GEARBOX

Gearbox v2.0

Smart Contract Security Assessment

Version: 2.1

August, 2022

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Gearbox smart contracts. The review focused solely on the security aspects of the Solidity implementation of the contract, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the Gearbox smart contracts contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the Gearbox smart contracts.

Overview

Gearbox is a generalised leverage protocol, which gives access to undercollateralised lending through an entirely new credit account primitive. Credit accounts enable the extension of credit to borrowers which is not built upon a reputation system, or "credit score".

Credit accounts consist of two types of collateral:

- Initial funds provided by the borrower when they open their credit account.
- Borrowed funds that are provided by the protocol's pool service.

Borrowers are able to utilise funds from liquidity providers to interact with a subset of trusted protocols. These interactions are accessible through various adapters which prevent funds from maliciously exiting the borrower's credit account.

Gearbox 2.0 introduces several new changes to the previous iteration:

- Upgrades legacy contracts to Solidity v0.8.

- Improves liquidation flows.
- Introduces multicall functionality for running batches of transactions through a credit account.
- Utilises USD price fees instead of ETH.

Liquidity providers earn yield on the fees generated by the Gearbox ecosystem. Upon closure or liquidation of the credit account, the borrowed funds are returned to the pool alongside any interest and fees accrued on the duration of the loan.

Security Assessment Summary

This review was conducted on the files accessed from a private Gearbox 2.0 repository. This repository was under active development by the Gearbox team during review, and four commit hashes were targeted throughout the course of the review.

The following commit hashes detail the targets (ordered with respect to time):

- 432a213
- 6c2bcdc
- 5d4ec27
- 0f500f0

The final scope of the review included the following files:

```
├── adapters
│   ├── convex
│   │   ├── ConvexV1_BaseRewardPool.sol
│   │   ├── ConvexV1_Booster.sol
│   │   ├── ConvexV1_ClaimZap.sol
│   │   └── ConvexV1_StakedPositionToken.sol
│   ├── curve
│   │   ├── CurveV1_2.sol
│   │   ├── CurveV1_3.sol
│   │   ├── CurveV1_4.sol
│   │   ├── CurveV1_Base.sol
│   │   ├── CurveV1_stETHGateway.sol
│   │   └── CurveV1_stETH.sol
│   ├── lido
│   │   ├── LidoV1.sol
│   │   └── LidoV1_WETHGateway.sol
│   ├── uniswap
│   │   ├── UniswapV2.sol
│   │   └── UniswapV3.sol
│   └── yearn
│       └── YearnV2.sol
├── core
│   ├── AccountFactory.sol
│   ├── ACL.sol
│   ├── ACLTrait.sol
│   ├── AddressProvider.sol
│   ├── ContractsRegister.sol
│   ├── DataCompressor.sol
│   └── WETHGateway.sol
├── credit
│   ├── CreditAccount.sol
│   ├── CreditConfigurator.sol
│   ├── CreditFacade.sol
│   └── CreditManager.sol
├── factories
│   ├── CreditManagerFactory.sol
```

```
│   ├── GenesisDeployer.sol
│   └── PoolFactory.sol
├── libraries
│   ├── Constants.sol
│   ├── Errors.sol
│   ├── PercentageMath.sol
│   ├── Types.sol
│   └── WadRayMath.sol
├── oracles
│   ├── curve
│   │   ├── CurveLP2PriceFeed.sol
│   │   ├── CurveLP3PriceFeed.sol
│   │   ├── CurveLP4PriceFeed.sol
│   │   └── CurveLPFactory.sol
│   ├── PriceOracle.sol
│   ├── yearn
│   │   └── YearnPriceFeed.sol
│   └── ZeroPriceFeed.sol
├── pool
│   ├── LinearInterestRateModel.sol
│   └── PoolService.sol
├── support
│   └── PathFinder.sol
└── tokens
    ├── DieselToken.sol
    ├── GearToken.sol
    └── PhantomERC20.sol
```

Note: third party libraries and dependencies were excluded from the scope of this assessment.

Retesting activities targeted commit d1d7e45, and focused exclusively on assessing the fixes to the issues raised in this report. Any additional functionalities which may have been introduced prior to this commit are deemed out-of-scope.

The manual code review section of the report focused on identifying any and all issues/vulnerabilities associated with the business logic implementation of the contracts. Specifically, their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout). Additionally, the manual review process focused on all known Solidity anti-patterns and attack vectors. These include, but are not limited to, the following vectors: re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers. For a more thorough, but non-exhaustive list of examined vectors, see [1, 2].

To support this review, the testing team used the following automated testing tools:

- Mythril: `https://github.com/ConsenSys/mythril`
- Slither: `https://github.com/trailofbits/slither`
- Surya: `https://github.com/ConsenSys/surya`

Output for these automated tools is available upon request.

Findings Summary

The testing team identified a total of 22 issues during this assessment. Categorized by their severity:

- Critical: 4 issues.
- High: 4 issues.
- Medium: 8 issues.
- Low: 3 issues.
- Informational: 3 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Gearbox smart contracts. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as “informational”.

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
GRB-01	Incorrect Parsing of Token Path Between Gearbox and Uniswap	Critical	Resolved
GRB-02	Liquidators Can Manipulate Pool's Profit or Loss Values	Critical	Resolved
GRB-03	Denial-of-Service (DoS) by Opening a Credit Account on Behalf of the `CreditFacade` Contract	Critical	Resolved
GRB-04	Liquidation Avoidance through Rejecting Native Ether Transfers	Critical	Resolved
GRB-05	Incomplete Implementation of Native Ether Transfers in `LidoV1_WETHGateway`	High	Resolved
GRB-06	`extraReward2` is Not Set in the `constructor()`	High	Resolved
GRB-07	Liquidation Avoidance by Front-Running `liquidateCreditAccount()` with Ownership Transfer	High	Resolved
GRB-08	`approve()` Will Revert on Any Credit Account	High	Resolved
GRB-09	Tokens Passed in Incorrect Order when Calling `fastCollateralCheck()`	Medium	Resolved
GRB-10	Incorrect Construction of `stakedPhantomToken`	Medium	Resolved
GRB-11	Premature Self-Destruct of `PoolFactory`	Medium	Resolved
GRB-12	Incorrect Token Amounts Used in `add_liquidity()`	Medium	Resolved
GRB-13	Incorrect Account Address Results in Missed Rewards	Medium	Resolved
GRB-14	Flashloan Protection Bypass Through Debt Increase and Account Closure Within The Same Block	Medium	Resolved
GRB-15	Chainlink Oracle Data Feeds Lack Validation	Medium	Resolved
GRB-16	Curve Adapters May Fail to Enable Tokens if a Zeroised `min_amounts` Array is Provided	Low	Resolved
GRB-17	Credit Accounts Can Pass a Full Collateral Check and Also be Liquidatable	Low	Resolved
GRB-18	Calls to `calcBorrowRate()` Can Fail if Variable is Set Incorrectly	Low	Resolved
GRB-19	Credit Accounts May be Returned with Residual Assets	Informational	Resolved
GRB-20	Incorrect `creditConfigurator` Upgrades Will Lock Configuration	Informational	Resolved
GRB-21	Miscellaneous `Gearbox` General Comments	Informational	Resolved
GRB-22	Retesting: `claimRewards()` Reverts Unexpectedly	Medium	Resolved

GRB-01	Incorrect Parsing of Token Path Between Gearbox and Uniswap		
Asset	`contracts/adapters/uniswap/UniswapV3.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

Note, this issue was also disclosed for the Gearbox V1 deployment via Immunifi's bug bounty program during the course of this review. It is further detailed in this post-mortem.

It is possible to manipulate the `path` parameter passed in to `exactOutput()` and other functions of `UniswapV3.sol` adapter to bypass collateral health check on token swaps, leading to theft of funds from the pool.

Uniswap decodes the parameters for each swap pool in a multi-pool `path` bytes array in the following order: `tokenA`, `fee`, `tokenB`. Each token address is encoded by 20 bytes and the pool swap fee encoded by 3 bytes. Any remaining bytes in the array are ignored, unless large enough (23 bytes for fee + address) to constitute another swap parameter. Note that `tokenA` or `tokenB` could be the output token depending on which Uniswap function is called.

Gearbox uses `_extractTokens()` to parse only the first and last tokens from the path to run a collateral check and ensure the safety of the swap. The first *20* bytes represent `tokenA` and the last *20* bytes are read as `tokenB`.

The difference in parsing methods allows a borrower to craft a malicious `path` as such: `tokenA`, `fee`, `tokenB`, `tokenC`.

Uniswap will execute a trade between `tokenA` and `tokenB` from this `path`, and ignore `tokenC`. However, Gearbox parses `tokenC` in the place of `tokenB` for the collateral check.

By deploying a fake ERC20 token and creating a pool for it on Uniswap, an attacker can force a credit account to trade at an unfair price while still passing all of Gearbox's health and safety checks. The attacker is able to withdraw the stolen tokens by minting fake ERC20 tokens and trading these against their own liquidity pool.

Recommendations

Validate that Gearbox decodes `params.path` the same way as seen in Uniswap's `SwapRouter` implementation. Careful consideration needs to be made to ensure that the correct byte offset is applied to `tokenIn` when performing multi-hop token swaps.

Resolution

The development team mitigated this issue in commit fa877cc.

GRB-02	Liquidators Can Manipulate Pool's Profit or Loss Values		
Asset	`contracts/credit/CreditFacade.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

Liquidators can manipulate the protocol's profit and loss accounting in `CreditFacade.liquidateCreditAccount()`, which can lead to theft of funds from the pool.

The call to `_multicall()` is done after `totalValue` is calculated. Hence, if a liquidator borrows additional funds before closing the target credit account, `totalValue` will be under-represented in comparison to `borrowedAmountWithInterest`.

This causes `CreditManager._calcClosePaymentsPure()` to miscalculate the `loss` amount as `amountToPool >= totalFunds`, which means `amountToPool = totalFunds`. The `totalFunds` will be less than `borrowedAMountWithInterest`, therefore the `loss` amount will be calculated as `borrowedAmountWithInterest - amountToPool`.

This vulnerability allows an attacker to continually burn diesel tokens held by the treasury at no cost to the attacker. The attacker can recover the tokens that they added to the borrower account.

Recommendations

Alter the logic of `CreditManager._calcCLosePaymentsPure()` to reliably compute correct payment amounts in the case of liquidations.

Resolution

The development team mitigated this issue in commit bdbabc by only allowing external calls in `_multicall` when called from `liquidateCreditAccount()`.

Note that the checks in the above commit were modified significantly in the final retesting target to the same effect.

GRB-03	Denial-of-Service (DoS) by Opening a Credit Account on Behalf of the `CreditFacade` Contract		
Asset	`contracts/credit/CreditFacade.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

Malicious actors can create a Denial-of-Service (DoS) condition on the Gearbox protocol by opening new `creditAccount` on behalf of `creditFacade`.

The `multicall()` functionality of a `creditAccount` temporarily transfers account ownership to `creditFacade` to call various functions on behalf of the owner. There is an assertion in `CreditManger.transferAccountOwnership()` which requires that the address receiving ownership does not have an open credit account.

This assertion creates a vulnerability to a denial-of-service attack in which the attacker opens a `creditAccount` on behalf of the `creditFacade`. The `creditFacade` would then be considered a registered owner of a `creditAccount` and any attempts to recieve temporary ownership of accounts via `multicall()` will fail.

Consequently, any other calls utilizing `multicall()` will fail (calls to adapters, closures and liquidations, etc).

Recommendations

Implement additional checks within `CreditFacade.openCreditAccount()` to ensure that the new credit account is not opened on behalf of a `creditFacade`.

Resolution

The development team mitigated this issue in commit 5d4ec27 by requiring the receiver of the newly opened account to first call `approveAccountTransfer()`.

This does effectively prevent the attack, but care should be taken that `approveAccountTransfer()` cannot be called from `creditFacade` (via `multicall()`). The current implementation blocks this because the approved selector methods for internal multicalls are hardcoded, and will revert with `UnknownMethodException()`.

GRB-04	Liquidation Avoidance through Rejecting Native Ether Transfers		
Asset	`contracts/credit/CreditFacade.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

Borrowers can avoid liquidations by creating `CreditAccount` from a contract address that reverts every time it receives native ETH transfers.

Gearbox is designed as a composable DeFi protocol. Therefore, it is possible to open a `CreditAccount` from a contract address.

This may create issues when `CreditManager.closeCreditAccount()` attempts to convert `remainingFunds` into native ETH and transfer them to the borrower, when the underlying token is WETH. Malicious borrowers could cause `WETHGateway.unwrapWETH()` to revert by having the `receive()` function of a borrowing contract actively revert.

This vulnerability could be exploited to avoid liquidations indefinitely.

Recommendations

Remove the option to unwrap `WETH` when transferring any remaining funds to the borrower. This can be implemented under the following adjustment:

```
// transfer remaining funds to borrower [Liquidation case only]
if (remainingFunds > 1) {
  _safeTokenTransfer(
    creditAccount,
    underlyingToken,
    borrower,
    remainingFunds,
    false // convertWETH == false
  );
}
```

Resolution

The development team mitigated the issue in commit fb616cc5 implementing the recommendations above.

GRB-05	Incomplete Implementation of Native Ether Transfers in `LidoV1_WETHGateway`		
Asset	`contracts/adapters/lido/LidoV1_WETHGateway.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: Medium	Likelihood: High

Description

`LidoV1_WETHGateway.sol` is missing `payable receive()` function and does not attach native `ETH` amount to `submit()` function calls of `stETH` token. As such, any call to this adapter attempting to unwrap `WETH` tokens will revert.

Gearbox credit accounts cannot hold native Ether for security reasons. As such, the `LidoV1_WETHGateway.sol` contract helps to facilitate native Ether deposits into the Lido platform by unwrapping `WETH` tokens before calling the `submit()` function.

Due to missing functionality of handling native `ETH` transfers, this adapter will always revert.

Recommendations

Update `submit()` such that the native Ether amount is passed alongside the function call.

Add `payable receive()` function to accept incoming `ETH` transfers.

Resolution

The development team mitigated this issue in commit 8614e53 by implementing the recommendations above.

GRB-06	`extraReward2` is Not Set in the `constructor()`		
Asset	`contracts/adapters/convex/ConvexV1_BaseRewardPool.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: Medium	Likelihood: High

Description

`_extraReward2` is never set in the constructor of `ConvexV1_BaseRewardPool.sol`.

Gearbox integrates the Convex platform by allowing borrowers to stake their Convex LP tokens. Gearbox makes use of a phantom ERC20 token to represent the borrower's position in the reward pool. Extra rewards can be attached to a reward pool on top of any Curve and Convex rewards.

Within the `constructor()` code, `_extraReward2` is never set and will end up storing the zero address. As a result, `extraReward2` will also contain the zero address and `extraReward1` will be overwritten with the address intended for `extraReward2`. The affected code is shown below:

```
if (extraRewardLength >= 1) {
  _extraReward1 = IRewards(
    IBaseRewardPool(_baseRewardPool).extraRewards(0)
  ).rewardToken();

  if (extraRewardLength >= 2) {
    _extraReward1 = IRewards(
      IBaseRewardPool(_baseRewardPool).extraRewards(1)
    ).rewardToken(); // assign to _extraReward2, not _extraReward1
  }
}
```

Recommendations

Modify the `constructor()` code to store the extra rewards in their correct, corresponding variables `_extraReward1` and `_extraReward2`.

Resolution

The development team mitigated this issue in commit 5ae20eb by implementing the recommendations above.

GRB-07	Liquidation Avoidance by Front-Running `liquidateCreditAccount()` with Ownership Transfer		
Asset	`contracts/credit/CreditFacade.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

Due to liquidations being performed on the borrower's address and not the `CreditAccount`, users can feasibly avoid liquidation by front-running calls to `liquidateCreditAccount()` and transferring `CreditAccount` ownership to a different address.

Considering that Gearbox intends to deploy their contracts on low-cost blockchains, it is entirely possible for users to sustain such an attack until the borrower's collateral does not sufficiently cover the loss. As a result, liquidity providers will have to cover the shortage in collateral when a liquidator closes a uncollateralised credit account.

Recommendations

Ensure that liquidations operate on the address of the credit account and not the borrower. Changes in credit account ownership should not affect the overall functionality of the Gearbox protocol.

Resolution

The development team mitigated this issue in commit 1e10d15 by reverting calls to `transferAccountOwnership()` when the account has a health factor below zero.

Note that the checks have been modified in the final retesting target, albeit to the same effect.

GRB-08	`approve()` Will Revert on Any Credit Account		
Asset	`contracts/credit/CreditFacade.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: Medium	Likelihood: High

Description

Various Gearbox functions pass the `creditAccount` variable as the `borrower` parameter of `CreditManager.approveCreditAccount()` causing calls to `CreditFacade.getCreditAccountOrRevert(borrower)` to always revert because the `creditAccount` does not exist.

Recommendations

Pass `msg.sender` as the `borrower` parameter in the `CreditManager.approveCreditAccount()` function calls.

Resolution

The development team mitigated this issue in multiple commits:

- 5e35d31 addressed calls from adapters
- 7cfef5f addressed calls from `CreditFacade.approve()`

GRB-09	Tokens Passed in Incorrect Order when Calling `fastCollateralCheck()`		
Asset	`contracts/adapters/convex/ConvexV1_BaseRewardPool.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

When calling the `fastCollateralCheck()` function to perform account's health check, tokens are passed in incorrect order not matching their corresponding balances, resulting in unexpected results when calculating `CreditAccount`'s health.

The issue persists in `_withdraw()` and `_withdrawAndUnwrap()` functions of `ConvexV1_BaseRewardPool.sol`.

Additionally, an incorrect token is referenced in `fastCollateralCheck()` of the `_withdrawAndUnwrap()` function.

Affected code in `_withdraw()` function:

```
uint256 balanceInBefore = IERC20(stakedPhantomToken).balanceOf(creditAccount);
uint256 balanceOutBefore = IERC20(cvxLPtoken).balanceOf(creditAccount);

// ...

if (msg.sender != creditFacade) {
  creditManager.fastCollateralCheck(
    creditAccount,
    cvxLPtoken,      // note, 'balanceInBefore' above is calculated on 'stakedPhantomToken', not 'cvxLPtoken'
    stakedPhantomToken,
    balanceInBefore,
    balanceOutBefore
  );
}
```

Affected code in `_withdrawAndUnwrap()` function:

```
uint256 balanceInBefore = IERC20(stakedPhantomToken).balanceOf(creditAccount);
uint256 balanceOutBefore = IERC20(curveLPtoken).balanceOf(creditAccount);  // note, 'curveLPtoken' referenced

// ...

if (msg.sender != creditFacade) {
  creditManager.fastCollateralCheck(
    creditAccount,
    cvxLPtoken,     // note, 'cvxLPtoken' referenced, when balances are calcualted on 'curveLPtoken'. Also incorrect ordering.
    stakedPhantomToken,
    balanceInBefore,
    balanceOutBefore
  );
}
```

Recommendations

Switch order of tokens passed in as parameters to `fastCollateralCheck()` function and ensure their order matches the order of their corresponding balances which are passed in as parameters to the same function.

Correct token references in `fastCollateralCheck()` to ensure they match their corresponding balances.

Resolution

The development team mitigated this issue originally in commit fa877cc.

Note that the implementation has been modified significantly from this referenced commit in the final retesting target due to the introduction of the `AbstractAdapter.sol` contract organizing this logic.

GRB-10	Incorrect Construction of `stakedPhantomToken`		
Asset	`contracts/adapters/convex/ConvexV1_BaseRewardPool.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

In the constructor code of `ConvexV1BaseRewardPoolAdapter`, the `stakedPhantomToken` is created with the pool's staking token passed in as first parameter, instead of the pool address itself.

As such, phantom token will not return correct balances, which will result in unexpected behaviour in other functions.

Affected code:

```
stakedPhantomToken = address(
  new ConvexStakedPositionToken(
    address(IBaseRewardPool(_baseRewardPool).stakingToken()),  // note, 'stakingToken()' address referenced, instead of pool address
    cvxLPtoken
  )
);
```

Note constructor declaration of `ConvexStakedPositionToken` - `constructor(address _pool, address _lptoken)`.

Recommendations

Modify call constructing `stakedPhantomToken` from `address(IBaseRewardPool(_baseRewardPool).stakingToken())` to `address(IBaseRewardPool(_baseRewardPool)`.

Resolution

The development team originally mitigated this issue in commit 5ae20eb by implementing the recommendations above.

Note that the deployment of `ConvexStakedPositionToken` was modified significantly in the final retesting target. It is no longer deployed within `ConvexV1BaseRewardPoolAdapter`, rather is deployed separately and passed in as a constructor argument. This allows multiple `CreditManager` deployments to share one `ConvexStakedPositionToken` deployment.

GRB-11	Premature Self-Destruct of `PoolFactory`		
Asset	`contracts/factories/PoolFactory.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

`PoolFactory` may be self-destructed while being sole owner of `ACL` contract.

During pool deployment, the ownership of `ACL` contract is transfered to `PoolFactory` to preform its configuration.

After `PoolFactory.configure()` is called, the ownership is transfered back to the configurator. Additional `PoolFactory.getRootBack()` function is also provided to recover ownership in the case of configuration failure.

The function `PoolFactory.destroy()` self destructs the PoolFactory contract and there are no check to prevent this function from being called while `PoolFactory` is the owner of `ACL`. If such event occurs, nobody will have ownership of `ACL`, which will break many Gearbox functions.

Recommendations

Implement additional checks to ensure that `ACL` ownership has been returned to the configurator before selfdestructing `PoolFactory`.

Resolution

The development team mitigated this issue in commit 59d9e0c.

The `destroy()` function has been implemented within `ContractUpgrader.sol` and inherited by `PoolFactory.sol` with an additional check to revert the call if `PoolFactory` currently owns `ACL`:

```
// from ContractUpgrader.sol
function destoy() external onlyOwner {
  if (ACL(addressProvider.getACL()).owner() == address(this))
    revert RootSelfDestoyException();
  selfdestruct(payable(msg.sender));
}
```

GRB-12	Incorrect Token Amounts Used in `add_liquidity()`		
Asset	`contracts/adapters/curve/CurveV1_stETHGateway.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

Incorrect amount of `token1` is used to transfer funds to the `CurveV1StETHPoolGateway`, resulting in unexpected behaviour.

`add_liquidity()` function on `L108` incorrectly references amount of `token1` to transfer to the gateway for further processing. This will result in incorrect amount transferred out of the borrower's address and may also cause reverts due to insufficient balance or exceeding approved transfer amount.

Affected code:

```
function add_liquidity(
  uint256[N_COINS] memory amounts,
  uint256 min_mint_amount
) external {
  if (amounts[0] > 0) {
    IERC20(token0).safeTransferFrom(
      msg.sender,
      address(this),
      amounts[0]
    );
    IWETH(token0).withdraw(amounts[0]);
  }

  if (amounts[1] > 0) {
    IERC20(token1).safeTransferFrom(
      msg.sender,
      address(this),
      amounts[0]    // note, 'amounts[0]' is referenced, instead of 'amounts[1]'
    );
  }
```

Recommendations

Change `token1` amount on `L108` of `add_liquidity()` function from `amounts[0]` to `amounts[1]`.

Resolution

The development team mitigated the issue in commit 5d506f0 implementing the recommendations above.

GRB-13	Incorrect Account Address Results in Missed Rewards		
Asset	`contracts/adapters/convex/ConvexV1_BaseRewardPool.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

Due to missing conversion of address passed in to `getReward(address _account, bool)` function from borrower's address to corresponding `CreditAccount`, users will miss accumulated rewards.

Address passed in to the function `getReward(address _account, bool)` is simply passed straight through to `creditManager.executeOrder()` without first being converted to corresponding Credit Account's address. As a result, further call to Convex's `BaseRewardPool` will fail, as account of this address does not exist.

Additionally, the `creditManager.executeOrder()` function is called passing in `msg.sender` as a borrower, instead of `_account`, resulting in further calls being executed from `msg.sender`'s credit account.

```
function getReward(address _account, bool)
  external
  override
  returns (bool)
{
  address creditAccount = creditManager.getCreditAccountOrRevert(
    _account
  );

  creditManager.executeOrder(
    msg.sender,  // this will result in request executed from 'msg.sender' Credit Account, not the provided '_account'
    address(baseRewardPool),
    msg.data    // this contains '_account' address, not 'creditAccount' address, as such the Convex getReward() call
  );         // call will fail as no such account exist
```

Recommendations

Rewrite `calldata` passed in to the `getReward(address _account, bool)` function to use corresponding Credit Account's address before using it further in `creditManager.executeOrder()`.

Modify first parameter passed in `creditManager.executeOrder()` to match intended borrower.

Resolution

The development team mitigated this issue in commit bcd3199.

The adapter `getReward(address,bool)` function now bypasses `creditManager.executeOrder()` and instead calls `IBaseRewardPool.getReward(address,bool)` directly, passing in the correct `creditAccount` address.

The gearbox team has acknowledged a potential griefing attack in a blog post caused by this design but has determined it a minor issue.

GRB-14	Flashloan Protection Bypass Through Debt Increase and Account Closure Within The Same Block		
Asset	`contracts/credit/CreditFacade.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

It is possible to bypass the flashloan attack protection.

Whilst the existing flashloan attack protection mechanism prevents users from opening and closing credit accounts within the same transaction (block), this check does not prevent users from increasing their debt position beyond the limits of their collaterization ratio (up to `maxBorrowedAmount`) and closing the account within the same transaction. This acts as a fee-less flashloan for borrowers.

Specifically, there are two ways to exploit this vulnerability:

- by calling functions of adding collateral, increasing debt and closing credit account sequentially, in the same block (e.g. when called as a single function from an external contract)
- by exploiting lack of collateral checks during the `_multicall()` operations in `CreditManger.closeCreditAccount()`, as the pool is assumed to be repaid, and non-underlying assets are assumed to be transferred, by the end of the transaction.

Recommendations

Implement checks to ensure `increaseDebt()` and `closeCreditAccount()` are not executed in the same block.

Alter the logic of `CreditFacade._multicall()` to limit its capabilities during account closures.

Resolution

The development team partially mitigated this issue in commit bdbabc.

Internal calls within `_multiCall()` are reverted when called via `closeCreditAccount()`, preventing the second flashloan scenario.

The possibility for the first of the two scenarios detailed above still exists (bypassing `multicall()`). This has been acknowledged by Gearbox in a blog post and determined an acceptable risk.

GRB-15	Chainlink Oracle Data Feeds Lack Validation		
Asset	`contracts/oracles/PriceOracle.sol` and `contracts/oracles/yearn/YearnPriceFeed.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

Chainink oracle data feeds lack validation to ensure that the data is fresh and from a complete round. Oracle malfunctions may lead to large scale liquidation events or alternatively it may put the protocol at risk of insolvency.

Recommendations

Validate the output of Chainlink's `latestRoundData()` function to match the following code snippet:

```
(
uint80 roundID,
int256 price,
,
uint256 updateTime,
uint80 answeredInRound
) = AggregatorV3Interface(priceFeeds[token]).latestRoundData();
require(
  answeredInRound >= roundID,
  "Chainlink Price Stale"
);
require(price > 0, "Chainlink Malfunction");
require(updateTime != 0, "Incomplete round");
```

It is important that these checks are added to all cases where `latestRoundData()` is used.

Resolution

The development team mitigated the issue in commit 59d9e0c implementing the recommendations above.

Several checks are implemented within `_checkAnswer()`, which is called on the outputs of `AggregatorV3Interface.latestRoundData()` wherever it is called.

GRB-16	Curve Adapters May Fail to Enable Tokens if a Zeroised `min_amounts` Array is Provided		
Asset	`contracts/adapters/curve/*`		
Status	**Resolved:** See Recommendations		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

Gearbox's Curve adapters may fail to enable tokens, resulting in inaccurate `CreditAccount` health check calculations.

Gearbox's Curve adapters will check and enable tokens if the credit account receives an amount of tokens greater than zero when removing liquidity. The check is performed in the `_remove_liquidity_int()` function, however, because the check is done on a potentially zeroised `min_amounts` array, it is possible that the credit account will fail to consider tokens removed from the liquidity pool.

As a result, this may lead to unexpected liquidations as the health of the user's credit account deteriorates due to this edge case.

Recommendations

Enforce that the `min_amounts` array contains no zero values. Alternatively, consider checking the before and after balance of each token is positive before calling `checkAndEnableToken()`.

Recommendations

The development team originally mitigated this issue in commit bc0ef58 by enabling all tokens when `remove_liquidity()` is called.

Note that the implementation has been significantly modified from this referenced commit in the final retesting target. The contract `CurveV1AdapterBase.sol` is inherited by each 2, 3, or 4 asset adapters to organize the shared `_remove_liquidity()` logic, to the same effect as above.

GRB-17	Credit Accounts Can Pass a Full Collateral Check and Also be Liquidatable		
Asset	`contracts/credit/CreditManager.sol and contracts/credit/CreditFacade.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

A `Credit Account` may be qualified as applicable for liquidation and pass collateral health check at the same time.

When a `CreditAccount` performs an operation on its assets, it will then perform a collateral check to ensure that the operation has not caused the health factor of the `Credit Account` to drop below 1.

However, the method of calculating the `CreditAccount` health factor differs between `CreditManager.fullCollateralCheck()` and `CreditFacade.liquidateCreditAccount()`. This can lead to a state where an account passes the `fullCollateralCheck()`, but can still be liquidated.

This is demonstrated in the unit test `test_exploit_liquidatable_and_fullCollateralPass()` in the test file for `CreditManager.sol`. The difference is that one function caclulates the health factor in terms of the underlying asset and the other calculates it in terms of dollars leads to a truncation issue with price feeds with accuracy below 18d.p. This truncation then leads to different quantities being evaluated on line [**639**] of `CreditManager.sol` and line [**288**] of `CreditFacade.sol`.

Recommendations

Change the calculation used for an account health factor to be consistent throughout different contracts.

When possible make use of unit testing and fuzzing to detect when edge-case scenarios give different outcomes.

Resolution

The development team mitigated this issue in commit 4947104 by modifying `liquidateCreditAccount()` and fullCollateralCheck() to calculate total value denominated in USD with similar logic.

Therefore, there should not be any cases where `fullCollateralCheck()` passes while also being liquidatable, and vice versa. This property appears to hold at surface level, but should be confirmed by more rigorous testing methods.

Note that the implementation of the above referenced commit has been modified significantly in the final retesting target. `CreditFacade.sol` now has the function `_isAccountLiquidatable()` and checks the return value of this function, to the same effect as above.

GRB-18	Calls to `calcBorrowRate()` Can Fail if Variable is Set Incorrectly		
Asset	`contracts/pool/LinearInterestRateModel.sol`		
Status	**Resolved:** See Recommendations		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

Setting `U_optimal = PERCENTAGE_FACTOR` can cause unexpected reverts due to division by zero.

On line [**46**] of `LinearInterestRateModel.sol` the check `U_optimal <= PERCENTAGE_FACTOR` is made, however setting `U_optimal = PERCENTAGE_FACTOR` can later cause problems as this results in `_U_Optimal_inverted_WAD = 0` on line [**55**].

Subsequently, division on line [**102**] will then always revert due to division by zero, meaning calls to `calcBorrowRate()` will always revert.

Recommendations

Change line [**46**] to be a strict inequality `U_optimal < PERCENTAGE_FACTOR` in order to prevent this issue occuring.

Recommendations

The development team mitigated this issue in commit 4947104 implementing the recommendations above.

GRB-19	Credit Accounts May be Returned with Residual Assets
Asset	`contracts/credit/CreditFacade.sol` and `contracts/core/AccountFactory.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

If a `creditAccount` is closed (or liquidated) with `sendAllAssets` parameter as false, the residual (non-underlying) assets will remain in the `creditAccount` if they are not transferred or converted to underlying via `_multicall()`.

These assets will be inherited by the next borrower assigned to this `creditAccount`.

Recommendations

Provide documentation so that users are aware of these implications and can take steps towards avoiding them.

Resolution

This issue has been acknowledged by Gearbox as intended behavior that will be documented.

GRB-20	Incorrect `creditConfigurator` Upgrades Will Lock Configuration
Asset	`contracts/credit/CreditConfigurator.sol` and `contracts/tokens/GearToken.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

If `CreditConfigurator.upgradeConfigurator()` is called by the configurator (owner of `ACL` contract) with an incorrect contract address, then `creditConfigurator` will be locked at this address and several parameters of `CreditManager` and `CreditFacade` will be locked forever.

Recommendations

Implement a two-step upgrade process, whereby the configurator sets the new `creditConfigurator` address and then the new `creditConfigurator` must claim the role.

Note: This vulnerability is also present in GearToken.sol

Resolution

The development team originally mitigated this issue in commit 4947104 by implementing several sanity checks on inputs to `upgradeConfigurator()`.

Note that the implementation has been significantly modified from the above reference commit in the final retesting target. The upgrade function now calls `_revertIfContractIncompatible(address)`, which organizes the sanity checks on the target of the upgrade. The referenced function was renamed to `CreditConfigurator.upradeCreditConfigurator()`, which provides some clarity to the exact effects of the function.

GRB-21	Miscellaneous `Gearbox` General Comments
Asset	`contracts/*`
Status	**Resolved:** See Resolution
Rating	Informational

Description

This section details miscellaneous findings in the `contracts` repository discovered by the testing team that do not have direct security implications:

1. **`core/DataCompressor.sol`**

   1a) Loop Limit Called Each Cycle
   To save gas these loop limits on line [**79**] and line [**94**] should be stored in a local variable rather than queried each cycle.

   1b) Loop Increment Called in Checked Maths
   To save gas this loop incrementing on line [**80**] and line [**95**] can be placed in an unchecked block at the end of the loop as it is already restricted by the loop termination condition so will not overflow.

   1c) Stale Code
   There is stale code in the following lines, this should be removed to avoid confusion: line [**139**], line [**153**] and line [**172**].

   1d) Typos
   line [**68**]: `particluar`, should be `particular`.
   line [**68**]: `account`, should be `accounts`.
   line [**188**]: `bborrower`, should be `borrower`.
   line [**217**]: `flg`, should be `flag`.
   line [**270**]: `particulr`, should be `particular`.
   line [**325**]: `non-standart`, should be `non-standard`.

   1e) Missing Inheritance
   Consider having `DataCompressor.sol` inherit from its interface `IDataCompressor.sol` to ensure that it is implemented correctly.

   1f) Misleading Modifier Naming
   Modifiers `registeredCreditManagerOnly(address creditManager)` and `registeredPoolOnly(address pool)` are intented to check that the target of the call is a credit manager or a pool. They do not check the caller of the function. Consider renaming to `targetIsCreditManager(address target)` to clarify intent.

   1g) Unused Local Variables
   The return value `ICreditFacade creditFacade` assigned to a local variable on line [**125**], line [**227**], line [**360**], and line [**404**] is not used and can be removed. The call assigning `uint256 allowedTokenCount` on line [**151**] can be removed.

2. **`core/ACL.sol`**

   2a) Missing Inheritance
   Consider having `ACL.sol` inherit from its interface `IACL.sol` to ensure that it is implemented correctly.

3. **credit/CreditManager.sol**

3a) Imports Hardhat Console
On line [**25**] there is an import that only functions when not on the mainnet and so should be removed.

3b) Confusing Comment
On line [**38**] the `Account Factory` is referred to as the account manager which may get confusing as the `Credit Manager` is the manager for credit accounts. The comment should be edited to make the differing roles clearer.

3c) Typo in Variable Name
On line [**84**] the variable `forbidenTokenMask` is introduced, it should be named `forbiddenTokenMask` here and at other places of use.

3d) Confusing Comment
On line [**89**] the comment suggests we record which block we last used the `fast check` mechanism when in fact the mapping records how often we have used the `fast check` mechanism in the same block.

3e) Inconsistent Naming Scheme
On line [**617**] refers to a variable `total` where in other contracts this calculation is refered to as `twv` short for `Threshold Weighted Value` while `total` is reserved for calculations of `Total value` as described in the whitepaper.

3f) Gas Optimization
On line [**727**] and line [**783**] `tokenMask > 0` can be replaced with `tokenMask != 0` which is 3 gas cheaper and acts the same when acting on unsigned integers.
The function `upgradeContracts` can be separated to upgrade `creditFacade` or `priceOracle` individually.

3g) Operation Order Unclear
On line [**727**] and line [**783**] multiple operations are carried out on the same line, use brackets to indicate the intended order of actions to improve readability.

3h) Missing Zero Address Checks
The function `upgradeContracts` prevent `creditFacade` or `priceOracle` from being set to the zero address. If this happens, important Gearbox functions will break. The mistake can be recovered by calling `upgradeContracts` again (wasting gas), but implementing zero address checks would be appropriate.

3i) Typos
line [**65**]: `Miltiplier`, should be `Multiplier`.
line [**71**]: `Adress`, should be `Address`.
line [**167**]: `transffered`, should be `transferred`.
line [**227**]: `liqution`, should be `liquidation`.
line [**279**]: `addrss`, should be `address`.
line [**280**]: `it it`, should be `if it`.
line [**345**]: `cunulativeIndex`, should be `cumulativeIndex`.
line [**347**]: `decresase`, should be `decrease`.
line [**348**]: `particall`, should be `partial`.
line [**349**]: `cumulativeIndex now`, should be `cumulativeIndexNow`.
line [**353**]: `fto`, should be `to`.
line [**805**]: `reverts of borrower has no one`, should be `reverts if borrower does not have one`.
line [**823**]: `Foloowing`, should be `Following`

3j) Public Function Could Be External
The public function `fullCollateralCheck()` can be declared as external.

4. **core/WETHGateway.sol**

4a) Typos
line [**117**]: `has`, should be `have`.

4b) Misleading Function Name
The function `_checkAllowance()` has side effects. It both checks the allowance and approves the maximum amount. Consider renaming the function to `_checkAndApproveAllowance()` to clarify the intent of the function.

4c) Misleading Modifier Name
The modifier `creditManagerOnly(address creditManager)` is intended to check that the caller is a `creditManager`. However, it expects that `msg.sender` be passed in as a parameter. Consider removing the parameter and instead check the caller directly to clarify usage.

4d) Unused Modifier
The modifer `wethCreditManagerOnly(address creditManager)` is not used and can be removed.

5. **`credit/CreditAccount.sol`**

5a) Inconsistent Use of `approve()` and `safeApprove()`
On line [**95**] and line [**109**] we have an inconsistent use of `ERC20`'s function `approve()` and the Open Zeppelin wrapper `safeApprove()`

5b) Typos
line [**54**]: `cause`, should be `because`.
line [**59**]: `Connects credit account to credit account address.`, should be `Connects credit account to credit manager address.`.

5c) Misleading Access Control
The functions `connectTo()` and `cancelAllowance()` do not have any modifiers and may be assumed to be callable by anyone. However, a require statement in the function restricts access to calls from `factory`. Consider implementing a `factoryOnly` modifer to clarify the access control.

6. **`credit/CreditConfigurator.sol`**

6a) Typo in Variable Name
On line [**216**] the variable `forbidenTokenMask` is introduced, it should be named `forbiddenTokenMask` here and at other places of use.

6b) Typos
line [**70**]: `condigures`, should be `configures`.
line [**134**]: `revers`, should be `reverts`.
line [**159**], line [**219**], line [**246**]: `cause`, should be `because`.
line [**244**]: `undelrying`, should be `underlying`.
line [**445**] `undelrying` should be `underlying`.
line [**491**]: `This internal function is check the need of additional sanity checks`, should be `This internal function conducts addtional sanity checks`

6c) Conflict Between Documentation and Implementation
Comments on line [**63**] through line [**73**] describe the correct deployment flow for `CreditManager`, `CreditFacade`, and `CreditConfigurator` contracts. However, these steps occur out of order in the factory contracts. Step 3 does not occur until Step 6, in the constructor of `CreditConfigurator`.

7. **`pool/LinearInterestRateModel.sol`**

7a) Inconsistent Notation
On line [**20**], line [**36**] until line [**39**] and line [**107**] a mix of commas and periods are used to denote a thousands seperator. Both should be avoided as depending on which country the reader is from they can be interpreted as decimal places. I.e. 10.000 may be mistaken for 10.000% If writing clarity is needed prefer 10_000 or 10 000.

7b) Mismatched Naming Scheme
On line [**79**] the variable name `U\_WAD` naming style may lead to readers assuming it is a constant.

7c) Constant Interest Model Parameters
It is not possible to update the interest model parameters and so if an asset is being over/underutilized this cannot be fixed by governance.

7d) Suitable Future Interest Models
The current interest model is a piecewise continuous function, it is important that if a different interest model is adopted it must also be a continuous function. Otherwise it may be possible to disproportionately affect the paid interest rate by donating a small amount of the underlying token to the pool.

7e) Typos
line [**50**]: `percetns`, should be `percents`.

8. **`credit/CreditFacade.sol`**

8a) Identical Function Names
Both `CreditFacade.sol` and `CreditManager.sol` contain a function called `transferAccountOwnership()` one of these should be changed to avoid confusion.

8b) Type In Variable Name
The `tvw` return variable assigned on line [**283**] should be called `twv` to match usage in other places.

8c) Typos
line [**403**]: `Requires`, should be `Required`.
line [**428**]: `cause account trasfership were trasffered here`, should be `because account ownership was transferred here`.
line [**436**]: `cause address(this) has no sense`, should be `because address(this) makes no sense`.
line [**477**]: `powerfull`, should be `powerful`.
line [**532**]: `who unexpect this`, should be `who does not expect this`.
line [**548**]: `borbidden`, should be `forbidden`.

8d) Public Functions Could Be External
The public functions `calcCreditAccountHealthFactor()` and `hasOpenedCreditAccount()` could be declared external.

8e) Unused Local Variables
The call assigning `address creditAccount` on line [**522**] can be removed.

9. **`core/AccountFactory.sol`**

9a) Typos
line [**157**]: `if no one in stock`, should be `if there are none in stock`.
line [**323**]: `accoutn` should be `account`
line [**345**]: `desinged` should be `designed`

10. **`oracles/PriceOracle.sol`**

10a) Inaccurate Comment
line [**56**] and line [**147**] suggest that the Chainlink price feed should be a `TOKENETH` pair however discussions with the Gearbox team suggested `TOKENUSD` price feeds were prefered due to better update characteristics. This is important to document clearly as ETH price feeds use 18d.p. while USD price feeds use only 8d.p.

10b) Gas Optimizations
On line [**152**] we load `priceFeeds[token]` from storage then again on line [**163**], the first instance should be stored in a local variable then accessed via that on line [**163**] to save gas.
On line [**161**] we create a local variable `timeStamp` however this is never used.

10c) Public Function Could Be External
The public function `convert()` could be declared external.

11. **`adapters/yearn/YearnV2.sol`**

11a) Magic Constants
On line [**105**] and line [**187**] the function selectors are hardcoded without an explanation of what they are. These should be turned into constant variables with suitable names and explanations of how they are derived.

11b) Conflict Between Documentation and Implementation
Comments on line [**161**] and line [**162**] suggest that the `uint256 maxLoss` parameter of `withdraw(uint256,address,uint256)` function should specify the maximum acceptable loss that can be sustained on withdrawal. However, the function does not implement such logic.

12. **pool/PoolService.sol**

12a) Public Functions Could Be External
The public functions `updateInterestRateModel()` and `setWithdrawFee()` could be declared external.

12b) Typos
line [**143**]: `amoung` should be `amount`
line [**309**]: `corretly` should be `correctly`
line [**317**]: `profitabe` should be `profitable`
line [**458**]: `undelying` should be `underlying`
line [**468**]: `Credif` should be `Credit`
line [**469**], line [**490**], line [**491**]: `credif` should be `credit`
line [**490**]: `particulat` should be `particular`

13. **adapters/convex/ConvexV1_Booster.sol**

13a) Stale Code
`_amount` input on line [**136**] is never used.

14. **adapters/convex/ConvexV1_BaseRewardPool.sol**

14a) Constant Return Statements
Several functions in this contract return bools however they always return true. This affects functions `_stake()` on line [**144**], `_withdraw()` on line [**196**], `_withdrawAndUnwrap()` on line [**243**], `getReward()` on line [**282**] and `getReward()` on line [**302**]. The return statements should either be removed or have multiple outcomes.

15. **adapters/curve/CurveV1_stETH.sol**

15a) Typo
line [**50**]: `Deisgned`, should be `Designed`.

16. **adapters/curve/CurveV1_stETHGateway.sol**

16a) Lack of `ReentrancyGuard` and no `nonReentrant` modifier on `exchange()`, `add_liquidity()` and `remove_liquidity()` functions

17. **adapters/lido/LidoV1.sol**

17a) Incorrect global variable value
line [**38**]: `_gearboxAdapterType`, is incorrectly set to `AdapterType.CONVEX_V1_CLAIM_ZAP`.

17b) No `nonReentrant` modifier on `submit()` function

18. **adapters/lido/LidoV1_WETHGateway.sol**

18a) Lack of `ReentrancyGuard` and no `nonReentrant` modifier on `submit()` function

19. **oracles/curve/CurveLP2PriceFeed.sol**

    19a) Magic Constant
    On line [**89**] there is a magic constant `10**18` this should be replaced with a named constant to improve readability.

20. **oracles/yearn/YearnPriceFeed.sol**

    20a) Gas Optimization
    On line [**117**] replace `_lowerBound > 0` with `_lowerBound != 0` as this is 3 gas cheaper and acts the same when acting on unsigned integers.

21. **libraries/Errors.sol**

    21a) Typo in Error Message
    line [**48**]: `CM_CAN_LIQUIDATE`, should be `CM_CANNOT_LIQUIDATE`

22. **tokens/GearToken.sol**

    22a) Typo
    line [**184**]: `to spends`, should be `to spend`

    22b) Redundant Check
    line [**430**] of `_transferTokens()` prevents the zero address from sending tokens. However, it is impossible for the zero address to hold tokens. Therefore, this check can be removed.

23. **factories/PoolFactory.sol**

    23a) Typo
    line [**104**]: `destroy`, should be `destroy`

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

The development team has acknowledged the comments detailed above, but is currently prioritizing improvement of contract efficiency and security. The issues will be addressed before production release.

GRB-22	Retesting: `claimRewards()` Reverts Unexpectedly		
Asset	`contracts/adapters/ConvexV1_ClaimZap.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Low	Likelihood: High

Description

The `claimRewards()` function of the Gearbox `ClaimZap` adapter makes calls to `BaseRewardPool.extraRewards(i)` and expects the zero address to be returned if the extra rewards are not set. However, these calls instead revert due to array index out of bounds access, resulting in `claimRewards()` always reverting in such cases. Users will be forced to claim rewards by other means, bypassing the `ClaimZap` adapter.

The problematic function is shown below, which is called from `claimRewards()`:

```
function _claimAndEnableRewards(
  address creditAccount,
  address[] calldata rewardContracts
) internal {
  address token;
  uint256 len = rewardContracts.length;

  for (uint256 i; i < len; ) {
    address rewardContract = rewardContracts[i];
    // ...
    address extraRewardContract1 = IBaseRewardPool(rewardContract)
    .extraRewards(0); // reverts if first rewards not set

    if (extraRewardContract1 != (address(0))) {
      // ...
      address extraRewardContract2 = IBaseRewardPool(rewardContract)
      .extraRewards(1); // reverts if second rewards not set

      if (extraRewardContract2 != (address(0))) {
        // ...
      }
    }
    unchecked {
      ++i;
    }
  }
}
```

Recommendations

The assignment calls for `extraRewardContract1` and `extraRewardContract2` should use try/catch statements to handle the possibility of reverted calls when extra rewards are not set.

However, these calls also seem to be redundant with those in `\_claimAndEnableExtraRewards()` and could be safely removed.

Resolution

The development team resolved this issue originally in commit 178bbd0 by implementing try/catch statements for both calls.

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are given along with this document. The `brownie` framework was used to perform these tests and the output is given below.

```
test_constructor                                PASSED   [0%]
test_stakeAll                                   PASSED   [1%]
test_stake                                      PASSED   [1%]
test_withdraw_claimFalse                        PASSED   [2%]
test_withdraw_claimTrue                         PASSED   [3%]
test_withdrawAll_claimFalse                     PASSED   [3%]
test_withdrawAll_claimTrue                      PASSED   [4%]
test_getReward                                  PASSED   [4%]
test_getReward_Account_ClaimExtrasFalse         PASSED   [5%]
test_withdrawAndUnwrap_claimFalse               PASSED   [6%]
test_withdrawAndUnwrap_claimTrue                PASSED   [6%]
test_withdrawAllAndUnwrap_claimFalse            PASSED   [7%]
test_withdrawAllAndUnwrap_claimTrue             PASSED   [7%]
test_constructor                                PASSED   [8%]
test_updateStakedPhantomTokensMap               PASSED   [9%]
test_deposit_stakeTrue                          PASSED   [9%]
test_deposit_stakeFalse                         PASSED   [10%]
test_depositAll_stakeTrue                       PASSED   [11%]
test_depositAll_stakeFalse                      PASSED   [11%]
test_withdraw                                   PASSED   [12%]
test_withdrawAll                                PASSED   [12%]
test_constructor                                PASSED   [13%]
test_constructor                                PASSED   [14%]
test_forbiddenFunctions                         PASSED   [14%]
test_balanceOf                                  PASSED   [15%]
test_constructor                                PASSED   [15%]
test_add_liquidity                              PASSED   [16%]
test_remove_liquidity                           PASSED   [17%]
test_constructor                                PASSED   [17%]
test_add_liquidity                              PASSED   [18%]
test_remove_liquidity                           PASSED   [19%]
test_constructor                                PASSED   [19%]
test_add_liquidity                              PASSED   [20%]
test_remove_liquidity                           PASSED   [20%]
test_constructor                                PASSED   [21%]
test_exchange                                   PASSED   [22%]
test_add_liquidity                              PASSED   [22%]
test_remove_liquidity                           PASSED   [23%]
test_get_dy                                     PASSED   [23%]
test_get_virtual_price                          PASSED   [24%]
test_constructor                                PASSED   [25%]
test_submit                                     PASSED   [25%]
test_constructor                                PASSED   [26%]
test_swapTokensForExactTokens                   PASSED   [26%]
test_swapExactTokensForTokens                   PASSED   [27%]
test_read                                       PASSED   [28%]
test_constructor                                PASSED   [28%]
test_extractTokens                              PASSED   [29%]
test_exactInputSingle                           PASSED   [30%]
test_exactAllInputSingle                        PASSED   [30%]
test_exactInput                                 PASSED   [31%]
test_exactOutputSingle                          PASSED   [31%]
test_exactOutput                                PASSED   [32%]
test_constructor                                PASSED   [33%]
test_deposit                                    PASSED   [33%]
test_deposit_uint256                            PASSED   [34%]
test_deposit_uint256_address                    PASSED   [34%]
test_withdraw                                   PASSED   [35%]
test_withdraw_uint256                           PASSED   [36%]
test_withdraw_uint256_address                   PASSED   [36%]
test_version                                    PASSED   [37%]
test_add_removePausableAdmin                    PASSED   [38%]
test_add_removeUnpausableAdmin                  PASSED   [38%]
test_isConfigurator                             PASSED   [39%]
```

```
test_renounceOwnership                                    PASSED   [39%]
test_constructor                                          PASSED   [40%]
test_creditAccount                                        PASSED   [41%]
test_countCreditAccountsInStock[10]                       SKIPPED  [41%]
test_countCreditAccountsInStock[100]                      SKIPPED  [42%]
test_countCreditAccountsInStock[1000]                     SKIPPED  [42%]
test_countCreditAccountsInStock[10000]                    SKIPPED  [43%]
test_takeOut                                              PASSED   [44%]
test_getters                                              PASSED   [44%]
test_version                                              PASSED   [45%]
test_addPool                                              PASSED   [46%]
test_addPool_multi                                        PASSED   [46%]
test_addCreditManager                                     PASSED   [47%]
test_addCreditManager_multi                               PASSED   [47%]
test_pause_unpause                                        PASSED   [48%]
test_constructor                                          PASSED   [49%]
test_add_remove_LiquidityETH                              PASSED   [49%]
test_unwrapWETH                                           PASSED   [50%]
test_receive                                              PASSED   [50%]
test_initialize                                           PASSED   [51%]
test_connectTo                                            PASSED   [52%]
test_updateParameters                                     PASSED   [52%]
test_cancelAllowance                                      PASSED   [53%]
test_safeTransfer                                         PASSED   [53%]
test_constructor_check                                    PASSED   [54%]
test_setLiquidationThreshold                              PASSED   [55%]
test_allow_forbid_Token                                   PASSED   [55%]
test_allow_forbid_Contract                                PASSED   [56%]
test_setLimits                                            PASSED   [57%]
test_setFees                                              PASSED   [57%]
test_upgradePriceOracle                                   PASSED   [58%]
test_upgradeCreditFacade                                  PASSED   [58%]
test_upgradeCreditConfigurator                            PASSED   [59%]
test_constructor                                          PASSED   [60%]
test_openCreditAccount_WETH                               PASSED   [60%]
test_close_CreditAccount                                  FAILED   [61%]
test_openCreditAccountMulticall                           PASSED   [61%]
test_openCreditAccountMulticall_increaseDebt              PASSED   [62%]
test_openCreditAccountMulticall_decreaseDebt              PASSED   [63%]
test_liquidateCreditAccount                               PASSED   [63%]
test_approve                                              PASSED   [64%]
test_approve_transfer_AccountOwnership                    PASSED   [65%]
test_liquidateAccount_poolFundsTheft                      PASSED   [65%]
test_liquidateAccount_contractAddress                     PASSED   [66%]
test_flashloanProtectionBypass                            XFAIL    [66%]
test_openCreditFacade_DoS                                 PASSED   [67%]
test_multicall_addCollateral_noAccount                    PASSED   [68%]
test_closeCreditAccount_increaseDebt_noCollateralCheck    PASSED   [68%]
test_constructor                                          PASSED   [69%]
test_openCreditAccount                                    PASSED   [69%]
test_manageDebt_increase                                  PASSED   [70%]
test_manageDebt_decrease                                  PASSED   [71%]
test_transferAccountOwnership                             PASSED   [71%]
test_approveCreditAccount                                 PASSED   [72%]
test_checkAndEnableToken                                  PASSED   [73%]
test_fastCollateralCheck                                  PASSED   [73%]
test_fullCollateralCheck                                  PASSED   [74%]
test_setParams                                            PASSED   [74%]
test_setLiquidationThreshold                              PASSED   [75%]
test_setForbidMask                                        PASSED   [76%]
test_changeContractAllowance                              PASSED   [76%]
test_upgradeContracts                                     PASSED   [77%]
test_setConfigurator                                      PASSED   [77%]
test_executeOrder                                         PASSED   [78%]
test_exploit_liquidatable_and_fullCollateralPass          PASSED   [79%]
test_constructor                                          PASSED   [79%]
test_constructor                                          PASSED   [80%]
test_getRootBack                                          PASSED   [80%]
test_destroy                                              PASSED   [81%]
test_constructor                                          PASSED   [82%]
```

```
test_getRoundData_reverts                          PASSED   [82%]
test_constructor                                   PASSED   [83%]
test_converts                                      PASSED   [84%]
test_constructor                                   PASSED   [84%]
test_getRoundData_reverts                          PASSED   [85%]
test_latestRoundData                               PASSED   [85%]
test_constructor                                   PASSED   [86%]
test_getModelParameters                            PASSED   [87%]
test_constructor                                   PASSED   [87%]
test_add_remove_Liquidity                          PASSED   [88%]
test_lend_repay_CreditAccount                      PASSED   [88%]
test_connect_forbid_CreditManager                  PASSED   [89%]
test_updateInterestRateModel                       PASSED   [90%]
test_setExpectedLiquidityLimit                     PASSED   [90%]
test_setWithdrawFee                                PASSED   [91%]
test_constructor                                   PASSED   [92%]
test_bestUniPath                                   PASSED   [92%]
test_convertPathToPathV3                           PASSED   [93%]
test_constructor                                   PASSED   [93%]
test_transferOwnership                             PASSED   [94%]
test_setMiner                                      PASSED   [95%]
test_allowTransfers                                PASSED   [95%]
test_approve                                       PASSED   [96%]
test_permit                                        PASSED   [96%]
test_transfer                                      PASSED   [97%]
test_transferTokens                                PASSED   [98%]
test_delegate                                      PASSED   [98%]
test_transferFrom                                  PASSED   [99%]
test_delegateBySig                                 PASSED   [100%]
```

Appendix B Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

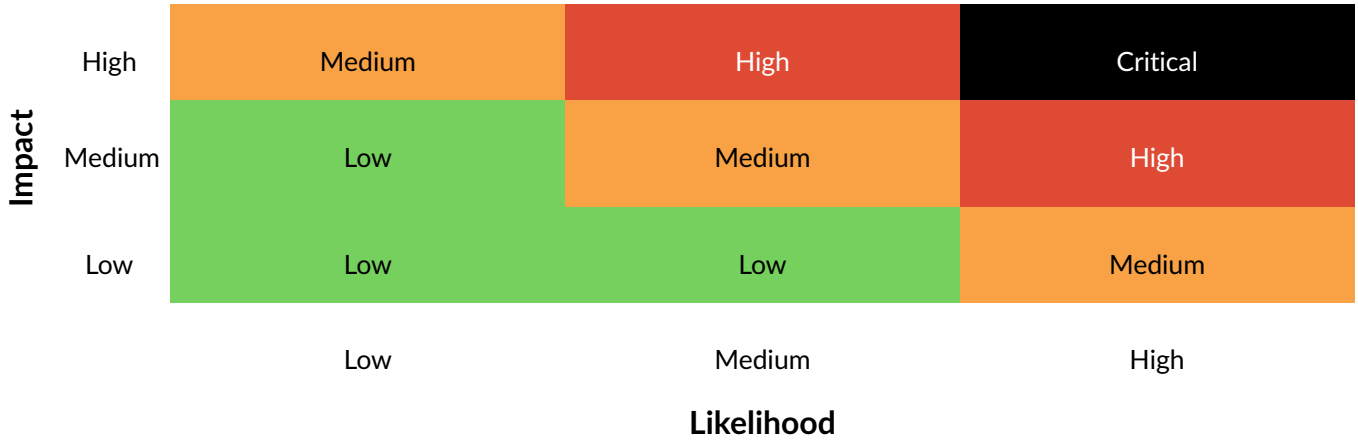

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] Sigma Prime. Solidity Security. Blog, 2018, Available: https://blog.sigmaprime.io/solidity-security.html. [Accessed 2018].

[2] NCC Group. DASP - Top 10. Website, 2018, Available: http://www.dasp.co/. [Accessed 2018].