Sushiswap
Security Review

Cantina Managed review by:
Saw-Mon and Natalie, Lead Security Researcher
Milo Truck, Associate Security Researcher

February 1, 2024

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

SushiSwap is a decentralized exchange which, similar to platforms like Uniswap and Balancer, uses a collection of liquidity pools to achieve this goal. Users first lock up assets into smart contracts, and traders then buy and sell cryptocurrencies from those pools, swapping out one token for another.

From Nov 13th to Nov 17th the Cantina team conducted a review of sushiswap RouteProcessor4.sol on commit hash 89babd24. The team identified a total of **26** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 3
- Low Risk: 4
- Gas Optimizations: 2
- Informational: 17

3 Findings

3.1 Medium Risk

3.1.1 `wrapNative()` wrongly transfers all native tokens out when unwrapping

Severity: Medium Risk

Context: RouteProcessor4.sol#L292-L297

Description: The `wrapNative()` function can be used to convert wrapped native tokens into native tokens. Afterwards, the native tokens are transferred to the `to` address.

```
if (directionAndFake & 2 == 0) {
  if (from == msg.sender) IERC20(tokenIn).safeTransferFrom(msg.sender, address(this), amountIn);
  IWETH(tokenIn).withdraw(amountIn);
}
//payable(to).transfer(address(this).balance);
(bool success,)= payable(to).call{value: address(this).balance}("");
```

However, instead of transferring `amountIn` worth of native tokens, which is the amount that was unwrapped, the entire native token balance is transferred out as `address(this).balance` is used. Should the router have any non-transient balance or temporarily hold any native tokens midway through the swap route, it will be drained into the `to` address here.

Recommendation: Only transfer `amountIn` worth of native tokens to the `to` address rather than the entire native token balance.

```
- (bool success,)= payable(to).call{value: address(this).balance}("");
+ (bool success,)= payable(to).call{value: amountIn}("");
```

Sushiswap: Fixed in commit c7053b38.

Cantina Managed: Fixed.

3.1.2 `swapCurve()` is incompatible with input tokens where `approve()` has no return value

Severity: Medium Risk

Context: RouteProcessor4.sol#L454

Description: In the `swapCurve()` function, approval for the input token to Curve's `pool` is granted by calling the token's `approve()` function directly.

```
IERC20(tokenIn).approve(pool, amountIn);
```

This is not compatible with ERC-20 tokens where the `approve()` function does not return a `bool`, most notably USDT.

`approve()` in the `IERC20` interface expects a `bool` to be returned. However, since USDT's `approve()` function does not return a `bool`, the function will revert when attempting to decode its return value.

Recommendation: Grant approvals using `safeIncreaseAllowance()` in OpenZeppelin's `SafeERC20` library instead.

```
- IERC20(tokenIn).approve(pool, amountIn);
+ IERC20(tokenIn).safeIncreaseAllowance(pool, amountIn);
```

Sushiswap: Fixed in commits bbb90746 and 5e854cc4.

Cantina Managed: Fixed.

3.1.3 Use `.call()` instead of `.transfer()` to transfer native tokens in `swapCurve()`

Severity: Medium Risk

Context: RouteProcessor4.sol#L466

Description: In the `swapCurve()` function, when the swap's output is in native tokens, it is transferred to the `to` address using `.transfer()`.

```
if(tokenOut == NATIVE_ADDRESS) {
  payable(to).transfer(amountOut);
} else { // ...
```

However, using `.transfer()` enforces a 2300 gas stipend on the `to` address, which will not work for addresses that consume a large amount of gas upon receiving native tokens. This might also break the functionality of the `swapCurve()` function in the future due to changes in gas prices in the EVM.

Recommendation: Use `.call()` to transfer native tokens instead.

```
- payable(to).transfer(amountOut);
+ payable(to).call{value: amountOut}("");
```

Sushiswap: Fixed in commit 03ee79b2.

Cantina Managed: Fixed.

3.2 Low Risk

3.2.1 `exchange` reverts when swapping a legacy curve pool with native token

Severity: Low Risk

Context: RouteProcessor4.sol#L450-L452

Description: When swapping a legacy curve pool with native token the `exchange` would revert:

```
if (tokenIn == NATIVE_ADDRESS) {
  amountOut = ICurve(pool).exchange{value: amountIn}(fromIndex, toIndex, amountIn, 0);
}
```

This is because legacy pool's `exchange` endpoint would not have a return parameter.

Recommendation: To cover this missed case one would need to also check the pool type and use the relevant `interface` to call `exchange`. Another solution is to unify all paths and query the balance before and after the exchange to calculate the `amountOut`. Although this second approach can be more expensive in some cases.

Sushiswap: Acknowledged, there are no such pools with more than $1000 so this change is not necessary.

Cantina Managed: Acknowledged.

3.2.2 `swapUniV2` does not check whether `tokenIn` belongs to the `pool`

Severity: Low Risk

Context: RouteProcessor4.sol#L342-L345

Description: `swapUniV2(...)` uses the decoded `tokenIn` to transfer tokens when `amountIn` is non-zero. Also `tokenIn` is used to recalculate `amountIn`. But this token has not been checked to see if it belongs to the `pool`.

Recommendation: Check to make sure `tokenIn` is either of the tokens belonging to the `pool`.

Sushiswap: Acknowleged. `stream` is created using our library; if created correctly, the contract will operate as per normal. Otherwise, `pool.swap()` will fail and revert, therefore there is no risk.

Cantina Managed: Acknowleged.

3.2.3 `distributeAndSwap()` should not be allowed to call `swap()` with `amount` of `0`

Severity: Low Risk

Context: RouteProcessor4.sol#L225-L230, RouteProcessor4.sol#L311-L328

Description: In the `distributeAndSwap()` function, `amountTotal` is distributed across multiple calls to `swap()` based on their respective `share`, which is specified by the user.

```
for (uint256 i = 0; i < num; ++i) {
  uint16 share = stream.readUint16();
  uint256 amount = (amountTotal * share) / 65535;
  amountTotal -= amount;
  swap(stream, from, tokenIn, amount);
}
```

However, the function does not check that `amount` is not `0`, which becomes problematic if `swap()` is used to call `bentoBridge()`.

When `bentoBridge()` is called with `amountIn` as `0` for deposits, it assumes the input tokens are at the `bentoBox` address and deposits the excess token balance into `bentoBox`. For withdrawals, it withdraws the entire token balance to the `to` address.

As such, the user could accidentally perform a deposit/withdrawal he did not intend to when `amount` is `0`, which will occur if:

- `amountTotal * shares / 65535` rounds down to `0`.
- The user mistakenly specifies `share` as `0`.

Recommendation: In the loop, check if `amount` is `0` and skip the call to `swap()` if so. This also applies to the `processInsideBento()` function, which contains the same logic.

```
  amountTotal -= amount;
+ if (amount == 0) continue;
  swap(stream, from, tokenIn, amount);
```

Sushiswap: Fixed in commit a1d42b5a. We decided against skipping the call to `swap()` since the swap code is already part of `stream` and should be processed. Instead of using `amountIn == 0` to represent that liquidity is already at the pool, it was changed to `from == INTERNAL_INPUT_SOURCE`.

Cantina Managed: Fixed.

3.2.4 Revert reason in `transferValueAndprocessRoute()` is bubbled up incorrectly

Severity: Low Risk

Context: RouteProcessor4.sol#L122-L123

Description: The `transferValueAndprocessRoute()` function performs a call to the `transferValueTo` address to transfer native tokens. If the call fails, the revert reason in `returnBytes` is bubbled up incorrectly using `string(abi.encodePacked(...))`.

```
(bool success, bytes memory returnBytes) = transferValueTo.call{value: amountValueTransfer}('');
require(success, string(abi.encodePacked(returnBytes)));
```

Recommendation: Use `revert` in assembly to bubble up the revert reason instead.

```
  (bool success, bytes memory returnBytes) = transferValueTo.call{value: amountValueTransfer}('');
- require(success, string(abi.encodePacked(returnBytes)));
+ if (!success) {
+   assembly {
+     revert(add(32, returnBytes), mload(returnBytes))
+   }
+ }
```

Sushiswap: Fixed in commit 4e34380d.

Cantina Managed: Fixed.

3.3 Gas Optimization

3.3.1 Storage update and decoding can be performed after the `require` statements in `uniswapV3SwapCallback(...)`

Severity: Gas Optimization

Context: RouteProcessor4.sol#L406-L417

Description: In case one of the require statements in the above context fails, it would be more gas efficient to apply the storage update and decoding of the provided `data` after checking all the requirements.

Recommendation: Storage update and decoding can be performed after the require statements in `uniswapV3SwapCallback(...)`:

```
require(msg.sender == lastCalledPool, 'RouteProcessor.uniswapV3SwapCallback: call from unknown source');
int256 amount = amount0Delta > 0 ? amount0Delta : amount1Delta;
require(amount > 0, 'RouteProcessor.uniswapV3SwapCallback: not positive amount');

lastCalledPool = IMPOSSIBLE_POOL_ADDRESS;
(address tokenIn) = abi.decode(data, (address));
IERC20(tokenIn).safeTransfer(msg.sender, uint256(amount));
```

Sushiswap: Fixed in commit ad327e28.

Cantina Managed: Fixed.

3.3.2 Rearrange conditions in `onlyOwnerOrPriviledgedUser()` by frequency

Severity: Gas Optimization

Context: RouteProcessor4.sol#L59

Description: The `onlyOwnerOrPriviledgedUser()` modifier contains two conditional statements, one to check if `msg.sender` is `owner` and the other to check if `msg.sender` is a privileged user.

```
modifier onlyOwnerOrPriviledgedUser() {
  require(msg.sender == owner() || priviledgedUsers[msg.sender] == true, "RP: caller is not the owner or a
↪  priviledged user");
  _;
}
```

These conditional statements should be ordered based on whichever condition is `true` more often to save gas.

Recommendation: Rearrange the conditional statements according to whichever party ends up calling `pause()` or `resume()` more often.

Sushiswap: Acknowledged, no change as functions with the `onlyOwnerOrPriviledgedUser()` modifier will rarely or never be called.

Cantina Managed: Acknowledged.

3.4 Informational

3.4.1 Define util functions to refactor logic

Severity: Informational

Context: RouteProcessor4.sol#L141-L142, RouteProcessor4.sol#L167-L168

Description: In the above context initial and final balances of input and output token are calculated as:

```
uint256 balanceInInitial  = tokenIn  == NATIVE_ADDRESS ? address(this).balance : IERC20(tokenIn
↪  ).balanceOf(msg.sender);
uint256 balanceOutInitial = tokenOut == NATIVE_ADDRESS ? address(to  ).balance : IERC20(tokenOut).balanceOf(to
↪         );
// ...
uint256 balanceInFinal    = tokenIn  == NATIVE_ADDRESS ? address(this).balance : IERC20(tokenIn
↪  ).balanceOf(msg.sender);
uint256 balanceOutFinal   = tokenOut == NATIVE_ADDRESS ? address(to  ).balance : IERC20(tokenOut).balanceOf(to
↪         );
```

Recommendation: It might be best to the following things:

1. Cache the conditional statements and reuse the result `a == b`.
2. Define util functions `_balanceIn(bool isNative, IERC20 tokenIn)` and `_balanceOut(bool isNative, IERC20 tokenOut, address to)`.

Sushiswap: Acknowledged, no change to the current implementation.

Cantina Managed: Acknowledged.

3.4.2 Slippage protection

Severity: Informational

Context: RouteProcessor4.sol#L170-L172

Description: Currently there is only one check for slippage protection and this is for the overall `amountOut` of `tokenOut`:

```
uint256 balanceOutFinal = tokenOut == NATIVE_ADDRESS ? address(to).balance : IERC20(tokenOut).balanceOf(to);
if (balanceOutFinal < balanceOutInitial + amountOutMin)
  revert MinimalOutputBalanceViolation(balanceOutFinal - balanceOutInitial);
```

The internal swaps might happen for other unrelated tokens. Currently there are no way to protect those swap from slippage.

- In `swapCurve` the min amount out is set to `0`, `pool.exchange(fromIndex, toIndex, amountIn, 0)`. And so slippage protection has been avoided.

Recommendation: A granular slippage protection can be applied using parameters for swap data in streams.

Sushiswap: Acknowledged. Adding additional slippage checks for each individual swap will create additional overhead, it is not obvious if these checks are worth it.

Cantina Managed: Acknowledged.

3.4.3 Only curve pools with less than `129` tokens are supported by the router

Severity: Informational

Context: RouteProcessor4.sol#L444-L445

Description: In `swapCurve` we have:

```
int128 fromIndex = int8(stream.readUint8());
int128 toIndex = int8(stream.readUint8());
```

where these two parameters are the indexes of the tokens in the pools. In general these parameters can have the type `int128`, but the way they are encoded in the `route` they can only go as high as `127`. So not all hypothetical curve pools would be supported in this router.

Recommendation: The above can be documented for the users.

Sushiswap: Acknowledged.

Cantina Managed: Acknowledged.

3.4.4 Subsequent calls to `swapCurve()` will revert if `exchange()` leaves an existing allowance behind

Severity: Informational

Context: RouteProcessor4.sol#L454

Description: The `swapCurve()` function grants an allowance of `amountIn` to `pool` whenever it is called to perform a swap.

```
IERC20(tokenIn).approve(pool, amountIn);
if (poolType == 0) amountOut = ICurve(pool).exchange(fromIndex, toIndex, amountIn, 0);
else {
  uint256 balanceBefore = IERC20(tokenOut).balanceOf(address(this));
  ICurveLegacy(pool).exchange(fromIndex, toIndex, amountIn, 0);
  // ...
```

However, this implementation is dangerous for ERC-20 tokens where `approve()` reverts if there is an existing non-zero allowance, such as USDT.

If the pool's `exchange()` function doesn't spend the entire allowance, it will leave a non-zero allowance behind. This will cause all future calls to `swapCurve()` with the same pool and `tokenIn` to revert.

Note that there currently isn't any Curve pool where `exchange()` does not spend the entire allowance, so this will only become a problem should future pools be different.

Recommendation: Perhaps scenarios such as above can be documented for the users.

Sushiswap: Fixed in commit bbb90746. If the original call to `approve()` reverts, `approve(..., 0)` is called to reset the existing allowance followed by `approve(..., amountIn)`.

Cantina Managed: Fixed.

3.4.5 `route` is not emitted in `processRouteInternal(...)`

Severity: Informational

Context: RouteProcessor4.sol#L176

Description: `route` is not emitted as data in the event below as it can consume a lot of gas:

```
emit Route(msg.sender, to, tokenIn, tokenOut, realAmountIn, amountOutMin, amountOut);
```

Recommendation: One can instead emit its hash for off-chain analysis if needed.

Sushiswap: Acknowledged.

Cantina Managed: Acknowledged.

3.4.6 `from` is always `msg.sender` or `address(this)`

Severity: Informational

Context: RouteProcessor4.sol#L269-L274, RouteProcessor4.sol#L363-L372

Description: See the following diagram:

By above the `from` address is always `msg.sender` or `address(this)`. For the line RouteProcessor4.sol#L368:

```
bentoBox.transfer(tokenIn, from, pool, amountIn);
```

The current implementation of the `IBentoBoxMinimal` is the following modifier:

```
// file: BentoBoxV1.sol

// Modifier to check if the msg.sender is allowed to use funds belonging to the 'from' address.
// If 'from' is msg.sender, it's allowed.
// If 'from' is the BentoBox itself, it's allowed. Any ETH, token balances (above the known balances) or
↪  BentoBox balances
// can be taken by anyone.
// This is to enable skimming, not just for deposits, but also for withdrawals or transfers, enabling better
↪  composability.
// If 'from' is a clone of a masterContract AND the 'from' address has approved that masterContract, it's
↪  allowed.
modifier allowed(address from) {
    if (from != msg.sender && from != address(this)) {
        // From is sender or you are skimming
        address masterContract = masterContractOf[msg.sender];
        require(masterContract != address(0), "BentoBox: no masterContract");
        require(masterContractApproved[masterContract][from], "BentoBox: Transfer not approved");
    }
    _;
}
```

so the `from` in the context of `swapTrident` can only be the the address of the `RP4` router and not the original `msg.sender` to `RP4` . And so the `from` does not need to be passed to this endpoint and one can use the `address(this)` directly.

So the route

is not possible for non-zero `amountIn` unless a different implementation of `IBentoBoxMinimal` is used.

Recommendation: `from` does not need to be passed to `swapTrident` and one can use the `address(this)` directly. Unless in some weird scenario a clone of the `RP4` master contract has been approved by the `from` address to be used in the `bentoBox`.

Note: The above recommendation depends on the implementation of `IBentoBoxMinimal`.

In general for all the child `private` function nodes of the above diagram one can pass a boolean variable `isMsgSender` and depending on its value use `msg.sender` or `address(this)`.

Sushiswap: Acknowledged. The scenario mentioned is possible only in swaps from inside Bento, which are not allowed now and most probably will never be. This will be fixed if we ever decide to support them.

Cantina Managed: Acknowledged.

3.4.7 Implementation of `processInsideBento()` can be simplified

Severity: Informational

Context: RouteProcessor4.sol#L246-L260

Description: The `processInsideBento()` function contains logic to distribute `amountTotal` over multiple calls to `swap()`:

```
uint8 num = stream.readUint8();

uint256 amountTotal = bentoBox.balanceOf(token, address(this));
unchecked {
  if (amountTotal > 0) amountTotal -= 1;      // slot undrain protection
  for (uint256 i = 0; i < num; ++i) {
    uint16 share = stream.readUint16();
    uint256 amount = (amountTotal * share) / 65535;
    amountTotal -= amount;
    swap(stream, address(this), token, amount);
  }
}
```

This logic is exactly the same as `distributeAndSwap()`, which is used to achieve the same purpose.

Recommendation: Refactor `processInsideBento()` to use `distributeAndSwap()` instead:

```
function processInsideBento(uint256 stream) private {
  address token = stream.readAddress();
  uint256 amountTotal = bentoBox.balanceOf(token, address(this));
  unchecked {
    if (amountTotal > 0) amountTotal -= 1;      // slot undrain protection
  }
  distributeAndSwap(stream, address(this), token, amountTotal);
}
```

Sushiswap: Fixed in commit 9ff434a3.

Cantina Managed: Fixed.

3.4.8 `safePermit()` has been removed from `SafeERC20`

Severity: Informational

Context: RouteProcessor4.sol#L186

Description: The `applyPermit()` function uses `safePermit()` from OpenZeppelin's `SafeERC20` library:

```
IERC20Permit(tokenIn).safePermit(msg.sender, address(this), value, deadline, v, r, s);
```

However, this function has been removed and will no longer be available from v5.0.0 onwards.

Recommendation: Refactor `applyPermit()` to not use `safePermit()` when the contract's OpenZeppelin version is upgraded.

Sushiswap: Acknowledged the warning.

Cantina Managed: Acknowledged.

3.4.9 `direction` and `zeroForOne` in `swapUniV2()` and `swapUniV3()` are unnecessary

Severity: Informational

Context: RouteProcessor4.sol#L338, RouteProcessor4.sol#L381

Description: Both `swapUniV2()` and `swapUniV3()` have a `direction` and `zeroForOne` parameter respectively in `stream` to specify the direction of the swap (i.e. whether the swap is from `token0` to `token1`, or vice versa).

However, the direction of the swap can programmatically be determined by comparing `tokenIn` to `token0` in the pool, which makes these parameters are unnecessary. This is what the Uniswap V2 router does.

Recommendation: In `swapUniV2()` and `swapUniV3()`, determine `direction` and `zeroForOne` by comparing `tokenIn` to `token0` and remove them from `stream`.

```
- uint8 direction = stream.readUint8();
+ uint8 direction = tokenIn == IUniswapV2Pair(pool).token0();
```

```
- bool zeroForOne = stream.readUint8() > 0;
+ bool zeroForOne = tokenIn == IUniswapV3Pool(pool).token0();
```

Sushiswap: Acknowledged, will not make the change since an additional external call might consume more gas.

Cantina Managed: Acknowledged.

3.4.10 Use a shared callback handler for `uniswapV3SwapCallback()` and `algebraSwapCallback()`

Severity: Informational

Context: RouteProcessor4.sol#L397-L434

Description: The contract implements the logic for handling swap callbacks from Uniswap V3 in `uniswapV3SwapCallback()`, which is a `public` function. To support Algebra pools as well, `algebraSwapCallback()` directly calls `uniswapV3SwapCallback()`.

However, it is a better practice to implement this logic in a common function to let both `uniswapV3SwapCallback()` and `algebraSwapCallback()` have `external` visibility.

Recommendation: Add an `internal` or `private` handler function to handle swap callback logic, and have both callback functions call this handler function instead. For example:

```
function uniswapV3SwapCallback(/*...*/) external { _handler(/*...*/); }
function   algebraSwapCallback(/*...*/) external { _handler(/*...*/); }

function _handler(/*...*/) private { /*...*/ }
```

Sushiswap: Acknowledged, no change as `uniswapV3SwapCallback()` is expected to be much more often variant, thus the current implementation is more gas-efficient.

Cantina Managed: Acknowledged.

3.4.11 Events not emitted when modifying storage parameters

Severity: Informational

Context: RouteProcessor4.sol#L63-L82

Description: The contract does not emit events when setting storage parameters, which might be useful for off-chain analysis.

These events are:

- Setting the immutable parameter `bentoBox` in the constructor.
- Adding to `priviledgedUsers[...]` in the constructor and `setPriviledge()`.
- Modifying `paused` in `pause()` and `resume()` when pausing/unpausing the contract.

Recommendation: Emit events when these parameters are modified in their respective functions.

Sushiswap: Acknowledged, these functions should never be called unless an emergency situation occurs.

Cantina Managed: Acknowledged.

3.4.12 Define named constants for possible states of `unlocked` and `paused`

Severity: Informational

Context: RouteProcessor4.sol#L48-L56, RouteProcessor4.sol#L76-L82

Description: The code uses the literals `1` and `2` directly to represent the paused/unpaused and lock/unlocked states of the contract.

```
modifier lock() {
    require(unlocked == 1, 'RouteProcessor is locked');
    require(paused == 1, 'RouteProcessor is paused');
    unlocked = 2;
    _;
    unlocked = 1;
}
```

To be more explicit, it is best to use named constants to represent the states of the contract.

Recommendation: Use named constants instead of `1` and `2`.

```
uint256 private constant NOT_ENTERED = 1;
uint256 private constant ENTERED = 2;
```

Same recommendation applied to `paused`.

Sushiswap: Fixed in commit 351b353d.

Cantina Managed: Fixed.

3.4.13 Use `type(uint16).max` instead of `65535`

Severity: Informational

Context: RouteProcessor4.sol#L227, RouteProcessor4.sol#L255

Description: In `distributeAndSwap()` and `processInsideBento()`, the value `65535` is used when calculating the `amount` distributed to each call to `swap()`.

```
uint256 amount = (amountTotal * share) / 65535;
```

This is equivalent to the maximum value for `uint16`.

Recommendation: Use `type(uint16).max` as a named constant and use that constant in the formulas instead.

Sushiswap: Fixed in commit 048acf73.

Cantina Managed: Fixed.

3.4.14 Use `uint256` instead of `uint` for variable types

Severity: Informational

Context: RouteProcessor4.sol

Description: Throughout the codebase, there is mix usage between `uint256` and `uint`. Although this inconsistency does not affect the contract's functionality, it is best to be explicit when specifying variable types.

Recommendation: Specify `uint256` as the type in the following lines:

- RouteProcessor4.sol#L37-L39.
- RouteProcessor4.sol#L42.
- RouteProcessor4.sol#L67.
- RouteProcessor4.sol#L146.

Sushiswap: Fixed in commit 68bce48b.

Cantina Managed: Fixed.

3.4.15 Avoid comparing a `bool` to `true` in `onlyOwnerOrPriviledgedUser()`

Severity: Informational

Context: RouteProcessor4.sol#L59

Description: In the `onlyOwnerOrPriviledgedUser()` modifier, comparing `priviledgedUsers[msg.sender]` with `== true` in the second condition is redundant.

```
modifier onlyOwnerOrPriviledgedUser() {
  require(msg.sender == owner() || priviledgedUsers[msg.sender] == true, "RP: caller is not the owner or a
↪  priviledged user");
  _;
}
```

Recommendation: Remove `== true` in the second condition.

```
- require(msg.sender == owner() || priviledgedUsers[msg.sender] == true, "RP: caller is not the owner or a
↪  priviledged user");
+ require(msg.sender == owner() || priviledgedUsers[msg.sender], "RP: caller is not the owner or a priviledged
↪  user");
```

Sushiswap: Fixed in commit befa9cd5.

Cantina Managed: Fixed.

3.4.16 Set explicit visibility for storage parameters

Severity: Informational

Context: RouteProcessor4.sol#L45

Description: `priviledgedUsers` is defined as:

```
mapping (address => bool) priviledgedUsers;
```

The compiler implicitly assumes that this storage parameter has a `public` visibility.

Recommendation: It would best to have explicit visibility for storage parameters. In this case define `priviledgedUsers` as:

```
mapping (address => bool) public priviledgedUsers;
```

Sushiswap: Fixed in commit 26a1bb74.

Cantina Managed: Fixed.

3.4.17 Typos and comments

Severity: Informational

Context: InputStream.sol, RouteProcessor4.sol, RouteProcessor4.sol#L26, RouteProcessor4.sol#L179, RouteProcessor4.sol#L180, RouteProcessor4.sol#L283, RouteProcessor4.sol#L332, RouteProcessor4.sol#L400, RouteProcessor4.sol#L437

Description/Recommendation:

- InputStream.sol - Comments and NatSpec docs missing.
- RouteProcessor4.sol - `priviledged` needs to be replaced by `privileged`.
- RouteProcessor4.sol#L26 -`version` perhaps needs to be `4`.
- RouteProcessor4.sol#L179 - NatSpec docs are missing for `applyPermit()`.
- RouteProcessor4.sol#L180 - Missing parenthesis at the beginning. Either remove the comment or specify that these are the parameters for `safePermit()`.
- RouteProcessor4.sol#L283, `wrapNative` might be a misleading function name, as this function also unwraps tokens. It might be best to name it `wrapOrUnwrapToken(...)`.
- RouteProcessor4.sol#L332 - NatSpec for `stream` is missing `fee`, which was added in `RP4`.
- RouteProcessor4.sol#L400 - Incorrect comment. If `amount0Delta` and `amount1Delta` are both `0`, `uniswapV3SwapCallback()` would revert due to the `amount > 0` check.
- RouteProcessor4.sol#L437 - NatSpec for `stream` has incorrect parameters.

Sushiswap: Fixed in commit 281dd466. No change to the following instances:

- RouteProcessor4.sol#L26 -`version` perhaps needs to be `4`.
- RouteProcessor4.sol#L283, `wrapNative` might be a misleading function name, as this function also unwraps tokens. It might be best to name it `wrapOrUnwrapToken(...)`.
- RouteProcessor4.sol#L400 - Incorrect comment. If `amount0Delta` and `amount1Delta` are both `0`, `uniswapV3SwapCallback()` would revert due to the `amount > 0` check.

Also the NatSpec for `InputStream.sol` does not include a documentation on how the stream are constructed.

Cantina Managed: Fixed.