yAudit Bunni Zap Review

Review Resources:

- None beyond the code repositories

Auditors:

- Jackson
- pandadefi

Table of Contents

Review Summary

Bunni Zap

Bunni Zap provides zapping contracts for making multiple Bunni interactions in a single transaction.

The contracts of the Bunni Zap Repo were reviewed over 10 days. The code review was performed by 2 auditors between March 5, 2023 and March 15, 2023. The repository was under active development during the review, but the review was limited to the latest commit at the start of the review. This was commit 4ab7c00f178ea16f5eaba0eca0884a644d99c02c for the Bunni Zap repo.

Scope

The scope of the review consisted of the following contracts at the specific commit:

- Multicall.sol
- SelfPermit.sol
- BunniLpZapIn.sol

After the findings were presented to the Bunni Zap team, fixes were made and included in

several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, Bunni Zap and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	All functions of the `BunniLpZapIn` contract are open for anyone to call. Therefore, no access control is used.
Mathematics	Good	The `BunniLpZapIn` is not intended to handle balances directly, therefore there are no mathematics in the contract.
Complexity	Minor	All of the functions in the `BunniLpZapIn` are deposit and withdrawal actions to dependency or input contracts.
Libraries	Average	The `SelfPermit` and `Multicall` functionality is copied from Uniswap V3. The `solmate`, `timeless`, `bunni`, and `gauge-foundry` dependencies are used primarily for interface purposes.
Decentralization	Good	There are no privileged actors in the `BunniLpZapIn` contract.
Code stability	Good	No changes were made to the code over the course of the audit.

Category	Mark	Description
Documentation	Medium	The only documentation available for the project is the README, however the functionality in `BunniLpZapIn` is simple.
Monitoring	Average	The underlying vault, gauge, and bunni hub dependencies emit events, however none of the `BunniLpZapIn` functions emit events themselves.
Testing and verification	Low	The `BunniLpZapIn` contract has 68% line coverage, 66% function coverage, and 56% branch coverage. In particular, none of the `doZeroExSwap()` function is covered by tests.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements
- Gas savings
  - Findings that can improve the gas efficiency of the contracts
- Informational
  - Findings including recommendations and best practices

Critical Findings

None.

High Findings

None.

Medium Findings

None.

Low Findings

1. Low - Add token revocation to functions granting approval (Jackson)

Technical Details

Currently only `enterWithUnderlying()` and `enterWithVaultShares()` revoke token approval after the function has finished executing.

Impact

Low. It is unlikely that this will result in loss of funds, but is still considered best practice.

Recommendation

Add token revocation everywhere token approval is granted.

Developer Response

Fixed in commit efba894b7e4c136008b065b2a7675e0a1c3f96cb

Gas Savings Findings

1. Gas - nonReentrant isn't needed (panda)

Several functions are using the `nonReentrant` modifier but aren't manipulable with re-entrancy.

Technical Details

`nonReentrant` is usually needed when a function is manipulating state variables that are used in other functions. In this case, the functions listed below are not manipulating state variables that are used in other functions, so the `nonReentrant` modifier is not needed.

List of functions:

- `zapIn` in `BunniLpZapIn.sol` line 79
- `zapInNoStake` in `BunniLpZapIn.sol` line 155
- `enterWithUnderlying` in `BunniLpZapIn.sol` line 221

- `enterWithVaultShares` in `BunniLpZapIn.sol` line 255
- `doZeroExSwap` in `BunniLpZapIn.sol` line 309

Impact

Gas savings.

Recommendation

Remove the `nonReentrant` modifier from the functions listed above.

Developer Response

Fixed in commit 96c5c7342e2c91afedbf8734bdfb32dbbe1cd5e9

Informational Findings

1. Informational - Payable functions don't receive ETH (panda)

Several external functions are marked as `payable` but are not used as such.

Technical Details

The following functions are marked as `payable` but are not used as such:

- `doZeroExSwap` in `BunniLpZapIn.sol` line 309
- `selfPermit` in `SelfPermit.sol` line 36
- `selfPermitIfNecessary` in `SelfPermit.sol` line 40
- `selfPermitAllowed` in `SelfPermit.sol` line 47
- `selfPermitAllowedIfNecessary` in `SelfPermit.sol` line 54

Impact

Informational. This is not a vulnerability but it is a good practice to remove the payable modifier if it is not used.

Recommendation

Remove the `payable` modifier from the functions listed above.

Developer Response

Acknowledged, no change. When `wrapEthInput()` is part of a multicall, `msg.value` of the multicall is non-zero, so other functions used in the multicall must be payable to avoid reverting.

2. Informational - Add events to functions (Jackson)

Technical Details

While the underlying dependencies in each function of the `BunniLpZapIn` contract emit events, the functions themselves do not emit events.

Impact

Informational.

Recommendation

Add events to the end of each `BunniLpZapIn` function.

Developer Response

Acknowledged, no change. We don't really expect to index data from this contract, and even if we do want to do so in the future we can just redeploy with events added.

Final remarks

Our primary concern is related to the assumption that the `BunniLpZapIn` contract will not hold any balances. An invariant test was written during the audit to ensure that this invariant is maintained and the intended use of this contract is that `zapIn()` is the last call in the multicall, which returns the token balances to the recipient. However, if this assumption is ever broken due to faulty configuration or input, the balances in the contract can easily be withdrawn by an attacker using the `useContractBalance` boolean, or nefarious input.

Aside from this, the contract is relatively small and fairly straightforward due to the assumption that balances will not be held in the contract. For this reason, the contract is considered relatively safe, as is evident by the lack of high, critical, and medium findings.
