yAudit Sickle Review

Review Resources:

- None beyond the code repositories

Auditors:

- Jackson
- fedebianu

Table of Contents

1 Review Summary

2 Scope

3 Code Evaluation Matrix

4 Findings Explanation

5 Critical Findings

6 High Findings

1. High - A transaction can be executed without the required signatures

2. High - Multisig could become permanently stuck

7 Low Findings

1. Low - There is no protection against return bombs

2. Low - If an Aave connector is introduced, users can avoid premiums on flashloans

3. Low - Lack of input checks for Morpho flashloan

8 Gas Saving Findings

1. Gas - Optimize for loops for gas savings

2. Gas - Simplify redundant logic in `callbackSafetyCheck()`

3. Gas - Dust sweep can be made optional

4. Gas - Result length check after low-level calls can be removed

9 Informational Findings

1. Informational - Unused imports

2. Informational - Follow naming conventions

3. Informational - Avoid duplicated code

4. Informational - Remove redundant prefix

10 Final Remarks

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 multi-sig for cross-chain deployments and new flashloan strategies.

The contracts of the Sickle Repo were reviewed over 10 days. The code review was performed by 2 auditors between May 27, 2024, and June 7, 2024. The repository was under active development during the review, but the review was limited to the latest commit at the start. This was commit 72c8dcc06a71e816ff8a08ced4413b3ffce94421 of the `sickle-contracts` repo.

A publicly accessible version of the repo with the fixes from this audit can be found here at commit 38293c660fd0079a5a1391309e8c3ba17d670260.

Scope

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

```
├── governance
│   └── SickleMultisig.sol
└── strategies
    ├── FlashloanStrategy.sol
    ├── LendingMigrator.sol
    ├── LendingStrategy.sol
    ├── LendingStrategyTwoAsset.sol
    ├── SimpleLendingStrategy.sol
    └── lending
        ├── FlashloanCallback.sol
        ├── FlashloanInitiator.sol
        └── LendingStructs.sol
```

After the findings were presented to the Vfat 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	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. However, one bug was related to missing multi-sig threshold logic.
Mathematics	Good	There are very few complex mathematical operations in the code reviewed.
Complexity	Medium	There is complexity around the flow of funds between the user, their Sickle, the flashloan provider, and the DeFi protocol that the Sickle interacts with.
Libraries	Average	Well-known libraries such as OpenZeppelin and Solmate are used. The team did write their own multi-sig rather than using an established multi-sig. This was done because established multisigs are not deployed on networks of interest to the Vfat team.
Decentralization	Good	A multi-sig was introduced to decentralize certain actions taken by the protocol.
Code stability	Good	Changes were made to the code base during the review but not to the contracts in scope.
Documentation	Low	No documentation was provided for the in-scope contracts. However, many of the functions in the contracts in scope contained NatSpec.
Monitoring	Low	Few of the functions in the contracts in scope contain events.
Testing and verification	Average	The contracts in scope range from 40% to 100% 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.

Critical Findings

None.

High Findings

1. High - A transaction can be executed without the required signatures

Technical Details

The `SickleMultisig` has a `threshold` state variable, which ensures the correct number of signers have signed a proposal before it can be executed. However, this state variable is unused throughout the contract, particularly when executing a transaction.

Impact

High. A transaction can be executed without the required number of signatures.

Recommendation

Ensure that the transaction has the required number of signatures before executing it.

Developer Response

Fixed in https://github.com/vfat-io/sickle-contracts/pull/242

2. High - Multisig could become permanently stuck

`signers` can fall below the `threshold`. Multisigs use a threshold to authorize transaction executions. If the number of signers falls below this threshold, the multisig will become permanently stuck, as there will not be enough signers to execute any transactions.

Technical Details

`_removeSigner()` allows for the removal of a signer from the multisig via a proposal. However, there are no security checks to prevent the number of signers from falling below the threshold, which could permanently lock the multisig after enabling `threshold` usage as per the H1 issue.

Impact

High. The multi-sig will become permanently stuck if a signer is removed and `signers` falls below `threshold`.

Recommendation

Ensure that `signers` can't fall below `threshold` in `_removeSigner()`:

```
     function _removeSigner(address signer) internal changesSettings {
         if (!isSigner[signer]) revert SignerAlreadyRemoved();
+        if (signers == threshold) {
+            revert SignersBelowThreshold();
+        }

         isSigner[signer] = false;
         signers -= 1;

         emit SignerRemoved(signer);
     }
```

Developer Response

Fixed in https://github.com/vfat-io/sickle-contracts/commit/c99058d5c8064946e922dacdb55a8a17099f1ec3

Low Findings

1. Low - There is no protection against return bombs

While multisigs are capable of executing arbitrary transactions, it is best practice to safeguard against potential vulnerabilities, such as a return bomb attack. This precaution ensures the multisig remains secure and functional.

Technical Details

The `_call()` low-level call could potentially return a large `result`, causing the multisig to run out of gas as it attempts to copy the data into memory.

Impact

Low. Proposals cannot be executed if a transaction returns a large amount of data, which may cause the multisig to run out of gas.

Recommendation

Use `ExcessivelySafeCall` to avoid return bombs.

Developer Response

Acknowledged.

2. Low - If an Aave connector is introduced, users can avoid premiums on flashloans

When a user repays their flashloan, they also pay a premium of 0.05% for Aave v3 and 0.09% for Aave v2. However, if the Aave connector is introduced and a user wants to leverage lending on Aave, repaying the flashloan and paying the premium can be avoided.

Technical Details

If we look at the code before the `executeOperation()` callback there are two ways that allow the receiver not to pay premiums:

1. the user is part of the `flashBorrowers` list (link to code)
2. the user has decided that for that specific asset flashloan it will open a borrow position (link to code)

The second scenario applies in this case. In this scenario, where you want a flashloan to open a leveraged lending position, you typically repay the flashloan and the premiums using the borrowed funds. However, by specifying that you want the flashloan as a borrow, you eliminate the need to repay it and the associated premiums.

To achieve this, specify `interestRateModes` as `STABLE` or `VARIABLE` in the `flashLoan()` call.

The same considerations apply for Aave v2 as stated in the docs.

Impact

Low. If the Aave connector is introduced, users will pay premiums that could be avoided and use more gas than necessary by repaying the flashloan.

Recommendation

Implement a different behavior for Aave flashloans, enabling the ability to pass different modes to `flashLoan()` and avoid the borrow in the callback.

Developer Response

Acknowledged and will be addressed if/when the Aave connector is introduced.

3. Low - Lack of input checks for Morpho flashloan

`initiateFlashloan()` accepts an arbitrary number of assets. However, some providers only offer flash loans for a specific amount. Uniswap providers have a check for this, but the Morpho provider lacks it.

Technical Details

Morpho `flashLoan()` only accepts a single token as a parameter. However, `initiateFlashloan()` does not include input checks for the Morpho provider, potentially leading to issues if multiple tokens are incorrectly assumed to be supported.

Impact

Low. While multiple tokens can be passed to the strategy call, the Morpho provider only processes a flashloan for one token. This could lead to unexpected behavior.

Recommendation

Add a check for the Morpho provider statement:

```
      } else if (providerType == FlashloanProvider.MORPHO) {
+         if (assets.length != 1) revert NotSingleAsset();
          bytes memory morphoParams =
              abi.encode(sickleAddress, assets[0], params);
          // storing the hash of the callback data for safety checks
          flashloanDataHash = keccak256(morphoParams);
          IMorpho(morpho).flashLoan(assets[0], amounts[0], morphoParams);
      } else {
```

Developer Response

Fixed in https://github.com/vfat-io/sickle-contracts/pull/245

Gas Saving Findings

1. Gas - Optimize for loops for gas savings

Caching the array length outside a loop saves reading it on each iteration.

Initializing the loop variable to zero is unnecessary because it is the default value.

Technical Details

In almost any for loop in the contracts, the array length can be cached before the for loop, and the loop variable initialization could be removed.

Impact

Gas savings.

Recommendation

e.g. cache the array length before the for loop and remove loop variable initialization in `calculatePremiums()`:

```
-    for (uint256 i = 0; i < amounts.length;) {
+    uint256 length = amounts.length;
+    for (uint256 i; i < length;) {
        if (amounts[i] > 0 && aaveV2FlashloanPremiumInBasisPoints > 0) {
            premiums[i] = (
                (amounts[i] * aaveV2FlashloanPremiumInBasisPoints - 1)
                    / 10_000
            ) + 1;
        } else {
            premiums[i] = 0;
        }

        unchecked {
            ++i;
        }
    }
```

Developer Response

Fixed in https://github.com/vfat-io/sickle-contracts/pull/249

2. Gas - Simplify redundant logic in `callbackSafetyCheck()`

There is redundancy in the logic of `callbackSafetyCheck()`.

Technical Details

In `callbackSafetyCheck()` the check `hashCheck != flashloanDataHash` will always pass if `flashloanDataHash` is zero, making the second check redundant.

Impact

Gas savings.

Recommendation

Remove the redundant check:

```
     modifier callbackSafetyCheck(bytes memory params) {
         bytes32 hashCheck = keccak256(params);
-        if (hashCheck != flashloanDataHash || flashloanDataHash == bytes32(0)) {
+        if (hashCheck  != flashloanDataHash) {
             revert InvalidFlashloanData();
         }
         if (currentFlashloanStatus != FlashloanStatus.FLASHLOAN_INITIATED) {
             revert FlashloanNotInitiated();
         }

         _;

         // resetting currentFlashloanStatus to FLASHLOAN_UNLOCKED after all
         // operations are finished
         currentFlashloanStatus = FlashloanStatus.FLASHLOAN_UNLOCKED;
     }
```

Developer Response

Fixed in https://github.com/vfat-io/sickle-contracts/pull/247

3. Gas - Dust sweep can be made optional

Dust is swept at the end of `deposit()` and `compound()` calls in `LendingStrategy`. However, if the strategy is deployed on mainnet, the gas costs can exceed the value of the dust itself.

Technical Details

This piece of code, which performs the dust sweep, is placed at the end of those strategy calls:

```
targets = new address[](1);
data = new bytes[](1);

targets[0] = address(this);
data[0] =
    abi.encodeCall(this._sickle_transfer_tokens_to_user, (sweepTokens));

sickle.multicall(targets, data);
```

Making the dust sweep optional could save gas. Users could choose to perform a sweep in the future when the accumulated dust reaches a substantial amount or during a withdrawal.

Impact

Gas savings.

Recommendation

Make the dust sweep optional:

```
+    if (sweepTokens.length > 0) {
        targets = new address[](1);
        data = new bytes[](1);

        targets[0] = address(this);
        data[0] =
            abi.encodeCall(this._sickle_transfer_tokens_to_user, (sweepTokens));

        sickle.multicall(targets, data);
+    }
```

Developer Response

Fixed in https://github.com/vfat-io/sickle-contracts/pull/248

4. Gas - Result length check after low-level calls can be removed

Technical Details

Result length check `if (result.length == 0) revert();` handles the case of empty low-level calls, but the subsequent YUL statement automatically manages this situation.

Impact

Gas savings.

Recommendation

Remove the `if (result.length == 0) revert();` checks in:

- `_call()`
- `_delegateTo()`
- `multicall()`
- `multicall()`

Developer Response

Partially fixed in https://github.com/vfat-io/sickle-contracts/pull/244/files

Informational Findings

1. Informational - Unused imports

Technical Details

- `ZapModule` in LendingStructs.sol is unused.
- `IERC20` and `SafeTransferLib` in FlashloanInitiator are unused.

Impact

Informational.

Recommendation

Remove the unused imports.

Developer Response

Fixed in https://github.com/vfat-io/sickle-contracts/pull/243

2. Informational - Follow naming conventions

Technical Details

Constants in the codebase lack a consistent format. Some start with lowercase letters, others with uppercase, and others are in uppercase. They should be named with all capital letters and underscores separating words.

The `external` and `public` functions in the `modules` folder contracts start with an underscore. Use the underscore prefix only for internal and private functions.

Impact

Informational.

Recommendation

Follow naming conventions as per Solidity style guide to improve readability.

Developer Response

Fixed in https://github.com/vfat-io/sickle-contracts/pull/253

3. Informational - Avoid duplicated code

Technical Details

The same code is used in two different if branches in `_sickle_transfer_token_to_user`.

Impact

Informational.

Recommendation

```
-     if (token == ETH) {
+     if (token == ETH || token == wrappedNativeAddress) {
         IWETH9(wrappedNativeAddress).withdraw(
             IWETH9(wrappedNativeAddress).balanceOf(address(this))
         );
```

```
            SafeTransferLib.safeTransferETH(recipient, address(this).balance);
-       } else if (token == wrappedNativeAddress) {
-           IWETH9(wrappedNativeAddress).withdraw(
-               IWETH9(wrappedNativeAddress).balanceOf(address(this))
-           );
-           SafeTransferLib.safeTransferETH(recipient, address(this).balance);
        } else {
            SafeTransferLib.safeTransfer(
                token, recipient, IERC20(token).balanceOf(address(this))
            );
        }
```

Developer Response

Fixed in https://github.com/vfat-io/sickle-contracts/commit/cf71c5a541398d639c9380285387b502201e1d15

4. Informational - Remove redundant prefix

Technical Details

The use of the `ZapModule` prefix in `LendingStrategy` is redundant because the `ZapModule` contract is inherited.

Impact

Informational.

Recommendation

Remove the `ZapModule` prefixes and use `this` instead to standardize the code as for other module calls.

Developer Response

Fixed in https://github.com/vfat-io/sickle-contracts/pull/246

Final Remarks

The new flashloan strategies that interact with the Sickles had few major bugs found due to the secure nature of the Sickle architecture. The major bugs that were found were in the new multi-sig contract, which does not interact with the Sickle architecture.