Sturdy contest
Findings & Analysis Report

2022-06-29

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit contest outlined in this document, C4 conducted an analysis of the Sturdy smart contract system written in Solidity. The audit contest took place between May 13—May 15 2022.

Wardens

73 Wardens contributed reports to the Sturdy contest:

1. Picodes
2. hyh
3. berndartmueller
4. WatchPug (jtp and ming)
5. IIIIIII
6. sorrynotsorry
7. jonah1005

8. StErMi
9. leastwood
10. MaratCerby
11. mtz
12. Dravee
13. rotcivegaf
14. 0x52
15. dipp
16. cccz
17. AuditsAreUS
18. defsec
19. 0x1f8b
20. simon135
21. robee
22. 0xliumin
23. 0xNazgul
24. oyc_109
25. joestakey
26. 0xf15ers (remora and twojoy)
27. hake
28. 0x4non
29. fatherOfBlocks
30. GimelSec (rayn and sces60107)
31. Funen
32. p4st13r4 (0x69e8 and 0xb4bb4)
33. 0xkatana
34. Hawkeye (0xwags and 0xmint)
35. Waze
36. sikorico

37. delfin454000
38. mics
39. bobirichman
40. kebabsec (okkothejawa and FlameHorizon)
41. TerrierLover
42. BouSalman
43. csanuragjain
44. p_crypt0
45. tintin
46. cryptphi
47. AlleyCat
48. z3s
49. hansfriese
50. Tomio
51. Cityscape
52. SooYa
53. ignacio
54. JC
55. Fitraldys
56. 0v3rf10w
57. samruna
58. CertoraInc (egjlmn1, OriDabush, ItayG, and shakedwinder)
59. isamjay
60. peritoflores
61. tabish
62. pedroais
63. saian
64. sseefried

This contest was judged by hickuphh3. The judge also competed in the contest as a warden, but forfeited their winnings.

Final report assembled by liveactionllama.

Summary

The C4 analysis yielded an aggregated total of 8 unique vulnerabilities. Of these vulnerabilities, 2 received a risk rating in the category of HIGH severity and 6 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 43 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 40 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the C4 Sturdy contest repository, and is composed of 5 smart contracts written in the Solidity programming language and includes 366 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website.

High Risk Findings (2)

[H-01] Hard-coded slippage may freeze user funds during market turbulence

Submitted by jonah1005, also found by berndartmueller, Picodes, IIIIIII, sorrynotsorry, and WatchPug

GeneralVault.sol#L125
GeneralVault set a hardcoded slippage control of 99%. However, the underlying yield tokens price may go down.
If Luna/UST things happen again, users' funds may get locked.

LidoVault.sol#L130-L137
Moreover, the withdrawal of the lidoVault takes a swap from the curve pool. 1 stEth worth 0.98 ETH at the time of writing.
The vault can not withdraw at the current market.

Given that users' funds would be locked in the lidoVault, I consider this a high-risk issue.

Proof of Concept

1 stEth = 0.98 Eth

LidoVault.sol#L130-L137

Recommended Mitigation Steps

There are different ways to set the slippage.

The first one is to let users determine the maximum slippage they're willing to take. The protocol front-end should set the recommended value for them.

```
function withdrawCollateral(
  address _asset,
  uint256 _amount,
  address _to,
  uint256 _minReceiveAmount
) external virtual {
    // ...
  require(withdrawAmount >= _minReceiveAmount, Errors.VT_WITHI
}
```

The second one is have a slippage control parameters that's set by the operator.

```
  // Exchange stETH -> ETH via Curve
  uint256 receivedETHAmount = CurveswapAdapter.swapExactTokens
    _addressesProvider,
    _addressesProvider.getAddress('STETH_ETH_POOL'),
    LIDO,
    ETH,
    yieldStETH,
    maxSlippage
  );
```

```
  function setMaxSlippage(uint256 _slippage) external onlyOper
      maxSlippage = _slippage;

      //@audit This action usually emit an event.
      emit SetMaxSlippage(msg.sender, slippage);
  }
```

These are two common ways to deal with this issue. I prefer the first one.
The market may corrupt really fast before the operator takes action.
It's nothing fun watching the number go down while having no option.

sforman2000 (Sturdy) confirmed

iris112 (Sturdy) commented:

> Fix the issue of require 99% of withdrawAmount sturdyfi/code4rena-may-2022#11

hickuphh3 (judge) commented:

> I realise there are 2 issues discussed here:

1. The high-risk severity relates to `GeneralVault`'s tight 1% slippage. Because it is inherited by vaults, it can cause withdrawals to fail and for user funds to be stuck.
2. However, in the context of the LIDO vault specifically, #69's first low-severity issue rightly points out that users can choose to withdraw their funds in stETH, then do conversion to ETH separately afterwards. Hence, funds won't actually be stuck. I would've therefore classified this a medium-severity issue. Nevertheless, it is expected that users will attempt to withdraw to ETH instead of stETH in times of market volatility.

[H-02] The check for value transfer success is made after the return statement in `_withdrawFromYieldPool` of `LidoVault`

Submitted by pedroais, also found by 0x52, 0xliumin, cccz, CertoraInc, fatherOfBlocks, GimelSec, hake, hickuphh3, hyh, IIIIIII, isamjay, mtz, oyc_109, p4st13r4, peritoflores, rotcivegaf, sorrynotsorry, StErMi, tabish, WatchPug, z3s, 0x4non, 0xf15ers, berndartmueller, dipp, Dravee, MaratCerby, saian, simon135, sseefried, and TerrierLover

Users can lose their funds

Proof of Concept

LidoVault.sol#L142

The code checks transaction success after returning the transfer value and finishing execution. If the call fails the transaction won't revert since require(sent, Errors.VT*COLLATERAL*WITHDRAW_INVALID); won't execute.

Users will have withdrawn without getting their funds back.

Recommended Mitigation Steps

Return the function after the success check

sforman2000 (Sturdy) confirmed

iris112 (Sturdy) commented:

> Fix the issue of return before require sturdyfi/code4rena-may-2022#9

hickuphh3 (judge) commented:

> Issue is rather clear-cut.

Medium Risk Findings (6)

[M-01] Possible lost msg.value

Submitted by rotcivegaf, also found by AuditsAreUS, berndartmueller, cccz, MaratCerby, dipp, and StErMi

GeneralVault.sol#L75-L89
LidoVault.sol#L79-L104
ConvexCurveLPVault.sol#L131-L149

Possible lost value in `depositCollateral` function call

Proof of Concept

In call `depositCollateral` can will send value and the asset can be an ERC20(!= address(0)), if `LidoVault` and `ConvexCurveLPVault` contract receive this call the fouds will lost.
Also in **LidoVault, L88**, if send as asset ETH(== address(0)) and send more value than `_amount` (msg.value > _amount), the exedent will lost.

Recommended Mitigation Steps

In **GeneralVault**, `depositCollateral` function:

- Check if the `msg.value` is zero when the `_asset` is ERC20(!= address(0))

- Check if the `msg.value` is equeal to `_amount` when the `_asset` ETH(== address(0))

```
function depositCollateral(address _asset, uint256 _amount) exte
  if (_asset != address(0)) { // asset = ERC20
    require(msg.value == 0, <AN ERROR FROM Errors LIBRARY>);
  } else { // asset = ETH
    require(msg.value == _amount, <AN ERROR FROM Errors LIBRARY>
  }

  // Deposit asset to vault and receive stAsset
  // Ex: if user deposit 100ETH, this will deposit 100ETH to Lic
  (address _stAsset, uint256 _stAssetAmount) = _depositToYieldPo

  // Deposit stAsset to lendingPool, then user will get aToken c
  ILendingPool(_addressesProvider.getLendingPool()).deposit(
    _stAsset,
    _stAssetAmount,
    msg.sender,
    0
  );

  emit DepositCollateral(_asset, msg.sender, _amount);
}
```

Also can remove the `require(msg.value > 0, Errors.VT_COLLATERAL_DEPOSIT_REQUIRE_ETH);` in **LidoVault, L88**

sforman2000 (Sturdy) confirmed

atozICT20 (Sturdy) commented:

> Fix the issue of ETH loss sturdyfi/code4rena-may-2022#3

[M-02] `UNISWAP_FEE` is hardcoded which will lead to significant losses compared to optimal routing

Submitted by Picodes, also found by hickuphh3

In `YieldManager`, `UNISWAP_FEE` is hardcoded, which reduce significantly the possibilities and will lead to non optimal routes. In particular, all swaps using ETH path will use the wrong pool as it will use the ETH / USDC 1% one due to this line.

Proof of Concept

For example for CRV / USDC, the optimal route is currently CRV -> ETH and ETH -> USDC, and the pool ETH / USDC with 1% fees is tiny compared to the ones with 0.3 or 0.1%. Therefore using the current implementation would create a significant loss of revenue.

Recommended Mitigation Steps

Basic mitigation would be to hardcode in advance the best Uniswap paths in a mapping like it's done for Curve pools, then pass this path already computed to the swapping library. This would allow for complex route and save gas costs as you would avoid computing them in `swapExactTokensForTokens`.

Then, speaking from experience, as `distributeYield` is `onlyAdmin`, you may want to add the possibility to do the swaps through an efficient aggregator like 1Inch or Paraswap, it will be way more optimal.

sforman2000 (Sturdy) confirmed

atozICT20 (Sturdy) commented:

> Fix the issue of hardcoded UNISWAP_FEE sturdyfi/code4rena-may-2022#12

[M-03] `processYield()` and `distributeYield()` may run out of gas and revert due to long list of extra rewards/yields

Submitted by lllllll, also found by leastwood and StErMi

Yields will not be able to be distributed to lenders because attempts to do so will revert.

Proof of Concept

The `processYield()` function loops overall of the extra rewards and transfers them

```
File: smart-contracts/ConvexCurveLPVault.sol   #1

105       uint256 extraRewardsLength = IConvexBaseRewardPool(bas
106       for (uint256 i = 0; i < extraRewardsLength; i++) {
107         address _extraReward = IConvexBaseRewardPool(baseRew
108         address _rewardToken = IRewards(_extraReward).rewarc
109         _transferYield(_rewardToken);
110       }
```

[ConvexCurveLPVault.sol#L105-L110]

There is no guarantee that the tokens involved will be efficient in their use of gas, and there are no upper bounds on the number of extra rewards:

```
    function extraRewardsLength() external view returns (uint256
        return extraRewards.length;
    }


    function addExtraReward(address _reward) external returns(bo
        require(msg.sender == rewardManager, "!authorized");
        require(_reward != address(0),"!reward setting");


        extraRewards.push(_reward);
        return true;
    }
```

[BaseRewardPool.sol#L105-L115]

Even if not every extra reward token has a balance, an attacker can sprinkle each one with dust, forcing a transfer by this function

`_getAssetYields()` has a similar issue:

```
File: smart-contracts/YieldManager.sol   #X
```

```
129       AssetYield[] memory assetYields = _getAssetYields(excl
130       for (uint256 i = 0; i < assetYields.length; i++) {
131         if (assetYields[i].amount > 0) {
132           uint256 _amount = _convertToStableCoin(assetYields
133           // 3. deposit Yield to pool for suppliers
134           _depositYield(assetYields[i].asset, _amount);
135         }
136       }
```

YieldManager.sol#L129-L136

Recommended Mitigation Steps

Include an offset and length as is done in `YieldManager.distributeYield()`.

sforman2000 (Sturdy) confirmed

atozICT20 (Sturdy) commented:

> Fix the issue of processYield()'s run out of gas due to convex's extra rewards sturdyfi/code4rena-may-2022#4

hickuphh3 (judge) commented:

> I've considered this issue. The reason why I chose not to raise it up is because adding reward tokens is restricted on Convex's side. Given the number of integrations they have, it's unlikely that they will add too many tokens or gas-guzzling ones that will cause claims to run OOG.

> Nevertheless, it is a possibility, albeit an unlikely one, so I'll let the issue stand (also since the sponsor confirmed it).

[M-04] ConvexCurveLPVault's `_transferYield` can become stuck with zero reward transfer

Submitted by hyh

Now there are no checks for the amounts to be transferred via _transferYield and _processTreasury. As reward token list is external and an arbitrary token can end up there, in the case when such token doesn't allow for zero amount transfers, the reward retrieval can become unavailable.

I.e. processYield() can be fully blocked for even an extended period, with some low probability, which cannot be controlled otherwise as pool reward token list is external.

Setting the severity to medium as reward gathering is a base functionality for the system and its availability is affected.

Proof of Concept

_transferYield proceeds with sending the amounts to treasury and yieldManager without checking:

ConvexCurveLPVault.sol#L74-L82

```
    // transfer to treasury
    if (_vaultFee > 0) {
      uint256 treasuryAmount = _processTreasury(_asset, yieldAmo
      yieldAmount = yieldAmount.sub(treasuryAmount);
    }

    // transfer to yieldManager
    address yieldManager = _addressesProvider.getAddress('YIELD_
    TransferHelper.safeTransfer(_asset, yieldManager, yieldAmour
```

ConvexCurveLPVault.sol#L205-L209

```
  function _processTreasury(address _asset, uint256 _yieldAmount
    uint256 treasuryAmount = _yieldAmount.percentMul(_vaultFee);
    IERC20(_asset).safeTransfer(_treasuryAddress, treasuryAmount
    return treasuryAmount;
  }
```

The incentive token can be arbitrary. Some ERC20 do not allow zero amounts to be sent:

https://github.com/d-xo/weird-erc20#revert-on-zero-value-transfers

In a situation of such a token added to reward list and zero incentive amount earned the whole processYield call will revert, making reward gathering unavailable until either such token be removed from pool's reward token list or some non-zero reward amount be earned. Both are external processes and aren't controllable.

Recommended Mitigation Steps

Consider running the transfers in _transferYield only when yieldAmount is positive:

```
+       if (yieldAmount > 0) {
            // transfer to treasury
            if (_vaultFee > 0) {
              uint256 treasuryAmount = _processTreasury(_asset,
              yieldAmount = yieldAmount.sub(treasuryAmount);
            }

            // transfer to yieldManager
            address yieldManager = _addressesProvider.getAddress
            TransferHelper.safeTransfer(_asset, yieldManager, yi
+   }
```

sforman2000 (Sturdy) confirmed

atozICT20 (Sturdy) commented:

> Fix the issue of some ERC20 tokens revert on zero value transfer
> sturdyfi/code4rena-may-2022#6

hickuphh3 (judge) commented:

> Agree with issue, there have been tokens that require the transfer amount to be non-zero. The other enabler is that the reward token list is arbitrary.

[M-05] Withdrawing ETH collateral with max uint256 amount value reverts transaction

Submitted by berndartmueller, also found by WatchPug

Withdrawing ETH collateral via the `withdrawCollateral` function using `type(uint256).max` for the `_amount` parameter reverts the transaction due to `_asset` being the zero-address and `IERC20Detailed(_asset).decimals()` not working for native ETH.

Proof of Concept

GeneralVault.sol#L121-L124

```
    if (_amount == type(uint256).max) {
        uint256 decimal = IERC20Detailed(_asset).decimals(); // @auc
        _amount = _amountToWithdraw.mul(this.pricePerShare()).div(10(
    }
```

Recommended mitigation steps

Check `_asset` and use hard coded decimal value (`18`) for native ETH.

sforman2000 (Sturdy) confirmed

atozICT20 (Sturdy) commented:

> Fix the issue of transaction fails due to calculate ETH's decimals sturdyfi/code4rena-may-2022#7

hickuphh3 (judge) commented:

> Good find! Stated in `_asset` description that null address is interpreted as ETH, which isn't a token, and therefore reverts when attempting to fetch its decimals.

[M-06] Yield can be unfairly divided because of MEV/Just-in-time stablecoin deposits

Submitted by mtz, also found by 0x52, hyh, jonah1005, leastwood, and sorrynotsorry

YieldManager.sol#L129-L134
YieldManager.sol#L160-L161

An attacker can use MEV (via gas auction or Flashbots or control of miners) to cause an unfair division of yield. By providing a very large (relative to the size of all other stablecoin deposits combined) stablecoin deposit Just-in-Time before an admin's call to **distributeYield** the stablecoin deposited by the attacker will receive a very large amount of the yield and the attacker can immediately withdraw their deposit after yield is distributed. We assume this allows an attacker to get a lot of the yield reward even though they haven't provided any deposit that has been borrowed. However, the exact mechanism for how yield is distributed to lenders of a particular stablecoin is in LendingPool.sol, which is out of scope. However it is implied in **the documentation of this repo** that it is based on the balance of that asset the lender has provided. We have confirmed that **in LendingPool.sol the yield is distributed based on the proportion of the asset provided**. However, even ignoring this, MEV can still be used to unfairly hurt lenders of other stablecoins.

Proof of Concept

1. An attacker watches the mempool for calls to **distributeYield** by the admin.
2. The attacker orders the block's transactions (most easily using a flashbots bundle) in the following order:
   i. Attacker deposits stablecoins to lend (ideally the stablecoin will be the one with the least volume).
   ii. admin's call to distributeYield happens.
   iii. Attacker withdraws their deposit.

The attacker has thus made the asset they deposited (and thus themselves) receive much of the yield even though they provide no value to Sturdy since none of their deposit is ever borrowed so the never do anything to earn yield for sturdy. This attack can be done by a whale or by borrowing (even from sturdy) assets and converting them to a stablecoins accepted by sturdy before i. and returning them after iii. This will essentially be cost free for the attacker, none of their capital will ever be tied up by borrowers.

Recommended Mitigation Steps

The simplest way to mitigate this is for the admin to use flashbots or some other means of submitting the distributeYield call that skips the mempool. This is only a partial mitigation since attackers can still withdraw right after yield is distributed and get lucky by depositing soon before the distribution thus still capture more yield than they should have.
A better mitigation could use something like snapshotting who has deposited since the last yield distribution and only give these depositers yield based on the size of their deposits the next time yield is distributed.

sforman2000 (Sturdy) confirmed and commented:

> We will use flashbots and vary when/how often yield is harvested to mitigate this.

hickuphh3 (judge) decreased severity to Medium and commented:

> I take reference to discussions on Discord and in a thread below: https://github.com/code-423n4/2022-03-biconomy-findings/issues/135

> To quote from 0xleastwood: "Protocol leaked value in has a broad context but I think most judges can agree that it would pertain to rewards being paid out a lower rate than expected. Or, users can extract small amounts (up to debate on what is considered to be small) from the protocol under certain assumptions."

> Hence, as per the TLDR risk assessment:
> `2 — Med: Assets not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but external requirements.`

> I would downgrade this to a medium severity.

Low Risk and Non-Critical Issues

For this contest, 43 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by IIIIIII received the top score from the judge.

The following wardens also submitted reports: Dravee, sorrynotsorry, MaratCerby, 0x1f8b, defsec, 0xNazgul, robee, StErMi, BouSalman, mtz, p4st13r4, joestakey, oyc_109, berndartmueller, rotcivegaf, csanuragjain, 0xf15ers, hyh, dipp, hake, Hawkeye, WatchPug, delfin454000, sikorico, mics, 0xliumin, TerrierLover, 0x4non, p_crypt0, fatherOfBlocks, 0xkatana, GimelSec, bobirichman, tintin, Picodes, cryptphi, hickuphh3, Waze, Funen, AlleyCat, kebabsec, *and* simon135.

Summary

See original submission for details.

[L-01] Mistaken null values cause `distributeYield()` to revert

There are no null checks in the `registerAsset()` function, so admins can mistakenly pass 0x0 to that function, which will cause the for-loop to revert when that asset is reached. I've marked this as 'Low' rather than 'Medium' since the admin can work around it by using the `_offset` and `_count` input arguments to the function.

There is 1 instance of this issue:

```
File: smart-contracts/YieldManager.sol   #1

118    function distributeYield(uint256 _offset, uint256 _count)
119      // 1. convert from asset to exchange token via uniswap
120      for (uint256 i = 0; i < _count; i++) {
121        address asset = _assetsList[_offset + i];
122:       require(asset != address(0), Errors.UL_INVALID_INDEX)
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/YieldManager.sol#L118-L122

[L-02] Can't remove old assets

There is no way to remove old assets added by calls to `registerAsset()`. A disgruntled admin, before their access is revoked, can add a lot of assets and

regularly sprinkle them with dust, so the new admins have to submit multiple calls to `distributeYield()` with different offsets and counts, to avoid the dust and possible reversion due to running out of gas

There is 1 instance of this issue:

```
File: smart-contracts/YieldManager.sol   #1

118    function distributeYield(uint256 _offset, uint256 _count)
119      // 1. convert from asset to exchange token via uniswap
120      for (uint256 i = 0; i < _count; i++) {
121        address asset = _assetsList[_offset + i];
122        require(asset != address(0), Errors.UL_INVALID_INDEX)
123        uint256 _amount = IERC20Detailed(asset).balanceOf(adc
124        _convertAssetToExchangeToken(asset, _amount);
125:     }
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/YieldManager.sol#L118-L125

[L-03] Missing checks for `approve()` 's return status

Some tokens, such as Tether (USDT) return false rather than reverting if the approval fails. Use OpenZeppelin's `safeApprove()` , which reverts if there's a failure, instead

There is 1 instance of this issue:

```
File: smart-contracts/YieldManager.sol   #1

221:     IERC20(_asset).approve(_lendingPool, _amount);
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/YieldManager.sol#L221

[L-04] Move `ETH` constant to child contract

All of the functions in the `GeneralVault` require `0x0` when referring to Ether, not the constant here. Having it here will lead to mistakes down the line. It's only used by `CurveswapAdapter`, so it only needs to be there (it currently is also defined there).

There is 1 instance of this issue:

```
File: smart-contracts/GeneralVault.sol   #1

47:     address constant ETH = 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeee
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/GeneralVault.sol#L47

[L-05] Unsafe casts and usage of `IERC20Detailed`

The `GeneralVault` is meant to be general, i.e. not specific to Lido or Curve, and therefore should not assume that the asset will always be `IERC20Detailed` (not all ERC20 contracts define `decimals()` since it's optional in the spec). Use `safeDecimals()` instead

There is 1 instance of this issue:

```
File: smart-contracts/GeneralVault.sol   #1

122:         uint256 decimal = IERC20Detailed(_asset).decimals();
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/GeneralVault.sol#L122

[L-06] Unused `receive()` function will lock Ether in contract

If the intention is for the Ether to be used, the function should call another function, otherwise it should revert

There is 1 instance of this issue:

```
File: smart-contracts/LidoVault.sol   #1

24:     receive() external payable {}
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/LidoVault.sol#L24

[L-07] `safeApprove()` is deprecated

Deprecated in favor of `safeIncreaseAllowance()` and `safeDecreaseAllowance()`. If only setting the initial allowance to the value that means infinite, `safeIncreaseAllowance()` can be used instead

There are 3 instances of this issue:

```
File: smart-contracts/ConvexCurveLPVault.sol   #1

141:       IERC20(curveLPToken).safeApprove(convexBooster, _amour
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/ConvexCurveLPVault.sol#L141

```
File: smart-contracts/ConvexCurveLPVault.sol   #2

146:       IERC20(internalAssetToken).safeApprove(address(_addres
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/ConvexCurveLPVault.sol#L146

```
File: smart-contracts/LidoVault.sol   #3
```

```
102:        IERC20(LIDO).safeApprove(address(_addressesProvider.ge
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/LidoVault.sol#L102

[L-08] Missing checks for `address(0x0)` when assigning values to `address` state variables

There is 1 instance of this issue:

```
File: smart-contracts/ConvexCurveLPVault.sol    #1

41:         curveLPToken = _lpToken;
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/ConvexCurveLPVault.sol#L41

[L-09] Open TODOs

Code architecture, incentives, and error handling/reporting questions/issues should be resolved before deployment

There is 1 instance of this issue:

```
File: smart-contracts/GeneralVault.sol    #1

77:         // Ex: if user deposit 100ETH, this will deposit 100ET
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/GeneralVault.sol#L77

[N-01] `override` function arguments that are unused should have the variable name removed or commented out to avoid compiler warnings

There are 2 instances of this issue:

```
File: smart-contracts/ConvexCurveLPVault.sol   #1

154:     function _getWithdrawalAmount(address _asset, uint256 _a
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/ConvexCurveLPVault.sol#L154

```
File: smart-contracts/LidoVault.sol   #2

109:     function _getWithdrawalAmount(address _asset, uint256 _a
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/LidoVault.sol#L109

[N-02] `public` functions not called by the contract should be declared `external` instead

Contracts are allowed to override their parents' functions and change the visibility from `external` to `public`.

There are 3 instances of this issue:

```
File: smart-contracts/CollateralAdapter.sol   #1

35:     function initialize(ILendingPoolAddressesProvider _provi
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-

contracts/CollateralAdapter.sol#L35

```
File: smart-contracts/GeneralVault.sol   #2

61:     function initialize(ILendingPoolAddressesProvider _provi
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/GeneralVault.sol#L61

```
File: smart-contracts/YieldManager.sol   #3

60:     function initialize(ILendingPoolAddressesProvider _provi
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/YieldManager.sol#L60

[N-03] `constant`s should be defined rather than using magic numbers

There are 6 instances of this issue:

```
File: smart-contracts/ConvexCurveLPVault.sol

/// @audit 0xF403C135812408BFbE8713b5A23a04b3D48AAE31
40:         convexBooster = 0xF403C135812408BFbE8713b5A23a04b3D48A
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/ConvexCurveLPVault.sol#L40

```
File: smart-contracts/GeneralVault.sol

/// @audit 99_00
125:         require(withdrawAmount >= _amount.percentMul(99_00), E
```

```
/// @audit 30_00
167:       require(_fee <= 30_00, Errors.VT_FEE_TOO_BIG);
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/GeneralVault.sol#L125

```
File: smart-contracts/LidoVault.sol

/// @audit 200
48:          200

/// @audit 1e18
73:        return 1e18;

/// @audit 200
136:           200
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/LidoVault.sol#L48

[N-04] Redundant cast

The type of the variable is the same as the type to which the variable is being cast

There is 1 instance of this issue:

```
File: smart-contracts/LidoVault.sol   #1

140:         (bool sent, bytes memory data) = address(_to).call{v
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/LidoVault.sol#L140

[N-05] Missing event for critical parameter change

There are 2 instances of this issue:

```
File: smart-contracts/ConvexCurveLPVault.sol   #1

37      function setConfiguration(address _lpToken, uint256 _poo
38        require(internalAssetToken == address(0), Errors.VT_IN
39
40        convexBooster = 0xF403C135812408BFbE8713b5A23a04b3D48A
41        curveLPToken = _lpToken;
42        convexPoolId = _poolId;
43        SturdyInternalAsset _interalToken = new SturdyInternal
44          string(abi.encodePacked('Sturdy ', IERC20Detailed(_l
45          string(abi.encodePacked('c', IERC20Detailed(_lpToker
46          IERC20Detailed(_lpToken).decimals()
47        );
48        internalAssetToken = address(_interalToken);
49:     }
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/ConvexCurveLPVault.sol#L37-L49

```
File: smart-contracts/YieldManager.sol   #2

64      function setExchangeToken(address _token) external onlyA
65        require(_token != address(0), Errors.VT_INVALID_CONFIG
66        _exchangeToken = _token;
67:     }
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/YieldManager.sol#L64-L67

[N-06] Use a more recent version of solidity

Use a solidity version of at least 0.8.13 to get the ability to use `using for` with a list of free functions

There are 3 instances of this issue:

```
File: smart-contracts/GeneralVault.sol   #1

2:    pragma solidity 0.6.12;
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/GeneralVault.sol#L2

```
File: smart-contracts/LidoVault.sol   #2

2:    pragma solidity 0.6.12;
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/LidoVault.sol#L2

```
File: smart-contracts/YieldManager.sol   #3

2:    pragma solidity 0.6.12;
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/YieldManager.sol#L2

[N-07] Use a more recent version of solidity

Use a solidity version of at least 0.8.4 to get `bytes.concat()` instead of `abi.encodePacked(<bytes>,<bytes>)`

Use a solidity version of at least 0.8.12 to get `string.concat()` instead of `abi.encodePacked(<str>,<str>)`

There is 1 instance of this issue:

```
File: smart-contracts/ConvexCurveLPVault.sol   #1

2:    pragma solidity 0.6.12;
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/ConvexCurveLPVault.sol#L2

[N-08] Variable names that consist of all capital letters should be reserved for `const` / `immutable` variables

If the variable needs to be different based on which class it comes from, a `view` / `pure` *function* should be used instead (e.g. like this).

There is 1 instance of this issue:

```
File: smart-contracts/LidoVault.sol   #1

127:      address LIDO = _addressesProvider.getAddress('LIDO');
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/LidoVault.sol#L127

[N-09] NatSpec is incomplete

There are 2 instances of this issue:

```
File: smart-contracts/GeneralVault.sol   #1

/// @audit Missing: '@return'
134       * @param _amount The amount to be withdrawn
135       */
136      function withdrawOnLiquidation(address _asset, uint256 _
137        external
138        virtual
139:       returns (uint256)
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/GeneralVault.sol#L134-L139

```
File: smart-contracts/YieldManager.sol   #2

/// @audit Missing: '@return'
104       * @param _tokenOut The address of token being received
105       */
106:      function getCurvePool(address _tokenIn, address _tokenOu
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/YieldManager.sol#L104-L106

[N-10] Event is missing `indexed` fields

Each `event` should use three `indexed` fields if there are three or more fields

There are 4 instances of this issue:

```
File: smart-contracts/GeneralVault.sol   #1

24:       event ProcessYield(address indexed collateralAsset, uint
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/GeneralVault.sol#L24

```
File: smart-contracts/GeneralVault.sol   #2

25:       event DepositCollateral(address indexed collateralAsset,
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/GeneralVault.sol#L25

```
File: smart-contracts/GeneralVault.sol    #3

26:     event WithdrawCollateral(address indexed collateralAsset
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/GeneralVault.sol#L26

```
File: smart-contracts/GeneralVault.sol    #4

27:     event SetTreasuryInfo(address indexed treasuryAddress, u
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/GeneralVault.sol#L27

[N-11] Consider allowing the passing of a referral code

There is 1 instance of this issue:

```
File: smart-contracts/GeneralVault.sol    #1

81        ILendingPool(_addressesProvider.getLendingPool()).depos
82          _stAsset,
83          _stAssetAmount,
84          msg.sender,
85          0
86:       );
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/GeneralVault.sol#L81-L86

[N-12] Remove commented out code

There is 1 instance of this issue:

```
File: smart-contracts/GeneralVault.sol   #1

144    // /**
145    //  * @dev Convert an `amount` of asset used as collatera
146    //  * @param _amountIn The amount of collateral asset
147    //  */
148:   // function convertOnLiquidation(address _assetOut, uint2
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/GeneralVault.sol#L144-L148

[N-13] Consider two-phase ownership transfer

Consider adding a two-phase transfer, where the current owner nominates the next owner, and the next owner has to call `accept*()` to become the new owner. This prevents passing the ownership to an account that is unable to use it.

There is 1 instance of this issue:

```
File: smart-contracts/YieldManager.sol   #1

26:  contract YieldManager is VersionedInitializable, Ownable {
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/YieldManager.sol#L26

sforman2000 (Sturdy) commented:

> Particularly high quality.

atozICT20 (Sturdy) commented:

> L1: Fixed
> L2: Fixed
> L3: Fixed

> L4: Invalid. ETH constant may be used in several child contract.
> L5: Admin can monitor it.
> L6: Invalid. LidoVault can be received ETH from CurveSwap
> L7: No need to change
> L8: Fixed
> L9: Fixed
> N1: Fixed
> N2: Fixed
> N3: Invalid
> N4: Fixed
> N5: Fixed
> N6: Fixed
> N7: Fixed
> N8: Fixed
> N9: No need to change
> N10: Invalid
> N11: Invalid
> N12: Fixed
> N13: No need to change

[hickuphh3 (judge) commented]():

> L5 should be addressed IMO. there is an inconsistency between addresses being used. The LIDO vault should be using the ETH constant instead of the null address for ETH, or vice versa. For someone who uses etherscan, he'll see the `ETH` constant define and assumes that he should be using that to specify ETH, then wonder why his tx will potentially revert in Metamask.

> **Low issues:** L1, L2, L3, L4, L5, L7, centralisation risk
> **NC issues:** L8, L9, N1, N2, N3 (more of sponsor acknowledged), N4, N5, N6, N7 (reasoning is diff from N6), N8, N9, N12, N13
> **Invalid:** L6, N10 (some fields are not worth the extra gas to index), N11 (no justification)

Gas Optimizations

For this contest, 40 reports were submitted by wardens detailing gas optimizations. The [report highlighted below]() by **IIIIIII** received the top score from the judge.

The following wardens also submitted reports: Dravee, WatchPug, MaratCerby, defsec, simon135, rotcivegaf, joestakey, 0xliumin, robee, hansfriese, StErMi, Funen, hickuphh3, Tomio, Waze, 0xf15ers, Cityscape, 0xNazgul, 0x4non, 0xkatana, oyc_109, hake, SooYa, sikorico, ignacio, z3s, Hawkeye, kebabsec, JC, Fitraldys, delfin454000, GimelSec, mtz, 0v3rf10w, bobirichman, mics, samruna, 0x1f8b, *and* fatherOfBlocks.

Summary

	Issue	Instances
1	Add `require()` for asset address checks before doing the exchange	1
2	Add an `unregisterAsset()` function	1
3	Multiple `address` mappings can be combined into a single `mapping` of an `address` to a `struct`, where appropriate	1
4	State variables should be cached in stack variables rather than re-reading them from storage	7
5	`internal` functions only called once can be inlined to save gas	6
6	`<array>.length` should not be looked up in every loop of a `for`-loop	1
7	Not using the named return variables when a function returns, wastes deployment gas	1
8	Use a more recent version of solidity	5
9	Using `> 0` costs more gas than `!= 0` when used on a `uint` in a `require()` statement	1
10	It costs more gas to initialize variables to zero than to let the default of zero be applied	5
11	`internal` functions not called by the contract should be removed to save deployment gas	2
12	`++i` costs less gas than `i++`, especially when it's used in `for`-loops (`--i` / `i--` too)	5
13	Using `private` rather than `public` for constants, saves gas	5
14	Duplicated `require()` / `revert()` checks should be refactored to a modifier or function	2
15	Empty blocks should be removed or emit something	6

	Issue	Instances
16	Functions guaranteed to revert when called by normal users can be marked `payable`	9
17	`public` functions not called by the contract should be declared `external` instead	3

Total: 61 instances over 17 issues

[G-01] Add `require()` for asset address checks before doing the exchange

The code below should verify that the address is either `0x0` or the LIDO address, in order to prevent wasting gas by doing all of the operations between this point and the actual check done in `_depositToYieldPool()`

There is 1 instance of this issue:

```
File: smart-contracts/LidoVault.sol   #1

109    function _getWithdrawalAmount(address _asset, uint256 _am
110      internal
111      view
112      override
113      returns (address, uint256)
114    {
115      // In this vault, return same amount of asset.
116      return (_addressesProvider.getAddress('LIDO'), _amount)
117:   }
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/LidoVault.sol#L109-L117

[G-02] Add an `unregisterAsset()` function

By unregistering and setting a mapping field to `0`, you'll be getting a Gsreset gas refund (2900 gas). If most `registerAsset()` calls are paired with

`unregisterAsset()` calls, transactions will be cheaper

There is 1 instance of this issue:

```
File: smart-contracts/YieldManager.sol   #1

73      function registerAsset(address _asset) external onlyAdmir
74        _assetsList[_assetsCount] = _asset;
75        _assetsCount = _assetsCount + 1;
76:     }
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/YieldManager.sol#L73-L76

[G-03] Multiple `address` mappings can be combined into a single `mapping` of an `address` to a `struct`, where appropriate

Saves a storage slot for the mapping. Depending on the circumstances and sizes of types, can avoid a Gsset (20000 gas) per mapping combined. Reads and subsequent writes can also be cheaper when a function requires both values and they both fit in the same storage slot

There is 1 instance of this issue:

```
File: smart-contracts/CollateralAdapter.sol   #1

27      mapping(address => address) internal _assetToVaults;
28      // External collateral asset -> internal collateral asse
29:     mapping(address => address) internal _collateralAssets;
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/CollateralAdapter.sol#L27-L29

[G-04] State variables should be cached in stack variables rather than re-reading them from storage

The instances below point to the second+ access of a state variable within a function. Caching will replace each Gwarmaccess (100 gas) with a much cheaper stack read.
Less obvious fixes/optimizations include having local storage variables of mappings within state variable mappings or mappings within state variable structs, having local storage variables of structs within mappings, having local memory caches of state variable structs, or having local caches of state variable contracts/addresses.

There are 7 instances of this issue:

```
File: smart-contracts/ConvexCurveLPVault.sol

/// @audit convexBooster
142:      IConvexBooster(convexBooster).deposit(convexPoolId, _a

/// @audit curveLPToken
138:      TransferHelper.safeTransferFrom(curveLPToken, msg.senc

/// @audit curveLPToken
141:      IERC20(curveLPToken).safeApprove(convexBooster, _amour

/// @audit internalAssetToken
146:      IERC20(internalAssetToken).safeApprove(address(_addres

/// @audit internalAssetToken
148:      return (internalAssetToken, _amount);
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/ConvexCurveLPVault.sol#L142

```
File: smart-contracts/YieldManager.sol

/// @audit _exchangeToken
202:      address _pool = _curvePools[_exchangeToken][_tokenOut]

/// @audit _exchangeToken
```

```
207:         _exchangeToken,
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/YieldManager.sol#L202

[G-05] `internal` functions only called once can be inlined to save gas

Not inlining costs 20 to 40 gas because of two extra `JUMP` instructions and additional stack operations needed for function calls.

There are 6 instances of this issue:

```
File: smart-contracts/ConvexCurveLPVault.sol

205:     function _processTreasury(address _asset, uint256 _yield
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/ConvexCurveLPVault.sol#L205

```
File: smart-contracts/LidoVault.sol

154:     function _processTreasury(uint256 _yieldAmount) internal
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/LidoVault.sol#L154

```
File: smart-contracts/YieldManager.sol

142:     function _getAssetYields(uint256 _totalYieldAmount) inte

178:     function _convertAssetToExchangeToken(address asset, uin
```

```
195     function _convertToStableCoin(address _tokenOut, uint256
196       internal
197:      returns (uint256 receivedAmount)

219:    function _depositYield(address _asset, uint256 _amount)
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/YieldManager.sol#L142

[G-06] `<array>.length` should not be looked up in every loop of a `for` -loop

The overheads outlined below are *PER LOOP*, excluding the first loop

- storage arrays incur a Gwarmaccess (100 gas)
- memory arrays use `MLOAD` (3 gas)
- calldata arrays use `CALLDATALOAD` (3 gas)

Caching the length changes each of these to a `DUP<N>` (3 gas), and gets rid of the extra `DUP<N>` needed to store the stack offset

There is 1 instance of this issue:

```
File: smart-contracts/YieldManager.sol   #1

130:      for (uint256 i = 0; i < assetYields.length; i++) {
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/YieldManager.sol#L130

[G-07] Not using the named return variables when a function returns, wastes deployment gas

There is 1 instance of this issue:

```
File: smart-contracts/YieldManager.sol    #1

200:         return _amount;
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/YieldManager.sol#L200

[G-08] Use a more recent version of solidity

Use a solidity version of at least 0.8.0 to get overflow protection without `SafeMath`
Use a solidity version of at least 0.8.2 to get compiler automatic inlining
Use a solidity version of at least 0.8.3 to get better struct packing and cheaper multiple storage reads
Use a solidity version of at least 0.8.4 to get custom errors, which are cheaper at deployment than `revert()/require()` strings
Use a solidity version of at least 0.8.10 to have external calls skip contract existence checks if the external call has a return value

There are 5 instances of this issue:

```
File: smart-contracts/CollateralAdapter.sol

2:    pragma solidity 0.6.12;
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/CollateralAdapter.sol#L2

```
File: smart-contracts/ConvexCurveLPVault.sol

2:    pragma solidity 0.6.12;
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/ConvexCurveLPVault.sol#L2

```
File: smart-contracts/GeneralVault.sol

2:    pragma solidity 0.6.12;
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/GeneralVault.sol#L2

```
File: smart-contracts/LidoVault.sol

2:    pragma solidity 0.6.12;
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/LidoVault.sol#L2

```
File: smart-contracts/YieldManager.sol

2:    pragma solidity 0.6.12;
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/YieldManager.sol#L2

[G-09] Using `> 0` costs more gas than `!= 0` when used on a `uint` in a `require()` statement

This change saves 6 gas per instance

There is 1 instance of this issue:

```
File: smart-contracts/GeneralVault.sol   #1

179:      require(yieldStAsset > 0, Errors.VT_PROCESS_YIELD_INVA
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/GeneralVault.sol#L179

[G-10] It costs more gas to initialize variables to zero than to let the default of zero be applied

There are 5 instances of this issue:

```
File: smart-contracts/ConvexCurveLPVault.sol

106:        for (uint256 i = 0; i < extraRewardsLength; i++) {
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/ConvexCurveLPVault.sol#L106

```
File: smart-contracts/GeneralVault.sol

218:        for (uint256 i = 0; i < length; i++) {
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/GeneralVault.sol#L218

```
File: smart-contracts/YieldManager.sol

120:        for (uint256 i = 0; i < _count; i++) {

130:        for (uint256 i = 0; i < assetYields.length; i++) {

156:        for (uint256 i = 0; i < length; i++) {
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/YieldManager.sol#L120

[G-11] `internal` functions not called by the contract should be removed to save deployment gas

If the functions are required by an interface, the contract should inherit from that interface and use the `override` keyword

There are 2 instances of this issue:

```
File: smart-contracts/GeneralVault.sol   #1

204:     function _getAssetYields(uint256 _WETHAmount) internal v
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/GeneralVault.sol#L204

```
File: smart-contracts/GeneralVault.sol   #2

235:     function _depositYield(address _asset, uint256 _amount)
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/GeneralVault.sol#L235

[G-12] `++i` costs less gas than `i++`, especially when it's used in `for`-loops (`--i` / `i--` too)

Saves 6 gas *PER LOOP*

There are 5 instances of this issue:

```
File: smart-contracts/ConvexCurveLPVault.sol

106:       for (uint256 i = 0; i < extraRewardsLength; i++) {
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/ConvexCurveLPVault.sol#L106

```
File: smart-contracts/GeneralVault.sol

218:       for (uint256 i = 0; i < length; i++) {
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/GeneralVault.sol#L218

```
File: smart-contracts/YieldManager.sol

120:       for (uint256 i = 0; i < _count; i++) {

130:       for (uint256 i = 0; i < assetYields.length; i++) {

156:       for (uint256 i = 0; i < length; i++) {
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/YieldManager.sol#L120

[G-13] Using `private` rather than `public` for constants, saves gas

If needed, the value can be read from the verified contract source code. Savings are due to the compiler not having to create non-payable getter functions for deployment calldata, and not adding another entry to the method ID table

There are 5 instances of this issue:

```
File: smart-contracts/CollateralAdapter.sol

22:     uint256 public constant VAULT_REVISION = 0x1;
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/CollateralAdapter.sol#L22

```
File: smart-contracts/GeneralVault.sol

55:     uint256 public constant VAULT_REVISION = 0x1;
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/GeneralVault.sol#L55

```
File: smart-contracts/YieldManager.sol

41:     uint256 public constant VAULT_REVISION = 0x1;

48:     uint256 public constant UNISWAP_FEE = 10000; // 1%

49:     uint256 public constant SLIPPAGE = 500; // 5%
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/YieldManager.sol#L41

[G-14] Duplicated `require()` / `revert()` checks should be refactored to a modifier or function

Saves deployment costs

There are 2 instances of this issue:

```
File: smart-contracts/ConvexCurveLPVault.sol   #1

101:       require(_token == _tokenFromConvex, Errors.VT_INVALID_
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/ConvexCurveLPVault.sol#L101

```
File: smart-contracts/YieldManager.sol   #2

203:      require(_pool != address(0), Errors.VT_INVALID_CONFIGU
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/YieldManager.sol#L203

[G-15] Empty blocks should be removed or emit something

The code should be refactored such that they no longer exist, or the block should do something useful, such as emitting an event or reverting. If the contract is meant to be extended, the contract should be `abstract` and the function signatures be added without any default implementation. If the block is an empty if-statement block to avoid doing subsequent checks in the else-if/else conditions, the else-if/else conditions should be nested under the negation of the if-statement, because they involve different classes of checks, which may lead to the introduction of errors when the code is later modified (`if(x){}else if(y){...}else{...}` => `if(!x){if(y){...}else{...}}`)

There are 6 instances of this issue:

```
File: smart-contracts/GeneralVault.sol

153:    function processYield() external virtual {}

158:    function pricePerShare() external view virtual returns (

246:    {}

255:    ) internal virtual returns (uint256) {}

265:    {}
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/GeneralVault.sol#L153

```
File: smart-contracts/LidoVault.sol

24:     receive() external payable {}
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/LidoVault.sol#L24

[G-16] Functions guaranteed to revert when called by normal users can be marked `payable`

If a function modifier such as `onlyOwner` is used, the function will revert if a normal user tries to pay the function. Marking the function as `payable` will lower the gas cost for legitimate callers because the compiler will not include checks for whether a payment was provided. The extra opcodes avoided are `CALLVALUE` (2), `DUP1` (3), `ISZERO` (3), `PUSH2` (3), `JUMPI` (10), `PUSH1` (3), `DUP1` (3), `REVERT` (0), `JUMPDEST` (1), `POP` (2), which costs an average of about 21 gas per call to the function, in addition to the extra deployment cost

There are 9 instances of this issue:

```
File: smart-contracts/CollateralAdapter.sol

43      function addCollateralAsset(
44        address _externalAsset,
45        address _internalAsset,
46        address _acceptVault
47:     ) external onlyAdmin {
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/CollateralAdapter.sol#L43-L47

```
File: smart-contracts/ConvexCurveLPVault.sol

37:     function setConfiguration(address _lpToken, uint256 _poo

87:     function processYield() external override onlyAdmin {
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/ConvexCurveLPVault.sol#L37

```
File: smart-contracts/GeneralVault.sol

165:    function setTreasuryInfo(address _treasury, uint256 _fee
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/GeneralVault.sol#L165

```
File: smart-contracts/LidoVault.sol

30:     function processYield() external override onlyAdmin {
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/LidoVault.sol#L30

```
File: smart-contracts/YieldManager.sol

64:     function setExchangeToken(address _token) external onlyA

73:     function registerAsset(address _asset) external onlyAdmi

92      function setCurvePool(
93        address _tokenIn,
94        address _tokenOut,
95        address _pool
96:     ) external onlyAdmin {
```

```
118:     function distributeYield(uint256 _offset, uint256 _count
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/YieldManager.sol#L64

[G-17] `public` functions not called by the contract should be declared `external` instead

Contracts are allowed to override their parents' functions and change the visibility from `external` to `public` and can save gas by doing so.

There are 3 instances of this issue:

```
File: smart-contracts/CollateralAdapter.sol    #1

35:     function initialize(ILendingPoolAddressesProvider _provi
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/CollateralAdapter.sol#L35

```
File: smart-contracts/GeneralVault.sol    #2

61:     function initialize(ILendingPoolAddressesProvider _provi
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/GeneralVault.sol#L61

```
File: smart-contracts/YieldManager.sol    #3

60:     function initialize(ILendingPoolAddressesProvider _provi
```

https://github.com/code-423n4/2022-05-sturdy/blob/78f51a7a74ebe8adfd055bdbaedfddc05632566f/smart-contracts/YieldManager.sol#L60

sforman2000 (Sturdy) commented:

> Particularly high quality.

iris112 (Sturdy) commented:

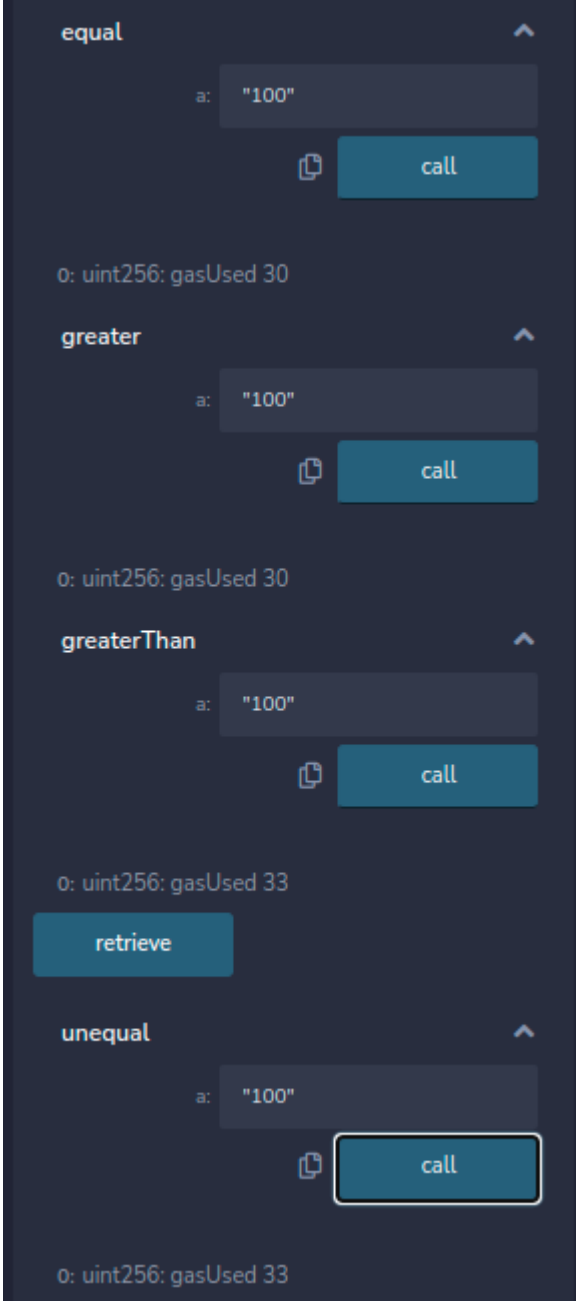
> `3. Multiple address mappings can be combined into a single mapping of an address to a struct, where appropriate`
> This is not correct.
> I have a simple test on remix and confirmed there is no effect. In fact the case of using struct type spent more gas (+65 gas)
> Let me know your example.

iris112 (Sturdy) commented:

> `2. Add an unregisterAsset() function`
> Yeah we need unregisterAsset function, but not sure about the resetting 0 should be efficient to reduce gas. Normally non-zero to non-zero is cheaper than zero to non-zero.
> I think we don't need additional feature to reset 0.

iris112 (Sturdy) commented:

> `9. Using > 0 costs more gas than != 0 when used on a uint in a require() statement`
> This is also not correct.
> I checked on remix with your same example, but *greater* is less than *not equal*.
> *greater: 30 gas*
> *equal: 30 gas*
> *greaterThan: 33 gas*
> *notequal: 33 gas*



Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top