Formal Verification of TridentRouter

Summary

This document describes the specification and verification of TridentRouter by SushiSwap using the Certora Prover. The work was undertaken while the contract was in development and concluded on Nov 4th, 2021.

The scope of our verification was TridentRouter contract. It is a router contract that helps to swap, mint, and burn across Trident pools.

The Certora Prover proved the implementation of TridentRouter is correct with respect to the formal rules written by the Certora and SushiSwap team.

The next section formally defines high-level specifications of TridentRouter. The results of running the Certora Prover are available at:

- TridentRouter

List of Main Issues Discovered

Severity: High

Issue:	Loss of assets
Description:	Amounts confused for BentoBox shares in multiple places
Mitigation/Fix:	Removed unnecessary/wrong conversions between amounts and BentoBox shares

Severity: High

Issue:	Loss of user's assets
Description:	TridentRouter's callback operations assume that the `cashedPool` will call them only on calls initiated by the TridentRouter. This is not a safe assumption since the pool can be a malicious pool.

Issue:	Loss of user's assets
Mitigation/Fix:	Only whitelisted pools should be allowed to call the callback functions

Severity: Recommendation

Issue:	Loss of user's assets
Description:	TridentRouter operations can be called with a different token than the tokens of the pool. The system will not lose any assets, but the users can. For example, a user transfers token A and token B for minting but the pool consists of token B and token C, then the user will not be able to mint and would lose token A.
Mitigation/Fix:	The caller is responsible for making sure that the input tokens match the pool tokens

Disclaimer

The Certora Prover takes as input a contract and a specification and formally proves that the contract satisfies the specification in all scenarios. Importantly, the guarantees of the Certora Prover are scoped to the provided specification, and the Certora Prover does not check any cases not covered by the specification.

We hope that this information is useful, but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

Notations

✔️ indicates the rule is formally verified on the latest reviewed commit. We write ✔️* when the rule was verified on a simplified version of the code (or under some assumptions).

![](https://i.imgur.com/rDhiM7e.png =20x20) indicates the rule was violated under one of the tested versions of the code.

✍️ indicates the rule is not yet formally specified.

🔁 indicates the rule is postponed (<due to other issues, low priority>) .

Verification of TridentRouter

TridentRouter by SushiSwap is an intermediary contract that allows users to use their Eth, BentoBox, or ERC20 tokens to perform different operations across Trident pools. The primary functions include swapping, minting, and burning. Users can swap using one pool, multiple pools, or a complex path (multiple input tokens to multiple output tokens using multiple paths, in different percentages). Furthermore, users can take advantage of TridentRouter's callbacks for flash swaps. The TridentRouter's callbacks use cached information (`cachedMsgSender` and `cachedPool`) from their caller functions.

The operations of TridentRouter are of two main types: native and non-native. Native operations transfer tokens from user's ERC20 accounts or use Eth, whereas non-native operations transfer tokens directly from the user's BentoBox account.

Assumptions

- complexPath and batch operations are out of scope for the verification.
- Most rules are only testing single hop swap functions. The ones that are testing multi hop swap functions assume 2 hops.
- Verified using a symbolic pool that implements the `IPool` interface.

Properties

1. Integrity of cached pool and msg.sender ✔️

- `cachedMsgSender` and `cachedPool` are always 1, except inside a callback functionת

2. Inverse of swapping ✔️

- Swapping tokenIn for tokenOut, followed by tokenOut for tokenIn should preserve user's balance and the final amountOut should be equal to the initial amountIn.

3. Assets are preserved ✔️

- Before and after adding liquidity using the TridentRouter, the assets of a user are preserved.

4. Safety of user's assets ✔️

- No operation may change some other user's assets.

5. Integrity of unwarpBento flag ✔️

- When using native operations with `unwrapBento = true` (pool would deposit assets to user's ERC20 account instead of BentoBox), user's BentoBox balances should not change.