yAudit Cove Boosties Review

Review Resources:

- Protocol documentation
- Cove Boosties Docs
- COVE Token Planning
- YSD, Yearn Strat, and coveYFI RFC
- Boosties Roles breakdown

Auditors:

- Sjkelleyjr (Jackson)
- adriro

Table of Contents

Review Summary

Cove Boosties

Boosties is a liquid locker and staking platform that allows users to efficiently benefit from Yearn v3 dYFI emissions on their gauge tokens. Through protocol-owned veYFI, users benefit from boosted Yearn rewards, which can be auto-compounded or manually managed, along with COVE token emissions.

The contracts of the Cove Boosties Repo were reviewed over 7 days. The code review was performed by 2 auditors between March 11 and March 19, 2024. The repository was under active development during the review, but the review was limited to the latest commit at the start of the review. This was commit b7564f528409a912ad3408ba1a861eed0b843811 for the Cove Boosties repo.

Scope

The scope of the review consisted of the following contracts at the specific commit:

```
src/rewards
├── BaseRewardsGauge.sol
├── ERC20RewardsGauge.sol
├── MiniChefV3.sol
├── RewardForwarder.sol
└── YSDRewardsGauge.sol
src/governance
└── CoveToken.sol
src
└── Yearn4626RouterExt.sol
```

After the findings were presented to the Cove team, fixes were made and included in several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, Cove and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Adequate access control is present in admin-controlled functionality, diversified through different roles.
Mathematics	Average	Issues related to rounding, decimals, and potential overflows were detected.
Complexity	Good	Complexity arising from the ecosystem and integrations is correctly managed.
Libraries	Good	The protocol uses an up-to-date version of the OpenZeppelin library.
Decentralization	Average	Contracts are not upgradeable, but the protocol still relies on trusted entities to oversee the protocol.
Code stability	Good	The repository was not under active development during the review.
Documentation	Good	The provided documentation was extensive and detailed. Code and contracts are well documented using NatSpec.
Monitoring	Good	Multiple events are emitted through the protocol life-cycle.
Testing and verification	Good	The codebase includes a complete test suite with unit, integration, fuzzing, invariant tests, and excellent coverage.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
    - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.
- Gas savings

    - Findings that can improve the gas efficiency of the contracts.
- Informational
    - Findings including recommendations and best practices.

Critical Findings

None.

High Findings

None.

Medium Findings

1. Medium - Max totals assets in YearnGaugeStrategy.sol ignores deposits coming from YSDRewardsGauge.sol

While the asset limit is checked in YSDRewardsGauge.sol, deposits to YearnGaugeStrategy.sol ignore assets deposited through the former.

Technical Details

The non-autocompounding version of the vault queries the strategy to validate the deposit limits.

```
086:     function _deposit(
087:         address caller,
088:         address receiver,
089:         uint256 assets,
090:         uint256 shares
091:     )
092:         internal
093:         virtual
094:         override(BaseRewardsGauge)
095:     {
096:         if (totalAssets() + assets > maxTotalAssets()) {
097:             revert MaxTotalAssetsExceeded();
098:         }
099:         BaseRewardsGauge._deposit(caller, receiver, assets, shares);
100:         IYearnStakingDelegate(yearnStakingDelegate).deposit(asset(), assets);
101:     }
```

Here, `maxTotalAssets()` is the available deposit limit of the strategy:

```
67:     function maxTotalAssets() public view virtual returns (uint256) {
68:         uint256 maxAssets = YearnGaugeStrategy(coveYearnStrategy).maxTotalAssets();
69:         uint256 totalAssetsInStrategy =
ITokenizedStrategy(coveYearnStrategy).totalAssets();
70:         if (totalAssetsInStrategy >= maxAssets) {
71:             return 0;
72:         } else {
73:             return maxAssets - totalAssetsInStrategy;
74:         }
75:     }
```

This indicates that the intention is to check if the current deposited assets in the gauge (`totalAssets()`) plus the new deposit (`assets`) don't exceed the available deposit limit in the strategy (`maxAssets - totalAssetsInStrategy`).

However, the same check if not accounted for in the strategy itself. YearnGaugeStrategy.sol defines `availableDepositLimit()` as:

```
116:     function availableDepositLimit(address) public view override returns (uint256)
{
117:         uint256 currentTotalAssets = TokenizedStrategy.totalAssets();
118:         uint256 currentMaxTotalAssets = _maxTotalAssets;
119:         if (currentTotalAssets >= currentMaxTotalAssets) {
120:             return 0;
121:         }
122:         // Return the difference between the max total assets and the current total
assets, an underflow is not possible
123:         // due to the above check
124:         unchecked {
125:             return currentMaxTotalAssets - currentTotalAssets;
126:         }
127:     }
```

For example, given the following state:

- `maxTotalAssets = 100`
- `totalAssetsInStrategy = 50`
- `totalAssetsInGauge = 20`

It won't be possible to deposit 50 tokens from YSDRewardsGauge.sol (as `50 + 20 + 50 > 100`) but it will be possible to deposit 50 tokens in YearnGaugeStrategy.sol (since `50 + 50 <= 100`).

Impact

Medium. Deployed assets may exceed the configured limit.

Recommendation

Either make YearnGaugeStrategy.sol aware of deposits through YSDRewardsGauge.sol, or check the number of assets using YearnStakingDelegate.sol, which should hold the total for both.

Developer Response

This was fixed in https://github.com/Storm-Labs-Inc/cove-contracts-boosties/pull/295

Moved deposit tracking / limit setting to YearnStakingDelegate.sol

2. Medium - Any `deposit` not followed by a `harvest` will revert when withdrawing

Technical Details

When a user deposits into a `MiniChefV3` for the first time, their `rewardDebt` is calculated as:

```
user.rewardDebt += amount * pool.accRewardPerShare / _ACC_REWARD_TOKEN_PRECISION;
```

where `pool.accRewardPerShare` is calculated in `updatePool()`:

```
        if (block.timestamp > pool.lastRewardTime) {
            uint256 lpSupply_ = lpSupply[pid];
            uint256 totalAllocPoint_ = totalAllocPoint;
            if (lpSupply_ != 0) {
                if (totalAllocPoint_ != 0) {
                    uint256 time = block.timestamp - pool.lastRewardTime;
                    uint256 rewardAmount = time * rewardPerSecond * pool.allocPoint /
totalAllocPoint_;
                    pool.accRewardPerShare += SafeCast.toUint128(rewardAmount *
_ACC_REWARD_TOKEN_PRECISION / lpSupply_);
                }
            }
            pool.lastRewardTime = uint64(block.timestamp);
            _poolInfo[pid] = pool;
            emit LogUpdatePool(pid, pool.lastRewardTime, lpSupply_,
pool.accRewardPerShare);
        }
```

Similarly, this amount is decremented when withdrawing as well. However, if no `harvest()` has occurred before calling `withdraw()` and any time has passed, L433 will underflow since the `pool.accRewardPerShare` will have increased without any corresponding increase to the user's `user.rewardDebt`.

See this POC, which is the `test_harvest()` unit test but with `harvest()` replaced with a withdrawal.

```
function test_withdrawUnderflow() public {
        miniChef.setRewardPerSecond(1e15);
        miniChef.add(1000, lpToken, IMiniChefV3Rewarder(address(0)));
        uint256 rewardCommitment = 10e25;
        rewardToken.mint(address(this), rewardCommitment);
        rewardToken.approve(address(miniChef), rewardCommitment);
        miniChef.commitReward(rewardCommitment);
        uint256 pid = miniChef.poolLength() - 1;
        uint256 amount = 1e18;
        lpToken.mint(alice, amount);

        vm.startPrank(alice);
        lpToken.approve(address(miniChef), amount);
        miniChef.deposit(pid, amount, alice);

        // Fast forward to accrue rewards
        vm.warp(block.timestamp + 1 days);

        uint256 initialRewardBalance = rewardToken.balanceOf(alice);
        uint256 pendingReward = miniChef.pendingReward(pid, alice);
        uint256 expectedTotalReward = miniChef.rewardPerSecond() * 1 days;
        assertEq(pendingReward, expectedTotalReward, "Pending rewards not accrued
correctly");

        vm.expectRevert(stdError.arithmeticError);
        vm.startPrank(alice);
        miniChef.withdraw(pid, amount, alice);
    }
```

Impact

Medium, a user will be unable to withdraw their funds after depositing. However, they can get their funds unstuck by calling `harvest()` before `withdraw()` to update their `rewardDebt`.

Recommendation

Subtract the minimum of `user.amount * pool.accRewardPerShare / _ACC_REWARD_TOKEN_PRECISION` and `user.rewardDebt` when withdrawing.

Developer Response

Fixed in https://github.com/Storm-Labs-Inc/cove-contracts-boosties/pull/292

Replaces `withdraw()` with `harvestAndWithdraw()` See tests for the updated behavior

3. Medium - RewardForwarder can be used to dilute reward distribution

By calling `forwardRewardToken()` with a minimal amount, a malicious actor can dilute the reward distribution process by extending the period another week.

Technical Details

RewardForwarder.sol is in charge of collecting the reward tokens and calling `depositRewardToken()` on their respective gauge.

Since this process is permissionless, anyone can call this function using a minimal amount to extend and dilute the reward distribution process. For example, a bad actor can transfer 1 wei of the reward token to the RewardForwarder.sol contract and then call `forwardRewardToken()`, which will take any pending tokens on the existing distribution and extend them over a new weekly period.

Impact

Medium. Risk of griefing in the reward distribution process.

Recommendation

Make `forwardRewardToken()` permissioned to a certain role, or impose a minimum limit on the number of reward tokens that would make sense to extend the period.

Developer Response

Fixed in https://github.com/Storm-Labs-Inc/cove-contracts-boosties/pull/301

4. Medium - Incorrect decimal normalization in YearnV2 calculations

The functions present in Yearn4626RouterExt.sol incorrectly assume that YearnV2 vaults have a decimal precision of 18.

Technical Details

In each of the preview functions, the implementation uses `1e18` to convert between assets and shares of YearnV2 vaults. This is to normalize the calculation given the multiplication or division by the vault's price per share (PPS). For example, `previewDeposits()` calculates the amount of shares using the following:

```
225:     sharesOut[i] =
226:         Math.mulDiv(assetsIn, 1e18, IYearnVaultV2(vault).pricePerShare(),
Math.Rounding.Down) - 1;
```

However, YearnV2 vaults take their decimals from the underlying asset's decimals. For example, the USDC vault has 6 decimals since the USDC token has 6 decimals, which means that its PPS is also given in 6 decimals precision.

Impact

Medium. Calculations will be incorrect when using a Yearn V2 vault with several decimals different from 18.

Recommendation

Take the decimals from the vault and normalize using that value. For example, in `previewDeposits()`:

```
sharesOut[i] = Math.mulDiv(assetsIn, 10 ** IERC20(vault).decimals(),
IYearnVaultV2(vault).pricePerShare(), Math.Rounding.Down) - 1;
```

`previewMints()`, `previewWithdraws()` and `previewRedeems()` should also be adjusted accordingly.

Developer Response

Fixed in https://github.com/Storm-Labs-Inc/cove-contracts-boosties/pull/288

5. Medium - Incorrect implementation of `previewMints()` and `previewWithdraws()` functions

Both of these functions work on expected output amounts, i.e. given a desired output amount calculate the required input amount. Their implementations should start at the end of the path and go backward calculating the amount of input tokens.

Technical Details

The `previewMints()` function takes a `sharesOut` amount and should calculate the amount of input assets at each step. The path here is represented as a succession of tokens that wrap each other (being `path[0]` the input token and `path[1]` the first vault).

```
path = [tokenIn, vault0, vault1, ..., vaultN]
```

To do so, the implementation loops through each vault calling `previewMint()` at each step.

```
263:         for (uint256 i; i < assetsInLength;) {
264:             address vault = path[i + 1];
265:             if (!Address.isContract(vault)) {
266:                 revert PreviewNonVaultAddressInPath(vault);
267:             }
268:             address vaultAsset = address(0);
269:             (bool success, bytes memory data) =
vault.staticcall(abi.encodeCall(IERC4626.asset, ()));
270:             if (success) {
271:                 vaultAsset = abi.decode(data, (address));
272:                 assetsIn[i] = IERC4626(vault).previewMint(sharesOut);
273:             } else {
274:                 (success, data) =
vault.staticcall(abi.encodeCall(IYearnVaultV2.token, ()));
275:                 if (success) {
276:                     vaultAsset = abi.decode(data, (address));
277:                     assetsIn[i] =
278:                         Math.mulDiv(sharesOut,
IYearnVaultV2(vault).pricePerShare(), 1e18, Math.Rounding.Up) + 1;
279:                 } else {
280:                     revert PreviewNonVaultAddressInPath(vault);
281:                 }
282:             }
283:
284:             if (vaultAsset != path[i]) {
285:                 revert PreviewVaultMismatch();
286:             }
287:             sharesOut = assetsIn[i];
288:
289:             /// @dev Increment the loop counter within an unchecked block to avoid
redundant gas cost associated with
290:             /// overflow checking.  This is safe because the loop's exit condition
ensures that `i` will not exceed
291:             /// `assetsInLength - 1`, preventing overflow.
292:             unchecked {
```

```
293:                 ++i;
294:             }
295:         }
```

This is incorrect since `sharesOut` is the expected result at the **last** vault. The implementation should loop backward, calculating the amount of assets in to get the desired `sharesOut` from the last vault first, then using that amount as the next `sharesOut` to get the amount of assets in for the penultimate vault, and so on.

Similarly, `previewWithdraws()` calculates the amount of input shares given a desired amount of output assets. Here the path is represented by the succession of vaults and the output token as the last element.

```
path = [vaultN, vaultN-1, ..., vault1, tokenOut]
```

Again, the implementation traverses the path from start to end:

```
317:         for (uint256 i; i < sharesInLength;) {
318:             address vault = path[i];
319:             if (!Address.isContract(vault)) {
320:                 revert PreviewNonVaultAddressInPath(vault);
321:             }
322:             address vaultAsset = address(0);
323:             (bool success, bytes memory data) =
vault.staticcall(abi.encodeCall(IERC4626.asset, ()));
324:             if (success) {
325:                 vaultAsset = abi.decode(data, (address));
326:                 sharesIn[i] = IERC4626(vault).previewWithdraw(assetsOut);
327:             } else {
328:                 (success, data) =
vault.staticcall(abi.encodeCall(IYearnVaultV2.token, ()));
329:                 if (success) {
330:                     vaultAsset = abi.decode(data, (address));
331:                     sharesIn[i] = Math.mulDiv(assetsOut, 1e18,
IYearnVaultV2(vault).pricePerShare(), Math.Rounding.Down);
332:                 } else {
333:                     // StakeDAO gauge token
334:                     // StakeDaoGauge.staking_token().token() is the yearn vault v2
token
335:                     (success, data) =
vault.staticcall(abi.encodeCall(IStakeDaoGauge.staking_token, ()));
336:                     if (success) {
337:                         vaultAsset = IStakeDaoVault(abi.decode(data,
(address))).token();
338:                         sharesIn[i] = assetsOut;
339:                     } else {
340:                         revert PreviewNonVaultAddressInPath(vault);
341:                     }
342:                 }
343:             }
344:             if (vaultAsset != path[i + 1]) {
345:                 revert PreviewVaultMismatch();
```

```
346:             }
347:             assetsOut = sharesIn[i];
348:
349:             /// @dev Increment the loop counter without checking for overflow.
This is safe because the for loop
350:             /// naturally ensures that `i` will not overflow as it is bounded by
`sharesInLength`, which is derived from
351:             /// the length of the `path` array.
352:             unchecked {
353:                 ++i;
354:             }
355:         }
```

This is incorrect too, as the implementation should start from the last vault (i.e. `path[length - 2]`) and work backward to calculate the amount of input shares. Since we want `assetsOut` of the output token (`path[length - 1]`) we should first query `previewWithdraw()` on the vault that unwraps the output token, which is `path[length - 2]`, and traverse the path in reverse order successively calling `previewWithdraw()`.

Impact

Medium. The implementation is broken and won't return the intended values.

Recommendation

Start with the last vault in the path, looping backward to calculate the amount of input assets in reverse order.

Developer Response

Fixed in https://github.com/Storm-Labs-Inc/cove-contracts-boosties/pull/298.

6. Medium - Incorrect value sent to rewarder callback in MiniChefV3.sol

The `harvest()` function notifies the rewarder using the `pendingReward_` amount instead of the actual reward specified by `rewardAmount`.

Technical Details

In the new version of the contract, MiniChefV3.sol stores unpaid rewards to the user if the available reward tokens are not enough to cover the harvested amount.

```
453:     function harvest(uint256 pid, address to) public {
454:         PoolInfo memory pool = updatePool(pid);
455:         UserInfo storage user = _userInfo[pid][msg.sender];
456:         uint256 accumulatedReward = user.amount * pool.accRewardPerShare /
_ACC_REWARD_TOKEN_PRECISION;
457:         uint256 pendingReward_ = accumulatedReward - user.rewardDebt +
user.unpaidRewards;
458:
459:         // Effects
460:         user.rewardDebt = accumulatedReward;
461:
462:         // Interactions
463:         uint256 rewardAmount = 0;
464:         if (pendingReward_ != 0) {
465:             uint256 availableReward_ = availableReward;
466:             uint256 unpaidRewards_ = 0;
467:             rewardAmount = pendingReward_ > availableReward_ ? availableReward_ :
pendingReward_;
468:             /// @dev unchecked is used as the subtraction is guaranteed to not
underflow because
469:             /// `rewardAmount` is always less than or equal to `availableReward_`.
470:             unchecked {
471:                 availableReward -= rewardAmount;
472:                 unpaidRewards_ = pendingReward_ - rewardAmount;
473:             }
474:             user.unpaidRewards = unpaidRewards_;
475:         }
476:
477:         emit Harvest(msg.sender, pid, rewardAmount);
478:
479:         if (pendingReward_ != 0) {
480:             if (rewardAmount != 0) {
```

```
481:                 REWARD_TOKEN.safeTransfer(to, rewardAmount);
482:             }
483:         }
484:
485:         IMiniChefV3Rewarder _rewarder = rewarder[pid];
486:         if (address(_rewarder) != address(0)) {
487:             _rewarder.onReward(pid, msg.sender, to, pendingReward_, user.amount);
488:         }
489:     }
```

The logic checks if `availableReward` is enough to cover the required amount given by `pendingReward_`. If tokens are not enough, it will send the available portion and store the rest in `user.unpaidRewards`. The actual amount sent to the user is `rewardAmount`, while `pendingReward_` has the current harvested amount plus any previous unpaid tokens.

Note that line 487 notifies the rewarder of the event, but it uses `pendingReward_` instead of `rewardAmount`. When the available reward tokens present in the contract are less than `pendingReward_`, then a user could repeatedly call `harvest()` and the implementation will notify the rewarder each time with a positive value for the `rewardAmount` argument. This could be used to exploit a rewarder if, for example, its implementation also distributes rewards based on this parameter.

A similar issue is present in `emergencyWithdraw()`. Here the `onReward()` callback is wrapped in a `try/catch` statement. A malicious user could intentionally send a low gas limit such that the call to `onReward()` fails, but the calling frame succeeds, which can be done using the "1/64 rule" defined in EIP-150. In this scenario, the user has removed their stake from MiniChefV3.sol, but the action has not been registered in the rewarder.

Impact

Medium. Depending on the nature of the rewarder implementation, these issues could be used to exploit the integration.

Recommendation

For `harvest()`, just change `pendingReward_` to `rewardAmount`.

For `emergencyWithdraw()`, if the callback needs to be optional to allow a potential failure, document and notify integrators of this particular behavior.

Developer Response

Fixed in https://github.com/Storm-Labs-Inc/cove-contracts-boosties/pull/296

Low Findings

1. Low - `depositRewardToken()` fails to check if the token is supported

The implementation of `depositRewardToken()` doesn't check if the given `rewardToken` is a valid reward token.

Technical Details

The manager role can deposit tokens for an unsupported reward token as the implementation doesn't check if `rewardToken` has been previously configured as a valid reward token. Unsupported tokens are not processed in `_checkpointRewards()`.

Impact

Low. Deposited funds could be lost. Requires user mistake.

Recommendation

Check that `reward.distributor != address(0)`.

```
    function depositRewardToken(address rewardToken, uint256 amount) external
nonReentrant {
        Reward storage reward = _rewardData[rewardToken];
+       address distributor = reward.distributor;
+       if (distributor == address(0)) {
+         revert InvalidDistributorAddress();
+       }
-       if (!(msg.sender == reward.distributor || hasRole(MANAGER_ROLE, msg.sender))) {
+       if (!(msg.sender == distributor || hasRole(MANAGER_ROLE, msg.sender))) {
            revert Unauthorized();
        }
```

Developer Response

Fixed in https://github.com/Storm-Labs-Inc/cove-contracts-boosties/pull/282

2. Low - Claimed tokens can overflow and become claimable

The BaseRewardsGauge.sol contracts track the amount of claimed reward tokens using a shared slot with the amount of claimable tokens. An overflow in the claimed counter could be used to drain the reward tokens.

Technical Details

The implementation of BaseRewardsGauge.sol uses a single storage slot to track the claimed and claimable amount of reward tokens for each user. The lower 128 bits store the claimed amount of tokens, while the upper 128 bits store the claimable amount.

When rewards are claimed, `claimData` is updated with the previously claimed amount (`totalClaimed`) plus the new claimable amount (`totalClaimable`).

```
416:             claimData[user][token] = claim ? totalClaimed + totalClaimable :
totalClaimed + (totalClaimable << 128);
```

As the updated amount is not validated to check if it fits within 128 bits, an overflow could occur in which the claimed amount starts overflowing into the claimable region.

Impact

Low. An overflow in the claimed amount of reward tokens could allow a user to withdraw more tokens than allocated, leading to a potential drain of the contract. However, this requires overflowing a 128-bit counter, which should be unlikely for most tokens.

Recommendation

Validate that `totalClaimed + totalClaimable` is no greater than the maximum value for the `uint128` type.

Developer Response

Acknowledged, won't fix.

We don't anticipate needing to support reward tokens with a max supply of more than `type(uint128).max`.

We have updated the natspec comments to reflect this limitation here:

- https://github.com/Storm-Labs-Inc/cove-contracts-boosties/pull/283
- https://github.com/Storm-Labs-Inc/cove-contracts-boosties/pull/307

3. Low - Rewards Gauge and MinichefV3 are incompatible with fee-on-transfer tokens

Fee-on-transfer (FOT) tokens cannot be used as reward tokens in BaseRewardsGauge.sol and MiniChefV3.sol.

Technical Details

The implementation of `depositRewardToken()` updates the reward accounting using the given `amount` and then transfers the tokens from the caller to the contract.

```
289:         emit RewardTokenDeposited(rewardToken, amount, newRate, block.timestamp);
290:         reward.rate = newRate;
291:         reward.lastUpdate = block.timestamp;
292:         reward.periodFinish = block.timestamp + _WEEK;
293:         // slither-disable-next-line weak-prng
294:         reward.leftOver = newRewardAmount % _WEEK;
295:         IERC20(rewardToken).safeTransferFrom(msg.sender, address(this), amount);
```

If the `rewardToken` is a FOT token, the amount of received tokens will be less than the specified `amount`, creating a discrepancy between the amount used in calculations, and the effective amount of received tokens.

Similarly, in `commitReward()` the `availableReward` variable is updated by the given `amount` without checking the actual transferred amount.

```
328:     function commitReward(uint256 amount) external {
329:         availableReward = availableReward + amount;
330:         emit LogRewardCommitted(amount);
331:         REWARD_TOKEN.safeTransferFrom(msg.sender, address(this), amount);
332:     }
```

Impact

Low. Reward accounting will be inconsistent as the effective amount of available tokens is less than the amount used in calculations.

Recommendation

If FOT tokens are expected to be used as the reward token, then first execute the transfer and record the difference in balance to calculate the effective amount of tokens received.

Developer Response

Acknowledged, won't fix.

We do not intend for these contracts to support rebasing or fee on transfer tokens.

We have updated the natspec comments to reflect that here: https://github.com/Storm-Labs-Inc/cove-contracts-boosties/pull/284

4. Low - `Yearn4626RouterExt`'s `previewDeposits()` can underflow

Technical Details

In `previewDeposits()`, if the vault is a `YearnVaultV2`, the `sharesOut` is calculated using this line

```
sharesOut[i] = Math.mulDiv(assetsIn, 1e18, IYearnVaultV2(vault).pricePerShare(),
    Math.Rounding.Down) - 1;
```

Since the division is rounded down, in certain edge cases it can round down to 0. 1 is then subtracted from this, which would cause an underflow and a revert.

Impact

Low. It's a view function, so the function can be re-tried without the offending vault.

Recommendation

Remove the `- 1`, or, if it's necessary, check if the division is 0 and revert with a message.

Developer Response

Fixed in https://github.com/Storm-Labs-Inc/cove-contracts-boosties/pull/302

Instead of relying on `pricePerShare()`, use the same logic from Vault.vy for calculating correct share values.

Also removed `depositToVaultV2()` function as IYearnVaultV2's `deposit(uint256 amount, address to) returns (uint256 shares)` `signature matches ERC4626's` deposit(uint256 amount, address to) returns (uint256 shares)`

In favor of using the existing `router.deposit()` function.

5. Low - Max deposit should be defined in YSDRewardsGauge.sol

The vault imposes a deposit limit but doesn't override `maxDeposit()` or `maxMint()`.

Technical Details

According to the ERC4626 standard, `maxDeposit()` and `maxMint()` should be aware of any limit in `deposit()` and `mint()`.

The YSDRewardsGauge.sol vault has a limit defined in `_deposit()` (used for both `deposit()` and `mint()`) but doesn't override the defaults, which are `type(uint256).max`.

Impact

Low. Failure to comply with the ERC4626 standard.

Recommendation

Override `maxDeposit()` and `maxMint()` to align these with the behavior of `_deposit()`. Note that the base ERC4626 implementation already checks `maxDeposit()` and `maxMint()`, so the explicit check could be removed from `_deposit()`.

Developer Response

This was fixed in:

- https://github.com/Storm-Labs-Inc/cove-contracts-boosties/pull/295
- https://github.com/Storm-Labs-Inc/cove-contracts-boosties/pull/305
- https://github.com/Storm-Labs-Inc/cove-contracts-boosties/pull/309

6. Low - `SafeCast` in `MiniChefV2`'s `updatePool()` can fail in extreme cases

Technical Details

Depending on `rewardPerSecond`, `pool.allocPoint`, `lpSupply`, and time elapsed between updates, the SafeCast when incrementing `pool.accRewardPerShare` can fail, bricking every function that calls `updatePool()`.

See this POC, which uses max values and waits 4 weeks to update the pool.

```
    function test_rewardShareSafecast() public {
        miniChef.setRewardPerSecond(miniChef.MAX_REWARD_TOKEN_PER_SECOND());
        miniChef.add(type(uint64).max, lpToken, IMiniChefV3Rewarder(address(0)));
        uint256 rewardCommitment = 10e25;
        rewardToken.mint(address(this), rewardCommitment);
        rewardToken.approve(address(miniChef), rewardCommitment);
        miniChef.commitReward(rewardCommitment);
        uint256 pid = miniChef.poolLength() - 1;
        uint256 amount = 1;
        lpToken.mint(alice, amount);

        vm.startPrank(alice);
        lpToken.approve(address(miniChef), amount);
        miniChef.deposit(pid, amount, alice);

        vm.warp(block.timestamp + 4 weeks);
        vm.expectRevert();
        miniChef.updatePool(pid);
    }
```

Impact

Low. The values used would have to be extreme to cause the cast to fail and can be altered to unbrick the contract.

Recommendation

Allow `pool.accRewardPerShare` to be a `uint256` instead of a `uint128` to remove the need for casting, require smaller maximums for `rewardPerSecond` and `pool.allocPoint`, or ensure `updatePool()` is called regularly.

Developer Response

Fixed in https://github.com/Storm-Labs-Inc/cove-contracts-boosties/pull/293 Changed accRewardPerShare and allocPoint types

7. Low - `pool.accRewardPerShare` can overflow in extreme cases

Technical Details

Depending on values for `rewardPerSecond` and `pool.allocPoint`, `lpSupply` a pool's `accRewardPerShare` can overflow in `updatePool()` bricking every function that calls updatePool().

See this POC, which uses max values and updates the pool twice every 2 weeks.

```
function test_rewardShareOverflow() public {
        miniChef.setRewardPerSecond(miniChef.MAX_REWARD_TOKEN_PER_SECOND());
        miniChef.add(type(uint64).max, lpToken, IMiniChefV3Rewarder(address(0)));
        uint256 rewardCommitment = 10e25;
        rewardToken.mint(address(this), rewardCommitment);
        rewardToken.approve(address(miniChef), rewardCommitment);
        miniChef.commitReward(rewardCommitment);
        uint256 pid = miniChef.poolLength() - 1;
        uint256 amount = 1;
        lpToken.mint(alice, amount);

        vm.startPrank(alice);
        lpToken.approve(address(miniChef), amount);
        miniChef.deposit(pid, 1, alice);

        vm.warp(block.timestamp + 2 weeks);
        miniChef.updatePool(pid);

        vm.warp(block.timestamp + 2 weeks);
        vm.expectRevert(stdError.arithmeticError);
        miniChef.updatePool(pid);
    }
```

Impact

Low. This requires extreme values and can be fixed by altering the values. However, these values fall within the current bounds of the system.

Recommendation

Allow `pool.accRewardPerShare` to be a `uint256` instead of a `uint128` and require smaller maximums for `rewardPerSecond` and `pool.allocPoint`.

Developer Response

Fixed in https://github.com/Storm-Labs-Inc/cove-contracts-boosties/pull/293 Changed accRewardPerShare and allocPoint types

8. Low - Incorrect rounding in YearnV2 calculations

There are a couple of calculations related to YearnV2 shares that have incorrect rounding.

Technical Details

In `previewWithdraws()`, the implementations the amount of input shares using the following calculation:

```
331:     sharesIn[i] = Math.mulDiv(assetsOut, 1e18, IYearnVaultV2(vault).pricePerShare(),
Math.Rounding.Down);
```

Since we are calculating the amount of required shares for a given output amount, the implementation should round up.

Similarly, in `previewRedeems()` the calculation is:

```
390:     assetsOut[i] = Math.mulDiv(sharesIn, IYearnVaultV2(vault).pricePerShare(), 1e18,
Math.Rounding.Up);
```

Since here we are calculating the amount of output asset for a given input shares, the implementation should round down.

Impact

Low. Calculations will use incorrect rounding, leading to small differences.

Recommendation

Change the rounding used in each calculation to always round in favor of the vault.

Developer Response

Fixed in https://github.com/Storm-Labs-Inc/cove-contracts-boosties/pull/287 Changed rounding in favor of the vault

Gas Saving Findings

1. Gas - `REWARD_TOKEN.safeTransfer()` can occur in if scope to reduce gas

Technical Details

In `MiniChefV3`'s `harvest()`, `if (pendingReward_ != 0) {` is checked to calculate the `rewardAmount`, it is then checked again after emitting the `Harvest` event to determine whether to transfer the REWARD_TOKEN or not. This can be done inside the same if block.

```
        if (pendingReward_ != 0) {
            uint256 availableReward_ = availableReward;
            uint256 unpaidRewards_ = 0;
            rewardAmount = pendingReward_ > availableReward_ ? availableReward_ :
pendingReward_;
            /// @dev unchecked is used as the subtraction is guaranteed to not underflow
because
            /// `rewardAmount` is always less than or equal to `availableReward_`.
            unchecked {
                availableReward -= rewardAmount;
                unpaidRewards_ = pendingReward_ - rewardAmount;
            }
            user.unpaidRewards = unpaidRewards_;
+           if (rewardAmount != 0) {
+               REWARD_TOKEN.safeTransfer(to, rewardAmount);
+           }
        }

        emit Harvest(msg.sender, pid, rewardAmount);

-        if (pendingReward_ != 0) {
-           if (rewardAmount != 0) {
-               REWARD_TOKEN.safeTransfer(to, rewardAmount);
-           }
-       }
```

Impact

Gas savings.

Recommendation

Move the above lines into the `if (pendingReward_ != 0) {}` block, so it's not checked twice unnecessarily.

Developer Response

Fixed in https://github.com/Storm-Labs-Inc/cove-contracts-boosties/pull/303.

2. Gas - Use `+=` to increment `lpSupply`

Technical Details

`lpSupply[pid]` is read twice, once to read the old value, and once to write the new value after incrementing it by `amount`, this can be replaced by `+=`.

Impact

Gas savings.

Recommendation

Use `+=` to increment the `lpSupply[pid]`, as is done in `withdraw()` when decrementing.

Developer Response

Fixed in https://github.com/Storm-Labs-Inc/cove-contracts-boosties/pull/290.

3. Gas - Contract validation can be omitted in Yearn4626RouterExt.sol

All the preview functions check that vaults present in the path are contracts. This isn't necessary since return data decoding would still fail if not.

Technical Details

While looping through the path, the implementation of each preview function checks that vault addresses have code. For example, in `previewDeposits()`:

```
212:             address vault = path[i + 1];
213:             if (!Address.isContract(vault)) {
214:                 revert PreviewNonVaultAddressInPath(vault);
215:             }
```

This isn't really necessary, as the decoding of `data` would still fail when trying to fetch the address:

```
218:             if (success) {
219:                 vaultAsset = abi.decode(data, (address));
220:                 sharesOut[i] = IERC4626(vault).previewDeposit(assetsIn);
221:             } else {
```

If `vault` doesn't have code, `success` will be true, but the call would fail because `data` would be empty and `abi.decode(data, (address))` will raise.

Impact

Gas savings.

Recommendation

Remove the `isContract()` check.

Developer Response

Acknowledged, won't fix.

For this issue, we would like to maintain the custom error for easier debugging on the frontend side. For context, the intended use of the preview functions is for off-chain viewing; therefore, gas is less of a concern.

4. Gas - Use unchecked math if no overflow risk

There are math operations that can be done unchecked arithmetic for gas savings.

Technical Details

- BaseRewardsGauge.sol#L280
- BaseRewardsGauge.sol#L407
- MiniChefV3.sol#L350
- MiniChefV3.sol#L375

Impact

Gas savings.

Recommendation

Use unchecked block if there is no overflow or underflow risk for gas savings.

Developer Response

Fixed in https://github.com/Storm-Labs-Inc/cove-contracts-boosties/pull/294

Informational Findings

1. Informational - Inaccurate documentation in CoveToken.sol

The documentation for CoveToken.sol mentions that when the contract is paused, transfers should be done between an allowed sender and receiver. However, in the implementation, only one party needs to be allowed.

Technical Details

The implementation of CoveToken.sol overrides the `_beforeTokenTransfer()` callback to restrict transfers when the contract is paused.

```
204:     /**
205:      * @dev Hook that is called before any transfer of tokens. This includes
minting and burning.
206:      *      It checks if the contract is paused and if so, only allows transfers
from allowed transferrers
207:      *      to allowed transferees.
208:      * @param from The address which is transferring tokens.
209:      * @param to The address which is receiving tokens.
210:      * @param amount The amount of tokens being transferred.
211:      */
212:     function _beforeTokenTransfer(address from, address to, uint256 amount)
internal override {
213:         // Check if the transfer is allowed
214:         // When paused, only allowed transferrers can transfer and only allowed
transferees can receive
215:         if (paused()) {
216:             if (!allowedSender[from]) {
217:                 if (!allowedReceiver[to]) {
218:                     revert Errors.TransferNotAllowedYet();
219:                 }
220:             }
221:         }
222:         super._beforeTokenTransfer(from, to, amount);
223:     }
```

The documentation indicates that both parties should be in their respective allowed list, however, one inclusion is required in the actual implementation.

Additionally, this document specifies that the name for the token is “Cove DAO” and that it has burnable capabilities, which differ from the reviewed implementation.

Impact

Informational.

Recommendation

Correct documentation.

Developer Response

Fixed in https://github.com/Storm-Labs-Inc/cove-contracts-boosties/pull/285.

We have corrected the token name and updated the documentation to consistently reflect that only one inclusion is the desired behaviour.

2. Informational - Duplicated code in `maxTotalAssets()`

The `maxTotalAssets()` function present in YSDRewardsGauge.sol is the same as `availableDepositLimit()` in YearnGaugeStrategy.sol.

Technical Details

YSDRewardsGauge.sol#L67-L75

```
67:     function maxTotalAssets() public view virtual returns (uint256) {
68:         uint256 maxAssets = YearnGaugeStrategy(coveYearnStrategy).maxTotalAssets();
69:         uint256 totalAssetsInStrategy =
ITokenizedStrategy(coveYearnStrategy).totalAssets();
70:         if (totalAssetsInStrategy >= maxAssets) {
71:             return 0;
72:         } else {
73:             return maxAssets - totalAssetsInStrategy;
74:         }
75:     }
```

YearnGaugeStrategy.sol#L116-L127

```
116:     function availableDepositLimit(address) public view override returns (uint256)
{
117:         uint256 currentTotalAssets = TokenizedStrategy.totalAssets();
118:         uint256 currentMaxTotalAssets = _maxTotalAssets;
119:         if (currentTotalAssets >= currentMaxTotalAssets) {
120:             return 0;
121:         }
122:         // Return the difference between the max total assets and the current total
assets, an underflow is not possible
123:         // due to the above check
124:         unchecked {
125:             return currentMaxTotalAssets - currentTotalAssets;
126:         }
127:     }
```

Impact

Informational.

Recommendation

`maxTotalAssets()` could delegate to `coveYearnStrategy.availableDepositLimit()`.

Developer Response

Fixed in:

- https://github.com/Storm-Labs-Inc/cove-contracts-boosties/pull/295
- https://github.com/Storm-Labs-Inc/cove-contracts-boosties/pull/305

Final Remarks

The Cove Boosties protocol enables the optimization of Yearn V3 dYFI emissions, allowing users to deposit their Yearn gauge tokens and benefit from protocol-owned veYFI boost. The yAudit team conducted a comprehensive review of the contracts related to the distribution of rewards within the protocol.

At its core, BaseRewardsGauge.sol provides an implementation of the classic staking algorithm in a multi-reward token fashion. This serves as the foundation for the two versions of the vault that handle both auto-compounded and non-autocompounded variants.

Additionally, the Cove team introduced a revamped version of Sushi's MiniChef contract, intended to handle incentive programs for the COVE token.

Although several issues of medium severity were identified, no critical findings were reported, reflecting the overall quality of the codebase and the team's organizational practices.

We value the thorough documentation provided, the extensive test suite present in the protocol, and the team's prompt response in addressing the issues identified during our review process.