SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for: Sense
Prepared by: Sherlock
Lead Security Expert: 0x52
Dates Audited: March 17 - March 23, 2023
Prepared on: April 11, 2023

Introduction

Sense is decentralized permissionless infrastructure, where teams can build and develop new yield primitives for DeFi.

Scope

sense-v1 @ 82abac25404d83b7aefaaeb46631f1d050dc4a4e

- sense-v1/pkg/core/src/Divider.sol
- sense-v1/pkg/core/src/Periphery.sol
- sense-v1/pkg/core/src/adapters/abstract/BaseAdapter.sol
- sense-v1/pkg/utils/src/Trust.sol

auto-roller @ 60b8b4d56346f053becafb6a9f50f75cebafcafa

- auto-roller/src/RollerPeriphery.sol

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Issues found

Medium	High
5	0

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

spyrosonic10
0x52
Bauer

0xAgro
martin
sayan_

Breeje
Saeedalipoor01988
tsvetanovv

Issue M-1: Multiple functions aren't payable so quotes that require protocol fees won't work correctly

Source: https://github.com/sherlock-audit/2023-03-sense-judging/issues/28

Found by

Bauer, 0x52

Summary

There are multiple functions that use quotes but that aren't payable. This breaks their compatibility with some quotes. As the 0x docs state: `Certain quotes require a protocol fee, in ETH, to be attached to the swap call.`

The following flows use a quote but the external/public starting function isn't payable:

RollerPeriphery

1) redeem

Periphery

1) removeLiquidity
2) combine
3) swapPT
4) swapYT
5) issue

Vulnerability Detail

See summary.

Impact

Functions won't be compatible with certain quotes causing wasted gas fees or bad rates for users

Code Snippet

https://github.com/sherlock-audit/2023-03-sense/blob/main/auto-roller/src/RollerPeriphery.sol#L104

https://github.com/sherlock-audit/2023-03-sense/blob/main/sense-v1/pkg/core/src/Periphery.sol#L325

https://github.com/sherlock-audit/2023-03-sense/blob/main/sense-v1/pkg/core/src/Periphery.sol#L409

https://github.com/sherlock-audit/2023-03-sense/blob/main/sense-v1/pkg/core/src/Periphery.sol#L433

https://github.com/sherlock-audit/2023-03-sense/blob/main/sense-v1/pkg/core/src/Periphery.sol#L240

https://github.com/sherlock-audit/2023-03-sense/blob/main/sense-v1/pkg/core/src/Periphery.sol#L263

Tool used

Manual Review

Recommendation

Add payable to these external/public functions

Discussion

jparklev

Confirmed: We've forgotten to add payable to the functions mentioned

Issue M-2: Multiple functions may leave excess funds in the contract that should be returned

Source: https://github.com/sherlock-audit/2023-03-sense-judging/issues/29

Found by

Bauer, 0x52, spyrosonic10

Summary

Periphery#combine may leave excess underlying in the contract due to _fromTarget unwrapping to underlying and the quote may not swap them all.

When using arbitrary tokens to swap to underlying the contract always moves in the full amount specified. There is no guarantee that the quote will consume all tokens. As a result the contract may leave excess sell tokens in the contract but it should return then to the receiver.

These functions include:

RollerPeriphery

1) deposit

Periphery

1) swapForPTs
2) addLiquidity
3) issue

RollerPeriphery#RollermintFromUnderlying uses adapter.scale and previewMint to determine the amount of underlying to transfer. The roller code will mean that previewMint will always perfectly reflect the exact exchange rate into the roller. However adapter.scale varies by adapter and isn't guaranteed to be exact. The result is that _transferFrom may take too much underlying. Since this underlying is wrapped to target the contract should return all excess target to receiver.

Vulnerability Detail

See summary.

Impact

Token may be left in the contract and lost

Code Snippet

https://github.com/sherlock-audit/2023-03-sense/blob/main/auto-roller/src/RollerPeriphery.sol#L175-L186

https://github.com/sherlock-audit/2023-03-sense/blob/main/auto-roller/src/RollerPeriphery.sol#L196

https://github.com/sherlock-audit/2023-03-sense/blob/main/sense-v1/pkg/core/src/Periphery.sol#L178

https://github.com/sherlock-audit/2023-03-sense/blob/main/sense-v1/pkg/core/src/Periphery.sol#L325

https://github.com/sherlock-audit/2023-03-sense/blob/main/sense-v1/pkg/core/src/Periphery.sol#L409

Tool used

Manual Review

Recommendation

Return excess tokens at the end of the function

Discussion

jparklev

We have this feature, for example, on `_swapSenseToken`, but not on the cases mentioned.

Our fix will be: Transfer non-used tokens back to the user.

Issue M-3: Periphery#_swapPTsForTarget won't work correctly if PT is mature but redeem is restricted

Source: https://github.com/sherlock-audit/2023-03-sense-judging/issues/32

Found by

0x52

Summary

Periphery#_swapPTsForTarget doesn't properly account for mature PTs that have their redemption restricted

Vulnerability Detail

Periphery.sol#L531-L551

```
function _swapPTsForTarget(
    address adapter,
    uint256 maturity,
    uint256 ptBal,
    PermitData calldata permit
) internal returns (uint256 tBal) {
    _transferFrom(permit, divider.pt(adapter, maturity), ptBal);

    if (divider.mscale(adapter, maturity) > 0) {
        tBal = divider.redeem(adapter, maturity, ptBal); <- @audit-issue always
        ↪  tries to redeem even if restricted
    } else {
        tBal = _balancerSwap(
            divider.pt(adapter, maturity),
            Adapter(adapter).target(),
            ptBal,
            BalancerPool(spaceFactory.pools(adapter, maturity)).getPoolId(),
            0,
            payable(address(this))
        );
    }
}
```

Adapters can have their redeem restricted meaning the even when they are mature they can't be redeemed. In the scenario that it is restricted Periphery#_swapPTsForTarget simply won't work.

Impact

Redemption will fail when redeem is restricted because it tries to redeem instead of swapping

Code Snippet

Tool used

Manual Review

Recommendation

Use the same structure as _removeLiquidity:

```
if (divider.mscale(adapter, maturity) > 0) {
    if (uint256(Adapter(adapter).level()).redeemRestricted()) {
        ptBal = _ptBal;
    } else {
        // 2. Redeem PTs for Target
        tBal += divider.redeem(adapter, maturity, _ptBal);
    }
```

Discussion

jparklev

Accepted: This is valid and is indeed something we should fix

Issue M-4: fillQuote uses transfer instead of call which can break with future updates to gas costs

Source: https://github.com/sherlock-audit/2023-03-sense-judging/issues/33

Found by

sayan_, Saeedalipoor01988, 0x52, tsvetanovv, martin, 0xAgro, Bauer, Breeje

Summary

Transfer will always send ETH with a 2300 gas. This can be problematic for interacting smart contracts if gas cost change because their interaction may abruptly break.

Vulnerability Detail

See summary.

Impact

Changing gas costs may break integrations in the future

Code Snippet

https://github.com/sherlock-audit/2023-03-sense/blob/main/sense-v1/pkg/core/src/Periphery.sol#L902-L932

Tool used

Manual Review

Recommendation

Use call instead of transfer. Reentrancy isn't a concern since the contract should only ever contain the callers funds.

Discussion

jparklev

Accepted: we should use .call instead of transfer when transferring ETH, specifically if the receiver is a contract that is integrating Sense.

Issue M-5: sponsorSeries() method fails when user want to swap for stake token using

Source: https://github.com/sherlock-audit/2023-03-sense-judging/issues/36

Found by

spyrosonic10

Summary

`sponsorSeries()` fails when user want to use `swapQuote` to swap for stake token to sponsor a series.

Vulnerability Detail

stake is token that user need to deposit (technically is pulled) to be able to sponsor a series for a given target. User has option to send `SwapQuote calldata quote` and swap any ERC20 token for stake token. Below is the code that doing transferFrom() of stakeToken not sellToken()

```
if (address(quote.sellToken) != ETH) _transferFrom(permit, stake, stakeSize);
 if (address(quote.sellToken) != stake) _fillQuote(quote);
```

Expected behaviour of this function is to pull `sellToken` from msg.sender when `address(quote.sellToken) != stake`. For example- stake token is WETH. User want to swap DAI for WETH in `sponsorSeries()`. In this case, user would be sending SwapQuote.sellToken = DAI and swapQuote.buyToke = WETH and expect that fillQuote() would swap it for WETH. This method will fail because sellToken not transferred from msg.sender.

Impact

sponsorSeries() fails when `address(quote.sellToken) != stake`

Code Snippet

https://github.com/sherlock-audit/2023-03-sense/blob/main/sense-v1/pkg/core/src/Periphery.sol#L116-L128

Tool used

Manual Review

Recommendation

Consider implementation of functionality to transferFrom `sellToken` from msg.sender with actual amount that is require to get exact amountOut greater or equal to `stakeSize`

Discussion

jparklev

Accepted:

This bug is valid but the below statement

> `sponsorSeries()` fails when user want to use `swapQuote` to swap for stake token to sponsor a series.

is not quite accurate.

The problem here is that here:

```
if (address(quote.sellToken) != ETH) _transferFrom(permit, stake, stakeSize);
```

we are sending wrong params to `_transferFrom`.

If we are making use of the `permit` feature, this would work fine because the `_transferFrom` **ignores** the params on that case.

On the contrary, if we want to make use of the traditional approval, this would revert since we will be trying to pull a the `stake` which has not been approved by the user.

Fix:

```
if (address(quote.sellToken) != ETH) _transferFrom(permit, quote.sellToken,
↪  quote.amount);
// quote.amount does not exist so we may need to add this param to the struct
```