bveCVX by BadgerDAO contest Findings & Analysis Report

2021-11-05

Table of contents

Overview

About C4

Code 432n4 (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

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

During the code contest outlined in this document, C4 conducted an analysis of the bveCVX smart contract system written in Solidity. The code contest took place between September 2—September 8 2021.

Wardens

7 Wardens contributed reports to the bveCVX by BadgerDAO code contest:

1. cmichel
2. tabish
3. hickuphh3
4. pauliax
5. JMukesh
6. pmerkleplant
7. patitonar

This contest was judged by ghoul.sol.

Final report assembled by moneylegobatman and CloudEllie.

Summary

The C4 analysis yielded an aggregated total of 21 unique vulnerabilities and 51 total findings. All of the issues presented here are linked back to their original submission.

Of these findings, 1 received a risk rating in the category of HIGH severity, 2 received a risk rating in the category of MEDIUM severity, and 18 received a risk rating in the category of LOW severity.

C4 analysis also identified 13 non-critical recommendations and 17 gas optimizations.

Scope

The code under review can be found within the C4 bveCVX code contest repository is composed of 219 smart contracts written in the Solidity programming language and includes 21,403 lines of Solidity code.

Severity Criteria

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

Vulnerabilities are divided into three primary risk categories: high, medium, and low.

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

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

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

High Risk Findings (1)

[H-01] `veCVXStrategy.manualRebalance` has wrong logic

Submitted by cmichel, also found by tabish

The `veCVXStrategy.manualRebalance` function computes two ratios `currentLockRatio` and `newLockRatio` and compares them.

However, these ratios compute different things and are not comparable:

- `currentLockRatio = balanceInLock.mul(10**18).div(totalCVXBalance)` is a **percentage value** with 18 decimals (i.e. `1e18 = 100%`). Its max value can at most be `1e18`.
- `newLockRatio = totalCVXBalance.mul(toLock).div(MAX_BPS)` is a **CVX token amount**. It's unbounded and just depends on the `totalCVXBalance` amount.

The comparison that follows does not make sense:

```
if (newLockRatio <= currentLockRatio) {
  // ...
}
```

Impact

The rebalancing is broken and does not correctly rebalance. It usually leads to locking nearly everything if `totalCVXBalance` is high.

Recommended Mitigation Steps

Judging from the `cvxToLock = newLockRatio.sub(currentLockRatio)` it seems the desired computation is that the “ratios” should actually be in CVX amounts and not in percentages. Therefore, `currentLockRatio` should just be `balanceInLock`. (The variables should be renamed as they aren’t really ratios but absolute CVX balance amounts.)

GalloDaSballo (BadgerDAO) acknowledged:

> Agree with the finding, the math is wrong

GalloDaSballo (BadgerDAO) confirmed:

> We will mitigate by deleting the function and using `manualRebalance` and `manualSendbCVXToVault` as way to manually rebalance

GalloDaSballo (BadgerDAO) patched:

> We mitigated by rewriting the code for `manualRebalance`

Medium Risk Findings (2)

[M-01] `SettV3.transferFrom` block lock can be circumvented

Submitted by cmichel

The `SettV3.transferFrom` implements a `_blockLocked` call to prevent users to call several functions at once, for example, `deposit` and then `transfer`ring the tokens.

```
function _blockLocked() internal view {
    require(blockLock[msg.sender] < block.number, "blockLocked")
}
```

However, as the block lock only checks `msg.sender`, an attacker can circumvent it using `transferFrom`:

- Attacker owns accounts `A`, `B` and `C`
- A deposits, `_lockForBlock(msg.sender) = _lockForBlock(A)` is called and `A` is locked.
- A approves B.
- B calls `transferFrom(from=A, to=C, amount)`. This passes the `_blockLocked() = _blockLocked(B)` check.
- C calls `withdraw()`.

Impact

The protection desired from the `_blockLocked` call does not work for this function. I assume the call is used to prevent flashloan attacks, but an attacker can bypass the protection on `transferFrom`.

Recommended Mitigation Steps

The block lock should be on the account that holds the tokens, i.e., on `sender` ("from" address), not on `msg.sender`. Parameterize `_blockLocked` to take an account parameter instead.

GalloDaSballo (BadgerDAO) acknowledged:

> The finding is correct, blockLock is ineffective

> The Recommended MItigation Steps proposes a DOS vector (I'll deposit 1 wei for you so you don't get to withdraw your funds)

[M-02] `CvxLocker.setBoost` wrong validation

Submitted by cmichel

The `CvxLocker.setBoost` function does not validate the `_max, _rate` parameters, instead it validates the already set **storage** variables.

```
// @audit this is checking the already-set storage variables, nc
require(maximumBoostPayment < 1500, "over max payment"); //max 1
require(boostRate < 30000, "over max rate"); //max 3x
```

Impact

Once wrong boost values are set (which are not validated when they are set), they cannot be set to new values anymore, breaking core contract functionality.

Recommended Mitigation Steps

Implement these two checks instead:

```
require(_max < 1500, "over max payment"); //max 15%
require(_rate < 30000, "over max rate"); //max 3x
```

GalloDaSballo (BadgerDAO) acknowledged:

> This can be an issue as our strat takes those variables at face value

GalloDaSballo (BadgerDAO) confirmed:

> As Badger we ended up upgrading the strategy to check for the validity of the inputs

C2tP-C2tP (BadgerDAO) commented:

> for convex side, we can layer on an admin contract that has the correct checks

Low Risk Findings (19)

[L-01] add zero address validation in constructor and initializer

Submitted by JMukesh

Impact

parameter used in constructor and initilizer are used to initialize the state variable, error in these can lead to redeployment of contract

Proof of Concept

- `veCVXStrategy.sol` L67
- `Controller.sol` L35

Tools Used

manual review

Recommended Mitigation Steps

add `address(0)` validation

GalloDaSballo (BadgerDAO) acknowledged:

> Agree on the issue and severity To be fair if we put a wrong value we just redeploy

[L-02] `CvxLocker.setStakeLimits` missing validation

Submitted by cmichel

The `CvxLocker.setStakeLimits` function does not check `_minimum <= _maximum`.

Recommended Mitigation Steps

Implement these two checks instead:

```
    require(_minimum <= _maximum, "min range");
    require(_maximum <= denominator, "max range");
```

[C2tP-C2tP (BadgerDAO) confirmed](#):

> will add an extra layer to call admin functions with fixed checks

[L-03] `CvxLocker.setApprovals` can be called by anyone

Submitted by cmichel

The `CvxLocker.setApprovals` function is callable by anyone, not only by the owner/admin.

Impact

It's okay for this function to be callable by anyone.

Recommended Mitigation Steps

Remove the comment that these are admin-only functions (`ADMIN CONFIGURATION` section) as this is not true for `setApprovals` and one does not know if it's intended to be admin-only or not.

[C2tP-C2tP (BadgerDAO) acknowledged](#)

[L-04] `CvxLocker.findEpochId` stops after 128 iterations

Submitted by cmichel

It's unclear why the binary search algorithm stops after `128` iterations.

Impact

It might not return the actual epoch id (although very unlikely as 128 iterations should be sufficient.)

Recommended Mitigation Steps

Use a `while (max > min)` loop instead of the `for` loop with a fixed number of iterations.

`uint256 mid = (min + max + 1) / 2;` is also not using safe math.

C2tP-C2tP (BadgerDAO) acknowledged

[L-05] Unbounded iteration in `CvxLocker.updateReward`

Submitted by cmichel

The `CvxLocker.updateReward` iterates over all `rewardTokens`.

Impact

The transactions can fail if the arrays get too big and the transaction would consume more gas than the block limit. This will then result in a denial of service for the desired functionality and break core functionality.

Recommended Mitigation Steps

Keep the number of `rewardTokens` small.

C2tP-C2tP (BadgerDAO) acknowledged

[L-06] Missing slippage/min-return check in `veCVXStrategy`

Submitted by cmichel, also found by tabish

The contracts are missing slippage checks which can lead to being vulnerable to sandwich attacks.

> A common attack in DeFi is the sandwich attack. Upon observing a trade of asset X for asset Y, an attacker frontruns the victim trade by also buying asset Y, lets the

> victim execute the trade, and then backruns (executes after) the victim by trading back the amount gained in the first trade. Intuitively, one uses the knowledge that someone's going to buy an asset, and that this trade will increase its price, to make a profit. The attacker's plan is to buy this asset cheap, let the victim buy at an increased price, and then sell the received amount again at a higher price afterwards.

See `veCVXStrategy._swapcvxCRVToWant`:

```
IUniswapRouterV2(SUSHI_ROUTER).swapExactTokensForTokens(
    toSwap,
    0, // @audit min. return of zero, no slippage check
    path,
    address(this),
    now
);
```

Impact

Trades can happen at a bad price and lead to receiving fewer tokens than at a fair market price. The attacker's profit is the protocol's loss.

Recommended Mitigation Steps

Add minimum return amount checks.

Accept a function parameter that can be chosen by the transaction sender, then check that the actually received amount is above this parameter.

Alternatively, check if it's feasible to send these transactions directly to a miner such that they are not visible in the public mempool.

GalloDaSballo (BadgerDAO) acknowledged:

> Not sure if this would be Medium in other bug bounties That said, yeah we use Flashbots to avoid being frontrun and yes the 0 for minOut can cause problems

ghoul-sol (judge) commented:

> Considering sponsor reply, this is low risk.

[L-07] Missing slippage/min-return check in `StrategyCvxHelper`

Submitted by cmichel

The contracts are missing slippage checks which can lead to being vulnerable to sandwich attacks.

> A common attack in DeFi is the sandwich attack. Upon observing a trade of asset X for asset Y, an attacker frontruns the victim trade by also buying asset Y, lets the victim execute the trade, and then backruns (executes after) the victim by trading back the amount gained in the first trade. Intuitively, one uses the knowledge that someone's going to buy an asset, and that this trade will increase its price, to make a profit. The attacker's plan is to buy this asset cheap, let the victim buy at an increased price, and then sell the received amount again at a higher price afterwards.

See `StrategyCvxHelper.harvest`:

```
_swapExactTokensForTokens(sushiswap, cvxCrv, cvxCrvBalance, get1
// @audit calls UniSwapper._swapExactTokensForTokens:
IUniswapRouterV2(router).swapExactTokensForTokens(balance, 0 /*
```

Impact

Trades can happen at a bad price and lead to receiving fewer tokens than at a fair market price. The attacker's profit is the protocol's loss.

Recommended Mitigation Steps

Add minimum return amount checks. Accept a function parameter that can be chosen by the transaction sender, then check that the actually received amount is above this parameter.

Alternatively, check if it's feasible to send these transactions directly to a miner such that they are not visible in the public mempool.

GalloDaSballo (BadgerDAO) acknowledged:

> Same as #57

ghoul-sol (judge) commented:

> Exactly the same problem reported but in a different place of the code. I'll keep it.

[L-08] Missing slippage/min-return check in `BaseStrategy`

Submitted by cmichel

The contracts are missing slippage checks which can lead to being vulnerable to sandwich attacks.

> A common attack in DeFi is the sandwich attack. Upon observing a trade of asset X for asset Y, an attacker frontruns the victim trade by also buying asset Y, lets the victim execute the trade, and then backruns (executes after) the victim by trading back the amount gained in the first trade. Intuitively, one uses the knowledge that someone's going to buy an asset, and that this trade will increase its price, to make a profit. The attacker's plan is to buy this asset cheap, let the victim buy at an increased price, and then sell the received amount again at a higher price afterwards.

See `BaseStrategy._swap`:

```
IUniswapRouterV2(uniswap).swapExactTokensForTokens(balance, 0 /*
```

Impact

Trades can happen at a bad price and lead to receiving fewer tokens than at a fair market price. The attacker's profit is the protocol's loss.

Recommended Mitigation Steps

Add minimum return amount checks. Accept a function parameter that can be chosen by the transaction sender, then check that the actually received amount is above this parameter.

Alternatively, check if it's feasible to send these transactions directly to a miner such that they are not visible in the public mempool.

GalloDaSballo (BadgerDAO) acknowledged:

> Same as #56

ghoul-sol (judge) commented:

> Same issue, different place. Keeping as is with low risk.

[L-09] StrategyCvxHelper: `safeApprove` instead of `approve`

Submitted by hickuphh3

This was probably an oversight since

- the veCVXStrategy contract used `safeApprove()` for token approvals
- `using SafeERC20Upgradeable for IERC20Upgradeable;` was declared

Recommended Mitigation Steps

Change

```
cvxToken.approve(address(cvxRewardsPool), MAX_UINT_256);
```

to

```
cvxToken.safeApprove(address(cvxRewardsPool), MAX_UINT_256);
```

GalloDaSballo (BadgerDAO) confirmed:

> Agree with finding

GalloDaSballo (BadgerDAO) patched:

> We ended up using safeApprove as suggested

[L-10] Swap conversion is susceptible to MEV flashbots

Submitted by hickuphh3

Impact

In `veCVXStrategy`, the `cvxCRV -> ETH -> CVX` conversion via sushiswap is done with 0 `minAmountOut`, making it susceptible to sandwich attacks / MEV flashbots. This is also true for `UniSwapper` inherited by `StrategyCvxHelper`.

Recommended Mitigation Steps

1. `veCVXStrategy`

   Ideally, the `harvest()` function should take in a `minAmountOut` parameter, but this breaks the Yearn architecture used. Using TWAPs / price oracles might alleviate the problem, but results in higher gas usage, and with multiple hops involved, may not be feasible.

   A simpler approach would be to have a configurable storage variable `minAmountOut`. Its value can then be adjusted such that harvesting can be done infrequently to save gas.

2. `UniSwapper`

   Ideally, each path registered in the `TokenSwapPathRegistry` should also have a `minAmount` mapping, that can be fetched together with the path.

GalloDaSballo (BadgerDAO) acknowledged:

> Agree with finding, we use private txs for harvests

[L-11] veCVXStrategy: Sub-optimal trading path

Submitted by hickuphh3

Impact

`_swapcvxCRVToWant()` swaps `cvxCRV -> ETH -> CVX` via sushiswap.

Looking at sushiswap analytics, this may also not be the most optimal trading path. The cvxCRV-CRV pool seems to have substantially better liquidity than the cvxCRV-ETH pool as reported here (Note that cvxCRV-CRV's liquidity is overstated, clicking into the pool gives a more reasonable amount). It is therefore better to do `cvxCRV -> CRV -> ETH -> CVX`, though this comes at the cost of higher gas usage.

Recommended Mitigation Steps

Switch the trading path to `cvxCRV -> CRV -> ETH -> CVX`, as it means more CVX tokens received, translating to higher APY, while the higher gas cost is borne by the caller.

Additionally, given how liquidity can shift between pools over time, the most optimal trade path may change accordingly. Hence, it may be beneficial to make the pool path configurable.

GalloDaSballo (BadgerDAO) confirmed:

> I love that you caught this Yeah we ended up using curve

[L-12] Functions not returning declared values

Submitted by pauliax

Impact

function `withdrawAll` in `BaseStrategy` declares 'returns (uint256 balance)', however, no actual value is returned. function reinvest in `MyStrategy` declares to return 'uint256 reinvested', however, it also actually does not return anything so they always get assigned a default value of 0.

Recommended Mitigation Steps

Either remove the return declarations or return the intended values. Otherwise, it may confuse other protocols that later may want to integrate with you.

GalloDaSballo (BadgerDAO) confirmed:

> We should remove the return values

[L-13] Frontrunning `distribute` functions

Submitted by pauliax

Impact

`distribute` and `distributeOther` can be frontrunned. Anyone can call these functions and receive `callIncentive`. A frontrunner can watch the mempool and copy the calldata to replicate the same tx. For example, a frontrunner calculates that replicating this tx will result in profit, he watches and copies the tx, then instantly sell these received incentives on AMM for profit. This may result in reverted txs, gas wasted, and a poor experience for legit users.

Recommended Mitigation Steps

This problem seems insurmountable in this case but you may want to consider adding restrictions on the callers or introducing any other possible prevention techniques.

C2tP-C2tP (BadgerDAO) acknowledged

[L-14] Faulty return value in `veCVXStrategy::reinvest()`

Submitted by pmerkleplant, also found by tabish

Impact

The function `reinvest` in the veCVXStrategy always returns 0 as the return variable `reinvested` is never updated. The function is `onlyGovernance` and the return value probably does not matter if the caller is a multi-sig. However, if a protocol is set as `onlyGovernance` the faulty return value would have to be ignored by the caller to not transition into an incorrect state.

Proof of Concept

The variable `reinvested` is declared as return variable (line 400) but not updated to reflect the actual amount reinvested which is saved in variable `toDeposit`.

Therefore always the default value is returned (0).

Recommended Mitigation Steps

Add `reinvested = toDeposit;` after line 412.

GalloDaSballo (BadgerDAO) confirmed:

> Agreed, we should just delete the return value as we don't need it

[L-15] `setKeepReward` function is unfinished

Impact

The `setKeepReward` function is unfinished.

Proof of Concept

`veCVXStrategy.sol` L203

Recommended Mitigation Steps

Either complete the function or follow the comment above the code and remove it.

GalloDaSballo (BadgerDAO) confirmed:

> Agree, should delete

[L-16] Don't include unused functions

Impact

The code includes unused functions, like `tend()`, L319. It's best practice to remove these. It will also save gas.

Recommended Mitigation Steps

Remove the unused function.

GalloDaSballo (BadgerDAO) confirmed:

> Agree

[L-17] Reentrancy on `distributeOther()`

Impact

The `distribute` function can be re-entered by fake tokens or tokens with callbacks. An attacker can use the callbacks on `safeTransfer` if a token has a callback to reenter an drain the entire balance of that particular token before the `notifyRewardAmount` is called.

I think this is only a medium issue because the attacker can only take tokens with callbacks, and I don't think any of the tokens you guys use have callbacks.

Proof of Concept

`CvxStakingProxy.sol` L153

Recommended Mitigation Steps

Be aware of this possibility. If you really want to, add a `nonReentrant` modifier to the function.

GalloDaSballo (BadgerDAO) acknowledged:

> Not my issue to handle, but should be low because it's dependent on the token, also you already know cvxCRV and CVX are ERC20 and don't have custom hooks on transfer

ghoul-sol (judge) commented:

> as per warden description this is unlikely to happen so making this low risk

[L-18] `ManualRebalance` will be frontrun for most of the tokens.

Impact

We have previously seen that the `harvest` function can be exploited for almost all the tokens at stake. Since `ManualRebalance` calls `harvest`, it is also unsafe and funds swapped using it will likely be lost.

Proof of Concept

`sol#L444` L453

Recommended Mitigation Steps

Adding an amount out minimum here will work that should be passed on to the `harvest` method.

GalloDaSballo (BadgerDAO) confirmed:

> Disagree with risk (should be medium like all other harvest findings), also we have optional harvest which means we can skip it, hence the finding is deceiving at best

GalloDaSballo (BadgerDAO) commented:

> We use private txs to mitigate

ghoul-sol (judge) commented:

> similar to #55 #56 and other front-running issues in this contest, it's low risk given the sponsor comments.

Non-Critical Findings (13)

- [N-01] use of floating pragma *Submitted by JMukesh*
- [N-02] lack of emission of events while setting fees *Submitted by JMukesh*
- [N-03] Unused event `StrategyCvxHelper.HarvestState` *Submitted by cmichel*
- [N-04] Unused event `StrategyCvxHelper.TendState` *Submitted by cmichel*
- [N-05] Unused event `veCVXStrategy.Debug` *Submitted by cmichel*
- [N-06] Unused event `veCVXStrategy.TreeDistribution` *Submitted by cmichel*

- [N-07] veCVXStrategy: Erroneous Comments *Submitted by hickuphh3*
- [N-08] Validations *Submitted by pauliax*
- [N-09] Order of parameters in KickReward event *Submitted by pauliax*
- [N-10] events in BaseStrategy are never emitted *Submitted by pauliax*
- [N-11] The comments incorrectly indicate the range in which `toLock` input should be given. *Submitted by tabish*
- [N-12] toLock in the comments is a % while in the code it is measured in bips.
- [N-13] Refactor code to use calculations at current epoch

Gas Optimizations (17)

- [G-01] state variable that are not changed throughout the contract should be declared as constant *Submitted by JMukesh*
- [G-02] Gas: `_onlyNotProtectedTokens` should use maps *Submitted by cmichel*
- [G-03] StrategyCvxHelper: Redundant re-initialisation of path array *Submitted by hickuphh3*
- [G-04] veCVXStrategy: Extra functions can be external instead of public *Submitted by hickuphh3*
- [G-05] veCVXStrategy: Redundancies *Submitted by hickuphh3*
- [G-06] veCVXStrategy: Unused return outputs from _processRewardsFees() *Submitted by hickuphh3*
- [G-07] Declare CvxLocker erc20 contract variables as immutable *Submitted by patitonar*
- [G-08] _processPerformanceFees is useless now *Submitted by pauliax*
- [G-09] Use cached _ethBalance *Submitted by pauliax*
- [G-10] Immutable variables *Submitted by pauliax*
- [G-11] tend() can be simplified *Submitted by pauliax*
- [G-12] lpComponent is useless *Submitted by pauliax*
- [G-13] public functions that can be external *Submitted by pauliax*
- [G-14] Calculation of valueInLocker *Submitted by pauliax*
- [G-15] Delete function `setKeepReward` *Submitted by pauliax*

Disclosures

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

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

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