Backd contest

2022-06-02

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 Backd smart contract system written in Solidity. The audit contest took place between April 21—April 27 2022.

Wardens

62 Wardens contributed reports to the Backd contest:

1. unforgiven
2. IllIllI
3. WatchPug
4. 0xDjango
5. shenwilly
6. sseefried
7. 0x52
8. Ruhum
9. fatherOfBlocks
10. wuwe1
11. defsec
12. Dravee
13. horsefacts
14. joestakey
15. pauliax
16. StyxRave
17. rayn

18. hubble (ksk2345 and shri4net)
19. robee
20. antonttc
21. Tomio
22. csanuragjain
23. TrungOre
24. sorrynotsorry
25. catchup
26. 0xkatana
27. reassor
28. berndartmueller
29. kenta
30. securerodd
31. gs8nrv
32. hake
33. z3s
34. 0v3rf10w
35. Funen
36. TerrierLover
37. oyc_109
38. simon135
39. Tadashi
40. dipp
41. kebabsec (okkothejawa and FlameHorizon)
42. peritoflores
43. jayjonah8
44. Kenshin
45. m4rio_eth
46. remora
47. MaratCerby
48. 0x1f8b
49. cccz
50. hyh
51. slywaters
52. 0x4non
53. 0xNazgul
54. NoamYakov
55. rfa
56. saian
57. 0xmint
58. tin537
59. danb
60. UnusualTurtle

This contest was judged by gzeon. 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 18 unique vulnerabilities. Of these vulnerabilities, 3 received a risk rating in the category of HIGH severity and 15 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 39 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 35 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 Backd contest repository, and is composed of 38 smart contracts written in the Solidity programming language and includes 4,630 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 (3)

[H-01] User can steal all rewards due to checkpoint after transfer

Submitted by 0xDjango, also found by unforgiven

StakerVault.sol#L112-L119

I believe this to be a high severity vulnerability that is potentially included in the currently deployed `StakerVault.sol` contract also. The team will be contacted immediately following the submission of this report.

In `StakerVault.sol`, the user checkpoints occur AFTER the balances are updated in the `transfer()` function. The user checkpoints update the amount of rewards claimable by the user. Since their rewards will be updated after transfer, a user can send funds between their own accounts and repeatedly claim maximum rewards since the pool's inception.

In every actionable function except `transfer()` of `StakerVault.sol`, a call to `ILpGauge(lpGauge).userCheckpoint()` is correctly made BEFORE the action effects.

Proof of Concept

Assume a certain period of time has passed since the pool's inception. For easy accounting, assume `poolStakedIntegral` of `LpGauge.sol` equals `1`. The `poolStakedIntegral` is used to keep track of the current reward rate.

Steps:

- Account A stakes 1000 LP tokens. `balances[A] += 1000`
- In the same `stakeFor()` function, `userCheckpoint()` was already called so A will already have `perUserShare[A]` set correctly based on their previously 0 balance and the current `poolStakedIntegral`.
- Account A can immediately send all balance to Account B via `transfer()`.
- Since the checkpoint occurs after the transfer, B's balance will increase and then `perUserShare[B]` will be updated. The calculation for `perUserShare` looks as follows.

```
perUserShare[user] += (
            (stakerVault.stakedAndActionLockedBalanceOf(user)).scaledMul(
                (poolStakedIntegral_ - perUserStakedIntegral[user])
            )
        );
```

Assuming Account B is new to the protocol, their `perUserStakedIntegral[user]` will default to `0`.

```
perUserShare[B] += 1000 * (1 - 0) = 1000
```

- B is able to call `claimRewards()` and mint all 1000 reward tokens.
- B then calls `transfer()` and sends all 1000 staked tokens to Account C.
- Same calculation occurs, and C can claim all 1000 reward tokens.
- This process can be repeated until the contract is drained of reward tokens.

Recommended Mitigation Steps

In `StakerVault.transfer()`, move the call to `ILpGauge(lpGauge).userCheckpoint()` to before the balances are updated.

chase-manning (Backd) confirmed and resolved

[H-02] function `lockFunds` in `TopUpActionLibrary` can cause serious fund lose. fee and Capped bypass. It's not calling `stakerVault.increaseActionLockedBalance` when transfers stakes.

Submitted by unforgiven

TopUpAction.sol#L57-L65

In function TopUpActionLibrary.lockFunds when transfers stakes from payer it doesn't call stakerVault.increaseActionLockedBalance for that payer so stakerVault.actionLockedBalances[payer] is not get updated for payer and stakerVault.stakedAndActionLockedBalanceOf(payer) is going to show wrong value and any calculation based on this function is gonna be wrong which will cause fund lose and theft and some restriction bypasses.

Proof of Concept

When user wants to create a TopUpAction. so he deposit his funds to Pool and get LP token. then stake the LP token in StakerVault and use that stakes to create a TopUp position with function TopUpAction.register. This function transfer user stakes (locks user staks) and create his position.

For transferring and locking user stakes it uses TopUpActionLibrary.lockFunds. function lockFunds transfers user stakes but don't call stakerVault.increaseActionLockedBalance for the payer which cause that stakerVault.actionLockedBalances[payer] to get different values(not equal to position.depositTokenBalance).

Function StakerVault.stakedAndActionLockedBalanceOf(account) uses stakerVault.actionLockedBalances[account] so it will return wrong value and any where in code that uses stakedAndActionLockedBalanceOf() is going to cause problems.

three part of the codes uses stakerVault.stakedAndActionLockedBalanceOf(): 1. LiqudityPool.depositFor() for checking user total deposits to be less than depositCap. 2. LiqudityPool._updateUserFeesOnDeposit() for updating user fee on new deposits. 3. userCheckpoint() for calculating user rewards. attacker can use #1 and #2 to bypass high fee payment and max depositCap and #3 will cause users to lose rewards.

The detail steps: 1- user deposit fund to Pool and get LP token. 2- user stakes LP token in StakerVault. 3- user approve TopUpAction address to transfer his staks in StakerVault. 3- user use all his stakes to create a position with TopUpAction.register() function. 3.1- register() will call lockFunds to transfer and lock user stakes. 3.2- lockFunds() will transfer user stakes with stakerVault.transferFrom() but don't call stakerVault.increaseActionLockedBalance() so StakerVault.actionLockedBalances[user] will be zero. 3.3- StakerVault.balance[useer] will be zero too because his stakes get transfers in 3.2 4- StakerVault.stakedAndActionLockedBalanceOf(user) will return zero (user has some locked stakes in TopUpAction but because of the bug calculation get out of sync)

In this moment user will lose all the rewards that are minted in LpGauge. because userCheckpoint() use stakerVault.stakedAndActionLockedBalanceOf(user) for calculating rewards which is zero and new rewards will be zero too.

Attacker can use this process to bypass "max deposit Cap" and deposit any amount of assets he wants. because LiqudityPool.depositFor(address,uint256,uint256) uses stakedAndActionLockedBalanceOf to check user deposits which is zero so Attacker can deposit & stake & register to make his balance zero and repeat this and in the end reset his TopUp positions to get back his large stakes which are multiple time bigger than "max deposit Cap"

Attacker can also use this process to bypass fee penalties for early withdraw. because LiqudityPool._updateUserFeesOnDeposit() to get user current balance use stakedAndActionLockedBalanceOf() which is zero. so the value of shareExisting variable become zero and newFeeRatio will be calculated based on feeOnDeposit which can be minFee if asset is already in wallet for some time.

Tools Used

VIM

Recommended Mitigation Steps

Add this line to TopUpActionLibrary.lockFunds() after stakerVault.transferFrom():

stakerVault.increaseActionLockedBalance(payer, amountLeft);

chase-manning (Backd) confirmed and resolved

[H-03] Customers cannot be `topUp()`ed a second time

Submitted by IllIllI

CompoundHandler.sol#L71 CompoundHandler.sol#L120 AaveHandler.sol#L53 TopUpAction.sol#L847

OpenZeppelin's `safeApprove()` will revert if the account already is approved and the new `safeApprove()` is done with a non-zero value.

```
function safeApprove(
    IERC20 token,
    address spender,
    uint256 value
) internal {
    // safeApprove should only be called when setting an initial allowance,
    // or when resetting it to zero. To increase and decrease it, use
    // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
    require(
        (value == 0) || (token.allowance(address(this), spender) == 0),
        "SafeERC20: approve from non-zero to non-zero allowance"
    );
    _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value
}
```

OpenZeppelin/SafeERC20.sol#L45-L58

Impact

Customers cannot be topped up a second time, which will cause them to be liquidated even though they think they're protected.

Proof of Concept

There are multiple places where `safeApprove()` is called a second time without setting the value to zero first. The instances below are all related to topping up.

Compound-specific top-ups will fail the second time around when approving the `ctoken` again:

```
File: backd/contracts/actions/topup/handlers/CompoundHandler.sol   #1

    function topUp(
        bytes32 account,
        address underlying,
        uint256 amount,
        bytes memory extra
    ) external override returns (bool) {
        bool repayDebt = abi.decode(extra, (bool));
        CToken ctoken = cTokenRegistry.fetchCToken(underlying);
        uint256 initialTokens = ctoken.balanceOf(address(this));

        address addr = account.addr();

        if (repayDebt) {
            amount -= _repayAnyDebt(addr, underlying, amount, ctoken);
            if (amount == 0) return true;
        }
```

```
66
67          uint256 err;
68          if (underlying == address(0)) {
69              err = ctoken.mint{value: amount}(amount);
70          } else {
71              IERC20(underlying).safeApprove(address(ctoken), amount);
```

CompoundHandler.sol#L50-L71

Compound-specific top-ups will also fail when trying to repay debt:

`File: backd/contracts/actions/topup/handlers/CompoundHandler.sol   #2`

```
62          if (repayDebt) {
63              amount -= _repayAnyDebt(addr, underlying, amount, ctoken);
64              if (amount == 0) return true;
65          }
```

CompoundHandler.sol#L62-L65

Aave-specific top-ups will fail for the `lendingPool`:

`File: backd/contracts/actions/topup/handlers/AaveHandler.sol   #3`

```
36      function topUp(
37          bytes32 account,
38          address underlying,
39          uint256 amount,
40          bytes memory extra
41      ) external override returns (bool) {
42          bool repayDebt = abi.decode(extra, (bool));
43          if (underlying == address(0)) {
44              weth.deposit{value: amount}();
45              underlying = address(weth);
46          }
47
48          address addr = account.addr();
49
50          DataTypes.ReserveData memory reserve = lendingPool.getReserveData(underlying);
51          require(reserve.aTokenAddress != address(0), Error.UNDERLYING_NOT_SUPPORTED);
52
53          IERC20(underlying).safeApprove(address(lendingPool), amount);
```

AaveHandler.sol#L36-L53

The `TopUpAction` itself fails for the `feeHandler`:

`File: backd/contracts/actions/topup/TopUpAction.sol   #4`

```
840      function _payFees(
        address payer,
        address beneficiary,
        uint256 feeAmount,
        address depositToken
    ) internal {
        address feeHandler = getFeeHandler();
        IERC20(depositToken).safeApprove(feeHandler, feeAmount);
```

TopUpAction.sol#L840-L847

I've filed the other less-severe instances as a separate medium-severity issue, and flagged the remaining low-severity instances in my QA report.

Recommended Mitigation Steps

Always do `safeApprove(0)` if the allowance is being changed, or use `safeIncreaseAllowance()`.

chase-manning (Backd) confirmed and resolved

Medium Risk Findings (15)

[M-01] `call()` should be used instead of `transfer()` on an `address payable`

Submitted by Dravee, also found by antonttc, berndartmueller, cccz, danb, horsefacts, hyh, IllIllI, MaratCerby, pauliax, rayn, UnusualTurtle, WatchPug, and wuwe1

This is a classic Code4rena issue:

- https://github.com/code-423n4/2021-04-meebits-findings/issues/2
- https://github.com/code-423n4/2021-10-tally-findings/issues/20
- https://github.com/code-423n4/2022-01-openleverage-findings/issues/75

Impact

The use of the deprecated `transfer()` function for an address will inevitably make the transaction fail when:

1. The claimer smart contract does not implement a payable function.
2. The claimer smart contract does implement a payable fallback which uses more than 2300 gas unit.
3. The claimer smart contract implements a payable fallback function that needs less than 2300 gas units but is called through proxy, raising the call's gas usage above 2300.

Additionally, using higher than 2300 gas might be mandatory for some multisig wallets.

Impacted lines:

```
backd/contracts/pool/EthPool.sol:
  30:         to.transfer(amount);

backd/contracts/strategies/BkdEthCvx.sol:
   77:             payable(vault).transfer(amount);
   93:         payable(vault).transfer(amount);
  117:         payable(vault).transfer(underlyingBalance);

backd/contracts/vault/EthVault.sol:
  29:         payable(to).transfer(amount);
  37:         payable(addressProvider.getTreasury()).transfer(amount);

backd/contracts/vault/VaultReserve.sol:
  81:             payable(msg.sender).transfer(amount);
```

Recommended Mitigation

I recommend using `call()` instead of `transfer()`.

chase-manning (Backd) confirmed and resolved

gzeon (judge) commented: > Sponsor confirmed. Judging this as Medium Risk.

[M-02] Its possible to lose total governance control by mistake

Submitted by hubble, also found by antonttc, csanuragjain, gs8nrv, rayn, reassor, and TrungOre

RoleManager.sol#L43-L46 RoleManager.sol#L115-L128

The impact of this vulnerability, i.e., losing all governance control is very High. There is a possibility, due to a corner case as described below.

Proof of Concept

Contract : RoleManager.sol Function : renounceGovernance()

```
Step 0:
  Let current governance role given to = CURRENT_GOV_ADDRESS
  so, getRoleMemberCount() for "governance" role will return = 1
```

```
Step 1: Add a new address say ALICE to governance role, by addGovernor(ALICE)
  now, ALICE also has governace role, and getRoleMemberCount() for "governance" role will

Step 2: Assume that ALICE renounces governance role, by renounceGovernance()
  now, ALICE does not have governance role, but getRoleMemberCount() for "governance" rol

Step 3: In some distant future, if there is a compromise of CURRENT_GOV_ADDRESS keys or c
  its decided to revoke governance role for CURRENT_GOV_ADDRESS via renounceGovernance(),
  It can be assumed that since getRoleMemberCount() for "governance" role returns = 2, at
  But now, CURRENT_GOV_ADDRESS does not have governance role, and the total governance co
```

Recommended Mitigation Steps

getRoleMemberCount() currently returns _roleMembers[role].length(); It should return the count only for _roles[role].members[account] = true;

Its recommended to add a new function to know who are the active members for any role, like getRoleMembers(bytes32 role) returning address account.

chase-manning (Backd) confirmed

gzeon (judge) decreased severity to Medium

[M-03] Lack of `safeApprove(0)` prevents some registrations, and the changing of stakers and LP tokens

Submitted by IllIllI, also found by defsec and Dravee

TopUpAction.sol#L50 LiquidityPool.sol#L721

OpenZeppelin's `safeApprove()` will revert if the account already is approved and the new `safeApprove()` is done with a non-zero value

```
function safeApprove(
    IERC20 token,
    address spender,
    uint256 value
) internal {
    // safeApprove should only be called when setting an initial allowance,
    // or when resetting it to zero. To increase and decrease it, use
    // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
    require(
        (value == 0) || (token.allowance(address(this), spender) == 0),
        "SafeERC20: approve from non-zero to non-zero allowance"
    );
```

```
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, v
    }
```

OpenZeppelin/SafeERC20.sol#L45-L58

Impact

Customers can be prevented from `register()`ing the same `token/stakerVaultAddress` as another customer; and once changed away from, stakers and lptokens can't be used in the future.

Proof of Concept

There are multiple places where `safeApprove()` is called a second time without setting the value to zero first.

`register()` calls `lockFunds()` for each user registration, and since users will use the same tokens and staker vaults, the second user's `register()` call will fail:

```
File: backd/contracts/actions/topup/TopUpAction.sol   #1

    function lockFunds(
        address stakerVaultAddress,
        address payer,
        address token,
        uint256 lockAmount,
        uint256 depositAmount
    ) external {
        uint256 amountLeft = lockAmount;
        IStakerVault stakerVault = IStakerVault(stakerVaultAddress);

        // stake deposit amount
        if (depositAmount > 0) {
            depositAmount = depositAmount > amountLeft ? amountLeft : depositAmount;
            IERC20(token).safeTransferFrom(payer, address(this), depositAmount);
            IERC20(token).safeApprove(stakerVaultAddress, depositAmount);
```

TopUpAction.sol#L36-L50

The changing of either the staker or an lp token is behind a time-lock, and once the time has passed, the changed variables rely on this function:

```
File: backd/contracts/pool/LiquidityPool.sol   #2

    function _approveStakerVaultSpendingLpTokens() internal {
        address staker_ = address(staker);
        address lpToken_ = address(lpToken);
        if (staker_ == address(0) || lpToken_ == address(0)) return;
721           IERC20(lpToken_).safeApprove(staker_, type(uint256).max);
722       }
```

LiquidityPool.sol#L717-L722

If a bug is found in a new `staker` or `lpToken` and the governor wishes to change back to the old one(s), the governor will have to wait for the time-lock delay only to find out that the old value(s) cause the code to revert.

I've filed the other more-severe instances as a separate high-severity issue, and flagged the remaining low-severity instances in my QA report.

Recommended Mitigation Steps

Always do `safeApprove(0)` if the allowance is being changed, or use `safeIncreaseAllowance()`.

chase-manning (Backd) confirmed

samwerner (Backd) commented: > It should be noted that the second example referring to `_approveStakerVaultSpendingLpTokens()` is not an issue. This is neither a member variable that can be updated nor is it behind a time lock. Both the `staker` and `lpToken` can only be set once and hence the `safeApprove` in the aforementioned function can only be called once.

chase-manning (Backd) resolved resolved

[M-04] `CvxCrvRewardsLocker` implements a swap without a slippage check that can result in a loss of funds through MEV

Submitted by Ruhum

The CvxCrvRewardsLocker contract swaps tokens through the CRV cvxCRV pool. But, it doesn't use any slippage checks. The swap is at risk of being frontrun / sandwiched which will result in a loss of funds.

Since MEV is very prominent I think the chance of that happening is pretty high.

Proof of Concept

Here's the swap: CvxCrvRewardsLocker.sol#L247-L252.

Recommended Mitigation Steps

Use a proper value for `minOut` instead of `0`.

chase-manning (Backd) confirmed

gzeon (judge) decreased severity to Medium and commented: > According to C4 Judging criteria: > > Unless there is something uniquely novel created by combining vectors, most submissions regarding vulnerabilities that are inherent to a particular system or the Ethereum network as a whole should be considered QA. Examples of such vulnerabilities include front running, sandwich attacks, and MEV. > > However since there is a configurable `minOut` that is deliberately set to 0, this seems to be a valid issue. I am judging this as Medium Risk.

[M-05] Chainlink's `latestRoundData` might return stale or incorrect results

Submitted by cccz, also found by 0x1f8b, 0xDjango, 0xkatana, berndartmueller, defsec, Dravee, horsefacts, hyh, IllIllI, kenta, rayn, reassor, sorrynotsorry, and WatchPug

On ChainlinkOracleProvider.sol and ChainlinkUsdWrapper.sol , we are using latestRoundData, but there is no check if the return value indicates stale data.

```
    function _ethPrice() private view returns (int256) {
        (, int256 answer, , , ) = _ethOracle.latestRoundData();
        return answer;
    }
...
    function getPriceUSD(address asset) public view override returns (uint256) {
        address feed = feeds[asset];
        require(feed != address(0), Error.ASSET_NOT_SUPPORTED);

        (, int256 answer, , uint256 updatedAt, ) = AggregatorV2V3Interface(feed).latestRound

        require(block.timestamp <= updatedAt + stalePriceDelay, Error.STALE_PRICE);
        require(answer >= 0, Error.NEGATIVE_PRICE);

        uint256 price = uint256(answer);
        uint8 decimals = AggregatorV2V3Interface(feed).decimals();
        return price.scaleFrom(decimals);
    }
```

This could lead to stale prices according to the Chainlink documentation:

https://docs.chain.link/docs/historical-price-data/#historical-rounds https://docs.chain.link/docs/faq/#how-can-i-check-if-the-answer-to-a-round-is-being-carried-over-from-a-previous-round

Proof of Concept

ChainlinkOracleProvider.sol#L55 ChainlinkUsdWrapper.sol#L64

Recommended Mitigation Steps

```
    function _ethPrice() private view returns (int256) {
        (uint80 roundID, int256 answer, , uint256 timestamp, uint80 answeredInRound) = _eth
        require(answeredInRound >= roundID, "Stale price");
        require(timestamp != 0,"Round not complete");
        require(answer > 0,"Chainlink answer reporting 0");
        return answer;
    }
...
    function getPriceUSD(address asset) public view override returns (uint256) {
        address feed = feeds[asset];
        require(feed != address(0), Error.ASSET_NOT_SUPPORTED);
        (uint80 roundID, int256 answer, , uint256 updatedAt, uint80 answeredInRound) = Aggre
        require(answeredInRound >= roundID, "Stale price");
        require(answer > 0," Error.NEGATIVE_PRICE");
        require(block.timestamp <= updatedAt + stalePriceDelay, Error.STALE_PRICE);

        uint256 price = uint256(answer);
        uint8 decimals = AggregatorV2V3Interface(feed).decimals();
        return price.scaleFrom(decimals);
    }
```

chase-manning (Backd) confirmed and resolved

[M-06] ERC777 tokens can bypass `depositCap` guard

Submitted by shenwilly, also found by wuwe1

LiquidityPool.sol#L523

When ERC777 token is used as the underlying token for a `LiquidityPool`, a depositor can reenter `depositFor` and bypass the `depositCap` requirement check, resulting in higher total deposit than intended by governance.

Proof of Concept

- An empty ERC777 liquidity pool is capped at 1.000 token.
- Alice deposits 1.000 token. Before the token is actually sent to the contract, `tokensToSend` ERC777 hook is called and Alice reenters `depositFor`.
- As the previous deposit hasn't been taken into account, the reentrancy passes the `depositCap` check.
- Pool has 2.000 token now, despite the 1.000 deposit cap.

Recommended Mitigation Steps

Add reentrancy guards to `depositFor`.

chase-manning (Backd) confirmed and resolved

[M-07] Inconsistency between constructor and setting method for `slippageTolerance`

Submitted by fatherOfBlocks, also found by shenwilly

StrategySwapper.sol#L38-L43 StrategySwapper.sol#L109-L114

In the setSlippageTolerance(L119) method you have certain requirements to set slippageTolerance, but in the constructor you don't.

Recommended Mitigation Steps

I would add the corresponding validations to the constructor.

chase-manning (Backd) confirmed and resolved

[M-08] `_decimalMultiplier` doesn't account for tokens with decimals higher than 18

Submitted by shenwilly, also found by pauliax, StyxRave, and WatchPug

StrategySwapper.sol#L287-L289 StrategySwapper.sol#L318-L320 StrategySwapper.sol#L335-L337

In `StrategySwapper`, swapping from or to tokens with decimals higher than 18 will always revert. This will cause inabilities for strategies to harvest rewards.

Proof of Concept

L288 will revert when `token_` has higher than 18 decimals.

```
return 10**(18 - IERC20Full(token_).decimals());
```

Recommended Mitigation Steps

Consider modifying how `_decimalMultiplier` works so it could handle tokens with higher than 18 decimals.

Update the calculation of `_minTokenAmountOut` and `_minWethAmountOut` to account when decimals are higher/lower than `18`.

chase-manning (Backd) confirmed and resolved

[M-09] getNewCurrentFees reverts when minFeePercentage > feeRatio

Submitted by shenwilly

LiquidityPool.sol#L694

Depositors won't be able to transfer or redeem funds temporarily.

The problem is caused by the implementation of `LiquidityPool.getNewCurrentFees`:

```
function getNewCurrentFees(
    uint256 timeToWait,
    uint256 lastActionTimestamp,
    uint256 feeRatio
) public view returns (uint256) {
    uint256 timeElapsed = _getTime() - lastActionTimestamp;
    uint256 minFeePercentage = getMinWithdrawalFee();
    if (timeElapsed >= timeToWait) {
        return minFeePercentage;
    }
    uint256 elapsedShare = timeElapsed.scaledDiv(timeToWait);
    return feeRatio - (feeRatio - minFeePercentage).scaledMul(elapsedShare);
}
```

The last line requires the current `feeRatio` to be higher than `minFeePercentage` or the function will revert. When this condition is broken, some critical functions such as transferring tokens and redeeming will be unusable. Affected users need to wait until enough time has elapsed and `getNewCurrentFees` returns `minFeePercentage` on L691.

This could happen if governance changes the `MinWithdrawalFee` to be higher than a user's feeRatio.

Proof of Concept

- Initial `MinWithdrawalFee` is set to 0, `MaxWithdrawalFee` is set to 0.03e18.
- Alice deposits fund and receives LP token. Alice's `feeRatio` is now set to 0.03e18 (the current `MaxWithdrawalFee`).
- Governance changes `MaxWithdrawalFee` to `0.05e18` and `MinWithdrawalFee` to `0.04e18`.
- `minFeePercentage` is now higher than Alice's `feeRatio` and she can't transfer nor redeem the LP token until `timeElapsed >= timeToWait`.

Recommended Mitigation Steps

Add a new condition in `getNewCurrentFees` L690 to account for this case:

```
if (timeElapsed >= timeToWait || minFeePercentage > feeRatio) {
    return minFeePercentage;
}
```

chase-manning (Backd) confirmed and resolved

[M-10] Griefer can extend period of higher withdrawal fees

Submitted by 0xDjango

LiquidityPool.sol#L790-L792

The `_updateUserFeesOnDeposit()` function in `LiquidityPool.sol` is used to update a user's withdrawal fees after an action such as deposit, transfer in, etc. The withdrawal fee decays toward a minimum withdrawal fee over a period of 1 or 2 weeks (discussed with developer). Since anyone can transfer lp tokens to any user, a griefer can transfer 1 wei of lp tokens to another user to reset their `lastActionTimestamp` used in the withdrawal fee calculation.

The developers nicely weight the updated withdrawal fee by taking the original balance/original fee vs the added balance/added fee. The attacker will only be able to extend the runway of the withdrawal fee cooldown by resetting the `lastActionTimestamp` for future calculations. Example below:

Proof of Concept

Assumptions: - MinWithdrawalFee = 0% //For easy math - MaxWithdrawalFee = 10% - timeToWait = 2 weeks

Steps

- User A has `100 wei` of shares
- User A waits 1 week (Current withdrawal fee = 5%)
- User B deposits, receives `1 wei` of shares, current withdrawal fee = 10%
- User B immediately transfers `1 wei` of shares to User A

Based on the formula to calculated User A's new feeRatio:

```
uint256 newFeeRatio = shareExisting.scaledMul(newCurrentFeeRatio) +
    shareAdded.scaledMul(feeOnDeposit);
```

In reality, User A's withdrawal fee will only increase by a negligible amount since the shares added were very small in proportion to the original shares. We can assume user A's current withdrawal fee is still 5%.

The issue is that the function then reset's User A's `lastActionTimestamp` to the current time. This means that User A will have to wait the maximum 2 weeks for the withdrawal fee to reduce from 5% to 0%. Effectively the cooldown runway is the same length as the original runway length, so the decay down to 0% will take twice as long.

```
meta.lastActionTimestamp = uint64(_getTime());
```

Recommended Mitigation Steps

Instead of resetting `lastActionTimestamp` to the current time, scale it the same way the `feeRatio` is scaled. I understand that this would technically not be the timestamp of the last action, so the variable would probably need to be renamed.

chase-manning (Backd) confirmed and resolved

[M-11] Position owner should set allowed slippage

Submitted by 0x52

TopUpAction.sol#L154 TopUpAction.sol#L187

The default swap slippage of 5% allows malicious keepers to sandwich attack topup. Additionally, up to 40% (_MIN_SWAPPER_SLIPPAGE) slippage allows malicious owner to sandwich huge amounts from topup

Proof of Concept

Keeper can bundle swaps before and after topup to sandwich topup action, in fact it's actually in their best interest to do so.

Recommended Mitigation Steps

Allow user to specify max swap slippage when creating topup similar to how it's specified on uniswap or sushiswap to block attacks from both keepers and owners.

chase-manning (Backd) confirmed and resolved

gzeon (judge) commented: > According to C4 Judging criteria > > Unless there is something uniquely novel created by combining vectors, most submissions regarding vulnerabilities that are inherent to a particular system or the Ethereum network as a whole should be considered QA. Examples of such vulnerabilities include front running, sandwich attacks, and MEV. > > However since Backd use keeper to run topup transactions, which presumably are bots and smart contracts that can fetch onchain price directly. A large (5% default, up to 40%) seems excessive and can lead to user losing fund. Judging this as Medium Risk.

[M-12] `CompoundHandler#topUp()` Using the wrong function selector makes native token `topUp()` always revert

Submitted by WatchPug

compound-finance/CEther.sol#L44-L47

```
function mint() external payable {
    (uint err,) = mintInternal(msg.value);
    requireNoError(err, "mint failed");
}
```

`mint()` for native cToken (`CEther`) does not have any parameters, as the `Function Selector` is based on `the function name with the parenthesised list of parameter types`, when you add a nonexisting `parameter`, the `Function Selector` will be incorrect.

CTokenInterfaces.sol#L316

```
function mint(uint256 mintAmount) external payable virtual returns (uint256);
```

The current implementation uses the same `CToken` interface for both `CEther` and `CErc20` in `topUp()`, and `function mint(uint256 mintAmount)` is a nonexisting function for `CEther`.

As a result, the native token `topUp()` always revert.

CompoundHandler.sol#L57-L70

```
CToken ctoken = cTokenRegistry.fetchCToken(underlying);
uint256 initialTokens = ctoken.balanceOf(address(this));

address addr = account.addr();

if (repayDebt) {
    amount -= _repayAnyDebt(addr, underlying, amount, ctoken);
    if (amount == 0) return true;
}

uint256 err;
if (underlying == address(0)) {
    err = ctoken.mint{value: amount}(amount);
}
```

See also:

- Compound's cToken mint doc

samwerner (Backd) confirmed and resolved

[M-13] `CEthInterface#repayBorrowBehalf()` reading non-existing returns makes `_repayAnyDebt()` with CEther always revert

Submitted by WatchPug

CTokenInterfaces.sol#L355-L358

```
function repayBorrowBehalf(address borrower, uint256 repayAmount)
        external
        payable
        returns (uint256);
```

`repayBorrowBehalf()` for native cToken (`CEther`) will return nothing, while the current `CEthInterface` interface defines the returns as `(uint256)`.

As a result, `ether.repayBorrowBehalf()` will always revert

CompoundHandler.sol#L117-L118

```
CEther cether = CEther(address(ctoken));
err = cether.repayBorrowBehalf{value: debt}(account);
```

Ref:

method	CEther	CErc20
mint()	revert	error code
redeem()	error code	error code
repayBorrow()	revert	error code
repayBorrowBehalf()	revert	error code

- Compound cToken Repay Borrow Behalf doc
- Compound CEther.repayBorrowBehalf()
- Compound CErc20.repayBorrowBehalf()

chase-manning (Backd) confirmed

[M-14] `CEthInterface#mint()` reading non-existing returns makes `topUp()` with native token always revert

Submitted by WatchPug

CTokenInterfaces.sol#L345

```
function mint() external payable returns (uint256);
```

`mint()` for native cToken (`CEther`) will return nothing, while the current `CEthInterface` interface defines the returns as `(uint256)`.

In the current implementation, the interface for `CToken` is used for both `CEther` and `CErc20`.

As a result, the transaction will revert with the error: `function returned an unexpected amount of data` when `topUp()` with the native token (ETH).

CompoundHandler.sol#L57-L70

```
CToken ctoken = cTokenRegistry.fetchCToken(underlying);
uint256 initialTokens = ctoken.balanceOf(address(this));

address addr = account.addr();

if (repayDebt) {
    amount -= _repayAnyDebt(addr, underlying, amount, ctoken);
    if (amount == 0) return true;
}

uint256 err;
if (underlying == address(0)) {
    err = ctoken.mint{value: amount}(amount);
}
```

Ref:

method	CEther	CErc20
mint()	revert	error code
redeem()	error code	error code
repayBorrow()	revert	error code
repayBorrowBehalf()	revert	error code

- Compound's cToken mint doc
- Compound CEther.mint()
- Compound CErc20.mint()

chase-manning (Backd) confirmed

[M-15] Malicious Stakers can grief Keepers

Submitted by sseefried

A Staker – that has their top-up position removed after `execute` is called by a Keeper – can always cause the transaction to revert. They can do this by deploying a smart contract to the `payer` address that has implemented a `receive()` function that calls `revert()`. The revert will be triggered by the following lines in `execute`

```
if (vars.removePosition) {
    gasBank.withdrawUnused(payer);
}
```

This will consume some gas from the keeper while preventing them accruing any rewards for performing the top-up action.

Proof of Concept

I have implemented a PoC in a fork of the contest repo. The attacker's contract can be found here.

Recommend Mitigation Steps

To prevent this denial of service attack some way of blacklisting badly behaved Stakers should be added.

chase-manning (Backd) confirmed

Low Risk and Non-Critical Issues

For this contest, 39 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by **IllIllI** received the top score from the judge.

The following wardens also submitted reports: horsefacts, sseefried, robee, defsec, hubble, 0xDjango, sorrynotsorry, berndartmueller, Dravee, joestakey, StyxRave, 0xkatana, csanuragjain, dipp, hake, kebabsec, pauliax, peritoflores, securerodd, z3s, 0v3rf10w, 0x52, catchup, fatherOfBlocks, Funen, jayjonah8, Kenshin, kenta, m4rio_eth, oyc_109, rayn, remora, Ruhum, simon135, Tadashi, TerrierLover, TrungOre, and antonttc.

Vulnerability details:

[L-01] The first withdrawal for each vault from the vault reserve has no delay

`_lastWithdrawal[vault]` will always be zero for new vaults, so the check is for `0 + minWithdrawalDelay` which will always be less than `block.timestamp`

```
File: backd/contracts/vault/VaultReserve.sol   #1

102    function canWithdraw(address vault) public view returns (bool) {
103        return block.timestamp >= _lastWithdrawal[vault] + minWithdrawalDelay;
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/vault/VaultReserve.sol#L102-L103

[L-02] `AaveHandler` does not extend `BaseHandler`

Unlike `CompoundHandler`, `AaveHandler` does not extend `BaseHandler`, which will cause storage problems in future versions

```
File: backd/contracts/actions/topup/handlers/AaveHandler.sol   #1

15 contract AaveHandler is ITopUpHandler {
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/actions/topup/handlers/AaveHandler.sol#L15

[L-03] 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

```
File: contracts/actions/topup/TopUpAction.sol   #1

176     receive() external payable {
177         // solhint-disable-previous-line no-empty-blocks
178     }
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/actions/topup/TopUpAction.sol#L176-L178

```
File: contracts/pool/EthPool.sol   #2

10     receive() external payable {}
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/pool/EthPool.sol#L10

```
File: contracts/strategies/BkdEthCvx.sol   #3

46     receive() external payable {}
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/strategies/BkdEthCvx.sol#L46

```
File: contracts/strategies/StrategySwapper.sol   #4

45     receive() external payable {}
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/strategies/StrategySwapper.sol#L45

```
File: contracts/vault/EthVault.sol   #5

13     receive() external payable {}
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/vault/EthVault.sol#L13

[L-04] Front-runable initializer

If the initializer is not executed in the same transaction as the constructor, a malicious user can front-run the `initialize()` call, forcing the contract to be redeployed. Most other initializers in this project are protected, but this one appears not to be.

```
File: backd/contracts/AddressProvider.sol   #1

53    function initialize(address roleManager) external initializer {
54        AddressProviderMeta.Meta memory meta = AddressProviderMeta.Meta(true, true);
55        _addressKeyMetas.set(AddressProviderKeys._ROLE_MANAGER_KEY, meta.toUInt());
56        _setConfig(AddressProviderKeys._ROLE_MANAGER_KEY, roleManager);
57    }
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/AddressProvider.sol#L53-L57

[L-05] `safeApprove()` is deprecated

Deprecated in favor of `safeIncreaseAllowance()` and `safeDecreaseAllowance()`

See original submission for instances.

[L-06] Missing checks for `address(0x0)` when assigning values to `address` state variables

```
File: contracts/actions/topup/TopUpActionFeeHandler.sol   #1

55         actionContract = _actionContract;
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/actions/topup/TopUpActionFeeHandler.sol#L55

```
File: contracts/CvxCrvRewardsLocker.sol   #2

151         treasury = _treasury;
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/CvxCrvRewardsLocker.sol#L151

```
File: contracts/StakerVault.sol   #3

66         token = _token;
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/StakerVault.sol#L66

```
File: contracts/strategies/ConvexStrategyBase.sol   #4

100         vault = vault_;
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/strategies/ConvexStrategyBase.sol#L100

```
File: contracts/strategies/ConvexStrategyBase.sol   #5

101         _strategist = strategist_;
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/strategies/ConvexStrategyBase.sol#L101

```
File: contracts/strategies/ConvexStrategyBase.sol   #6

182         communityReserve = _communityReserve;
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/strategies/ConvexStrategyBase.sol#L182

```
File: contracts/strategies/ConvexStrategyBase.sol   #7

261         _strategist = strategist_;
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/strategies/ConvexStrategyBase.sol#L261

[L-07] `abi.encodePacked()` should not be used with dynamic types when passing the result to a hash function such as `keccak256()`

Use `abi.encode()` instead which will pad items to 32 bytes, which will prevent hash collisions (e.g. `abi.encodePacked(0x123,0x456)` => `0x123456` => `abi.encodePacked(0x1,0x23456)`, but `abi.encode(0x123,0x456)` => `0x0...1230...456`). "Unless there is a compelling reason, `abi.encode` should be preferred". If there is only one argument to `abi.encodePacked()` it can often be cast to `bytes()` or `bytes32()` instead.

```
File: contracts/actions/topup/handlers/CTokenRegistry.sol   #1

67               keccak256(abi.encodePacked(ctoken.symbol())) == keccak256(abi.encodePacke
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/actions/topup/handlers/CTokenRegistry.sol#L67

[L-08] `address.call{value:x}()` should be used instead of `payable.transfer()`

The use of `payable.transfer()` is heavily frowned upon because it can lead to the locking of funds. The `transfer()` call requires that the recipient has a `payable` callback, only provides 2300 gas for its operation. This means the following cases can cause the transfer to fail:

- The contract does not have a `payable` callback
- The contract's `payable` callback spends more than 2300 gas (which is only enough to emit something)
- The contract is called through a proxy which itself uses up the 2300 gas

```
File: backd/contracts/vault/VaultReserve.sol   #1

81            payable(msg.sender).transfer(amount);
```

uses the `onlyVault` modifier, and vaults currently have empty `payable` callbacks, so they don't currently revert https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/vault/VaultReserve.sol#L81

```
File: backd/contracts/vault/EthVault.sol   #2

29        payable(to).transfer(amount);
```

uses the `onlyPoolOrGovernance` modifier, and pools currently have an empty `payable` callback, so they don't currently rever. Governance is currently deployed and not seeing issues, so presumably it also has an empty `payable` callback https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/vault/EthVault.sol#L29

```
File: backd/contracts/vault/EthVault.sol   #3

37        payable(addressProvider.getTreasury()).transfer(amount);
```

the treasury is currently deployed and not seeing issues, so presumably it also has an empty `payable` callback https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/vault/EthVault.sol#L37

```
File: backd/contracts/strategies/BkdEthCvx.sol   #4

77            payable(vault).transfer(amount);
```

vaults currently have an empty `payable` callback https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/strategies/BkdEthCvx.sol#L77

```
File: backd/contracts/strategies/BkdEthCvx.sol   #5

93        payable(vault).transfer(amount);
```

vaults currently have an empty `payable` callback https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/strategies/BkdEthCvx.sol#L93

```
File: backd/contracts/strategies/BkdEthCvx.sol   #6

117        payable(vault).transfer(underlyingBalance);
```

vaults currently have an empty `payable` callback https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/strategies/BkdEthCvx.sol#L117

[L-09] Upgradeable contract is missing a `__gap[50]` storage variable to allow for new storage variables in later versions

See this link for a description of this storage variable. While some contracts may not currently be sub-classed, adding the variable now protects against forgetting to add it in the future.

```
File: contracts/LpToken.sol   #1

10 contract LpToken is ILpToken, ERC20Upgradeable {
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/LpToken.sol#L10

[L-10] Math library unnecessarily overflows during some operations

In the example below, `a + b` may overflow even though the division that comes later would prevent it. This particular case can be prevented by doing `(a & b) + (a ^ b) / b`. There are other functions with similar issues. See this library for ways of doing math without this sort of issue.

```
File: backd/libraries/ScaledMath.sol   #1

40    function divRoundUp(uint256 a, uint256 b) internal pure returns (uint256) {
41        return (a + b - 1) / b;
42    }
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/libraries/ScaledMath.sol#L40-L42

[N-01] `_prepareDeadline()`, `_setConfig()`, and `_executeDeadline()` should be `private`

These functions have the ability to bypass the timelocks of every setting. No contract besides the `Preparable` contract itself should need to call these functions, and having them available will lead to exploits. The contracts that currently call `_setConfig()` in their constructors should be given a new function `_initConfig()` for this purpose. The `Vault` calls some of these functions as well, and should be changed to manually inspect the deadline rather than mucking with the internals, which is error-prone. The mappings should also be made `private`, and there should be public getters to read their values

```
File: backd/contracts/utils/Preparable.sol   #1

    /**
     * @notice Execute uint256 config update (with time delay enforced).
     * @dev Needs to be called after the update was prepared. Fails if called before time
     * @return New value.
     */
    function _executeUInt256(bytes32 key) internal returns (uint256) {
        _executeDeadline(key);
        uint256 newValue = pendingUInts256[key];
        _setConfig(key, newValue);
        return newValue;
    }

    /**
     * @notice Execute address config update (with time delay enforced).
     * @dev Needs to be called after the update was prepared. Fails if called before time
     * @return New value.
     */
    function _executeAddress(bytes32 key) internal returns (address) {
        _executeDeadline(key);
        address newValue = pendingAddresses[key];
        _setConfig(key, newValue);
        return newValue;
    }
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/utils/Preparable.sol#L115-L137

[N-02] Open TODOs

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

```
File: contracts/actions/topup/TopUpAction.sol   #1

713         // TODO: add constant gas consumed for transfer and tx prologue
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/actions/topup/TopUpAction.sol#L713

```
File: contracts/strategies/ConvexStrategyBase.sol   #2

4 // TODO Add validation of curve pools
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/strategies/ConvexStrategyBase.sol#L4

```
File: contracts/strategies/ConvexStrategyBase.sol   #3

5 // TODO Test validation
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/strategies/ConvexStrategyBase.sol#L5

[N-03] `payable` function does not reject payments to ERC20 tokens

```
File: backd/contracts/vault/VaultReserve.sol   #1

50        if (token == address(0)) {
51            require(msg.value == amount, Error.INVALID_AMOUNT);
52            _balances[msg.sender][token] += msg.value;
53            return true;
54        }
55        uint256 balance = IERC20(token).balanceOf(address(this));
```

After the if-statement there should be a `require(0 == msg.value)` to ensure no Ether is being used when updating ERC20 balances. This is non-critical since the function has the `onlyVault` modifier, and presumably vaults would be coded never to deposit Ether to ERC20 tokens https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/vault/VaultReserve.sol#L50-L55

[N-04] Adding a `return` statement when the function defines a named return variable, is redundant

```
File: contracts/pool/PoolFactory.sol   #1

216         return addrs;
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/pool/PoolFactory.sol#L216

[N-05] `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`.

```
File: contracts/actions/topup/TopUpAction.sol   #1

742     function prepareTopUpHandler(bytes32 protocol, address newHandler)
743         public
744         onlyGovernance
745         returns (bool)
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/actions/topup/TopUpAction.sol#L742-L745

```
File: contracts/CvxCrvRewardsLocker.sol   #2

222     function withdraw(address token, uint256 amount) public onlyGovernance returns (bool
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/CvxCrvRewardsLocker.sol#L222

[N-06] `constants` should be defined rather than using magic numbers

See original submission for instances.

[N-07] Large multiples of ten should use scientific notation (e.g. `1e6`) rather than decimal literals (e.g. `1000000`), for readability

```
File: contracts/utils/CvxMintAmount.sol   #1

7     uint256 private constant _CLIFF_SIZE = 100000 * 1e18; //new cliff every 100,000 tokens
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/utils/CvxMintAmount.sol#L7

```
File: contracts/utils/CvxMintAmount.sol   #2

9     uint256 private constant _MAX_SUPPLY = 100000000 * 1e18; //100 mil max supply
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/utils/CvxMintAmount.sol#L9

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

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

```
File: contracts/actions/topup/handlers/CTokenRegistry.sol   #1
```

```
2 pragma solidity 0.8.9;
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/actions/topup/handlers/CTokenRegistry.sol#L2

```
File: contracts/actions/topup/TopUpActionFeeHandler.sol   #2
```

```
2 pragma solidity 0.8.9;
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/actions/topup/TopUpActionFeeHandler.sol#L2

```
File: contracts/actions/topup/TopUpAction.sol   #3
```

```
2 pragma solidity 0.8.9;
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/actions/topup/TopUpAction.sol#L2

[N-09] Constant redefined elsewhere

Consider defining in only one contract so that values cannot become out of sync when only one location is updated. A cheap way to store constants in a single location is to create an `internal constant` in a `library`. If the variable is a local cache of another contract's value, consider making the cache variable internal or private, which will require external users to query the contract with the source of truth, so that callers don't get out of sync.

See original submission for instances.

[N-10] Inconsistent spacing in comments

Some lines use `// x` and some use `//x`. The instances below point out the usages that don't follow the majority, within each file

```
File: contracts/utils/CvxMintAmount.sol   #1
```

```
8     uint256 private constant _CLIFF_COUNT = 1000; // 1,000 cliffs
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/utils/CvxMintAmount.sol#L8

```
File: contracts/utils/CvxMintAmount.sol   #2

11         IERC20(address(0x4e3FBD56CD56c3e72c1403e103b45Db9da5B9D2B)); // CVX Token
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/utils/CvxMintAmount.sol#L11

[N-11] Typos

See original submission for instances.

[N-12] File is missing NatSpec

```
File: contracts/access/Authorization.sol (various lines)   #1
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/access/Authorization.sol

```
File: contracts/access/RoleManager.sol (various lines)   #2
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/access/RoleManager.sol

```
File: contracts/oracles/ChainlinkUsdWrapper.sol (various lines)   #3
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/oracles/ChainlinkUsdWrapper.sol

```
File: contracts/oracles/OracleProviderExtensions.sol (various lines)   #4
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/oracles/OracleProviderExtensions.sol

```
File: contracts/pool/Erc20Pool.sol (various lines)   #5
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/pool/Erc20Pool.sol

```
File: contracts/pool/EthPool.sol (various lines)   #6
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/pool/EthPool.sol

```
File: contracts/utils/CvxMintAmount.sol (various lines)   #7
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/utils/CvxMintAmount.sol

```
File: contracts/vault/Erc20Vault.sol (various lines)   #8
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/vault/Erc20Vault.sol

```
File: contracts/vault/EthVault.sol (various lines)   #9
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/contracts/vault/EthVault.sol

```
File: libraries/AddressProviderMeta.sol (various lines)   #10
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/libraries/AddressProviderMeta.sol

```
File: libraries/Errors.sol (various lines)   #11
```

https://github.com/code-423n4/2022-04-backd/blob/c856714a50437cb33240a5964b63687c9876275b/backd/libraries/Errors.sol

[N-13] NatSpec is incomplete

See original submission for instances.

[N-14] Event is missing `indexed` fields

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

See original submission for instances.

chase-manning (Backd) resolved and commented: > I consider this report to be of particularly high quality.

gzeon (judge) commented: > Nice submission, warden covered basically all the low risk and non-critical issues. Would be nice if there was an index.

Gas Optimizations

For this contest, 35 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **joestakey** received the top score from the judge.

The following wardens also submitted reports: Tomio, IllIllI, Dravee, catchup, defsec, securerodd, 0xkatana, kenta, robee, slywaters, sorrynotsorry, 0v3rf10w, 0x1f8b, 0x4non, 0xNazgul, fatherOfBlocks, Funen, NoamYakov, pauliax, rfa, saian, TerrierLover, WatchPug, MaratCerby, 0xDjango, 0xmint, hake, horsefacts, oyc_109, rayn, simon135, Tadashi, tin537, and z3s.

[G-01] Caching storage variables in memory to save gas

PROBLEM

Anytime you are reading from storage more than once, it is cheaper in gas cost to cache the variable in memory: a SLOAD cost 100gas, while MLOAD and MSTORE cost 3 gas.

In particular, in `for` loops, when using the length of a storage array as the condition being checked after each loop, caching the array length in memory can yield significant gas savings if the array length is high.

PROOF OF CONCEPT

Instances include:

AaveHandler.sol scope: `topUp()`

- `weth` is read twice

```
AaveHandler.sol:44
AaveHandler.sol:45
```

- `lendingPool` is read 4 times

```
AaveHandler.sol:50
AaveHandler.sol:53
AaveHandler.sol:60
AaveHandler.sol:65
```

CompoundHandler.sol scope: `_getAccountBorrowsAndSupply()`

- `comptroller` is read (2 + `assets.length`) times. Number of read depends on the length of `assets` as it is in a for loop

```
CompoundHandler.sol:132
CompoundHandler.sol:134
CompoundHandler.sol:142
```

CTokenRegistry.sol scope: `_isCTokenUsable()`

- `comptroller` is read 3 times

```
CTokenRegistry.sol:77
CTokenRegistry.sol:79
CTokenRegistry.sol:80
```

TopUpAction.sol scope: `resetPosition()`

- `addressProvider` is read twice

```
TopUpAction.sol:284
TopUpAction.sol:295
```

scope: `execute()`

- `addressProvider` is read 3 times

TopUpAction.sol:562
TopUpAction.sol:604
TopUpAction.sol:632

BkdEthCvx.sol scope: `_withdraw()`

- `vault` is read twice

BkdEthCvx.sol:77
BkdEthCvx.sol:93

BkdTriHopCvx.sol scope: `_withdraw()`

- `vault` is read twice

BkdTriHopCvx.sol:175
BkdTriHopCvx.sol:201

ConvexStrategyBase.sol scope: `addRewardToken()`

- `_strategySwapper` is read twice

ConvexStrategyBase.sol:279
ConvexStrategyBase.sol:280

scope: `harvestable()`

- `crvCommunityReserveShare` is read twice

ConvexStrategyBase.sol:307
ConvexStrategyBase.sol:311

- `_rewardTokens.length()` is read `_rewardTokens.length()` times. Number of read depends on the length of `_rewardsTokens` as it is in a for loop

ConvexStrategyBase.sol:313

scope: `_harvest()`

- `_rewardTokens.length()` is read `_rewardTokens.length()` times. Number of read depends on the length of `_rewardsTokens` as it is in a for loop

ConvexStrategyBase.sol:380

scope: `_sendCommunityReserveShare()`

- `cvxCommunityReserveShare` is read twice

ConvexStrategyBase.sol:398
ConvexStrategyBase.sol:409

Vault.sol scope: `_handleExcessDebt()`

- `reserve` is read 3 times

```
Vault.sol:645
Vault.sol:648
Vault.sol:649
```

scope: `_handleExcessDebt()`

- `totalDebt` is read twice

```
Vault.sol:657
Vault.sol:658
```

Vault.sol scope: `stakeFor()`

- `token` is read 4 times

```
Vault.sol:324
Vault.sol:331
Vault.sol:338
Vault.sol:339
```

scope: `unStakeFor()`

- `token` is read 4 times

```
Vault.sol:365
Vault.sol:376
Vault.sol:382
Vault.sol:384
```

MITIGATION

cache these storage variables in memory

[G-02] Calldata instead of memory for RO function parameters

PROBLEM

If a reference type function parameter is read-only, it is cheaper in gas to use calldata instead of memory. Calldata is a non-modifiable, non-persistent area where function arguments are stored, and behaves mostly like memory.

Try to use calldata as a data location because it will avoid copies and also makes sure that the data cannot be modified.

PROOF OF CONCEPT

Instances include:

RoleManager.sol scope: `hasAnyRole()`

```
RoleManager.sol:73: bytes32[] memory roles
```

AaveHandler.sol scope: `topUp()`

```
AaveHandler.sol:40: bytes memory extra
```

CompoundHandler.sol scope: `topUp()`

```
CompoundHandler.sol:54: bytes memory extra
```

TopUpAction.sol scope: `getHealthFactor()`

```
TopUpAction.sol:760: bytes memory extra
```

TopUpKeeperHelper.sol scope: `canExecute()`

```
TopUpKeeperHelper.sol:108: ITopUpAction.RecordKey memory key
```

scope: `_canExecute()`

```
TopUpKeeperHelper.sol:131: ITopUpAction.RecordWithMeta memory position
```

scope: `_positionToTopup()`

```
TopUpKeeperHelper.sol:145: ITopUpAction.RecordWithMeta memory position
```

scope: `_shortenTopups()`

```
TopUpKeeperHelper.sol:159: TopupData[] memory topups
```

Erc20Pool.sol scope: `initialize()`

```
Erc20Pool.sol:15: string memory name_
```

EthPool.sol scope: `initialize()`

```
EthPool.sol:13: string memory name_
```

LiquidityPool.sol scope: `initialize()`

```
LiquidityPool.sol:702: string memory name_
```

LpToken.sol scope: `initialize()`

```
LpToken.sol:29: string memory name_
LpToken.sol:30: string memory symbol_
```

MITIGATION

Replace `memory` with `calldata`

[G-03] Comparisons with zero for unsigned integers

PROBLEM

`>0` is less gas efficient than `!0` if you enable the optimizer at 10k AND you're in a require statement. Detailed explanation with the opcodes here

PROOF OF CONCEPT

Instances include:

TopUpAction.sol scope: `register()`

`TopUpAction.sol:210`

scope: `execute()`

`TopUpAction.sol:554`

TopUpActionFeeHandler.sol scope: `claimKeeperFeesForPool()`

`TopUpActionFeeHandler.sol:123`

LiquidityPool.sol scope: `updateDepositCap()`

`LiquidityPool.sol:401`

scope: `calcRedeem()`

`LiquidityPool.sol:471`
`LiquidityPool.sol:473`

scope: `redeem()`

`LiquidityPool.sol:549`

Vault.sol scope: `withdrawFromReserve()`

`Vault.sol:164`

BkdLocker.sol scope: `depositFees()`

`BkdLocker.sol:90`
`BkdLocker.sol:91`
`BkdLocker.sol:136`

MITIGATION

Replace `> 0` with `!0`

[G-04] Comparison Operators

PROBLEM

In the EVM, there is no opcode for >= or <=. When using greater than or equal, two operations are performed: > and =.

Using strict comparison operators hence saves gas

PROOF OF CONCEPT

Instances include:

TopUpAction.sol

```
TopUpAction.sol:61
TopUpAction.sol:212
TopUpAction.sol:224
TopUpAction.sol:328
TopUpAction.sol:360
TopUpAction.sol:361
TopUpAction.sol:500
TopUpAction.sol:501
TopUpAction.sol:576
TopUpAction.sol:584
TopUpAction.sol:724
```

TopUpActionFeeHandler.sol

```
TopUpActionFeeHandler.sol:54
TopUpActionFeeHandler.sol:151
TopUpActionFeeHandler.sol:163
TopUpActionFeeHandler.sol:196
TopUpActionFeeHandler.sol:208
```

ChainLinkOracleProvider.sol

```
ChainLinkOracleProvider.sol:41
ChainLinkOracleProvider.sol:57
```

EthPool.sol

```
EthPool.sol:442
EthPool.sol:208
EthPool.sol:518
EthPool.sol:525
EthPool.sol:551
EthPool.sol:562
EthPool.sol:690
EthPool.sol:811
EthPool.sol:812
```

BkdEthCvx.sol

```
BkdEthCvx.sol:76
```

BkdTriHopCvx.sol

```
BkdTriHopCvx.sol:174
```

ConvexStrategyBase.sol

```
ConvexStrategyBase.sol:197
ConvexStrategyBase.sol:214
```

StrategySwapper.sol

```
StrategySwapper.sol:110
```

CvxMintAmount.sol

```
CvxMintAmount.sol:21
```

Preparable.sol

```
Preparable.sol:29
Preparable.sol:110
```

Vault.sol

```
Vault.sol:88
Vault.sol:89
Vault.sol:90
Vault.sol:167
Vault.sol:264
Vault.sol:323
Vault.sol:392
Vault.sol:437
Vault.sol:482
Vault.sol:712
Vault.sol:763
```

VaultReserve.sol

```
VaultReserve.sol:59
VaultReserve.sol:75
VaultReserve.sol:103
```

BkdLocker.sol

```
BkdLocker.sol:119
BkdLocker.sol:140
BkdLocker.sol:281
```

Controller.sol

```
Controller:98
```

CvxCrvRewardsLocker.sol

```
CvxCrvRewardsLocker:84
```

GasBank.sol

```
GasBank:68
GasBank:76
```

StakerVault.sol

```
StakerVault:107
StakerVault:150
StakerVault:153
StakerVault:324
StakerVault:368
StakerVault:371
```

MITIGATION

Replace `<=` with `<`, and `>=` with `>`. Do not forget to increment/decrement the compared variable

example:

```
-require(maxFee >= minFee, Error.INVALID_AMOUNT);
+require(maxFee > minFee - 1, Error.INVALID_AMOUNT);
```

When the comparison is with a constant storage variable, you can also do the increment in the storage variable declaration

example:

```
-require(maxFee <= _MAX_WITHDRAWAL_FEE, Error.INVALID_AMOUNT)
+require(maxFee < _MAX_WITHDRAWAL_FEE_PLUS_ONE, Error.INVALID_AMOUNT)
```

However, when `1` is negligible compared to the variable (with is the case here as the variable is in the order of `10**16`), it is not necessary to increment.

[G-05] Custom Errors

PROBLEM

Custom errors from Solidity 0.8.4 are cheaper than revert strings (cheaper deployment cost and runtime cost when the revert condition is met) while providing the same amount of information, as explained here

Custom errors are defined using the error statement

PROOF OF CONCEPT

Instances include:

RoleManager.sol

```
RoleManager.sol:44
RoleManager.sol:110
RoleManager.sol:111
```

AaveHandler.sol

```
AaveHandler.sol:51
```

CompoundHandler.sol

```
CompoundHandler.sol:74
CompoundHandler.sol:80
CompoundHandler.sol:123
CompoundHandler.sol:141
CompoundHandler.sol:148
```

TopUpAction.sol

```
TopUpAction.sol:67
TopUpAction.sol:98
TopUpAction.sol:185
TopUpAction.sol:209
TopUpAction.sol:210
TopUpAction.sol:211
TopUpAction.sol:212
TopUpAction.sol:213
TopUpAction.sol:217
TopUpAction.sol:218
TopUpAction.sol:224
TopUpAction.sol:282
TopUpAction.sol:328
TopUpAction.sol:359
TopUpAction.sol:546
TopUpAction.sol:553
TopUpAction.sol:554
TopUpAction.sol:560
TopUpAction.sol:575
TopUpAction.sol:583
TopUpAction.sol:676
TopUpAction.sol:723
TopUpAction.sol:928
```

TopUpActionFeeHandler.sol

```
TopUpActionFeeHandler.sol:67
TopUpActionFeeHandler.sol:68
TopUpActionFeeHandler.sol:87
TopUpActionFeeHandler.sol:123
TopUpActionFeeHandler.sol:151
TopUpActionFeeHandler.sol:161
TopUpActionFeeHandler.sol:196
TopUpActionFeeHandler.sol:206
```

ChainLinkOracleProvider.sol

```
ChainLinkOracleProvider.sol:31
ChainLinkOracleProvider.sol:41
ChainLinkOracleProvider.sol:53
ChainLinkOracleProvider.sol:57
ChainLinkOracleProvider.sol:58
```

Erc20Pool.sol

```
Erc20Pool.sol:20
Erc20Pool.sol:30
```

EthPool.sol

```
EthPool.sol:25
EthPool.sol:26
```

LiquidityPool.sol

```
LiquidityPool.sol:76
LiquidityPool.sol:136
LiquidityPool.sol:137
LiquidityPool.sol:155
LiquidityPool.sol:179
LiquidityPool.sol:208
LiquidityPool.sol:331
LiquidityPool.sol:333
LiquidityPool.sol:387
LiquidityPool.sol:399
LiquidityPool.sol:400
LiquidityPool.sol:401
LiquidityPool.sol:441
LiquidityPool.sol:471
LiquidityPool.sol:473
LiquidityPool.sol:517
LiquidityPool.sol:525
LiquidityPool.sol:549
LiquidityPool.sol:551
LiquidityPool.sol:562
LiquidityPool.sol:811
LiquidityPool.sol:812
```

PoolFactory.sol

```
PoolFactory.sol:159
PoolFactory.sol:162
PoolFactory.sol:165
PoolFactory.sol:170
PoolFactory.sol:180
PoolFactory.sol:184
```

BkdTriHopCvx.sol

```
BkdTriHopCvx.sol:133
BkdTriHopCvx.sol:147
```

ConvexStrategyBase.sol

```
ConvexStrategyBase.sol:117
ConvexStrategyBase.sol:144
ConvexStrategyBase.sol:197
ConvexStrategyBase.sol:198
ConvexStrategyBase.sol:214
ConvexStrategyBase.sol:215
ConvexStrategyBase.sol:260
ConvexStrategyBase.sol:273
```

StrategySwapper.sol

```
StrategySwapper.sol:69
StrategySwapper.sol:110
StrategySwapper.sol:111
StrategySwapper.sol:123
```

StrategySwapper.sol:124
StrategySwapper.sol:139

Preparable.sol

Preparable.sol:28
Preparable.sol:29
Preparable.sol:86
Preparable.sol:98
Preparable.sol:110
Preparable.sol:111

Erc20Vault.sol

Erc20Vault.sol:20

Vault.sol

Vault.sol:88
Vault.sol:89
Vault.sol:90
Vault.sol:164
Vault.sol:165
Vault.sol:167
Vault.sol:194
Vault.sol:195
Vault.sol:198
Vault.sol:264
Vault.sol:392
Vault.sol:429
Vault.sol:762

VaultReserve.sol

VaultReserve.sol:51
VaultReserve.sol:59
VaultReserve.sol:73
VaultReserve.sol:75

AddressProvider.sol

AddressProvider.sol:64
AddressProvider.sol:70
AddressProvider.sol:96
AddressProvider.sol:100
AddressProvider.sol:170
AddressProvider.sol:179

```
AddressProvider.sol:188
AddressProvider.sol:230
AddressProvider.sol:231
AddressProvider.sol:249
AddressProvider.sol:259
AddressProvider.sol:284
AddressProvider.sol:285
AddressProvider.sol:314
AddressProvider.sol:417
AddressProvider.sol:423
```

BkdLocker.sol

```
BkdLocker.sol:58
BkdLocker.sol:90
BkdLocker.sol:91
BkdLocker.sol:118
BkdLocker.sol:136
BkdLocker.sol:207
```

Controller.sol

```
Controller:32
Controller:33
Controller:80
```

CvxCrvRewardsLocker.sol

```
CvxCrvRewardsLocker:83
CvxCrvRewardsLocker:135
```

GasBank.sol

```
GasBank:42
GasBank:68
GasBank:69
GasBank:76
GasBank:91
```

LpToken.sol

```
LpToken:34
```

StakerVault.sol

```
StakerVault:70
StakerVault:93
StakerVault:106
```

```
StakerVault:107
StakerVault:139
StakerVault:150
StakerVault:153
StakerVault:203
StakerVault:224
StakerVault:324
StakerVault:340
StakerVault:367
StakerVault:371
```

SwapperRegistry.sol

```
SwapperRegistry:35
```

MITIGATION

Replace require statements with custom errors.

[G-06] Default value initialization

PROBLEM

If a variable is not set/initialized, it is assumed to have the default value (0, false, 0x0 etc depending on the data type). Explicitly initializing it with its default value is an anti-pattern and wastes gas.

PROOF OF CONCEPT

Instances include:

RoleManager.sol

```
RoleManager.sol:80
RoleManager.sol:110
RoleManager.sol:111
```

CompoundHandler.sol

```
CompoundHandler.sol:135
```

CTokenRegistry.sol

```
CTokenRegistry.sol:61
```

TopUpAction.sol

```
TopUpAction.sol:188
TopUpAction.sol:452
TopUpAction.sol:456
TopUpAction.sol:479
TopUpAction.sol:506
TopUpAction.sol:891
```

TopUpActionKeeperHandler.sol

```
TopUpActionKeeperHandler.sol:43
TopUpActionKeeperHandler.sol:46
TopUpActionKeeperHandler.sol:72
TopUpActionKeeperHandler.sol:93
TopUpActionKeeperHandler.sol:165
```

LiquidityPool.sol

```
LiquidityPool.sol:483
```

ConvexStrategyBase.sol

```
ConvexStrategyBase.sol:313
ConvexStrategyBase.sol:380
```

Vault.sol

```
Vault.sol:42
Vault.sol:135
Vault.sol:583
```

BkdLocker.sol

```
BkdLocker.sol:133
BkdLocker.sol:310
```

Controller.sol

```
Controller:114
Controller:117
```

CvxCrvRewardsLocker.sol

```
CvxCrvRewardsLocker:43
```

StakerVault.sol

```
StakerVault:144
StakerVault:260
```

MITIGATION

Remove explicit initialization for default values.

[G-07] Prefix increments

PROBLEM

Prefix increments are cheaper than postfix increments.

PROOF OF CONCEPT

Instances include:

RoleManager.sol

```
RoleManager.sol:80
```

CompoundHandler.sol

```
CompoundHandler.sol:135
```

CTokenRegistry.sol

```
CTokenRegistry.sol:61
```

TopUpAction.sol

```
TopUpAction.sol:188
TopUpAction.sol:456
TopUpAction.sol:479
TopUpAction.sol:506
TopUpAction.sol:891
```

TopUpActionKeeperHandler.sol

```
TopUpActionKeeperHandler.sol:43
TopUpActionKeeperHandler.sol:46
TopUpActionKeeperHandler.sol:50
TopUpActionKeeperHandler.sol:72
TopUpActionKeeperHandler.sol:93
TopUpActionKeeperHandler.sol:165
```

ConvexStrategyBase.sol

```
ConvexStrategyBase.sol:313
ConvexStrategyBase.sol:380
```

BkdLocker.sol

```
BkdLocker.sol:310
```

Controller.sol

```
Controller:117
```

StakerVault.sol

```
StakerVault:260
```

MITIGATION

change `variable++` to `++variable`

[G-08] Redundant code

IMPACT

Redundant code should be avoided as it costs unnecessary gas

PROOF OF CONCEPT

Instances include:

Preparable.sol

```
Preparable.sol:140:
address oldValue = currentAddresses[key];
currentAddresses[key] = value;
pendingAddresses[key] = address(0);
deadlines[key] = 0;
emit ConfigUpdatedAddress(key, oldValue, value);
return value;
```

We can update `currentAddresses[key]` after emitting the event to save the gas of the declaration of `oldValue`:

```
+emit ConfigUpdatedAddress(key, currentAddresses[key], value);
pendingAddresses[key] = address(0);
deadlines[key] = 0;
currentAddresses[key] = value;
return value;
```

MITIGATION

see Proof of Concept for mitigation steps.

[G-09] Require instead of &&

IMPACT

Require statements including conditions with the `&&` operator can be broken down in multiple require statements to save gas.

PROOF OF CONCEPT

Instances include:

TopUpAction.sol

```
TopUpAction:360:
require(
            newSwapperSlippage >= _MIN_SWAPPER_SLIPPAGE &&
                newSwapperSlippage <= _MAX_SWAPPER_SLIPPAGE,
            Error.INVALID_AMOUNT
        );
```

ConvexStrategyBase.sol

```
ConvexStrategyBase:274:
require(
            token_ != address(_CVX) && token_ != address(underlying) && token_ != address(_C
            Error.INVALID_TOKEN_TO_ADD
        );
```

SwapperRegistry.sol

```
SwapperRegistry:35:
require(
            fromToken != toToken &&
                fromToken != address(0) &&
                toToken != address(0) &&
                newSwapper != address(0),
            Error.INVALID_TOKEN_PAIR
        );
```

MITIGATION

Breakdown each condition in a separate `require` statement (though require statements should be replaced with custom errors)

```
require( newSwapperSlippage >=_MIN_SWAPPER_SLIPPAGE);
require(newSwapperSlippage <= _MAX_SWAPPER_SLIPPAGE,
           Error.INVALID_AMOUNT)
```

[G-10] Tight Variable Packing

PROBLEM

Solidity contracts have contiguous 32 bytes (256 bits) slots used in storage. By arranging the variables, it is possible to minimize the number of slots used within a contract's storage and therefore reduce deployment costs.

address type variables are each of 20 bytes size (way less than 32 bytes). However, they here take up a whole 32 bytes slot (they are contiguous).

As bool type variables are of size 1 byte, there's a slot here that can get saved by moving them closer to an address

PROOF OF CONCEPT

Instances include:

VaultStorage.sol

```
VaultStorage.sol:11:
address public pool;
uint256 public totalDebt;
bool public strategyActive;
```

MITIGATION

Place `strategyActive` after `pool` to save one storage slot

```
address public pool;
+bool public strategyActive;
uint256 public totalDebt;
```

[G-11] Unchecked arithmetic

PROBLEM

The default "checked" behavior costs more gas when adding/diving/multiplying, because under-the-hood those checks are implemented as a series of opcodes that, prior to performing the actual arithmetic, check for under/overflow and revert if it is detected.

if it can statically be determined there is no possible way for your arithmetic to under/overflow (such as a condition in an if statement), surrounding the arithmetic in an `unchecked` block will save gas

PROOF OF CONCEPT

Instances include:

LiquidityPool.sol

```
LiquidityPool.sol:751: underlyingBalance - underlyingToWithdraw; //because of the condition
```

BkdEthCvx.sol

```
BkdEthCvx.sol:83: uint256 requiredUnderlyingAmount = amount - underlyingBalance; //because o
```

BkdTriHopCvx.sol

```
BkdTriHopCvx.sol:181: uint256 requiredUnderlyingAmount = amount - underlyingBalance; //becau
```

CvxMintAmount.sol

```
CvxMintAmount.sol:24: uint256 remaining = _CLIFF_COUNT - currentCliff; //because of the conc
```

Vault.sol

```
Vault.sol:24: uint256 remaining = _CLIFF_COUNT - currentCliff; //because of the condition li
```

```
Vault.sol:125: uint256 requiredWithdrawal = amount - availableUnderlying_; //because of the
```

```
Vault.sol:130: uint256 newTarget = (allocated - requiredWithdrawal) //because of the conditi
```

```
Vault.sol:141: uint256 totalUnderlyingAfterWithdraw = totalUnderlying - amount; //because of
```

```
Vault.sol:440: waitingForRemovalAllocated = _waitingForRemovalAllocated - withdrawn; //becau
```

```
Vault.sol:444: uint256 profit = withdrawn - allocated; //because of the condition line 443,
```

```
Vault.sol:452: allocated -= withdrawn; //because of the condition line 443, the underflow ch
```

```
Vault.sol:591: uint256 profit = allocatedUnderlying - amountAllocated; //because of the conc
```

```
Vault.sol:595: profit -= currentDebt; //because of the condition line 593, the underflow che
```

```
Vault.sol:600: currentDebt -= profit; //because of the condition line 593, the underflow che
```

```
Vault.sol:605: uint256 loss = amountAllocated - allocatedUnderlying; //because of the condit
```

```
Vault.sol:784: uint256 withdrawAmount = allocatedUnderlying - target; //because of the condi
```

```
Vault.sol:790: uint256 depositAmount = target - allocatedUnderlying; //because of the condit
```

StakerVault.sol

```
StakerVault.sol:164: uint256 srcTokensNew = srcTokens - amount; //because of the condition l
```

MITIGATION

Place the arithmetic operations in an `unchecked` block

chase-manning (Backd) resolved and commented: > I consider this report to be of particularly high quality.

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.