yAudit Bunni Review

Review Resources:

- README

Auditors:

- Jackson
- devtooligan

Table of Contents

Review Summary

Bunni

From Bunni's README:

> Bunni enables projects to create ERC-20 LP tokens on Uniswap V3 and plug them into existing liquidity incentivization contracts designed for Uniswap V2/Sushiswap.

The main branch of the Bunni repo was reviewed over 14 days, 5 of which were used to create an initial overview of the contract. The code review was performed between August 29 and September 11, 2022. The code was reviewed by 2 auditors for a total of 52 man hours (devtooligan: 27 hours, and Jackson 25 hours). The review was limited to one specific commit.

Scope

Code Repo Commit

The commit reviewed was fc3535c35ca182dee5684e0a9dd9c152cc33047e. The review covered the entire repository at this specific commit but focused on the contracts directory.

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. Bunni and third parties should use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	The onlyOwner modifier was only applied to the `sweepTokens()` and `setProtocolFee()` functions. Access controls existed on the relevant callback functions in LiquidityManagement.sol for deposits into the underlying Uniswap liquidity pool and BunniToken. msg.sender is properly used so that the user cannot perform actions they should not be able to. Access controls are applied where needed.
Mathematics	Average	Solidity 0.7.6 is used, which does not provide overflow and underflow protect. However, Uniswap's FullMath and LowGasSafeMath libraries are used where overflow and underflow protection is necessary.
Complexity	Average	Many function names and implementations are borrowed from UniswapV3 contracts and Solmate. This reduces the amount of custom development work necessary. The primary source of complexity is in the handling of callbacks from UniswapV3 pools and proper accounting in the BunniHub.
Libraries	Average	Since the contracts are using Solidity 0.7.6 some library contracts were copied into the directory rather than being imported. These dependencies include Solmate's CREATE3, ERC20, and SafeTransferLib. However, due to these being copied into the project, they do not match the latest versions of these projects.
Decentralization	Good	Only two functions contain `onlyOwner` modifiers, `sweepTokens()` and `setProtocolFee()` in the BunniHub.
Code stability	Good	Changes were reviewed at a specific commit hash and the scope was not expanded after the review was started. The code reviewed had nearly all features implemented.
Documentation	Good	Descriptive NatSpec comments are found in the interface contracts. The comments accurately describe the function purpose and function input/output arguments. The

Category	Mark	Description
		BunniHub `compound()` function is particularly well commented. The README contains useful information about the intended use and purpose of Bunni.
Monitoring	Average	The core functions in BunniHub contain events, however, the other auxilliary functions do not.
Testing and verification	Average	Bunni uses Foundry for testing. Foundry coverage was used to produce a coverage report that indicates roughly 50% coverage. However, the majority of core logic is found in the src/ directory, which has roughly 75% coverage. The coverage could be improved to test for the edge cases introduced by modifications to the forked Uniswap contracts as demonstrated by the findings. Fuzz and fork tests could also be easily included due to the use of Foundry.

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

1. Critical - Anyone can call `uniswapV3MintCallback()` stealing the protocol fees

(devtooligan, Jackson)

Technical Details

`LiquidityManagement::uniswapV3MintCallback()` only checks that `msg.sender == address(decodedData.pool)`, which any attacker can trivially fulfill by setting `decodedData.pool` to an address they own. The function then transfers `decodedData.pool.token0()` and `decodedData.pool.token1()` to `msg.sender`, the amount of which is specified by the caller.

Impact

Critical, anyone can take the protocol fees stored in BunniHub.

Recommendation

Use the Uniswap V3 factory to verify that the caller is indeed the Uniswap pool associated with the desired tokens and fee tier.

Developer response (zeframL)

This was fixed in commit f6fa8cd1ef06e71ddc25ac15e82c9fe9859a9aec by using the Uniswap V3 factory to verify that the caller is indeed the Uniswap pool associated with the desired tokens and fee tier.

2. Critical - Anyone can call the PeripheryPayments `sweepToken()` stealing the protocol fees (Jackson)

Technical Details

PeripheryPayments is a dependency of the LiquidityManagement contract, which is a dependency of the BunniHub contract. PeripheryPayments has a `sweepToken()` with no access control which allows a caller to transfer any token from the BunniHub contract to any recipient.

Impact

Critical. Anyone can take the protocol fees stored in BunniHub.

Recommendation

Remove or access control the PeripheryPayments `sweepToken()` function.

Developer response (zeframL)

Fixed in commit cfead992384cf959350439eb1ca79d73b0c528ac by removing PeripheryPayments.

3. Critical - Anyone can call the PeripheryPayments `unwrapWETH9()` stealing the weth stored in BunniHub (Jackson)

Technical Details

PeripheryPayments is a dependency of the LiquidityManagement contract, which is a dependency of the BunniHub contract. PeripheryPayments has a `unwrapWETH9()` with no access control which allows a caller to the contract's weth balance from the BunniHub contract to any recipient.

Impact

Critical. Anyone can take the weth stored in BunniHub.

Recommendation

Remove or access control the PeripheryPayments `unwrapWETH9()` function.

Developer response (zeframL)

Fixed in commit cfead992384cf959350439eb1ca79d73b0c528ac by removing PeripheryPayments.

High Findings

None.

Medium Findings

None.

Low Findings

1. Low - General complexity around ETH management including certain circumstances where ETH could be lost (devtooligan)

Technical Details

The `BunniHub` contract currently only has one `payable` function that can receive ETH which is the `receive()` found in `PeripheryPayments`. That function requires that the sender is WETH9. In general `BunniHub` will not be interacting with ETH but there is functionality embedded with the `Uniswap` imports related to the management of ETH which adds complexity and some risk.

```
abstract contract PeripheryPayments is IPeripheryPayments, PeripheryImmutableState {
    receive() external payable {
        require(msg.sender == WETH9, 'Not WETH9');
    }
```

If for some reason someone unwrapped WETH and set the recipient to the BunniHub, then the contract would no hold the ETH. This ETH could be taken by anyone via `refundEth()` or it could also be included used the next time anyone deposited into a WETH pool since `pay()` includes some logic that is designed to work with with unwrapping WETH:

```
    function pay(
        address token,
        address payer,
        address recipient,
        uint256 value
    ) internal {
        if (token == WETH9 && address(this).balance >= value) {
            // pay with WETH9
            IWETH9(WETH9).deposit{value: value}(); // wrap only what is needed to pay
            IWETH9(WETH9).transfer(recipient, value);
        } else if (payer == address(this)) {
            ...
```

Impact

Low. It is a real edge case where a user could lose ETH via `pay()` but the general complexity of including the ETH management is more of a concern.

Recommendation

Consider removing ETH management logic, including the `receive()` and `refundEth()` functions in PeripheryPayments, as well as the conditional logic found in `pay()`.

Developer response (zeframL)

Fixed in commit cfead992384cf959350439eb1ca79d73b0c528ac by removing PeripheryPayments.

2. Low - Calling `withdraw()` forfeits uncollected LP fees (devtooligan)

Technical Details

Fees are collected and reinvested via `compound()`, but when `withdraw()` is called, fees are not collected.

Impact

Low. Depending on the size of the deposit and the amount of time since `compound()` was called, a depositor lose out on significant portion of fees.

Recommendation

Consider adding a second function `compoundAndWithdraw()` or else adding a bool to the current `withdraw()` which explicitly indicates whether or not `compound()` should be called prior to `withdraw()`.

Developer response (zeframL)

This is fine, since `BunniHub` inherits `Multicall` where the caller can batch multiple calls in the same transaction. If the user is okay with paying the extra gas cost, the user can choose to call `compound()` before `withdraw()`.

Gas Savings Findings

1. Gas - Optimize the `sweepTokens()` loop (devtooligan)

Technical Details

```
@@ -436,12 +453,16 @@ contract BunniHub is
-        for (uint256 i = 0; i < tokenList.length; i++) {
+        uint256 tokenListLength = tokenList.length;
+        for (uint256 i = 0; i < tokenListLength;) {
             SafeTransferLib.safeTransfer(
                 tokenList[i],
                 recipient,
                 tokenList[i].balanceOf(address(this))
             );
+            unchecked {
+                ++i;
```

```
+            }
        }
```

Incrementing `i` can be done in an unchecked block because the number of tokens will not overflow saving 108 gas per per iteration. The length of an array can be stored outside of the for loop for additional gas savings 3 gas per iteration.of as is suggested here. Incrementing with `++i` instead of `i++` saves ~5 gas per iteration from reduced stack operations.

Impact

Gas savings ~115 per token submitted.

Developer response (zeframL)

Fixed in commit 8587a86bb5f45a5bcf82d10ce22cca04989b2abd.

Informational Findings

1. Informational - Upgrade to at least solidity 0.8.4 (devtooligan)

Technical Details

Currently there is no underflow or overflow protection at the compiler level in Bunni, which means SafeMath style libraries are necessary, which are gas inefficient compared to the compiler and potentially less safe. Also see hrkrshnn's related comment.

Impact

Informational.

Recommendation

Uniswap V3 now has a 0.8 branch of their contracts. Consider using this branch and upgrading to a more modern version of Solidity.

Developer response (zeframL)

Fixed in commit d2a7f886ce619096c4c79c7a89c4bcc06e941a81.

2. Informational - Use latest versions of unmodified libraries (devtooligan)

Technical Details

The contracts are using Solidity 0.7.6 some library contracts (CREATE3, ERC20, and

SafeTransferLib) were copied into the directory rather than being imported. However, due to these being copied into the project and also because of some modifications, they do not match the latest versions of these projects.

Impact

Informational.

Recommendation

We recommend using the latest versions of the libraries, and using the libraries without modification.

Developer response (zeframL)

Fixed in commit d2a7f886ce619096c4c79c7a89c4bcc06e941a81. Note that `ERC20` and `SafeTransferLib` still need to be copied and modified, because we need `ERC20` to inherit from `IERC20` in order to allow `BunniToken` to inherit from both `ERC20` and `IERC20`.

3. Informational - Consider Foundry scripting for the deployment scripts (Jackson)

Technical Details

Currently bash scripts are used for the deployment scripts. While they are currently relatively simple, as the project progresses this may no longer be the case.

Impact

Informational.

Recommendation

Since Foundry is used in the project for testing, consider migrating to Foundry for the deployment scripts as well, as it provides a few benefits over a bash script, to include compile time checking and transaction simulation.

Developer response (zeframL)

Fixed in 86739fcec723fc7be7d2386152d369597c532ef3.

4. Informational - Typo in README (Jackson)

Technical Details

There is a typo in the README “perfer”.

Impact

Informational.

Recommendation

Change “perfer” to “prefer”.

Developer response (zeframL)

Fixed in commit 5ebca63def23b21004efbbff84a7f105438a6308.

5. Informational - Consider adding npm related steps to the README (Jackson)

Technical Details

npm is used to manage dependencies but this information is not present in the README.

Impact

Informational.

Recommendation

Consider adding the dependency management steps to the README for future contributors.

Developer response (zeframL)

Fixed in commit 5ebca63def23b21004efbbff84a7f105438a6308.

6. Informational - Consider removing the WETH unwrapping logic (Jackson)

Technical Details

One of the critical findings is related to WETH management. If it’s not necessary it is a source of unecessary additional complexity that may lead to other future vulnerabilities.

Impact

Informational.

Recommendation

Consider removing this functionality from all contracts and treat WETH as any other token in order to simplify the logic.

Developer response (zeframL)

Fixed in commit cfead992384cf959350439eb1ca79d73b0c528ac by removing

PeripheryPayments.

Final remarks

devtooligan

We conducted extensive testing based on a mainnet fork to ensure realistic interaction with Uniswap deployed contracts. We stress tested the `deposit()` and `withdraw()` by conducting unit and fuzz tests on multiple scenarios including changing the existing liquidity (through deposit/withdraws) before/after transactions as well as affecting the market price of the pair itself through Uniswap trades. In all of these tests, the target amount being tested was unaffected, remaining constant to the wei. We noted some seemingly unnecessary complexity and risk associated with WETH/ETH logic which can probably be removed completely. Additionally, all functions in the SafeTransferLib implementation were stress tested with fuzzing and no counterexamples were found. Overall, I found the contracts to be clean, well documented, and using best practices.

Jackson

I spent quite a bit of time trying to manipulate the underlying pool in an attempt to get more BunniToken shares, or more underlying tokens from the pool when withdrawing. But, none of these approaches proved fruitful. This is primarily because the ticks used with the Uniswap V3 liquidity pools mean that deposits and withdrawals are contained to those liquidity ranges and those liquidity ranges only even if the pool itself is not trading in those liquidity ranges. As is said in the Bunni README, the project is an ERC20 wrapper over the Uniswap V3 LP NFTs, which makes the system somewhat simple and therefore hard to game and relatively safe from a security perspective.