yAudit Sickle Review

Review Resources:

- None beyond the code repositories

Auditors:

- Jackson
- Drastic Watermelon

Table of Contents

Review Summary

Sickle

Sickle provides an extensible on-chain account. This on-chain account uses protocol-provided strategies to perform various on-chain functions, such as farming, sweeping tokens, or taking flash loans. The strategies themselves use shared modules and connectors to interact with defi protocols. This audit also included a new governance system, although there are no immediate plans to use it at this time.

The contracts of the Sickle Repo were reviewed over 15 days. Two auditors performed the code review between April 15, 2024 and May 3, 2024. 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 4f21de8cc1269dfbc05a3eeb5e792ec82012f11d for the `sickle-contracts` repo.

Scope

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

```
.
├── Compounder.sol
├── ConnectorRegistry.sol
├── Sickle.sol
├── SickleFactory.sol
├── SickleRegistry.sol
├── base
│   ├── Admin.sol
│   ├── Multicall.sol
│   ├── NonDelegateMulticall.sol
│   └── SickleStorage.sol
├── connectors
│   ├── AerodromeGaugeConnector.sol
│   ├── AerodromeRouterConnector.sol
│   ├── CompoundMarketConnector.sol
│   ├── ComptrollerConnector.sol
│   ├── ComptrollerRegistry.sol
│   ├── Curve2TokenConnector.sol
│   ├── Curve2TokenConnectorInt128.sol
│   ├── CurveDepositor3Connector.sol
│   ├── EqualizerGaugeConnector.sol
│   ├── EqualizerRouterConnector.sol
│   ├── GaugeRegistry.sol
│   ├── MasterChefConnector.sol
│   ├── MasterChefWithReferrerConnector.sol
│   ├── MasterchefV3Connector.sol
│   ├── ParaswapConnector.sol
│   ├── PrismaConnector.sol
│   ├── UniswapV2Connector.sol
│   ├── UniswapV3Connector.sol
│   ├── UniswapV3PoolRegistry.sol
│   └── VelocoreConnector.sol
├── governance
│   ├── SickleGovernor.sol
│   └── SickleVote.sol
```

```
├── interfaces
│   ├── IFarmConnector.sol
│   ├── ILendingConnector.sol
│   ├── ILiquidityConnector.sol
│   └── external
│       ├── IERC20PermitAllowed.sol
│       ├── IMasterChef.sol
│       ├── IMasterchefV3.sol
│       ├── IWETH.sol
│       ├── aerodrome
│       │   ├── IGauge.sol
│       │   ├── IPool.sol
│       │   ├── IPoolFactory.sol
│       │   ├── IRouter.sol
│       │   └── IVoter.sol
│       ├── compound-v2
│       │   ├── CTokenInterfaces.sol
│       │   ├── ComptrollerInterface.sol
│       │   ├── ComptrollerStorage.sol
│       │   ├── EIP20NonStandardInterface.sol
│       │   ├── ErrorReporter.sol
│       │   ├── IRewardDistributor.sol
│       │   └── InterestRateModel.sol
│       ├── equalizer
│       │   ├── IEqualizerGauge.sol
│       │   └── IEqualizerRouter.sol
│       ├── flashloans
│       │   ├── IBalancerVault.sol
│       │   ├── IFlashLoanReceiver.sol
│       │   ├── ILendingPoolV2.sol
│       │   └── IPoolV3.sol
│       ├── uniswap
│       │   ├── INonfungiblePositionManager.sol
│       │   ├── ISwapRouter.sol
│       │   ├── IUniswapV2Factory.sol
│       │   ├── IUniswapV2Pair.sol
```

```
│       │   ├── IUniswapV2Router02.sol
│       │   ├── IUniswapV3Factory.sol
│       │   └── IUniswapV3Pool.sol
│       └── velocore
│           └── IVelocoreLens.sol
├── libraries
│   └── FeesLib.sol
├── strategies
│   ├── FarmStrategy.sol
│   ├── FlashloanStrategy.sol
│   ├── MigrationStrategy.sol
│   ├── NftFarmStrategy.sol
│   ├── SweepStrategy.sol
│   └── modules
│       ├── AccessControlModule.sol
│       ├── DelegateModule.sol
│       ├── FeesModule.sol
│       ├── MsgValueModule.sol
│       ├── NftTransferModule.sol
│       ├── SwapModule.sol
│       ├── TransferModule.sol
│       └── ZapModule.sol
└── test
    ├── Target.sol
    ├── TestERC20.sol
    └── TestMasterChef.sol
```

After the findings were presented to the Sickle 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, vfat, and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	One of the challenges in attacking the Sickle protocol is that the Sickles are partitioned by user and only whitelisted strategies can multi-call into the Sickles. These whitelisted strategies can only interact with whitelisted connectors.
Mathematics	Good	There are very few complex mathematical operations in the code reviewed.
Complexity	Medium	There is complexity around the number of protocols that Sickles are expected to interact with and the interaction between Sickles, Strategies, and Connectors.
Libraries	Good	The Sickle project makes good use of well-known libraries.
Decentralization	Medium	When a Sickle is deployed, it is largely owned by a user. However, the list of Strategies and Connectors allowed to interact with a Sickle is controlled entirely by the protocol team. At some point in the future, this whitelisting privilege may be transferred to a contract with on-chain voting capabilities.
Code stability	Medium	The repository was actively developed during the review. A follow-up audit was scheduled for new code developed by the team during this audit.
Documentation	Low	No documentation was provided besides the README in the repo. However, some of the functions are documented using NatSpec.
Monitoring	Low	Few of the functions emit events.

Category	Mark	Description
Testing and verification	Good	The project has 95% line and function coverage and 73% branch coverage. In addition, many of the test functions are fuzzed.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low Impact
  - These findings 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

1. High - `MasterchefV3Connector` deposits can be blocked

Technical Details

In `MasterchefV3Connector`'s `deposit()` there is a check if the balance of the sickle is equal to 1, and if not, a revert occurs. This can be DOS'd by minting NFTs to the sickle address such that a sickle can never deposit.

See the following POC using `PancakeV3.t.sol` for setup:

```
function test_brick_deposit() public {
        // a random address w/ a lot of usdc and usdt
        address usdc_whale = 0x40ec5B33f54e0E8A33A975908C5BA1c14e5BbbDf;
        vm.startPrank(usdc_whale);
        // usdc but could be any token

IERC20(0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48).approve(address(Pancake.NFT_MANAGER),
type(uint256).max);
        // usdt but could be any token

SafeERC20.safeIncreaseAllowance(IERC20(0xdAC17F958D2ee523a2206206994597C13D831ec7),
address(Pancake.NFT_MANAGER), type(uint256).max);

        INonfungiblePositionManager.MintParams memory params =
INonfungiblePositionManager.MintParams({
            token0: address(0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48),
            token1: address(0xdAC17F958D2ee523a2206206994597C13D831ec7),
            fee: 100,
            tickLower: -9,
            tickUpper: 12,
            amount0Desired: 6177807, // can be smaller than this, some random amount
taken from an on-chain tx.
            amount1Desired: 7399999,
            amount0Min: 0,
            amount1Min: 0,
            recipient: address(ctx.sickle), // mint to the sickle so it can't be
deposited to anymore
            deadline: block.timestamp + 60
        });

        INonfungiblePositionManager(Pancake.NFT_MANAGER).mint(params);

        vm.stopPrank();

        // reverts w/ SickleNftBalanceError()
```

```
        test_deposit_DepositsUsingETH(1 ether);
}
```

Impact

High. Sickles can be blocked from depositing into MasterchefV3 contracts.

Recommendation

Remove the revert and allow the user to supply the `index` and `tokenId` of the token they want to deposit using the `extraData`.

Developer Response

This was a known issue but considered low severity because attackers have no incentive to perform this attack: NFTs can be swept from Sickles, and new amended connectors can be deployed if there are grief attacks on the old ones.

That said, the proposed solution made sense, we aren't able to supply the tokenId but can supply the user's previous balance of NFTs, which is also going to be the `index` of the next NFT. Typically a user has 0 NFTs, so the index of the first NFT is 0.

This has been implemented in this commit https://github.com/vfat-io/sickle-contracts/commit/11b09b69f9896f4008a5357fbbc72ecedf938d26

Medium Findings

1. Medium - Fees can be bypassed by passing a token with no balance as a `tokenIn` or `tokenOut`

Technical Details

When fees are charged in `_sickle_charge_fees()`, the balance of `tokenOut` is checked as the `feeBasis`. However, in various strategies such as `FarmStrategy`'s `compound()` when no zap is needed, the `tokenIn` or `tokenOut` is not used. This means that any value can be used here. If a `tokenOut` with no balance is supplied, the `feeBasis` will be 0, avoiding fees.

See the following POC, which is `AerodromeStrategy.t.sol`'s `test_compound_CompoundsIntoSamePool()` test, replacing the `tokenIn` with `0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE` and only compounding once.

```
function test_feeBypassWithETH() public {
        uint256 amountIn = 1 ether;

        test_deposit_DepositsUsingETH(amountIn);

        vm.prank(Org.COLLECTOR);
        IERC20(Base.VAMM_WETH_DAI).approve(
            address(ctx.collectorSickle), type(uint256).max
        );

        vm.startPrank(sickleOwner);

        vm.roll(6_422_188 + 60_000);
        vm.warp(block.timestamp + 120_000); // AERO rewards are calculated based
        // on block.timestamp

        uint256 balanceBefore =
            IGauge(Base.WETH_DAI_GAUGE).balanceOf(address(ctx.sickle));

        uint256 earned = IGauge(Base.WETH_DAI_GAUGE).earned(address(ctx.sickle));

        SwapData[] memory swaps = new SwapData[](2);
        swaps[0] =
            _getSwapData(Base.AERO, Base.WETH, false, earned * 9910 / 10_000);

        uint256 intermediateEstimate = _estimateIntermediateAmount(
            Base.VAMM_AERO_WETH, Base.AERO, earned * 9910 / 10_000
        );

        uint256 swapInEstimate = _estimateSwapInAmount(
            Base.VAMM_WETH_DAI, false, Base.WETH, intermediateEstimate
        );

        swaps[1] = _getSwapData(Base.WETH, Base.DAI, false, swapInEstimate);
```

```
        ZapModule.ZapInData memory zapData = ZapModule.ZapInData({
            tokenIn: 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE,
            amountIn: earned,
            swaps: swaps,
            addLiquidityData: _getAerodromeAddLiquidityData(
                Base.VAMM_WETH_DAI, Base.WETH, Base.DAI, false
            )
        });

        uint256 collectorBalanceBeforeCompoundFee = ctx.registry.collector().balance;

        farmStrategy.compound(
            address(ctx.sickle),
            Base.WETH_DAI_GAUGE,
            Base.WETH_DAI_GAUGE,
            zapData,
            "",
            ""
        );

        uint256 balanceAfter =
            IGauge(Base.WETH_DAI_GAUGE).balanceOf(address(ctx.sickle));

        // the user's balance increased as expected
        assertGt(balanceAfter, balanceBefore);
        uint256 collectorBalanceAfterCompoundFee = ctx.registry.collector().balance;

        // the collector's balance remains unchanged.  No fee has been paid
        assertEq(
            collectorBalanceAfterCompoundFee, collectorBalanceBeforeCompoundFee
        );
}
```

Impact

Medium

Recommendation

Pass the token used in whatever operation is performed to `_sickle_charge_fees()` rather than always `tokenIn` or `tokenOut`.

Developer Response

We are aware of these vectors but there is no straightforward mitigation, for example the contracts are typically not aware of the output tokens of any zaps as we use opaque aggregators. For the time being we are taking the idealistic approach that we are offering a valuable service to users and our fees are fairly priced. In the future if we notice abuse of this capability we can look at possible strategies to prohibit or diminish it.

Low Findings

1. Low - Wrapped native tokens may stay in sickle after `compoundFor()`.

Technical Details

`compoundFor()` is intended to be called by someone other than the sickle owner. To compensate the caller for their gas used, a `feeZapData` is passed in. This data is used to swap into the wrapped native token of the network, unwrap this wrapped native token, and return the value to `msg.sender`. However, `estimateCost` is passed here, rather than the sickle's balance of the wrapped native token. Suppose a wrapped native token balance greater than `estimateCost` is received during the swap. In that case, this excess wrapped native balance may be stuck in the sickle, depending on whether the wrapped native token is in the addLiquidityData.tokens list or not.

Impact

Low. If this occurs, the user can recover the wrapped native tokens using the `SweepStrategy`.

Recommendation

Sweep any leftover wrapped native tokens back to the user as is done for any other dust left in the sickle at the end of `compoundFor()`.

Developer Response

This has been refactored in main with the addition of a sweepTokens array to the call params: https://github.com/vfat-io/sickle-

contracts/blob/main/contracts/strategies/FarmStrategy.sol#L147 https://github.com/vfat-io/sickle-contracts/commit/82e74dffaebf49605bffbf6e8b7491c85a489f37#diff-284157fe21efb2c4e53c03b657f7fcc839c90f2ccf1cacd31b192a2cc669f112R44

2. Low - Sickles may execute arbitrary calls

A Sickle can be made to execute an arbitrary external call to any target with any calldata.

Technical Details

By leveraging `ParaswapConnector.swapExactTokensForTokens`, a Sickle owner can execute any arbitrary call by calling into `FarmStrategy.harvest()` with a carefully constructed payload. In particular, such a payload must contain a `zapData.swaps` array that encodes the parameters of the to-be-executed arbitrary call.

Following is a PoC of how a Sickle can be made to deploy another Sickle, becoming its admin: gist

Impact

Low.

Recommendation

Impose strict checks on the call executed by `ParaswapConnector`, whitelisting calls to their Augustus Swapper contract.

Developer Response

This has been refactored in main with the addition of an immutable router address https://github.com/vfat-io/sickle-contracts/blob/main/contracts/connectors/AggregatorConnector.sol#L14 https://github.com/vfat-io/sickle-contracts/commit/e33f97070418ad9b2e84e4698bf6ae8a787441da

3. Low - ZapModule fails to verify value consistency among redundant fields

`ZapModule.zapIn()` and `ZapModule.zapOut()` doesn't enforce some consistency requirements between redundant fields.

Technical Details

`ZapModule.zapIn()` and `ZapModule.zapOut()` respectively accept a `ZapInData` and `ZapOutData` struct as an argument. Both methods fail to verify that `zapData.amountIn == zapData.swaps[0].amountIn`. Furthermore, they lack checks to verify that `tokenIn` or `tokenOut` matches the first or last token of the swap path specified by `zapData.swaps`.

Notice that this issue, in combination with the fee-bypass issue, enables users to bypass entirely any zap fee charged by protocol, as they can forward the tokens with an external transfer and get `ZapModule` to swap those tokens, while not enforcing that `tokenIn` or `tokenOut` match the swap origin or destination asset.

Impact

Low.

Recommendation

Eliminate redundant fields from the cited structs or enforce them to have consistent values.

Developer Response

We have refactored this as well in main, probably best for it to be reviewed in the addendum audit as most structs have been changed. For instance zapIn/zapOutData no longer have tokenIn/amountIn tokenOut/amountOut. https://github.com/vfat-io/sickle-contracts/blob/main/contracts/strategies/modules/ZapModule.sol#L12 https://github.com/vfat-io/sickle-contracts/commit/3f97a12ef813f98c0803f0c5401302356a9f3b81

Gas Saving Findings

1. Gas - Cache array length when iterating over it

When iterating over a storage array, caching its length in a variable is less expensive than accessing it from storage.

Technical Details

Given that caching the array length in a variable avoids having to access the contract's storage at each iteration, the gas cost of a `for` loop can be noticeably reduced, as can be seen in the following PoC

This optimization can be implemented at:

- ConnectorRegistry.sol#L38
- ConnectorRegistry.sol#L71
- ConnectorRegistry.sol#L95

Impact

Gas savings.

Recommendation

Implement the suggested optimization.

Developer Response

Amended here https://github.com/vfat-io/sickle-contracts/pull/210/commits/2124ed0f5819f175eb39d76f94f739bddad3eb86

2. Gas - Store custom registries as a map

Technical Details

In `ConnectorRegistry`, custom connectors are added by pushing to the `customRegistries` list. Later, when `isCustomRegistry()` is called, the `customRegistries` list is iterated to check whether a given target is in the `customRegistries` list. If the target was also stored as a mapping, this lookup could be done in constant time.

Impact

Gas.

Recommendation

When adding a custom registry, store it as a mapping for fast look-up.

Developer Response

Amended here https://github.com/vfat-io/sickle-contracts/pull/210/commits/2124ed0f5819f175eb39d76f94f739bddad3eb86

3. Gas - Avoid writing default values

Explicitly writing default values to variables consumes unnecessary gas.

Technical Details

Writing a default value to a variable is generally avoidable to save writes to memory/storage.

Avoid writing the default value in the following code locations:

- FlashloanStrategy.sol#L115

Impact

Gas savings.

Recommendation

Implement the suggested changes.

Developer Response

Acknowledged but I think this looks better being explicit when its an enum, plus being in the constructor it is a one off cost.

Informational Findings

1. Informational - Missing event emissions

Important state changes should be followed by an event emission to improve the ability to monitor and analyze a contract's state through time.

Technical Details

- `Compounder.setApprovedCompounder()` should emit an event.
- `Compounder.compoundFor()` could emit an event upon a successful call.

Impact

Informational.

Recommendation

Add suggested event logs.

Developer Response

Added here https://github.com/vfat-io/sickle-contracts/commit/aa8dd66d92eed6c265e9d356172b88b0b88411b1

2. Informational - Unused imports

Unused imports may be removed to reduce contract size and reduce a contract's deployment cost.

Technical Details

The following imports aren't used:

- SickleStorage.sol#L6
- AerodromeRouterConnector.sol#L6
- ComptrollerConnector.sol#L6
- EqualizerRouterConnector.sol#L6
- MasterChefWithReferrerConnector.sol#L5
- FarmStrategy.sol#L9
- FeesLib.sol#L5
- SickleGovernor.sol#12

Impact

Informational.

Recommendation

Remove unused imports.

Developer Response

Removed here https://github.com/vfat-io/sickle-contracts/commit/0733fc270e62634823d859721121e98fd323d182

3. Informational - Upgrade OpenZeppelin dependency

The OpenZeppelin contracts dependency is on version 4.8.3, which is outdated. Consider updating to a newer version.

Technical Details

OpenZeppelin library v4.8.3 is not the latest version available. Consider upgrading to v4.9.6, which fixes some issues in v4.8.3. In particular, v4.9.4 mitigates the Multicall + ERC2771 issue.

Impact

Informational.

Recommendation

Update the OpenZeppelin dependency to include the latest fixes to their contracts.

Developer Response

Addressed in https://github.com/vfat-io/sickle-contracts/pull/213

4. Informational - Use `msg.sender` to retrieve sickle in `FarmStrategy`'s `compound()`

Technical Details

The paradigm in the `FarmStrategy` is to use `msg.sender` to retrieve the user's sickle from the factory before interacting with it, except for `compoundFor()` which is intended to be called by someone other than the sickle's owner.

```
Sickle sickle = Sickle(payable(factory.sickles(msg.sender)));
```

This paradigm is not followed in `compound()` even though it is intended to be called by the sickle's owner, unlike `compoundFor()`.

Impact

Informational.

Recommendation

For consistency's sake, use the same paradigm as the other functions in FarmStrategy that retrieve the user's sickle from the factory.

Developer Response

Removed here https://github.com/vfat-io/sickle-contracts/commit/dd2ceeec072e8714a3594c4ff0e6c19d8856c76c#diff-284157fe21efb2c4e53c03b657f7fcc839c90f2ccf1cacd31b192a2cc669f112R138

5. Informational - `_transferAssets()` can be used in place of loops in `FlashloanStrategy`

Technical Details

In `FlashloanStrategy`'s `_uniswapCallback()` a `_transferAssets()` function is used to transfer assets to the sickle, and back to the flash loan provider.

This function can be re-used in any flash loan callback that involves the transfer of multiple assets to the sickle and back to the flash loan provider, such as `executeOperation()`

Impact

Informational.

Recommendation

Re-use the existing `_transferAssets()` function rather than repeating code.

Developer Response

Amended here https://github.com/vfat-io/sickle-contracts/commit/0cec69d80dbecfa27ff81a0736cc9590432fac0c

6. Informational - Curve token connectors can have their swaps sandwiched

Technical Details

As part of the `swapExactTokensForTokens()` implementation of the two Curve connectors, if the balance of the sickle is less than the `swapData.amountIn`, a new `minAmountOut` is calculated as follows:

```
swapData.minAmountOut = swapData.minAmountOut * balance / swapData.amountIn * 999 /
1000;
```

Depending on the values of `swapData.minAmountOut`, `balance`, and `swapData.amountIn`, this could round down 0, leaving the swap with no slippage protection.

Impact

Informational, since according to the team, the Curve connectors are only used for testing.

Recommendation

If the newly calculated `minAmountOut` rounds down to 0, revert.

Developer Response

Acknowledged but no action required as these swaps were added for testing purposes, production swaps use aggregators.

7. Informational - Prisma rewards may get stuck

Rewards accrued by a Sickle in Prisma can get temporarily stuck.

Technical Details

Given that `PrismaConnector.claim()` attempts to call `TokenLocker.withdrawWithPenalty()` if the user has accrued more than `1e18 PRISMA` and that Prisma's TokenLocker.sol may disable penalty withdrawals voluntarily, some users may have their `CRV` and `PRISMA` rewards locked temporarily until the Prisma team enables penalty withdrawals again, or requiring the Sickle team to deploy a new connector that avoids such call altogether.

Impact

Informational. Rare cases will lead to momentarily locked rewards.

Recommendation

Enable the users to select whether the `locker.withdrawWithPenalty()` call should be executed or not based on an input flag they provide.

Developer Response

Acknowledged but no action required for the reason stated (momentary nature of the issue).

Final Remarks

The architecture of Sickles makes it challenging to steal funds from users, as Sickles utilize the minimum amount of storage needed while also minimizing the funds held by the Sickle itself. However, a few attack vectors exist to DOS a Sickle or avoid paying the protocol fees. In general, the team put thought into protecting users and wrote an adequate test suite, also utilizing fuzz tests.

The `CompoundV2Migrator.sol` and `CompoundV2Strategy.sol` files were included in the repo at the commit reviewed. However, after the audit started, these files were deemed out of scope, and thus, any issues found in these contracts were omitted from this report.
