following internal/private variables in the `Bridge.sol` contract should begin with _:

```
mapping(uint256 => mapping(IBridge.AdapterType => IBridge.EvmChainSettings)) private s_evmChainSettings;
mapping(uint256 => uint256) private s_nonEvmChainToEvmChain;
mapping(address => IBridge.ERC721Wrapped) private s_WERC721FromOriginERC721Address;
mapping(address => address) private s_originERC721FromWERC721Address;
address payable private treasureAddress;
uint256 private systemFeePercent = 10;
uint256 private systemRoyaltyPercent = 0 ether;
```

- The following internal/private variables in `LayerZeroAdapter.sol` contract should begin with -:

```
IBaseAdapter.MessageReceive[] private s_pendingMessagesToExecute;
uint256 private s_messagesExecutedCount;
```

- The following internal/private variables in the `WERC721.sol` contract should begin with _:

```
address private s_bridgeAddress;
mapping(uint256 tokenId => string) private s_tokenURIs;
```

- The following internal/private variables in the `WERC721.sol` contract in uniswap-v2-nft project should begin with _:

```
uint private unlocked = 1;
```

Recommendation: Consider updating the functions and variables above to adhere to the Solidity style guide. For example:

```
function initCodeHash() external pure returns (bytes32) {
    return keccak256(type(UniswapV2Pair).creationCode);
}
```

Sweep n' Flip: Acknowledged.

Cantina Managed: Acknowledged.

3.5.6 Underflow in `mint()` function if liquidity added is less than `MINIMUM_LIQUIDITY`

Severity: Informational

Context: UniswapV2Pair.sol#L115

Description: The `mint()` function in `UniswapV2Pair.sol` has an underflow error when adding liquidity below the `MINIMUM_LIQUIDITY` value.

Though the function panics with an underflow error, handling the error could be more informative for the end user.

Proof of Concept: The below proof of concept explains the issue by adding `1e2` liquidity to the pair contract, which reverts with the `panic: arithmetic underflow or overflow` error:

```
function test_uniswapV2Router() public {
    uint[] memory tokenIds = new uint[](0);

    vm.startPrank(user);
    liqToken.approve(address(router), 100e18);
    collection.setApprovalForAll(address(router), true);
    router.addLiquidityCollection(address(liqToken), address(collection), 1e2, tokenIds, 1e2, user,
    ↪  block.timestamp);
}
```

Recommendation: Check for underflow before the panic and return a more meaningful error message as follow:

```
function mint(address to) external lock returns (uint liquidity) {
    // ...
    if (_totalSupply == 0) {
        uint sqrt = Math.sqrt(amount0 * amount1);
        require (sqrt >= MINIMUM_LIQUIDITY, "SweepnFlip: LESS_THAN_MINIMUM_LIQUIDITY");
        liquidity = sqrt - MINIMUM_LIQUIDITY;
        _mint(address(0), MINIMUM_LIQUIDITY); // permanently lock the first MINIMUM_LIQUIDITY tokens
    }
    // ...
}
```

Sweep n' Flip: This issue also applies to the original Uniswap V2 implementation. We would rather maintain the original implementation.

Cantina Managed: Acknowledged.

3.5.7 `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.

3.5.8 Usage of `ecrecover` in `UniswapV2ERC20.sol`

Severity: Informational

Context: UniswapV2ERC20.sol#L81

Description: The `permit` function in the `UniswapV2ERC20.sol` contract directly utilizes the `ecrecover` opcode for signature verification.

The `ecrecover` opcode lacks built-in security checks, such as ensuring the signature is not malleable (e.g., verifying the s value is in the lower half of the curve). These checks must be manually implemented to prevent edge-case issues.

Nonetheless the implementation is secure and there are no immediate security vulnerabilities identified. This function is present on Uniswap V2 deployed code and it been proven robust.

Recommendation: It is recommended to use the `ECDSA` library from OpenZeppelin, which provides robust utilities for signature verification, including checks for non-malleability and other potential pitfalls. This improves the maintainability and readability of the code while adhering to security best practices.

Replace the current `ecrecover` usage with:

```
bytes32 digest = keccak256(
    abi.encodePacked(
        "\x19\x01",
        DOMAIN_SEPARATOR,
        keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, value, nonce, deadline))
    )
);
address recoveredAddress = ECDSA.recover(digest, v, r, s);
require(recoveredAddress != address(0) && recoveredAddress == owner, "UniswapV2: INVALID_SIGNATURE");
```

This change ensures the `permit` function benefits from the well-tested OpenZeppelin implementation and reduces the risk of subtle issues stemming from the direct use of `ecrecover`.

Sweep n' Flip: This would deviate from the original Uniswap V2 implementation.

Cantina Managed: Acknowledged.

3.5.9 Consider inheritance of openzeppelin `ERC20.sol` contract on `WERC721.sol`

Severity: Informational

Context: WERC721.sol#L80-L81

Description: The `WERC721.sol` contracts acts as an extended ERC20.sol contract. The basic functionalities have been implemented without using the current standard from OpenZeppelin.

Although there has not been any security identified issue, the OpenZeppelin implementation performs important sanity checks and controlled errors in a more accurate way.

Recommendation: Consider inheriting from ERC20.sol contract from OZ.

Sweep n' Flip: I agree that OZ is a solid choice. Will consider.

Cantina Managed: Acknowledged.

3.5.10 Presence of commented-out functions in `UniswapV2Router01Collection`

Severity: Informational

Context: UniswapV2Router01Collection.sol#L172-L202

Description: The `UniswapV2Router01Collection` contract contains two commented-out functions: `removeLiquidityETHWithPermitCollection` and `removeLiquidityWithPermitCollection`. While the presence of commented-out code does not introduce security vulnerabilities, it may hinder code readability and maintainability. Commented code can cause confusion, especially for external reviewers, developers new to the codebase, or during audits, as it obscures the intent and state of the code.

Recommendation: To adhere to clean code practices and improve maintainability, it is recommended to remove the commented-out functions. If the functionality is intended for future implementation, consider using version control or adding a clear note in the project's documentation to outline its planned development.

Removing unused commented code helps ensure the codebase remains concise and easier to understand, fostering better collaboration and reducing the likelihood of misinterpretation during development or review processes.

Sweep n' Flip: This should be removed.

Cantina Managed: Acknowledged.

3.5.11 Inverse delegated pair is not populated

Severity: Informational

Context: UniswapV2Factory.sol#L66

Description: On the UniswapV2 protocol and in SwipNFlip implementation, every time a pair is created the inverse pair is stored as well on the storage. This helps to retrieve if the pair is created without ordering the addresses.

However the delegated mapping does not store the inverse delegated pair. Although with the current system desing there does not seem to exist any security related issue, it is still advised to follow the stablished patter with delegated pairs as well.

Recommendation: Consider storing the delegated inverse pair.

Sweep n' Flip: Not really necessary. As pointed out elsewhere, delegation is an internal implementation artifact that should be oblivious to the AMM users. Except for being consistent it does not really matter.

Cantina Managed: Acknowledged.

3.5.12 Uncontrolled panics on mathematical operations

Severity: Informational

Context: UniswapV2Library.sol#L98

Description: Unlike the original `UniswapV2Library.sol`, the Sweep n' Flip version does not use **** as solidity already checks and prevents overflow. However, as there are no checks in place to ensure the success of the mathematical operation, the contract panics without a controlled error in some situations.

```
uint numerator = reserveIn * amountOut * 10000;
uint denominator = (reserveOut - amountOut) * netFee;
amountIn = numerator / denominator + 1;
```

The possible impacts include:

- The `swapTokensForExactTokensCollection` function in `UniswapV2Router01Collection.sol` can cause an arithmetic underflow when attempting to swap for the entire reserve of wrapped NFT tokens. This leads to a denial of service when users try to perform certain swaps, trying to buy all the NFTs from the pool.

Proof of Concept:

```
function test_forceZeroDivisionPanic() public {
    usdc.mint(address(this), 10000e6);
    usdc.approve(address(routerCollection), 1000e6);
    uint256[] memory tokenIds = new uint256[](1);
    for(uint i; i < 10; ++i) {
        apes.mint(address(this), i);
        apes.approve(address(routerCollection), i);
    }

    tokenIds[0] = 3;

    address wrapper = factory.createWrapper(address(apes));
    address pair = factory.createPair(address(wrapper), address(usdc));

    (,,uint256 liquidity) = routerCollection.addLiquidityCollection(
        address(usdc), address(apes), 1000e6, tokenIds, 1, address(this), 1 days
    );

    usdc.approve(address(routerCollection), 9000e6);

    uint256[] memory amounts;

    address[] memory path = new address[](2);
    path[0] = address(usdc);
    path[1] = address(apes);

    amounts = routerCollection.swapTokensForExactTokensCollection(
        tokenIds, 4000e6, path, true, address(wrapper), 1 days
    );
}
```

Recommendation: Consider controlling potential mathematical errors to avoid uncontrolled execution panics. Adding a following check might clarify why the action failed, but it won't necessarily fix the issue.

```
require(amountOut < reserveOut, "SweepnFlipLibrary: INSUFFICIENT_RESERVE"); // Add this line
```

Sweep n' Flip: The pools always lock at least 1 NFT per design. The fact that the error is non descriptive could be improved but note that this is a contract is in size severely close to the EVM's limit.

Cantina Managed: Acknowledged.

3.5.13 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.

3.5.14 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.

3.5.15 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.

3.5.16 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.