Sandclock v2 Audit Report

Apr 27, 2023

WATCHPUG

Table of Contents

Summary

This report has been prepared for Sandclock v2 Audit Report smart contract, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Overview

Project Summary

Project Name	**Sandclock v2 Audit Report**
Codebase	**https://github.com/lindy-labs/sandclock-contracts**
Commit	**b7f942f3182716107323f2c1039992104cf3186b**
Language	**Solidity**

Audit Summary

Delivery Date	**Apr 27, 2023**
Audit Methodology	**Static Analysis, Manual Review**
Total Isssues	**9**

[WP-C1] Lack of initiator check of the flashloan callback allows the attacker to drain the vault

Critical

Issue Description

There is no initiator check in the `flashloan` callback function.

This allows an attacker to drain the vault by initiating a `flashloan` with a fake token. They can then trick the vault contract into sending all the WETH balance to `balancerVault` .

https://github.com/balancer/balancer-v2-monorepo/blob/master/pkg/vault/contracts/FlashLoans.sol#L38

```
37  function flashLoan(
38      IFlashLoanRecipient recipient,
39      IERC20[] memory tokens,
40      uint256[] memory amounts,
41      bytes memory userData
42  ) external override nonReentrant whenNotPaused {
43      InputHelpers.ensureInputLengthMatch(tokens.length, amounts.length);
44
45      uint256[] memory feeAmounts = new uint256[](tokens.length);
46      uint256[] memory preLoanBalances = new uint256[](tokens.length);
47
48      // Used to ensure `tokens` is sorted in ascending order, which ensures token
    uniqueness.
49      IERC20 previousToken = IERC20(0);
50
51      for (uint256 i = 0; i < tokens.length; ++i) {
52          IERC20 token = tokens[i];
53          uint256 amount = amounts[i];
54
55          _require(token > previousToken, token == IERC20(0) ? Errors.ZERO_TOKEN :
    Errors.UNSORTED_TOKENS);
56          previousToken = token;
57
58          preLoanBalances[i] = token.balanceOf(address(this));
59          feeAmounts[i] = _calculateFlashLoanFeeAmount(amount);
60
```

```
        _require(preLoanBalances[i] >= amount, Errors.INSUFFICIENT_FLASH_LOAN_BALANCE);
        token.safeTransfer(address(recipient), amount);
    }

    recipient.receiveFlashLoan(tokens, amounts, feeAmounts, userData);

    for (uint256 i = 0; i < tokens.length; ++i) {
        IERC20 token = tokens[i];
        uint256 preLoanBalance = preLoanBalances[i];

        // Checking for loan repayment first (without accounting for fees) makes for simpler debugging, and results
        // in more accurate revert reasons if the flash loan protocol fee percentage is zero.
        uint256 postLoanBalance = token.balanceOf(address(this));
        _require(postLoanBalance >= preLoanBalance, Errors.INVALID_POST_LOAN_BALANCE);

        // No need for checked arithmetic since we know the loan was fully repaid.
        uint256 receivedFeeAmount = postLoanBalance - preLoanBalance;
        _require(receivedFeeAmount >= feeAmounts[i], Errors.INSUFFICIENT_FLASH_LOAN_FEE_AMOUNT);

        _payFeeAmount(token, receivedFeeAmount);
        emit FlashLoan(recipient, token, amounts[i], receivedFeeAmount);
    }
}
```

https://github.com/lindy-labs/sandclock-contracts/blob/f3a62502638d9de496ad05df381fc56cae59d0e8/src/steth/scUSDC.sol#L227-L258

```
function receiveFlashLoan(address[] memory, uint256[] memory amounts, uint256[] memory, bytes memory userData)
    external
{
    if (msg.sender != address(balancerVault)) {
        revert InvalidFlashLoanCaller();
    }

    uint256 flashLoanAmount = amounts[0];
    (uint256 collateral, uint256 debt) = abi.decode(userData, (uint256, uint256));

    aavePool.repay(address(weth), debt, C.AAVE_VAR_INTEREST_RATE_MODE, address(this));
    aavePool.withdraw(address(asset), collateral, address(this));

    asset.approve(address(swapRouter), type(uint256).max);

    ISwapRouter.ExactOutputSingleParams memory params = ISwapRouter.ExactOutputSingleParams({
        tokenIn: address(asset),
        tokenOut: address(weth),
        fee: 500,
        recipient: address(this),
        deadline: block.timestamp,
        amountOut: flashLoanAmount,
        amountInMaximum: type(uint256).max, // ignore slippage
        sqrtPriceLimitX96: 0
    });

    swapRouter.exactOutputSingle(params);

    asset.approve(address(swapRouter), 0);

    weth.safeTransfer(address(balancerVault), flashLoanAmount);
}
```

POC

Given:

- 1 WETH = 2000 USDC
- scUSDC have: 1M USDC as collateral, 250 WETH (worth $500k)

1. Attacler called `balancerVault` with:

- `recipient` = `scUSDC`
- `userData` = { collateral: 1M debt: 250 }
- `tokens` = [WETH]
- `amounts` = [250]

scUSDC will repay 250 WETH to `aavePool` and withdraw 1M USDC back to scUSDC

1. Attacler called `balancerVault` with:

- `recipient` = `scUSDC`
- `userData` = { collateral: 0 debt: 0 }
- `tokens` = [FAKE_TOKEN]
- `amounts` = [250]

scUSDC will repay 250 WETH TO `balancerVault` and get 250 `FAKE_TOKEN`

Recommendation

Consider setting a storage variable when initiating the flashloan and check the value of that storage variable in the callback function `receiveFlashLoan()` to ensure that the flashloan originates from the current contract.

Status

✓ Fixed

[WP-H2] Improper setting of `slippageTolerance` may prevent `rebalance()` and so that when the aave position is underwater, it may get liquidated

High

Issue Description

https://github.com/lindy-labs/sandclock-contracts/blob/f3a62502638d9de496ad05df381fc56cae59d0e8/src/steth/scUSDC.sol#L154-L197

```solidity
function rebalance() public onlyKeeper {
    uint256 initialBalance = getUsdcBalance();
    uint256 currentBalance = initialBalance;
    uint256 collateral = getCollateral();
    uint256 invested = getInvested();
    uint256 debt = getDebt();
    uint256 profit = _calculateWethProfit(invested, debt);

    // 1. sell profits
    if (profit > invested.mulWadDown(DEBT_DELTA_THRESHOLD)) {
        uint256 withdrawn = _disinvest(profit);
        currentBalance += _swapWethForUsdc(withdrawn);
        invested -= withdrawn;
    }

    uint256 floatRequired =
        _calculateTotalAssets(currentBalance, collateral, invested, debt).mulWadDown(floatPercentage);
    uint256 excessUsdc = currentBalance > floatRequired ? currentBalance - floatRequired : 0;

    // 2. deposit excess usdc as collateral
    if (excessUsdc >= rebalanceMinimum) {
        aavePool.supply(address(asset), excessUsdc, address(this), 0);
        collateral += excessUsdc;
        currentBalance -= excessUsdc;
    }

    // 3. rebalance to target ltv
    uint256 targetDebt = getWethFromUsdc(collateral.mulWadDown(targetLtv));
    uint256 delta = debt > targetDebt ? debt - targetDebt : targetDebt - debt;

    if (delta <= targetDebt.mulWadDown(DEBT_DELTA_THRESHOLD)) return;

    if (debt > targetDebt) {
        _disinvest(delta);
        aavePool.repay(address(weth), delta, C.AAVE_VAR_INTEREST_RATE_MODE, address(this));
    } else {
        aavePool.borrow(address(weth), delta, C.AAVE_VAR_INTEREST_RATE_MODE, 0, address(this));
        scWETH.deposit(delta, address(this));
    }

    emit Rebalanced(
        targetLtv, debt, targetDebt, collateral - excessUsdc, collateral, initialBalance, currentBalance
    );
}
```

https://github.com/lindy-labs/sandclock-contracts/blob/f3a62502638d9de496ad05df381fc56cae59d0e8/src/steth/scUSDC.sol#L405-L418

```
function _swapWethForUsdc(uint256 _wethAmount) internal returns (uint256) {
    ISwapRouter.ExactInputSingleParams memory params = ISwapRouter.ExactInputSingleParams({
        tokenIn: address(weth),
        tokenOut: address(asset),
        fee: 500,
        recipient: address(this),
        deadline: block.timestamp,
        amountIn: _wethAmount,
        amountOutMinimum: getUsdcFromWeth(_wethAmount).mulWadDown(slippageTolerance),
        sqrtPriceLimitX96: 0
    });

    return swapRouter.exactInputSingle(params);
}
```

We believe that the `rebalance()` function is intended to rebalance the position if the health factor of the Aave position is low, rather than using `exitAllPositions()` .

The `rebalance()` function withdraws funds from scWETH to pay off a portion of the loan for deleveraging.

However, there is one step in the rebalance procedure that involves swapping WETH to USDC (`_swapWethForUsdc()` called at L165). Currently, an admin-configured `slippageTolerance` is used to calculate and determine the slippage control (`amountOutMinimum`).

If the `slippageTolerance` is set to a lower value than the current pool states offer (WETHUSDC 0.05%), then the swap is likely to fail, resulting in the entire `rebalance()` transaction failing.

If this occurs during a low health factor situation and the much-needed rebalance cannot be completed, it may result in the liquidation of the Aave position.

Recommendation

Consider allowing the keeper to specify a `slippageTolerance` or `amountOutMinimum` using a parameter of the `rebalance()` function.

Status

✓ Fixed

[WP-M4] `_disinvest(delta)` may not withdrawn extact `delta` , as a result `rebalance()` may revert

Medium

Issue Description

This issue has been fixed before the report is out.

https://github.com/lindy-labs/sandclock-contracts/blob/f3a62502638d9de496ad05df381fc56cae59d0e8/src/steth/scUSDC.sol#L154-L197

```
function rebalance() public onlyKeeper {
@@ 155,184 @@

    if (debt > targetDebt) {
        _disinvest(delta);
        aavePool.repay(address(weth), delta, C.AAVE_VAR_INTEREST_RATE_MODE, address(this));
    } else {
        aavePool.borrow(address(weth), delta, C.AAVE_VAR_INTEREST_RATE_MODE, 0, address(this));
        scWETH.deposit(delta, address(this));
    }

    emit Rebalanced(
        targetLtv, debt, targetDebt, collateral - excessUsdc, collateral, initialBalance, currentBalance
    );
}
```

https://github.com/lindy-labs/sandclock-contracts/blob/f3a62502638d9de496ad05df381fc56cae59d0e8/src/steth/scUSDC.sol#L399-L403

```
function _disinvest(uint256 _wethAmount) internal returns (uint256 amountWithdrawn) {
    uint256 shares = scWETH.convertToShares(_wethAmount);

    amountWithdrawn = scWETH.redeem(shares, address(this), address(this));
}
```

Status

✓ Fixed

[WP-L5] Too much precision loss in `getUsdcFromWeth()`

Low

Issue Description

C.WETH_USDC_DECIMALS_DIFF = 1e12

https://github.com/lindy-labs/sandclock-contracts/blob/f3a62502638d9de496ad05df381fc56cae59d0e8/src/steth/scUSDC.sol#L264-L268

```
264  function getUsdcFromWeth(uint256 _wethAmount) public view returns (uint256) {
265      (, int256 usdcPriceInWeth,,,) = usdcToEthPriceFeed.latestRoundData();
266
267      return (_wethAmount /
     C.WETH_USDC_DECIMALS_DIFF).divWadDown(uint256(usdcPriceInWeth));
268  }
```

There is a significant amount of precision loss in `_wethAmount / C.WETH_USDC_DECIMALS_DIFF` especially when `_wethAmount` is low. As the constant `WETH_USDC_DECIMALS_DIFF` is a rather big number of `1e12` .

Recommendation

```
264  function getUsdcFromWeth(uint256 _wethAmount) public view returns (uint256) {
265      (, int256 usdcPriceInWeth,,,) = usdcToEthPriceFeed.latestRoundData();
266
267      return _wethAmount.divWadDown(uint256(usdcPriceInWeth) *
     C.WETH_USDC_DECIMALS_DIFF);
268  }
```

Status

✓ Fixed

[WP-L6] The user should specify the slippage control as they will bear the swap cost

Low

Issue Description

https://github.com/lindy-labs/sandclock-contracts/blob/f3a62502638d9de496ad05df381fc56cae59d0e8/src/steth/scWETH.sol#L243-L266

```solidity
function redeem(uint256 shares, address receiver, address owner) public override returns (uint256 assets) {
    if (msg.sender != owner) {
        uint256 allowed = allowance[owner][msg.sender]; // Saves gas for limited approvals.

        if (allowed != type(uint256).max) allowance[owner][msg.sender] = allowed - shares;
    }

    // Check for rounding error since we round down in previewRedeem.
    require((assets = previewRedeem(shares)) != 0, "ZERO_ASSETS");

    beforeWithdraw(assets, shares);

    _burn(owner, shares);

    uint256 balance = asset.balanceOf(address(this));

    if (assets > balance) {
        assets = balance;
    }

    emit Withdraw(msg.sender, receiver, owner, assets, shares);

    asset.safeTransfer(receiver, assets);
}
```

https://github.com/lindy-labs/sandclock-contracts/blob/

f3a62502638d9de496ad05df381fc56cae59d0e8/src/steth/scWETH.sol#L411-L420

```
function beforeWithdraw(uint256 assets, uint256) internal override {
    uint256 float = asset.balanceOf(address(this));
    if (assets <= float) {
        return;
    }

    uint256 missing = (assets - float);

    _withdrawToVault(missing);
}
```

https://github.com/lindy-labs/sandclock-contracts/blob/f3a62502638d9de496ad05df381fc56cae59d0e8/src/steth/scWETH.sol#L366-L383

```
function _withdrawToVault(uint256 amount) internal {
    uint256 debt = getDebt();
    uint256 collateral = getCollateral();

    uint256 flashLoanAmount = amount.mulDivDown(debt, collateral - debt);

    address[] memory tokens = new address[](1);
    tokens[0] = address(weth);

    uint256[] memory amounts = new uint256[](1);
    amounts[0] = flashLoanAmount;

    // needed otherwise counted as loss during harvest
    totalInvested -= amount;

    // take flashloan
    balancerVault.flashLoan(address(this), tokens, amounts, abi.encode(false, amount));
}
```

https://github.com/lindy-labs/sandclock-contracts/blob/f3a62502638d9de496ad05df381fc56cae59d0e8/src/steth/scWETH.sol#L273-L328

```solidity
function receiveFlashLoan(address[] memory, uint256[] memory amounts, uint256[] memory, bytes memory userData)
    external
{
    if (msg.sender != address(balancerVault)) {
        revert InvalidFlashLoanCaller();
    }

    // the amount flashloaned
    uint256 flashLoanAmount = amounts[0];

    // decode user data
    (bool isDeposit, uint256 amount) = abi.decode(userData, (bool, uint256));

    amount += flashLoanAmount;

    // if flashloan received as part of a deposit
@@ 289,305 @@
    else {
        // repay debt + withdraw collateral
        if (flashLoanAmount >= getDebt()) {
            aavePool.repay(address(weth), type(uint256).max, C.AAVE_VAR_INTEREST_RATE_MODE, address(this));
            aavePool.withdraw(address(wstETH), type(uint256).max, address(this));
        } else {
            aavePool.repay(address(weth), flashLoanAmount, C.AAVE_VAR_INTEREST_RATE_MODE, address(this));
            aavePool.withdraw(address(wstETH), _ethToWstEth(amount), address(this));
        }

        // unwrap wstETH
        uint256 stEthAmount = wstETH.unwrap(wstETH.balanceOf(address(this)));

        // stETH to eth
        curvePool.exchange(1, 0, stEthAmount, _stEthToEth(stEthAmount).mulWadDown(slippageTolerance));

        // wrap eth
        weth.deposit{value: address(this).balance}();
    }

    // payback flashloan
    asset.safeTransfer(address(balancerVault), flashLoanAmount);
}
```

While redeeming shares into underlying assets, exchanging from `stETH` to `ETH` using `curvePool.exchange()` may result in partial loss due to slippage.

The user is bearing the loss caused by slippage. Therefore, they should be able to specify the slippage themselves instead of having it specified for them.

Otherwise, the attacker may sandwich their `redeem()` transaction and force them to pay for the max slippage.

The same case in `scUSDC.sol#redeem()` , `_disinvest()` may incur a slippage cost. Therefore, there should be a parameter for the user to specify the maximum slippage.

https://github.com/lindy-labs/sandclock-contracts/blob/f3a62502638d9de496ad05df381fc56cae59d0e8/src/steth/scUSDC.sol#L338-L362

```
338  function beforeWithdraw(uint256 _assets, uint256) internal override {
339      uint256 initialBalance = getUsdcBalance();
340      if (initialBalance >= _assets) return;
341
342      uint256 collateral = getCollateral();
343      uint256 debt = getDebt();
344      uint256 invested = getInvested();
345      uint256 total = _calculateTotalAssets(initialBalance, collateral, invested,
     debt);
346      uint256 profit = _calculateWethProfit(invested, debt);
347      uint256 floatRequired = total > _assets ? (total -
     _assets).mulWadUp(floatPercentage) : 0;
348      uint256 usdcNeeded = _assets + floatRequired - initialBalance;
349
350      // first try to sell profits to cover withdrawal amount
351      if (profit != 0) {
352          uint256 withdrawn = _disinvest(profit);
353          uint256 usdcReceived = _swapWethForUsdc(withdrawn);
354
355            if (initialBalance + usdcReceived >= _assets) return;
356
357            usdcNeeded -= usdcReceived;
358        }
359
360        // if we still need more usdc, we need to repay debt and withdraw collateral
361        _repayDebtAndReleaseCollateral(debt, collateral, invested, usdcNeeded);
362    }
```

https://github.com/lindy-labs/sandclock-contracts/blob/f3a62502638d9de496ad05df381fc56cae59d0e8/src/steth/scUSDC.sol#L364-L376

```
364    function _repayDebtAndReleaseCollateral(uint256 _debt, uint256 _collateral, uint256 _invested, uint256 _usdcNeeded)
365        internal
366    {
367        // handle rounding errors when withdrawing everything
368        _usdcNeeded = _usdcNeeded > _collateral ? _collateral : _usdcNeeded;
369        // to keep the same ltv, weth debt to repay has to be proportional to collateral withdrawn
370        uint256 wethNeeded = _usdcNeeded.mulDivUp(_debt, _collateral);
371        wethNeeded = wethNeeded > _invested ? _invested : wethNeeded;
372
373        uint256 withdrawn = _disinvest(wethNeeded);
374        aavePool.repay(address(weth), withdrawn, C.AAVE_VAR_INTEREST_RATE_MODE, address(this));
375        aavePool.withdraw(address(asset), _usdcNeeded, address(this));
376    }
```

Status

ⓘ Acknowledged

[WP-L7] Frontrun the oracle price update with mint/redeem can profit from the sudden change of price per share

Low

Issue Description

https://github.com/lindy-labs/sandclock-contracts/blob/162c7e5b13907e28560e3e9a3d8c17a3d5ba60ee/src/steth/scUSDC.sol#L378-L393

```
function _calculateTotalAssets(uint256 _float, uint256 _collateral, uint256 _invested, uint256 _debt)
    internal
    view
    returns (uint256 total)
{
    total = _float + _collateral;

    uint256 profit = _calculateWethProfit(_invested, _debt);

    if (profit != 0) {
        // account for slippage when selling weth profits
        total += getUsdcFromWeth(profit).mulWadDown(slippageTolerance);
    } else {
        total -= getUsdcFromWeth(_debt - _invested);
    }
}
```

https://github.com/lindy-labs/sandclock-contracts/blob/162c7e5b13907e28560e3e9a3d8c17a3d5ba60ee/src/steth/scUSDC.sol#L264-L268

```
function getUsdcFromWeth(uint256 _wethAmount) public view returns (uint256) {
    (, int256 usdcPriceInWeth,,,) = usdcToEthPriceFeed.latestRoundData();

    return (_wethAmount / C.WETH_USDC_DECIMALS_DIFF).divWadDown(uint256(usdcPriceInWeth));
}
```

Whenever the usdcToEthPriceFeed updates, it brings a chance for MEV. If the oracle price of ETH goes up, then the price per share will also rise.

The MEV attacker can frontrun deposit and backrun withdraw to make a profit.

Recommendation

Consider using a simpler approach for burn/mint which always adds and removes USDC and ETH proportionally so that the oracle price of ETH can’t impact the results.

Status

ⓘ **Acknowledged**

[WP-L8] `exitAllPositions()` should specify a slippage control to prevent MEV / sandwich attack

Low

Issue Description

https://github.com/lindy-labs/sandclock-contracts/blob/f3a62502638d9de496ad05df381fc56cae59d0e8/src/steth/scUSDC.sol#L202-L221

```
202 function exitAllPositions() external onlyAdmin {
203     uint256 debt = getDebt();
204 
205     if (getInvested() >= debt) {
206         revert VaultNotUnderwater();
207     }
208 
209     uint256 wethBalance = scWETH.redeem(scWETH.balanceOf(address(this)), address(this), address(this));
210     uint256 collateral = getCollateral();
211 
212     address[] memory tokens = new address[](1);
213     tokens[0] = address(weth);
214 
215     uint256[] memory amounts = new uint256[](1);
216     amounts[0] = debt - wethBalance;
217 
218     balancerVault.flashLoan(address(this), tokens, amounts, abi.encode(collateral, debt));
219 
220     emit EmergencyExitExecuted(msg.sender, wethBalance, debt, collateral);
221 }
```

https://github.com/lindy-labs/sandclock-contracts/blob/f3a62502638d9de496ad05df381fc56cae59d0e8/src/steth/scUSDC.sol#L227-L258

```
227 function receiveFlashLoan(address[] memory, uint256[] memory amounts, uint256[] memory, bytes memory userData)
228     external
```

```
{
    if (msg.sender != address(balancerVault)) {
        revert InvalidFlashLoanCaller();
    }

    uint256 flashLoanAmount = amounts[0];
    (uint256 collateral, uint256 debt) = abi.decode(userData, (uint256, uint256));

    aavePool.repay(address(weth), debt, C.AAVE_VAR_INTEREST_RATE_MODE, address(this));
    aavePool.withdraw(address(asset), collateral, address(this));

    asset.approve(address(swapRouter), type(uint256).max);

    ISwapRouter.ExactOutputSingleParams memory params = ISwapRouter.ExactOutputSingleParams({
        tokenIn: address(asset),
        tokenOut: address(weth),
        fee: 500,
        recipient: address(this),
        deadline: block.timestamp,
        amountOut: flashLoanAmount,
        amountInMaximum: type(uint256).max, // ignore slippage
        sqrtPriceLimitX96: 0
    });

    swapRouter.exactOutputSingle(params);

    asset.approve(address(swapRouter), 0);

    weth.safeTransfer(address(balancerVault), flashLoanAmount);
}
```

Recommendation

Consider adding slippage control parameters to `exitAllPositions()` and requiring the final balances to be greater than the specified minimal amounts.

Status

✓ Fixed

[WP-L9] Using the same oracle for WETH price to ensure the LTV is align with the health factor of the AAVE position

Low

Issue Description

https://github.com/lindy-labs/sandclock-contracts/blob/f3a62502638d9de496ad05df381fc56cae59d0e8/src/steth/scUSDC.sol#L312-L321

```
function getLtv() public view returns (uint256) {
    uint256 debt = getDebt();

    if (debt == 0) return 0;

    uint256 debtPriceInUsdc = getUsdcFromWeth(debt);

    // totalDebt / totalSupplied
    return debtPriceInUsdc.divWadUp(getCollateral());
}
```

https://github.com/lindy-labs/sandclock-contracts/blob/f3a62502638d9de496ad05df381fc56cae59d0e8/src/steth/scUSDC.sol#L264-L268

```
function getUsdcFromWeth(uint256 _wethAmount) public view returns (uint256) {
    (, int256 usdcPriceInWeth,,,) = usdcToEthPriceFeed.latestRoundData();

    return (_wethAmount / C.WETH_USDC_DECIMALS_DIFF).divWadDown(uint256(usdcPriceInWeth));
}
```

The price oracle provider address on Aave is configurable, while the `usdcToEthPriceFeed` on `scUSDC.sol` is immutable.

If Aave updates their oracle provider, there is a chance that the LTV calculated using `usdcToEthPriceFeed` may be different from that calculated based on their oracle feed.

Recommendation

Consider using Aave’s LTV for better accuracy.

Status

✓ Fixed

[WP-L10] When calculating `totalAssets()` , the `performanceFee` belonging to `treasury` should be excluded

Low

Issue Description

https://github.com/lindy-labs/sandclock-contracts/blob/f3a62502638d9de496ad05df381fc56cae59d0e8/src/steth/scWETH.sol#L137-L159

```
function harvest() external onlyKeeper {
    // reinvest
    _rebalancePosition();

    // store the old total
    uint256 oldTotalInvested = totalInvested;
    uint256 assets = totalAssets();

    if (assets > oldTotalInvested) {
        totalInvested = assets;

        // profit since last harvest, zero if there was a loss
        uint256 profit = assets - oldTotalInvested;
        totalProfit += profit;

        uint256 fee = profit.mulWadDown(performanceFee);

        // mint equivalent amount of tokens to the performance fee beneficiary ie the treasury
        _mint(treasury, convertToShares(fee));

        emit Harvest(profit, fee);
    }
}
```

When calculating `totalAssets()` , the pending `performanceFee` that belongs to the `treasury` must be excluded.

Otherwise, every time `harvest()` is called, it will cause a sudden drop in the price per share.

This is because new shares will be minted to `treasury` without increasing the `totalAssets` .

Status

ⓘ **Acknowledged**

Appendix

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by WatchPug; however, WatchPug does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Smart Contract technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.