yAcademy OpenMEV review

Review Resources: Wiki Docs and whitepaper

Residents:

- Jackson
- engin33r

Table of Contents

[TOC]

Review Summary

OpenMEV

The purpose of OpenMEV is a DEX with some addition MEV protection. While enabling exchanges with UniSwap and SushiSwap, it also protects against direct MEV arbitrage (arb) between the two platforms by performing the arb within the DEX swap process.

The main branch of the OpenMEV Repo was reviewed over 22 days, 4 of which were used to create an initial overview of the contract. The code review was performed between May 12 and June 3, 2022. The code was reviewed by 2 residents for a total of XX man hours (engn33r: XX hours, and Jackson: XX hours). The repository was under active development during the review, but the review was limited to one specific commit.

Scope

Code Repo Commit

The commit reviewed was 8648277c0a89d0091f959948682543bdcf0c280b. The review covered the entire repository at this specific commit but focused on the contracts directory.

After the findings were presented to the OpenMEV team, fixes were made and included in commit 8648277c0a89d0091f959948682543bdcf0c280b.

The review was a time-limited review to provide rapid feedback on potential vulnerabilities. The review was not a full audit. The review did not explore all potential attack vectors or areas of vulnerability and may not have identified all potential issues.

yAcademy and the residents make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAcademy and the residents do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. Yearn and third parties should use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	The onlyOwner modifier was applied on functions that set key protocol state variables in Registry.sol and BaseGauge.sol. Elsewhere, msg.sender is used so that the user can only control their own deposits. Access controls are applied where needed.
Mathematics	Average	Solidity 0.8.13 is used, which provides overflow and underflow protect. No unchecked code exists and no low-level bitwise operations are performed. There was no unusually complex math, except perhaps some of the vyper code functions borrowed from Curve which were not modified.
Complexity	Low	The inheritance structure, duplicate function names, lack of clear comments, and inclusion of vyper code all together can make the code hard to follow at times. There is a lack of clarity around the off-chain voting process, and no comments in the code or repository issues/PRs clarify how the off-chain voting expected to integrate with on-chain voting variables.
Libraries	Good	Only basic Open Zeppelin contracts such as SafeERC20, Ownable, and Math are imported, no other external libraries are used. Fewer and simpler external dependencies is always a plus for security.
Decentralization	Average	The onlyOwner modifier indicates there is some centralization risk, but if the owner is a Yearn Finance multisig, the risk is reduced. Contracts such as VotingEscrow.vy support migration, so some form of ownership is required.
Code stability	Average	Changes were reviewed at a specific commit and the scope was not expanded after the review was started. However, development was ongoing when the review was performed so the code was not fully frozen, which means deployed code may vary from what was reviewed.

Category	Mark	Description
Documentation	Low	Comments existed in many places, but were lacking in key areas. As one example, identically named _updateReward functions existed in Gauge.sol, ExtraReward.sol and VeYfiRewards.sol, but no comments existed on either function and no explanation of the differences of these identically-named function existed. It would be best if more thorough comments and documentation was added throughout the code to better explain the purpose of different functions and specific math that is performed. No developer documentation like gitbooks was observed for veYFI at the time of review.
Monitoring	Good	Events were added to all important functions that modified state variables.
Testing and verification	Average	Brownie test coverage was passing with some warnings at the commit reviewed. Test coverage was above 80% in most contract functions but not all. There was a lack of testing integration with snapshot.org voting contracts and off-chain voting strategies.

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

High Findings

1. High - Using normal functions for fee-on-transfer tokens causes value loss (engn33r)

Uniswap's code relies on the assumption that functions without direct support for fee-on-transfer tokens, like `removeLiquidityETH`, will revert. This assumption is invalid in OpenMevRouter. The difference is that Uniswap routers are designed to not hold token balance, which the etherscan token balance confirms. In comparison, the docs for OpenMevRouter.sol show it stores value that is later collected with the `harvest()` function. If enough fee-on-transfer tokens are held by the OpenMevRouter contract, functions such as `removeLiquidityETH()` can be called instead of `removeLiquidityETHSupportingFeeOnTransferTokens()` and the function will not revert. This leads to the OpenMevRouter contract losing value due to the fees paid for the fee-on-transfer transfer.

Proof of concept

The NatSpec comment for `removeLiquidityETHSupportingFeeOnTransferTokens()` includes

```
Identical to removeLiquidityETH, but succeeds for tokens that take a fee on transfer
```

The only difference in these functions, and what is implied to cause the revert condition in `removeLiquidityETH()`, is the amount used in `safeTransfer()`. `removeLiquidityETH()` has an amount of `amountToken`, while `removeLiquidityETHSupportingFeeOnTransferTokens()` uses `ERC20(token).balanceOf(address(this)) - balanceBefore`. This does cause a revert in Uniswap's code because of the Uniswap assumption that the router holds no token balance, but OpenMevRouter can hold a token balance.

The process of value loss is:

1. Fee-on-transfer token is held by the router. This can happen either with an initial deposit by the Manifold team or from backrun arbitrage profits. The devs suggested the tokens that will be sent to the router will likely be tokens that Aave does not support flashloans for, which could include lesser known tokens with fee-on-transfer support.
2. User wants to remove liquidity from WETH-ERC20 pair where the ERC20 has a non-zero fee-on-transfer. Instead of using `removeLiquidityETHSupportingFeeOnTransferTokens()`, the user calls `removeLiquidityETH()`.
3. The code of `removeLiquidityETHSupportingFeeOnTransferTokens()` and `removeLiquidityETH()` is identical except for the amount in `ERC20(token).safeTransfer()`. The `amountToken` value used in `removeLiquidityETH()` is greater than the amount of fee-on-transfer tokens received from the `removeLiquidity()` call, so the amount transferred to the user will include some of the token balance that was held by the router before the user's remove liquidity interaction.
4. Result: The router lost value in the form of the transfer-on-fee token

Impact

High. Value can be lost from the router if the router stores fee-on-transfer tokens. While it may be unlikely for OpenMevRouter.sol to hold many fee-on-transfer tokens (note: USDT could become fee-on-transfer in the future), value loss would occur if the scenario does arise and no protections prevent against this.

Recommendation

Add a stricter check to functions not designed for fee-on-transfer tokens. For example, a rewrite of `removeLiquidityETH()` logic:

```
ensure(deadline);
address weth = WETH09;
uint256 balanceBefore = ERC20(token).balanceOf(address(this));
(amountToken, amountETH) = removeLiquidity(
    token,
    weth,
    liquidity,
    amountTokenMin,
    amountETHMin,
    address(this),
    deadline
);
if (amountToken != ERC20(token).balanceOf(address(this)) - balanceBefore) revert TokenIsF
ERC20(token).safeTransfer(to, amountToken);
IWETH(weth).withdraw(amountETH);
SafeTransferLib.safeTransferETH(to, amountETH);
```

Medium Findings

1. Medium - Failed Flashloan Arbitrage Reverts the Original Swap (Jackson)

If one of the backrun flashloan arbitrages fails to return a profit, the original swap is reverted.

Proof of concept

These lines include the revert for each flashloan [1, 2]

Impact

Medium. While this will not involve a loss of user funds, it will result it a poor user experience when user swaps are unecessarily reverted.

Recommendation

Use a try-catch when executing the flashloans such that if they revert, the entire swap is not also reverted.

Low Findings

1. Low - Edge case suboptimal arb profit (engn33r)

There can be cases where `contractAssetBalance >= optimalAmount` is not true, but using the available contractAssetBalance is still cheaper than using a flashloan with a fee. For example, if `contractAssetBalance = optimalAmount - 1`, using `contractAssetBalance` will normally produce a superior result to using a flashloan.

Proof of concept

The logic branch checks if `contractAssetBalance >= optimalAmount`, otherwise a flashloan is used https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L896

Impact

Low. This is an edge case that may be rare, but can reduce the profits of the router. Hypothetically this could be gamed by liquidity providers looking to increase yield through flashloan fees on certain assets in Aave or Kashi, because the fees are paid by OpenMevRouter arb profits.

Recommendation

When calculating the optimalAmount for the backrun process, account for the profit loss due to Aave or Kashi fees.

2. Low - Max approval granted to spender (Jackson)

Maximum approvals should be avoided, particularly when the necessary amount is known.

Proof of concept

`ERC20(token).safeApprove(spender, type(uint256).max);` in `_approveTokenIfNeeded` approves the spend to spent the entire balance.

Impact

Low. Assuming nothing problematic occurs this is not a problem. However, it is a level of protection in case of attack.

Recommendation

Only approve what is necessary for the transaction when it is known prior to granting approval.

3. Low - No Check For Aave Flashloan Balance (Jackson)

`_backrunSwaps` in `OpenMevRouter` checks that Kashi has the necessary liqudity to take a flashloan against, but does not check that Aave does as well.

Proof of concept

L915 of OpenMevRouter

https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L915

Impact

Low. It is unlikely that Aave will not have the necessary liquidity for the flashloan.

Recommendation

Check that Aave contains the necessary liquidity at the time of the flashloan as is done for Kashi.

Gas Savings Findings

1. Gas - Use `_isNonZero()` for gas savings (engn33r)

There is a gas efficient `_isNonZero()` function that is not used in two places. Otherwise, `!= 0` is preferred to `> 0` when comparing a uint to zero.

Proof of Concept

Two instances of this were found:

- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevZapper.sol#L68
- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L1151

Impact

Gas savings

Recommendation

Replace `> 0` with `!= 0` to save gas. Even better, use the existing `_isNonZero()` function in OpenMevLibrary.sol.

2. Gas - Use `_inc()` instead of ++ and `_dec()` instead of -- (engn33r)

Gas efficient functions `_inc()` and `_dec()` should be used to replace normal increments and decrements. Otherwise, if these functions were not available, use prefix is preferred to postfix for gas efficiency. In other words, use `++i` instead of `i++`.

Proof of concept

There is one instance of an increment improvement:

- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevZapper.sol#L66

There are two instances of a double decrement that could be replaced with `_dec(_decr())` or with `unchecked { length - 2; }`:

- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/libraries/OpenMevLibrary.sol#L274
- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/libraries/OpenMevLibrary.sol#L313

Impact

Gas savings

Recommendation

Increment with prefix addition and not postfix in for loops. Even better, use `_inc()` and `_dec()`.

3. Gas - Bitshifting is cheaper than multiplication or division (engn33r)

Bitshifting is cheaper than multiplication or division. Multiplication and division can be replaced by a bitshift easily when a multiple of two is involved.

Proof of concept

There are two instance of divide by 2 operations that can use bitshifting for gas efficiency:

- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevZapper.sol#L137

- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevZapper.sol#L172
- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevZapper.sol#L204
- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/libraries/Uint512.sol#L344

Impact

Gas savings

Recommendation

Replace multiplication and vision by a bitshift when a power of two is involved.

4. Gas - Unnecessary zero initialization (engn33r)

Initializing an int or uint to zero is unnecessary, because solidity defaults int/uint variables to a zero value. Removing the initialization to zero can save gas.

Proof of Concept

Several instances of this were found:

- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/libstd/OpenMevErrors.sol#L71
- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L1152

Impact

Gas savings

Recommendation

Remove the explicit uint variable initializations to zero.

5. Gas - Payable functions can save gas (engn33r)

If there is no risk of a function accidentally receiving ether, such as a function with the onlyOwner modifier, this function can use the payable modifier to save gas.

Proof of concept

The following functions have the onlyOwner modifier and can be marked as payable

- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L1141
- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/TwoStepOwnable.sol#L66
- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/TwoStepOwnable.sol#L80

Impact

Gas savings

Recommendation

Mark functions that have onlyOwner as payable for gas savings. This might not be aesthetically pleasing, but it works.

6. Gas - Avoid && logic in require statements (engn33r)

Using && logic in require statements uses more gas than using separate require statements. Dividing the logic into multiple require statements is more gas efficient.

Proof of concept

One instance of require with && logic was found:

- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/ERC20.sol#L151

Impact

Gas savings

Recommendation

Replace require statements that use && by dividing up the logic into multiple require statements.

7. Gas - Declare constant internal when possible (engn33r)

Declaring constant with internal visibility is cheaper than public constants. This is already applied to all constants in the code except one.

Proof of concept

The `bento` constant should be internal if possible:

- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L89

Impact

Gas savings

Recommendation

Make constant variables internal for gas savings.

8. Gas - Replace require with errors in OpenMevRouter (Jackson)

Two require statements can be replaced with custom errors in OpenMevRouter.

Custom errors are already used elsewhere in OpenMevRouter and are more gas-efficient than require statements: https://blog.soliditylang.org/2021/04/21/custom-errors/

Proof of concept

One in `_addLiquidity` (`require(amountAOptimal <= amountADesired);`) and another in `addLiquidityETH` (`require(IWETH(weth).transfer(pair, amountETH));` , which can be replaced with `safeTransfer` as is done in `swapExactETHForTokens`).

- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L138
- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L226

Impact

Gas savings

Recommendation

Use solidity custom errors instead of require statements.

9. Gas - Remove unused code (Jackson)

`RESERVE_SELECTOR` is not used in `OpenMevLibrary` and can be removed, neither are `_require()` or `_revert()` in `OpenMevErrors` .

Proof of concept

1. https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/libraries/OpenMevLibrary.sol#L36
2. https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/libstd/OpenMevErrors.sol#L9
3. https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/libstd/OpenMevErrors.sol#L16

Impact

Gas savings

Recommendation

Remove unused code to save gas on deployment.

10. Gas - Use simple comparison (engn33r)

Using a compound comparison such as ≥ or ≤ uses more gas than a simple comparison check like >, <, or ==. Compound comparison operators can be replaced with simple ones for gas savings.

Proof of concept

The `_addLiquidity()` function in OpenMenRouter.sol contains https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L131-L143

```
if (amountBOptimal <= amountBDesired) {
    // require(amountBOptimal >= amountBMin, 'UniswapV2Router: INSUFFICIENT_B_AMOUNT');
    if (amountBOptimal < amountBMin) revert InsufficientBAmount();
    // revert InsufficientBAmount({ available: amountBOptimal, required: amountBMin });
    (amountA, amountB) = (amountADesired, amountBOptimal);
} else {
    uint256 amountAOptimal = OpenMevLibrary.quote(amountBDesired, reserveB, reserveA);
    require(amountAOptimal <= amountADesired);
    // require(amountAOptimal >= amountAMin, 'UniswapV2Router: INSUFFICIENT_A_AMOUNT');
    if (amountAOptimal < amountAMin) revert InsufficientAAmount();
    // revert InsufficientAAmount({ available: amountAOptimal, required: amountAMin });
    (amountA, amountB) = (amountAOptimal, amountBDesired);
}
```

By switching around the if/else clauses, we can replace the compound operator with a simple one

```
if (amountBOptimal > amountBDesired) {
    uint256 amountAOptimal = OpenMevLibrary.quote(amountBDesired, reserveB, reserveA);
    require(amountAOptimal <= amountADesired);
    // require(amountAOptimal >= amountAMin, 'UniswapV2Router: INSUFFICIENT_A_AMOUNT');
    if (amountAOptimal < amountAMin) revert InsufficientAAmount();
    // revert InsufficientAAmount({ available: amountAOptimal, required: amountAMin });
    (amountA, amountB) = (amountAOptimal, amountBDesired);
} else {
    // require(amountBOptimal >= amountBMin, 'UniswapV2Router: INSUFFICIENT_B_AMOUNT');
    if (amountBOptimal < amountBMin) revert InsufficientBAmount();
    // revert InsufficientBAmount({ available: amountBOptimal, required: amountBMin });
    (amountA, amountB) = (amountADesired, amountBOptimal);
}
```

Another instance of this improvement is found with the comparison `>= 1`. Two other instances of this are in OpenMevLibrary.sol (lines 270 and 331), but to show the example from `_swapSupportingFeeOnTransferTokens()` https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L729

```
swaps[i].isBackrunable = ((1000 * amountInput) / reserveInput) >= 1;
```

Because `>= 1` equates to `> 0`, and G1 shows how `!= 0` or `_isNonZero()` is better than `> 0`, the comparison can be simplified to

```
swaps[i].isBackrunable = _isNonZero(((1000 * amountInput) / reserveInput));
```

Impact

Gas savings

Recommendation

Replace compound comparison operators with simple ones for gas savings.

11. Gas - Combine reserve value checks (engn33r)

`getAmountOut()` in OpenMevLibrary.sol checks if the reserve values with `_isZero()`. Most locations where `OpenMevLibrary.getAmountOut()` is called also use the check `if (reserve0 < 1000 || reserve1 < 1000)` before `getAmountOut()` is called. Rather than duplicating similar checks, gas could be saved by consistently checking reserve values before calling `getAmountOut()`, or requiring `reserve0 < 1000 && reserve1 < 1000` inside `getAmountOut()`.

Proof of concept

Most places where `OpenMevLibrary.getAmountOut()` in OpenMevZapper results in duplicated reserve checks.
https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevZapper.sol

Impact

Gas savings

Recommendation

Remove duplicated reserves checks to save gas

12. Gas - Use msg global vars directly (engn33r)

Using msg.sender and msg.value without caching is slightly more gas efficient than caching the value.

Proof of concept

msg.value is unnecessarily cached in:

- `addLiquidityETH()` https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L214
- `swapETHForExactTokens()` https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L666
- `swapETHAndStakeLiquidity()` https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevZapper.sol#L166

msg.value can replace swaps[0].amountIn

- `swapExactETHForTokens()` https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L564 and https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L566

Impact

Gas savings

Recommendation

Improve gas efficiency by removing the caching of msg global vars to use the global vars directly

13. Gas - Remove duplicate internal function call (engn33r)

`ensure()` gets called twice in ETH-related functions. The first call happens at the start of `addLiquidityETH()` or `removeLiquidityETH()`, and the second call happens when this function calls `addLiquidity()` or `removeLiquidity()`. However, this only helps in the case where no revert occurs, otherwise reverting earlier is better.

Proof of concept

One example First call: https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L212 Second call: https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L177

Impact

Gas savings

Recommendation

Remove the `ensure()` call at the start of the ETH-related functions in OpenMevRouter.sol.

14. Gas - deadline special case not aligned with permit (engn33r)

From EIP-2612:

```
The deadline argument can be set to uint(-1) to create Permits that effectively never exp
```

In contrast, `ensure()` implies a value of zero for a deadline that never expires

```
if (deadline < block.timestamp && _isNonZero(deadline)) revert Expired();
```

Proof of concept

EIP-2612 text https://eips.ethereum.org/EIPS/eip-2612#rationale

`ensure()` function
https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L98

Impact

Gas savings

Recommendation

Use the same permit approach as EIP-2612. This simplifies and aligns the check in `ensure()` to match Uniswap's check.

Uniswap code: https://github.com/Uniswap/v2-core/blob/8b82b04a0b9e696c0e83f8b2f00e5d7be6888c79/contracts/UniswapV2ERC20.sol#L82
`require(deadline >= block.timestamp, 'UniswapV2: EXPIRED');`

Revised OpenMevRouter.sol `ensure()` logic: `if (deadline < block.timestamp) revert Expired();`

15. Gas - Replace `pair.swap()` with `_asmSwap()` (engn33r)

One instance of `pair.swap()` has not been replaced with `_asmSwap()` for gas efficiency.

Proof of concept

https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L699

Impact

Gas savings

Recommendation

Replace all instances of `pair.swap()` with `_asmSwap()`. This may allow the swap to be moved out of `_swapSupportingFeeOnTransferTokensExecute()` and into `_swapSupportingFeeOnTransferTokens()`.

16. Gas - Remove a sortTokens call (engn33r)

`_swapSupportingFeeOnTransferTokens()` in OpenMevRouter.sol calls `sortTokens()` twice. Caching the outputs from the first call can remove the need for the 2nd call.

Proof of concept

The first `sortTokens()` call:
https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L714

The second `sortTokens()` call happens in `pairFor()`:
https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L714

Impact

Gas savings

Recommendation

Cache the outputs from the first `sortTokens()` call, then replace `OpenMevLibrary.pairFor()` with `OpenMevLibrary._asmPairFor()`.

17. Gas - Missing curly brace (engn33r)

The final if statement in `withdrawLiquidityAndSwap()` is missing curly braces. The code added in OpenMevZapper not found in Beefy is designed to save gas, but the curly braces are necessary to provide the gas savings. Otherwise the token swap always happens even if `desiredTokenOutMin` of `desiredToken` are already available to send to the user.

Proof of concept

If statement with missing curly braces
https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevZapper.sol#L107-L115

Impact

Gas savings

Recommendation

The revised code should read

```
        if (desiredTokenOutMin > ERC20(desiredToken).balanceOf(address(this))) {
            desiredSwapAmount = desiredTokenOutMin - ERC20(desiredToken).balanceOf(addres
            router.swapExactTokensForTokens(
                ERC20(swapToken).balanceOf(address(this)),
                desiredSwapAmount,
                path,
                address(this),
                block.timestamp
            );
        }
```

18. Gas - More efficient variable swap (engn33r)

`_arb()` has an inefficient approach to swapping variables around.

https://blog.oliverjumpertz.dev/solidity-quick-tip-efficiently-swap-two-variables

Proof of concept

The original variables in line 1114
https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevZapper.sol#L1114

The inefficiently swapped variables in line 1128
https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevZapper.sol#L1128

Impact

Gas savings

Recommendation

Swap the variables around in line 1128 like this:

```
(amount0Out, amount1Out) = (amount1Out, amount0Out);
```

19. Gas - Reduce number of swaps (engn33r)

There are three steps in the swap and arb process. The steps are: 1. Perform the user swap with factory0 2. Perform arb with a swap in the opposite direction with optimalAmount on factory0 3. Continue the arb with a swap in the initial direction on factory1. The first two steps (swap and arb on the same factory liquidity pool) can be combined because the 2nd step is effectively reversing a part of the first step. Because the end goal is to remove a price differential between the Uniswap and SushiSwap pools, this can be achieved by splitting the initial user swap between the Uniswap and SushiSwap pools to optimize the overall exchange rate rather than by arbing a larger swap that happens in a single LP. The profit for OpenMevRouter can be taken from the improved exchange rate (returning the user tokens based on the exchange rate if the swap happened in only one LP) rather than taking profit from the arb.

Proof of concept

Consider the constant product diagram Constant Product Swaps

Point 1 shows the liquidity pool amounts before OpenMevRouter interaction, point 2 shows the amounts after the OpenMevRouter user swap, and point 3 shows the amounts after the first backrun of the arb process. These two steps can be combined to arrive from point 1 to point 3, skipping to need to swap to arrive at point 2. The math in OpenMevRouter.sol would need changing, but gas savings from removing one swap may be enough to reduce overall gas consumption.

Impact

Gas savings

Recommendation

Remove a swap by combining the user swap and the first step of the backrun that reverse the user swap by exchanging output token to input token.

Informational Findings

1. Informational - OpenMevRouter should inherit from IFlashBorrower and IOpenMevRouter (Jackson)

OpenMevRouter should also inherit from IFlashBorrower and IOpenMevRouter aside from TwoStepOwnable.

Impact

Type safety.

2. Informational - The ETHERSCAN_API key is present in plaintext (Jackson)

`ETHERSCAN_API` is present in plaintext in test_Swaps.py

Impact

Malicious use of your Etherscan API key.

3. Informational - SafeTransferLib does not match Solmate's main branch (Jackson)

The SafeTransferLib does not match Solmate's latest implementation. Consider whether an update would be useful or save gas.

Impact

Possible gas savings.

4. Informational - Incorrect comment (engn33r, Jackson)

A comment in OpenMevRouter.sol has an extra function argument that doesn't exist in the code.

Elsewhere, in `addLiquidityETH()`

Proof of concept

The comment on line 1001 doesn't match the code in line 1002 https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L1001-L1002

Impact

Informational

Recommendation

Remove the extra function argument

5. Informational - Replace magic numbers with constants (engn33r)

Constant variables should be used in place of magic numbers to prevent typos. For one example, the magic number 1000 is found in multiple places in OpenMevRouter.sol and should be replaced with a constant. Using a constant also adds a description using the variable name to explain what this value is for.

Proof of concept

There are many instances of the value 1000. Consider replacing this magic number with a constant internal variable named MINIMUM_LIQUIDITY like Uniswap does:

- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L726

- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L729
- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevZapper.sol#L57-L58
- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevZapper.sol#L136
- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevZapper.sol#L171

Impact

Informational

Recommendation

Use constant variables instead of magic numbers

6. Informational - Typos (engn33r)

`balanaceToDistribute` might be better named `balanceToDistribute`. `isBackrunable` might be better named `isBackrunnable`.

Proof of concept

- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L1147
- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/libraries/OpenMevLibrary.sol#L28

Impact

Informational

Recommendation

Fix typos

7. Informational - Hard coded Aave token list (engn33r)

Aave can modify their list of supported tokens that support flashloans. The `aaveList()` function in OpenMevLibrary.sol stores a hard coded list of these tokens, meaning OpenMevRouter does not support a way of updating its internal list of tokens supporting Aave flashloans.

The list in the contract does match the list of supported Aave tokens at the time of this review: https://aave.github.io/aave-addresses/mainnet.json

Proof of concept

The hard coded list of tokens in OpenMevLibrary.sol https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/libraries/OpenMevLibrary.sol#L343

Impact

Informational

Recommendation

Store Aave token addresses in a state variable that has a setter function with the onlyOwner modifier to enable changes to the Aave token list.

8. Informational - Inconsistency in WETH transfers (engn33r)

There is one inconsistent instance of WETH transfer. Consider using a consistent approach for gas savings and code simplification.

Proof of concept

The one instance of a WETH transfer with `require(IWETH(weth).transfer(pair, amount));`

- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L226

All other instances use `IWETH(weth).deposit{ value: amount }();`

- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L225
- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L564
- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L679

Impact

Informational

Recommendation

Use consistent WETH transfer approach.

9. Informational - safeApprove vulnerable to double withdraw (engn33r)

Using `approve()` or `safeApprove()` adds the risk of a double withdrawal: https://docs.openzeppelin.com/contracts/4.x/api/token/erc20#IERC20-approve-address-uint256-

The same race condition applies to `permit()`: https://eips.ethereum.org/EIPS/eip-2612#security-considerations

Furthermore, the `safeApprove()` function is deprecated per OpenZeppelin docs: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/57725120581e27ec469e1c7e497a4008aafff818/contracts/token/ERC20/utils/SafeERC20.sol#L39-L58

Proof of concept

One relevant `safeApprove()` call was found:

- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L1043

Permit is used in several functions in OpenMevRouter.sol:

- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L327
- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L366
- https://github.com/manifoldfinance/OpenMevRouter/blob/8648277c0a89d0091f959948682543bdcf0c280b/contracts/OpenMevRouter.sol#L428

Impact

Informational. This has not been shown to be a notable problem on mainnet, but better solutions do exist.

Recommendation

Use `safeIncreaseAllowance()` or `safeDecreaseAllowance()` instead of `safeApprove()`.

10. Informational - Same frontrunning weaknesses as Uniswap/SushiSwap (engn33r)

While the description of this protection is to prevent MEV extraction with a specific form of MEV, there is no protection for other forms of MEV. This is acknowledge by the devs in project documentation, with acknowledgement that Uniswap does not protect against this eiher. Attack vectors such as frontrunning or an uncle bandit attack can extract value from transactions that swap with OpenMevRouter.sol because only backrun arbitrage MEV protection is built into the OpenMevRouter design.

Proof of concept

Project documentation explaining these attack vectors still remain: https://github.com/manifoldfinance/OpenMevRouter/wiki/V01-Router-Spec-Doc#front-running-and-transaction-reordering

Impact

Informational

Recommendation

Clarify user documentation to make it clear that `amountOutMin` or a similarly named function argument is still an important slippage setting in OpenMevRouter.sol and OpenMevZapper.sol.

Final remarks

engn33r

Jackson

About yAcademy

yAcademy is an ecosystem initiative started by Yearn Finance and its ecosystem partners to bootstrap sustainable and collaborative blockchain security reviews and to nurture aspiring security talent. yAcademy includes a fellowship program and a residents program. In the fellowship program, fellows perform a series of periodic security reviews and presentations during the program. Residents are past fellows who continue to gain experience by performing security reviews of contracts submitted to yAcademy for review (such as this contract).

Appendix and FAQ