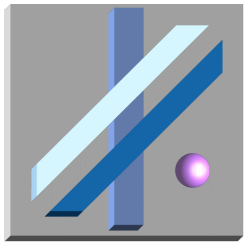

Fixed Point Solutions, LLC

Sense Protocol Assessment

2022/03/15
Prepared by: Kurt Barry

1. Scope

The contents of pkg/core/ were audited at commit:
https://github.com/sense-finance/sense-v1/tree/e560dcc21ae16814e52524f4d477821e480bc7a7

The contents of pkg/space/ were audited at commit:
https://github.com/sense-finance/sense-v1/tree/b5f18c3fb5ab5324a1c5fd6825966ca22b8615d0

Note: later in the Sense development process “Zeros” were renamed to “Principal Tokens” or “PTs”, and “Claims” were renamed to “Yield Tokens” or “YTs”. This report largely predates this naming change and thus uses the original naming scheme.

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 Incorrect Adapter Index Value Emitted in `_setAdapter`

Severity: Low

Code Location:
https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Divider.sol#L562

Description: The `adapterCounter` storage variable is used as the `id` argument in the `AdapterChanged` event in the `_setAdapter` function. However, this is always incorrect. When adding or turning on an adapter, this value will always be one more than the actual id. When turning off an adapter, there is no guaranteed correspondence whatsoever to the correct adapter id. This may cause confusion and create an obstacle to accurately monitoring and alerting on adapter status changes.

Response: Fixed in PR72.

SC.2 Loss of Yield for Claim Recipients on Claims They Already Hold

Severity: High

Code Location:
https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Divider.sol#L369

Description: When a Claim transfer is performed, the value accrued to date is sent to the sender while the receiver has their `lscales` entry updated to the most recent `adapter.scale()` value for the series to prevent them from collecting the same yield. If the receiver already held Claims prior to the transfer, they will lose all uncollected yield on those Claims due to the update to their `lscales` value. This occurs even if the amount transferred is zero--so for just the gas cost of a zero Claim transfer, any Claim holder's yield can be wiped out. This could be particularly high-impact if large amounts of Claims are held in a single smart contract, for example as part of a structured product. This loss of yield also occurs when a current Claim holder that issues themselves additional Claims from the same series.

Probably the simplest way to mitigate this is to also collect accrued value for transfer recipients before updating their `lscales` value. However, that might not be a desirable behavior from a user perspective, as they will suddenly acquire a target balance as well as a claim balance. A smart contract holding claims, in particular, may not have logic to handle target tokens. Another solution is possible: computing a synthetic `lscales` value for the recipient that keeps their claimable yield unchanged and also accrues yield correctly going forward. Define:

C := current claim balance of account about to be sent more claims
LS := current `lscales` value of account about to be sent more claims
X := amount of claims transferred to account
MS := current `maxscale`
LS' := synthetic `lscales` value that will be solved for

LS' is derived by setting the yield available to collect of the recipient before and after the transfer equal:

C/LS - C/MS = (C + X)/LS' - (C + X)/MS

Solving for LS' yields:

LS' = (C + X)/(C/LS + X/MS)

which is just the claim-weighted harmonic mean of LS and MS.

To prove that yield is computed correctly at any future `maxscale` value FS, one must show that:

(C/LS - C/FS) + (X/MS - X/FS)

is equivalent to:

(C + X)/LS' - (C + X)/FS

This can be done straightforwardly via substitution of the formula for LS' into the latter expression. Though more complex than simply collecting yield for the recipient of a transfer, this solution may prove superior in terms of user experience and composability of Claims.

Response: Fixed in PR78 (errors in PR corrected in later commits).

SC.3 Zero Redemption Shortfall Is Calculated Incorrectly in `redeemZero()`

Severity: Low

Code Location:
[1]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Divider.sol#L273
[2]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Divider.sol#L266

Description: Because the assignment at [2], the subtraction at [1] will always result in a `shortfall` value of zero, and hence no correction will ever be applied if an actual shortfall in available target for the zero holder exists. This can be remedied by adding an `else` clause to the `if (tBalClaimActual != 0)` statement and moving the assignment [2] inside of it.

Response: Duplicate from another audit; fixed in PR96.

SC.4 Claim Holders Can Collect Farming Rewards Indefinitely After Redemption

Severity: Low

Code Location:
[1]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Divider.sol#L391

Description: After a series has matured, any attempt to collect yield on Claims results in the holder's Claim balance being fully burned at the code location [1]; however, there is no associated call to `adapter.notify()` to keep consistent accounting of the amount of target assigned to the associated address. Assuming the underlying adapter is a `CropAdapter` that still contains some amount of target (which will generally be true if multiple series exist for the adapter, or will be true for a new series created), the (former) Claim holder can call `collect()` again at a future date and collect some amount of accrued farming rewards (since these are solely determined by the `tBalance` mapping entry for their address in the adapter). This allows Claim holders to obtain rewards they should not be entitled to, and dilutes the rewards distributed to others.

Response: Fixed in PR121.

SC.5 Protocol Fees In Space Are Diluted by LP Deposits and Withdrawals

Severity: Low

Code Location:
[1]https://github.com/sense-finance/sense-v1/blob/b5f18c3fb5ab5324a1c5fd6825966ca22b8615d0/pkg/space/src/Space.sol#L149
[2]https://github.com/sense-finance/sense-v1/blob/b5f18c3fb5ab5324a1c5fd6825966ca22b8615d0/pkg/space/src/Space.sol#L226

Description: Both the `onJoinPool` and `onExitPool` functions distribute protocol fees after doing calculations that invoke `totalSupply()` to determine either how many BPTs the user is to be issued or the quantity of each reserve token they will receive. In both cases this reduces the actual fees collected. In an extreme case where a pool has a single depositor that withdraws all of their liquidity, all fees are lost. Similarly, large deposits dilute protocol fees. Protocol fees should be accounted for prior to any other calculations that require `totalSupply()` as an input to avoid this.

Response: Fixed in PR137.

SC.6 No Verification When Setting `Space` Fee Parameter Values

Severity: Low

Code Location:
[1]https://github.com/sense-finance/sense-v1/blob/b5f18c3fb5ab5324a1c5fd6825966ca22b8615d0/pkg/space/src/SpaceFactory.sol#L67
[2]https://github.com/sense-finance/sense-v1/blob/b5f18c3fb5ab5324a1c5fd6825966ca22b8615d0/pkg/space/src/SpaceFactory.sol#L42

Description: The Space AMM applies a different fee parameter depending on the direction of the swap (`g1` when swapping targets to Zeros, and `g2` when swapping Zeros to targets). This is due to the fact in one case the effective interest rate should be discounted, while in the other it should be marked up. This implies that there are hard numerical bounds for these parameters to prevent accidentally paying fees to traders instead of liquidity providers, e.g. `g1` must be less than one in fixed-point terms, and vice versa for `g2`. The functions that set these values could perform input validation to ensure this.

Response: Fixed in PR129.

SC.7 Laggy Pricing for Target Assets in Fuse Pools

Severity: Low

Code Location:
https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/fuse/src/oracles/Target.sol#L41

Description: Using the last recorded scale value instead of an updated one could lead to mispricing assets in lending pools, resulting e.g. in spurious liquidations. The risk should be low if interactions with code that does update cached scale values are executed frequently. One mitigation might be to impose a time constraint (e.g. one day) on how old a scale value is allowed to be, although this is imperfect given that assets will have different and time-varying interest rates.

Response: “A risk would still exist, but https://github.com/sense-finance/sense-v1/pull/139 will help substantially for protocols that have a cache'd scale & healthy update schedule. We'll accept the risk in this case. For protocols without their own cache, we'll consider running our own service to ensure the cache'd scale doesn't become too stale. (e.g. once a day)”

SC.8 Incorrect Underlying Token Assignment in Zero Oracle

Severity: Low

Code Location:
https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/fuse/src/oracles/Zero.sol#L98

Description: The assignments in the two branches here should use the opposite index, e.g. if the Zero is index 0, then `underlying` should be assigned the address of the token at index 1. As a consequence, attempts to query the Zero price will fail (depending on architecture, either due to exceeding the maximum EVM call depth, or the caller, which is assumed to implement an oracle interface, simply not having a way to price Zeros itself). This would be quickly noticed in practice and wouldn’t be exploitable within a standard lending market architecture.

Response: Fixed in PR126.

SC.9 Zero Token Assumed to Be the 0-Index Token in a Pool in the Zero Oracle

Severity: Medium

Code Location:
https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/fuse/src/oracles/Zero.sol#L94

Description: As written, the code assumes the Zero token will be at index 0 for the pool, which will not always be the case. This would lead to returning an erroneous price, which could be exploitable in some scenarios, potentially even allowing undercollateralized borrowing if the erroneous price is sufficiently higher than the true price.

Response: Fixed in PR125.

SC.10 Space LP Share Price Can Be Manipulated With Flash Loans

Severity: Medium

Code Location:
https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/fuse/src/oracles/LP.sol#L87

Description: Relying on the current pool reserves in the manner done here to compute Space LP token prices allows manipulation to be done using flash loans. Many practical attacks have resulted from such pricing techniques. A safer, although more complex, method would be to use the pool invariant to compute “equilibrium” values for the two reserves and use those instead.

To give an example of a possible attack, consider explicitly the case where the series is still far from settlement, meaning the Space pool behaves more like a constant product than a constant sum. We will assume that the prices obtained from oracles are not affected (or at least not strongly affected) by any activity the attacker makes within a single transaction (if this is not the case–the price oracles are vulnerable to manipulation and that is a separate issue). While Balancer pools are not vulnerable to “donation” attacks since they employ internal tracking of balances (since all tokens are held in a central Vault contract), the slippage effect still allows manipulation of balances. By flash borrowing a large number of Zeros and swapping them in the Space pool for Target, the price reported by this oracle implementation can be made erroneously high. This is because the average execution price for the swap, due to slippage, will be lower than the market prices, which do not update–so if e.g. $100 of Target is removed, >$100 of Zeros will be added (to emphasize–based on the non-manipulable market prices). The resulting erroneously high LP token could enable economic attacks similar to the famous attack on the Cream lending protocol.

As mentioned above, this can be mitigated by computing “equilibrium” balances, i.e. the balances the pool would have assuming the swap price reflects the market price for one asset in terms of the other. Complete formulae are given below.

Let:
p_t := current market price of the Target token in the reference asset
p_z := current market price of the Zero token in the reference asset
r_t := current target reserves of the Space pool

s_i := scale value for the Target asset when the Space pool was created (note: multiplying a Target quantity by a scale computes the amount of Underlying that amount of target is redeemable for at that scale; the Space implementation tracks the initial scale and uses it to separate interest income to be exclusively for LPs)
s := current scale of the Target asset
r_z := current "real" Zero reserves
v := current virtual Zero reserves (identical to the total supply of LP tokens)

The current YieldSpace invariant at time-to-maturity t is computed as:

$$k = (r_t \cdot s_i)^{1-t} + (r_z + v)^{1-t}$$

The implied market interest is computed as:

$$r = \left(\frac{p_t/s}{p_z}\right)^{1/t} - 1$$

Define the equilibrium values we will solve for:
R_z := equilibrium "real" Zero reserves
R_t := equilibrium target reserves

To obtain these values, we must solve the simultaneous system of equations (see the YieldSpace paper for a justification of [2]):

$$[1]\ k = (R_t \cdot s_i)^{1-t} + (R_z + v)^{1-t}$$
$$[2]\ r = \frac{R_z+v}{R_t \cdot s_i} - 1$$

This can be done, for example, by using [2] to isolate an expression for R_t in terms of R_z, substituting that expression into [1], and solving the resulting single-variable equation for R_z. With the solution for R_z in hand, it can be substituted back into the expression derived from [2] for R_t to complete the solution. The results are:

$$R_z = \left(\frac{k}{1+\frac{1}{(1+r)^{1-t}}}\right)^{\frac{1}{1-t}}$$

$$R_t = \frac{R_z+v}{s_i \cdot (r+1)} = \frac{1}{s_i \cdot (r+1)}\left(\frac{k}{1+\frac{1}{(1+r)^{1-t}}}\right)^{\frac{1}{1-t}}$$

With these values, the price (p_{LP}) of a single Space LP token can be computed as:

$$p_{LP} = (R_t \cdot p_t + R_z \cdot p_z)/v$$

This is somewhat involved, but it guarantees that the value of LP tokens cannot be manipulated except to the extent that the trusted oracle prices can be manipulated.

Response: Fixed in commit 1d3df05cf7de02b38a3abdbc386f912da8c340c2. See also Fixed Point Solutions's further audit of Space after it was moved into its own repository for discussion and resolution of an issue with the `getImpliedRateFromPrice` function used in the fix (finding SC.3 in that report).

SC.11 `Periphery.addLiquidityFromUnderlying` Doesn't Transfer Tokens Correctly

Severity: Low

Code Location:
https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Periphery.sol#L260

Description: The `_addLiquidity()` function attempts to transfer Target tokens from `msg.sender`, but here, it is expected that the caller possesses the Underlying token. The lines above the invocation of `_addLiquidity()` attempt to wrap Underlying to Target, but do not transfer the caller's Underlying to the `Periphery` first (and would need to transfer those tokens back to the caller for the call to `_addLiquidity()` to succeed). One simple fix would be to do all token transfers and conversions explicitly in both `addLiquidityFromTarget()` and `addLiquidityFromUnderlying`, and modify `_addLiquidity()` to assume the necessary tokens have been moved to the `Periphery`.

Response: Fixed in PR130.

SC.12 Assumption of Pool Token Ordering in `Periphery._addLiquidity`

Severity: Low

Code Location:
https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Periphery.sol#L456

Description: The calculation of `zBalInTarget` assumes a particular ordering for tokens inside the Space pool, but this will not necessarily be the case, leading to unexpected behavior for callers attempting to provide liquidity via `Periphery` methods.

Response: Fixed.

SC.13 Incorrect Calculation of Amount of Target Needed to Issue Zeros in `Periphery._addLiqudity`

Severity: Low

Code Location:
https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Periphery.sol#L456

Description: Whichever balance in the denominator represents the amount of Target currently in the pool should be multiplied by a factor of `adapter.scale()` (which appropriate precision adjustments).

Response: Fixed.

SC.14 Token Location Mismatch in `Periphery.migrateLiquidity`

Severity: Low

Code Location:
https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Periphery.sol#L322

Description: The execution of `_removeLiquidity()` will result in the withdrawn liquidity assigned to the `Periphery` as Target tokens; however, `_addLiquidity()` will pull Target tokens from `msg.sender`, resulting in incorrect behavior.

Response: Fixed in PR130.

SC.15 Unwrapping WSTETH Lacks Slippage Protection

Severity: Medium

Code Location:
https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/adapters/lido/WstETHAdapter.sol#L90

Description: The calculated minDy parameter for Curve's swap function comes from the expected result of the swap according to the current state of the Curve pool, which means this operation is just accepting whatever price the pool offers. This may lead to value extraction opportunities (e.g. sandwich attacks) that negatively impact users of the Sense Protocol.

Response: Fixed in commit af5081c5ccef8d2143e85bc94dff540c48e535e5.

3.2 Usability and Incentives

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

U.1 Claim Swapping Functions in `Periphery` Lack Slippage Protection

Severity: Low

Code Location:
[1]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Periphery.sol#L145
[2]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Periphery.sol#L158
[3]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Periphery.sol#L207
[4]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Periphery.sol#L221

Description: These swap functions don't include any slippage protection parameter, whereas other swapping functions do. Users swapping through the Periphery directly will therefore be fully exposed to value extraction via transaction sandwiching or similar techniques. While callers can mitigate this by using a proxy contract that imposes slippage checks (in particular, most DEX aggregators possess this functionality), it might be better for adoption and general user experience if slippage protection is added to these functions.

Response: Slippage protection added to the flash loan component in commit f9fd5820a7a3ceb32decfe604cee15a52c94a8cd. See also discussion in PR#173.

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 Many Unnecessary SLOADs When Reading Adapter Params in `Divider` and `Periphery`

Severity: Low

Code Location:
[1]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Divider.sol#L92
[2]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Divider.sol#L135

[3]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Divider.sol#L502
[4]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Periphery.sol#L78

Description: Even though most of the values returned by `adapterParams()` are unneeded, these calls will still perform expensive SLOAD operations for the discarded values. In all three cases in the `Divider` contract, the same values are being accessed, so these calls could be optimized by adding a function to the `BaseAdapter` contract that reads and returns only these three values. The call to `adapterParams()` in the Periphery contract is reading two out of the same three variables, so it could either reuse the three-value reading function for only a small inefficiency, or a two-value reading function could be defined as well.

Response: Improved in commit ca4787d02c8a94787ec3a4b7da83c8784432d34f.

G.2 Unnecessary Storage Reads When Emitting Events

Severity: Low

Code Location:
[1]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Divider.sol#L449
[2]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Divider.sol#L456
[3]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Divider.sol#L469

Description: In these cases the calldata value is the same as the storage value and can be used instead, which should circumvent a warm SLOAD, saving a small amount of gas.

Response: Fixed in commit 486a0065b5a219e6c4f97c8f30571ea5a3e28b0e.

G.3 Initialization Flag in `TokenHandler` is Unnecessary

Severity: Low

Code Location:
https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Divider.sol#L634

Description: Since the value of `divider` will always be zero prior to initialization and non-zero afterwards, the check on `inited` in `init()` can be replaced with `require(divider != address(0))` and the check on `inited` in `deploy()` can simply be omitted as the check on `msg.sender` will be sufficient. Since those are the only two usages of `inited`, it can then be eliminated.

Note: more complex creation sequences for the `Divider` and `TokenHandler` could allow for complete elimination of the need to initialize the `TokenHandler` and allow the `divider` field to be `immutable`, further lowering interaction costs. This, however, would require deploying at least one extra contract to act as an address registry and may not be worth the extra complexity.

Response: Fixed in commit 81acde6870a64db967eff3ceb613fb9ce3829ddb.

G.4 Adapter Params Could Be Separated and Made `immutable`

Severity: Low

Code Location:
https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/adapters/BaseAdapter.sol#L33

Description: These parameters cannot be modified after initialization; the BaseAdapter and BaseAdapterFactory could be modified to accommodate passing these values in the constructor so they could be immutable, likely resulting in overall significant savings over the lifetime of an adapter (though the initial deployment would be more costly).

Response: Fixed in PR134.

G.5 Overcomplicated Fee Calculation

Severity: Low

Code Location:
https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Divider.sol#L165

Description: This entire conditional could be replaced with a single line by switching the order of multiplication:

```
fee = tBal.fmul(issuanceFee, 1 ether);
```

This further eliminates the need to read the target's decimals on line 157 or to compute `tBase`. In addition to saving gas, this change makes the code more readable and minimizes rounding error.

Response: Fixed in commit cd360b0adb8ce39b77f8f73bd923853ff0677fa1.

G.6 Unnecessary Safe Math Applications

Severity: Low

Code Location:
[1]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/adapters/CropAdapter.sol#L69
[2]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/adapters/CropAdapter.sol#L71

Description: Code in the indicated locations could be restructured to eliminate some or all safe math via the use of unchecked { … } blocks due to the presence of conditionals that ensure overflow/underflow is impossible.

Response: Fixed [1] in commit f208811eefe7708b431979af2ebd87786f644826.

G.7 Excessive Storage Reads During Space Pool Creation

Severity: Low

Code Location:
https://github.com/sense-finance/sense-v1/blob/b5f18c3fb5ab5324a1c5fd6825966ca22b8615d0/pkg/space/src/SpaceFactory.sol#L60

Description: Seven extra SLOAD operations are performed when obtaining the address of the contract for Zeros of a given series. The function table for the `Divider` is already fairly large, so it is likely worthwhile to add a function that allows retrieving just the Zero address. Alternatively, a deployment strategy for Zero (and Claim) contracts that generates deterministic addresses could be used, allowing them to be computed instead of read. The CREATE3 library found in the solmate dependency could be used for this purpose.

Response: Tried deterministic addresses, but the operation became more expensive–may revisit later.

G.8 `WstETHAdapter._scale()` Can Be Optimized to Eliminate an SLOAD and an Contract Call

Severity: Low

Code Location:
https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/adapters/lido/WstETHAdapter.sol#L71

Description: This function calls the `stEthPerToken()` function of the WSTETH contract, reading the WSTETH contract address from storage (via `adapterParams.target`). However, the `stEthPerToken()` function itself makes an external call `getPooledEthByShares(1 ether)` to the STETH contract. Further, `WstETHAdapter` already hardcodes the STETH contract

address as a constant. Thus the body of `_scale()` can be replaced with a direct call to `getPooledEthByShares(1 ether)`, eliminating both the SLOAD to read the target address, and the intermediate call to `stEthPerToken()`.

Response: Fixed in PR143.

G.9 Likely Storage Read Optimization in `CAdapter` Functions

Severity: **Low**

Code Location:
[1]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/adapters/compound/CAdapter.sol#L89
[2]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/adapters/compound/CAdapter.sol#L105

Description: The Solidity compiler tends not to memoize storage reads in most cases. In these two locations, a single storage read could be guaranteed by first reading `adapterParams.target` into a stack variable, and only then casting and calling to get the underlying token. Though the cost for a warm SLOAD is only 100 gas post-EIP-2929, it is still usually worthwhile to eliminate them where possible.

Response: Obsolete (Underlying and Target addresses now cached in the adapter).

3.4 Code Quality

CQ.1 Typos

Severity: **Informational**

Code Location:
[1]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Divider.sol#L551
[2]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Divider.sol#L239
[3]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Divider.sol#L243
[4]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Divider.sol#L330
[5]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Divider.sol#L116
[6]https://github.com/sense-finance/sense-v1/blob/b5f18c3fb5ab5324a1c5fd6825966ca22b8615d0/pkg/space/src/Space.sol#L446

Description:
[1] “FNCTIONS” → “FUNCTIONS”
[2] its → it’s
[3] equivelent → equivalent
[4] users → user’s
[5] transfer → transfers
[6] “to a swap” → “to swap”

Response: [1-5] fixed in PR99.

CQ.2 Missing Function Parameter Comments

Severity: **Informational**

Code Location:
[1]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Divider.sol#L81
[2]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Periphery.sol#L114
[3]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Periphery.sol#L128

Description:
[1] The `sponsor` parameter is omitted.
[2] The minAccepted parameter is omitted.
[3] Same as [2].

Response: [1] addressed in PR99.

CQ.3 Inconsistent Use of Underscore Prefixes in `BaseAdapter.sol`

Severity: **Informational**

Code Location:
[1]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/adapters/BaseAdapter.sol#L49
[2]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/adapters/BaseAdapter.sol#L97

Description: At [1], a contract storage field is prefixed with an underscore but this is not done for the other storage fields. At [2] a local stack variable is prefixed with an underscore, but this is not done for other local stack variables. The code would be easier to audit and maintain if underscore prefixes were used with semantic consistency.

Response: Fixed in PR99.

CQ.4 Inaccurate Comment in `Space.getIndices()`

Severity: **Informational**

Code Location:
https://github.com/sense-finance/sense-v1/blob/b5f18c3fb5ab5324a1c5fd6825966ca22b8615d0/pkg/space/src/Space.sol#L483

Description: The field `zeroi` is `immutable`, so in fact no SLOAD is performed here, so this comment can be removed.

Response: Fixed in commit b99c536fddf725aa877aa9bf458c41660cbc9f68.

CQ.5 Unused Import

Severity: **Informational**

Code Location:
https://github.com/sense-finance/sense-v1/blob/b5f18c3fb5ab5324a1c5fd6825966ca22b8615d0/pkg/space/src/Space.sol#L10

Description: This import is unused.

Response: Fixed in PR122.

CQ.6 Unresolved TODOs

Severity: **Informational**

Code Location:
[1]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/fuse/src/PoolManager.sol#L189
[2]https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Periphery.sol#L328

Description: These TODOs should be resolved, either by adding logic, or deleting them if they are no longer needed.

Response: [1] addressed in PR138; [2] addressed in commit 5b86a971946f04f4110835651207e6c06420be33.

CQ.7 Dead Link

Severity: **Informational**

Code Location:
https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/fuse/src/oracles/Zero.sol#L77

Description: The link in this comment should go to documentation for Balancer's built-in oracles, but instead a "Page not found" error results.

Response: Fixed in PR122.

CQ.8 Inaccurate Comment for Return Value of `_flashBorrow()`

Severity: **Informational**

Code Location:
https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Periphery.sol#L524

Description: This comment states that `_flashBorrow()` returns the number of Claims issued, but in fact it returns the amount of Target obtained from a sale of Claims.

Response: Fixed in PR122.

CQ.9 Inaccurate Parameter Name in `FactoryChanged` Event

Severity: **Informational**

Code Location:
https://github.com/sense-finance/sense-v1/blob/e560dcc21ae16814e52524f4d477821e480bc7a7/pkg/core/src/Periphery.sol#L623

Description: The first argument is named `adapter`, but the value actually corresponds to an adapter factory; hence, a more accurate name could be used.

Response: Fixed in PR122.

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.

None.