Documentation
https://docs.google.com/document/d/1zOjFnNggXS4H5IpcziBxeqFvZ7RyEq2wtbMPOY6VovE/
https://docs.google.com/document/d/1Jm8jpyWjjesRB1yhLPfISqV3gqEKVDVWmzfJcekgoWc/edit
https://docs.google.com/document/d/1NyJDyiuPKnzFqCYWe_U8HUTbLkxyyZFIb56to-Rthw8/edit

BSTN.sol

Issue 1

Consider overriding the internal _transfer function to disallow token transfers to the token contract. This will prevent users from sending BSTN tokens to the token contract, resulting in loss of funds.

```
require(_to != address(this));
```

CDaiDelegate.sol

ok

CErc20.sol

Issue 1

https://medium.com/chainsecurity/trueusd-compound-vulnerability-bc5b696d29e2
sweepToken can cause pricing issues of the cToken if the underlying token has more than 1 address (e.g. TUSD described in the blog above).
A recommended change would be to check the balance of the underlying before and after the transfer to ensure that the balance has not changed. Alternatively, a simpler but incomplete fix would be to only allow the admin to call this function.

Notes
sweepToken can transfer to admin any token except underlying, callable by anyone.
isNative is not present unlike in compound.
_delegateCompLikeTo to delegate underlying token by admin call

CErc20Delegate.sol

ok

CErc20Delegator.sol

ok

CErc20Immutable.sol

OK

CEther.sol

Issue 1

doTransferOut uses .transfer, so if it is sending Ether to a smart contract with a fallback function that costs more than 2300 gas, such as a gnosis safe, it will revert.

Consider making a middle man contract with a default fallback function, that allows the user to send cEther tokens which will then be redeemed. The middle man contract will receive the redeemed native ETH, then send them back to the user (e.g. gnosis safe) using call instead of transfer, so that the fallback function gas consumption will not cause a revert when receiving the native ETH.

CToken.sol

Issue 1 [fixed]

If an underlying has any callback functionality, reentrancy is possible in another CToken contract as doTransferOut is done before the storage update. This is an issue which has happened with CREAM (AMP token), and AGAVE and Hundred finance (XDAI and other tokens on chain have erc677 functionality).

Move the doTransferOut after the change of state in the following functions:
redeemFresh

```
    /* We write previously calculated values into storage */
    totalSupply = vars.totalSupplyNew;
    accountTokens[redeemer] = vars.accountTokensNew;
    doTransferOut(redeemer, vars.redeemAmount); //<- move to here
```

borrowFresh

```
    /* We write the previously calculated values into storage */
    accountBorrows[borrower].principal = vars.accountBorrowsNew;
    accountBorrows[borrower].interestIndex = borrowIndex;
    totalBorrows = vars.totalBorrowsNew;

    doTransferOut(borrower, borrowAmount); // <- move to here
```

Ref:
https://github.com/compound-finance/compound-protocol/pull/152/commits/55daa663cefb409770a4e405d34d1e3755c4b5a3

Issue 2

In _acceptAdmin, msg.sender == address(0) conditional is unnecessary.
if (msg.sender != pendingAdmin || msg.sender == address(0)) {

Issue 3

borrowRatePerBlock and supplyRatePerBlock refer to block, when actually, timestamp is being used.

Other references to block instead of timestamp in the code include the following
CToken: L405
(MathError mathErr, uint blockDelta) = subUInt(currentBlockTimestamp, accrualBlockTimestampPrior);
L423
(mathErr, simpleInterestFactor) = mulScalar(Exp({mantissa: borrowRateMantissa}), blockDelta);

CTokenInterfaces.sol

Unitroller.sol

Issue 1

In _acceptAdmin, msg.sender == address(0) conditional is unnecessary.
if (msg.sender != pendingAdmin || msg.sender == address(0)) {

Comptroller.sol

Issue 1 [Fixed]

Possible loss of precision in liquidateCalculateSeizeTokens, if denominator decimals is much higher than numerator decimals (e.g. wnear 24 decimals vs usdc 6 decimals)

```
    Exp memory numerator;
    Exp memory denominator;
    Exp memory ratio;

    numerator = mul_(
        Exp({mantissa: liquidationIncentiveMantissa}),
        Exp({mantissa: priceBorrowedMantissa})
    );
    denominator = mul_(
        Exp({mantissa: priceCollateralMantissa}),
        Exp({mantissa: exchangeRateMantissa})
    );
    ratio = div_(numerator, denominator);

    seizeTokens = mul_ScalarTruncate(ratio, actualRepayAmount);
```

As a division is done to calculate the ratio, before the multiplication is done to calculate seize tokens, there can be some precision issues if wnear is the denominator and a small decimal coin like usdc is used as the numerator.

Consider multiplying the numerator by actualRepayAmount before dividing by the denominator to get ratio. seizeTokens will just be a truncation of the ratio.

E.g fix

```
    numerator = mul_(mul_(
        Exp({mantissa: liquidationIncentiveMantissa}),
        Exp({mantissa: priceBorrowedMantissa})
    ), actualRepayAmount);
    denominator = mul_(
        Exp({mantissa: priceCollateralMantissa}),
        Exp({mantissa: exchangeRateMantissa})
    );
ratio = div_(numerator, denominator);
seizeTokens = truncate(ratio);
```

ComptrollerInterface.sol

ok

ComptrollerStorage.sol

ok

EIP20Interface.sol

ok

EIP20NonStandardInterface.sol

ok

ErrorReporter.sol

ok

Exponential.sol

ok

ExponentialNoError.sol

Ok

SafeMath.sol

Ok

HomoraMath.sol

ok

InterestRateModel.sol

ok

JumpRateModel.sol

ok

JumpRateModelV2.sol

ok

LegacyInterestRateModel.sol

ok

LegacyJumpRateModelV2.sol

ok

BaseJumpRateModelV2.sol

ok

WhitePaperInterestRateModel.sol

ok

Maximillion.sol

Ok

AggregatorV2V3Interface.sol

ok

FluxOracle.sol

Issue 1

Add a check that feed.latestAnswer() is a positive non 0, since it returns a int256, before parsing to uint256

```
 function latestAnswer() external view returns (int256);
```

Issue 2

Admin can overwrite the prices to any arbitrary value. The private key of the admin should be managed securely as a compromised key could result in manipulation of the oracle prices. If the oracle prices are not intended to be set manually, the admin should be a secured multisig.

Issue 3

Instead of getting latestAnswer, which is a v2 interface function, consider using latestRoundData, which is a v3 interface function and checking require(answeredInRound >= roundID) to ensure that the data is the latest.

FluxOraclev1.sol

Issue 1

Add a check that feed.latestAnswer() is a positive non 0, since it returns a int256, before parsing to uint256

```
 function latestAnswer() external view returns (int256);
```

Issue 2

Admin can overwrite the prices to any arbitrary value. The private key of the admin should be managed securely as a compromised key could result in manipulation of the oracle prices. If the oracle prices are not intended to be set manually, the admin should be a secured multisig.

Issue 3

Instead of getting latestAnswer, which is a v2 interface function, consider using latestRoundData, which is a v3 interface function and checking require(answeredInRound >= roundID) to ensure that the data is the latest.

Issue 4

In getChainlinkPrice, if the feed.decimals is ever greater than 18, it will lead to revert due to underflow in subtraction.
uint decimalDelta = uint(18).sub(feed.decimals());

Consider putting back the logic from FluxOracle.

LPOracle.sol

Issue 1

Add a check that feed.latestAnswer() is a positive non 0, since it returns a int256, before parsing to uint256
 function latestAnswer() external view returns (int256);

Issue 2

Admin can overwrite the prices to any arbitrary value. The private key of the admin should be managed securely as a compromised key could result in manipulation of the oracle prices. If the oracle prices are not intended to be set manually, the admin should be a secured multisig.

Issue 3

Instead of getting latestAnswer, which is a v2 interface function, consider using latestRoundData, which is a v3 interface function and checking require(answeredInRound >= roundID) to ensure that the data is the latest.

AuroraStNear.sol

Ok

StNearFeed.sol

Issue 1

Add the following checks in the constructor

```
                    require(_stNearPrice > 1.06e10, "StNearPrice too low");
                    require(_stNearPrice < 1.1e10, "StNearPrice too high");
```

Issue 2

Add a check that nearFeed.latestAnswer() is a positive non 0, since it returns a int256, before parsing to uint256

```
 function latestAnswer() external view returns (int256);
```

Issue 3

Admin can overwrite the prices to any arbitrary value. The private key of the admin should be managed securely as a compromised key could result in manipulation of the oracle prices. If the oracle prices are not intended to be set manually, the admin should be a secured multisig.

Issue 4

Instead of getting latestAnswer, which is a v2 interface function, consider using latestRoundData, which is a v3 interface function and checking require(answeredInRound >= roundID) to ensure that the data is the latest.

NEAROracle.sol

Issue 1

Add a check that feed.latestAnswer() is a positive non 0, since it returns a int256, before parsing to uint256

```
 function latestAnswer() external view returns (int256);
```

Issue 2

Admin can overwrite the prices to any arbitrary value. The private key of the admin should be managed securely as a compromised key could result in manipulation of the oracle prices. If the oracle prices are not intended to be set manually, the admin should be a secured multisig.

Issue 3

Instead of getting latestAnswer, which is a v2 interface function, consider using latestRoundData, which is a v3 interface function and checking require(answeredInRound >= roundID) to ensure that the data is the latest.

TwapFeed.sol

Issue 1

decimals can be constant as it is declared once and never changed.

Issue 2

The following could result in underflow as normal arithmetic is used.
pairDecimalsDelta = asToken0 ? 18 + token1.decimals() - token0.decimals() : 18 + token0.decimals() - token1.decimals();

Issue 3

The following functions can be external as they are never called in the same contract:
latestAnswer

UQ112x112.sol

ok

SimplePriceOracle.sol

N/A

PriceOracle.sol

N/A

Reservoir.sol

ok

RewardDistributor.sol

Issue 1

In setRewardAddress, there is a lack of check that rewardType < rewardAddresses.length

Issue 2

In setRewardSpeedInternal, updateRewardBorrowIndex and updateRewardSupplyIndex should be called each time regardless of what the currentRewardSupplySpeed or currentRewardBorrowSpeed is.

This is because if the reward or supply for a token is paused (set to 0) for a long time, then unpaused (set to non 0) after that, a user who has supplied/borrowed before the pause and unpause time would be getting the rewards since the pause time, not the unpause time.

This is due to the fact that the supply index is not updated when the current reward speed is 0, during the unpause function call, setting from 0 to non 0.

Note that this will only affect markets that have been paused for rewards, then unpaused.

Issue 3

claimReward should not be payable as no native ETH is to be sent when calling the function
https://github.com/bastion-protocol/bastion-protocol/blob/master/contracts/RewardDistributor.sol#L510

CarefulMath.sol

Ok

Timelock.sol

ok

Not used

Governance/
Comp.sol
GovernorAlpha.sol
GovernorBravoDelegate.sol
GovernorBravoDelegateG1.sol
GovernorBravoDelegator.sol
GovernorBravoInterfaces.sol