yAudit Sickle Strategies Review

Review Resources:

- None beyond the code repositories.

Auditors:

- Invader-tak
- Devtooligan

Review Summary

Sickle

This review of Sickle protocol focused on the introduction of several new strategies to be used by the Sickles, including FarmStrategy, NftFarmStrategy, and MultiFarmStrategy, with supporting contracts and automation.

The contracts of the Sickle Protocol repo were reviewed over 15 days. Two auditors performed the code review between September 23rd, 2024, and October 11th, 2024. The repository was not under active development during the review, and the review was limited to the latest commit 96262fdfa694775b97a4e9407168df5b34a00429 for the Sickle repo.

In addition, commit 14cd3781a4e2aa3552e9d8a6d0230af57610409c was added to the review's scope. This commit reduced code complexity in FarmStrategy and NftFarmStrategy by streamlining deposit and withdrawal flows and moving reusable modifiers to transferLib.

All these changes are reflected in commit 899e7aaff58320f01f4aa5a9f906d0e41599a085 of Sickle's public repo.

Table of Contents

Scope

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

```
├── contracts
│   ├── Automation.sol
│   ├── NftSettingsRegistry.sol
│   ├── PositionSettingsRegistry.sol
│   ├── libraries
│   │   ├── FeesLib.sol
│   │   ├── NftSettingsLib.sol
│   │   ├── NftTransferLib.sol
│   │   ├── NftZapLib.sol
│   │   ├── PositionSettingsLib.sol
│   │   ├── SwapLib.sol
│   │   ├── TransferLib.sol
│   ├── strategies
│   │   ├── FarmStrategy.sol
│   │   ├── MultiFarmStrategy.sol
│   │   ├── NftFarmStrategy.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, Sickle and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Automation and registries implement sufficient access control to ensure that actors are either administrators, approved contracts, or a user-specific sickle
Mathematics	Good	The contracts utilize mathematical operations effectively, especially in liquidity and token management functions. They handle edge cases well, such as ensuring valid ranges for ticks.
Complexity	Average	The contracts are moderately complex due to the interactions between multiple components, including connectors with external contracts. While the structure is logical, parts are challenging to reason about without knowledge of the integrations.
Libraries	Good	Good use of industry-standard libraries such as OZ and Solmate. Additionally, the team has created library-like contracts, like FeesLib, NftSettingsLib, etc., for constructing calls. This enhances modularity and code reuse. This is a strong point, as it promotes cleaner code and reduces redundancy.
Decentralization	Average	The contracts allow for some level of decentralization through admin roles and potential multisig implementations. However, the functionality of many of the strategies and the sickle's interaction with them depends on complex off-chain calculations and components; it would be difficult for a user to interact with them without assistance. Vfat provides open-source tooling for users to interact with the contracts through a simplified interface to mitigate this. Approved automators are also controlled by the protocol admins, adding a layer of centralization to the automation.
Code stability	Good	The contracts are well-structured and follow best practices, contributing to their stability.

Category	Mark	Description
Documentation	Low	While there are comments and natspec for some contracts, the documentation could be improved to provide clearer guidance on the purpose and usage of each function and contract.
Monitoring	Average	While there generally was good event coverage, there were functions that were missing relevant events
Testing and verification	Good	The contracts have an extensive testing framework in place, and the team is working on expanding their test coverage.

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.

Low Findings

1. Low - Configuration values not validated when updating position settings

The setPositionSettings Sickle function does not validate the values of the `settings` argument.

And the setPositionSettings function used to set reward behavior does not validate `rewardConfig`.

Technical Details

The function does not contain the checkConfigValues modifier like the related setPositionSettings owner function does.

This modifier checks the new settings values for misconfigurations as well as limits on slippage and price impact.

```
    modifier checkConfigValues(
        PositionSettings memory settings
    ) {
        if (
            settings.rewardConfig.rewardBehavior != RewardBehavior.Harvest
                && settings.rewardConfig.harvestTokenOut != address(0)
        ) {
            revert InvalidTokenOut();
        }
        if (!settings.autoExit) {
            if (
                settings.exitConfig.triggerPriceLow != 0
                    || settings.exitConfig.triggerPriceHigh != 0
                    || settings.exitConfig.exitTokenOutLow != address(0)
                    || settings.exitConfig.exitTokenOutHigh != address(0)
                    || settings.exitConfig.slippageBP != 0
                    || settings.exitConfig.priceImpactBP != 0
            ) {
                revert AutoExitNotSet();
            }
        } else {
            if (
                settings.exitConfig.triggerPriceLow == 0
                    && settings.exitConfig.triggerPriceHigh == 0
            ) {
                revert ExitTriggersNotSet();
            }
            if (settings.exitConfig.slippageBP > MAX_SLIPPAGE_BP) {
                revert InvalidSlippageBP();
            }
            if (
                settings.exitConfig.priceImpactBP > MAX_PRICE_IMPACT_BP
                    || settings.exitConfig.priceImpactBP == 0
            ) {
                revert InvalidPriceImpactBP();
```

```
            }
        }
        _;
    }
```

Impact

At worst, when a high value is set for slippage and/or price impact, swaps are susceptible to front-running and can lose significant value. However, the likelihood of this happening is low because the owner would have to make a mistake when setting the values.

Recommendation

Always perform the `checkConfigValues` when updating the settings via the Sickle or owner.

Validate the `rewardConfig` argument of the setPositionSettings function used for setting reward behavior with the related logic found in `checkConfigValues`.

Developer Response

Acknowledged - Fixed in #282

2. Low - FarmStrategy functions lack enforcement of sweep tokens

The FarmStrategy functions do not check if the `sweepTokens` array argument is empty.

Technical Details

The following functions do not contain the `sweepTokensRequired` modifier that their NftStrategy counterparts do have:

- deposit
- increase
- compound
- withdraw
- harvest
- simpleHarvest
- simpleExit
- depositTwoTokens
- increaseTwoTokens
- exit (for both `harvestSweepTokens` and `withdrawSweepTokens`)

Impact

Without the check, tokens may be leftover in the Sickle.

The project team has stated:

> The sweepTokens array should always be used in order to ensure there are no leftover tokens in the Sickle at the end of each action.

Recommendation

Add the missing modifier to the functions above.

Developer Response

Acknowledged - Fixed in #290

Gas Saving Findings

1. Gas - Do zero address check before loading variables

Transfer lib loads the sickle before checking that the transfer token is valid.

Technical Details

Loading of sickle is unnecessary when the Zero address check fails.

```
    function transferTokenToUser(
        address token
    ) public payable {
        address recipient = Sickle(payable(address(this))).owner();
        if (token == address(0)) {
            return;
        }
        ...
```

Impact

Gas - minor gas saving

Recommendation

Perform checks that allow a call to fail early to save gas

Developer Response

Acknowledged - Fixed here vfat-io/sickle-contracts#286

2. Gas - Remove redundant operations

Redundant operations can be refactored to save gas in _check_rebalance_config:

```
        if (config.cutoffTickLow < MIN_TICK || config.cutoffTickLow > MAX_TICK)
        {
            revert InvalidMinTickLow();
        }
        if (
            config.cutoffTickHigh < MIN_TICK || config.cutoffTickHigh > MAX_TICK
        ) {
            revert InvalidMaxTickHigh();
        }
        if (config.cutoffTickLow >= config.cutoffTickHigh) {
            revert InvalidMinMaxTickRange();
        }
```

Impact

Gas - minor gas saving

Recommendation

Consider applying logic similar to:

```
if (config.cutoffTickLow < MIN_TICK) revert...
if (config.cutoffTickLow >= config.cutoffTickHigh) revert...
if (config.cutoffTickHigh > MAX_TICK) revert...
```

Developer Response

Acknowledged - Fixed in #282

3. Gas - Initialize memory variables outside of loops

In setPositionSettings, the `key` memory variable is re-initialized repeatedly in a loop.

Impact

This extends memory every time a new key is assigned, which wastes gas.

Recommendation

Initialize the variable once before the loop, and the same memory location will be reused.

Developer Response

Acknowledged - Fixed in #282

4. Gas - Remove the redundant check in TransferLib

The transferTokenFromUser contains a redundant check:

```
        if (tokensIn.length == 0) {
            revert TokenInRequired();
        }
```

This check is already performed by the `checkIncrease`, `singleTokenIncrease`, and `twoTokenIncrease` modifiers in the strategies.

Impact

Minor gas savings.

Recommendation

Remove redundant checks.

Developer Response

Acknowledged - Fixed in #286

5. Gas - Unnecessary double resets of allowance

The following pattern appears in several contracts:

- set approval to 0
- set approval to some value x
- reset approval to 0

One of the zero approvals could safely be removed without causing any issues elsewhere in the codebase.

Technical Details

Example from `SwapLib.sol`

```
function _swap(
    SwapParams memory swapParams
) internal {
    ....
    // In case there is USDT dust approval, revoke it
    SafeTransferLib.safeApprove(tokenIn, swapParams.router, 0);

    SafeTransferLib.safeApprove(
        tokenIn, swapParams.router, swapParams.amountIn
    );

    ...

    // Revoke any approval after swap in case the swap amount was estimated
    SafeTransferLib.safeApprove(tokenIn, swapParams.router, 0);
}
```

The comment mentions potential lingering approvals causing issues, but it's unclear from the codebase where these would originate.

edit

The issue occurred with old strategies that didn't reset the approvals to 0.

Impact

Gas - Minor gas saving

Recommendation

If legacy strategies do not impact the project, removing one of the approvals would result in a small gas savings.

Developer Response

Acknowledged - Won't fix. The double reset is necessary if the user's Sickle has leftover previous USDT allowance. Much earlier versions of the strategies (used by the same Sickles) did not set it to zero, so if we changed this now some old users might not be able to use USDT.

Informational Findings

1. Informational - Missing events in PositionSettingsRegistry

Events are not emitted when setting connector registry in `constructor` and ` setConnectorRegistry.`

Events are not emitted when setting position settings in `setPositionSettings`.

Recommendation

Emit `PositionSettingsSet` and `ConnectionRegistrySet` events.

Developer Response

Acknowledged - Fixed in #282

2. Informational - Incorrect event definition in Automation

The ExitedFor and HarvestedFor events incorrectly include the poolIndex as an address, but it should be a uint.

Impact

No impact since the compiler uses the definition of the other events with the same name and signature.

Recommendation

Update event name to `NftHarvestedFor` and `NftExitedFor` and use the correct type for `tokenId`.

Developer Response

Acknowledged - Fixed in #288

3. Informational - Missing sanity checks in FarmStrategy in token increase modifiers

The twoTokenIncrease modifier in `FarmStrategy` does not check that the two tokens are the same or if they are address(0).

Impact

Unexpected behavior could result from two token increases of the same token.

Recommendation

Add suggested checks.

Developer Response

Acknowledged - Fixed in #285

4. Informational - Prefer the use of constants to literals

The `FeesLib` library function `chargeFee` uses the magic value `0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE`

Recommendation

Use constants for readability and maintainability.

Developer Response

Acknowledged - Fixed in #286

Final Remarks

The code in scope had a shallow attack surface since the main strategy contracts do not hold tokens, contain storage, or utilize much complexity. The complex operations for creating calldata sent to the Sickle's multicall are performed off-chain. The strategies are more of a routing contract to organize and forward calls to the Sickle. Each user has a separate Sickle, so funds are not commingled, and even the Sickles themselves are not intended to hold funds directly but rather farming positions. The overall quality of the code is high, the functions are clear and concise, and the codebase is straightforward to reason about.