Badger-Vested-Aura contest Findings & Analysis Report

2022-08-02

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

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

During the audit contest outlined in this document, C4 conducted an analysis of the Badger-Vested-Aura smart contract system written in Solidity. The audit contest took place between June 15—June 18 2022.

Wardens

56 Wardens contributed reports to the Badger-Vested-Aura contest:

1. unforgiven
2. GimelSec (rayn and sces60107)
3. zzzitron
4. berndartmueller
5. kirk-baird
6. minhquanym
7. rfa
8. tabish
9. PumpkingWok
10. IllIllI
11. sorrynotsorry
12. cccz

13. kenzo
14. scaraven
15. 0xKitsune
16. dipp
17. defsec
18. reassor
19. Tadashi
20. oyc_109
21. Chom
22. c3phas
23. robee
24. 0xNazgul
25. Picodes
26. 0x1f8b
27. hyh
28. Meera
29. joestakey
30. TerrierLover
31. 0xFar5eer
32. codexploder
33. Waze
34. _Adam
35. gzeon
36. Funen
37. 0xNineDec
38. a12jmx
39. saian
40. 242
41. asutorufos

42. cryptphi
43. Czar102
44. Sm4rty
45. Ov3rf10w
46. 0xDjango
47. 0x52
48. georgypetrov
49. 0xkatana
50. fatherOfBlocks
51. simon135
52. MiloTruck
53. TomJ
54. JC
55. sach1r0

This contest was judged by Jack the Pug.

Final report assembled by itsmetechjay.

Summary

The C4 analysis yielded an aggregated total of 5 unique vulnerabilities. Of these vulnerabilities, 2 received a risk rating in the category of HIGH severity and 3 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 35 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 28 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the C4 Badger-Vested-Aura contest repository, and is composed of 1 smart contracts written in the Solidity programming language and includes 440 lines of Solidity code.

Severity Criteria

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

Vulnerabilities are divided into three primary risk categories: high, medium, and low/non-critical.

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

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

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

High Risk Findings (2)

[H-01] Attacker can call sweepRewardToken() when `bribesProcessor==0` and reward funds will be lost because there is no check in sweepRewardToken() and _handleRewardTransfer() and _sendTokenToBribesProcessor()

Submitted by unforgiven, also found by GimelSec, and zzzitron

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L107-L113 https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L405-L413

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L421-L425

Impact

If the value of `bribesProcessor` was `0x0` (the default is `0x0` and `governance()` can set to `0x0`) then attacker can call `sweepRewardToken()` make contract to send his total balance in attacker specified token to `0x0` address.

Proof of Concept

The default value of `bribesProcessor` is `0x0` and `governance` can set the value to `0x0` at any time. Rewards are stacking in contract address and they are supposed to send to `bribesProcessor`.

This is `sweepRewardToken()` and `_handleRewardTransfer()` and `_sendTokenToBribesProcessor()` code:

```
    /// @dev Function to move rewards that are not protected
    /// @notice Only not protected, moves the whole amount using _
    /// @notice because token paths are hardcoded, this function i
    /// @notice Will not notify the BRIBES_PROCESSOR as this coulc
    function sweepRewardToken(address token) public nonReentrant {
        _onlyGovernanceOrStrategist();
        _onlyNotProtectedTokens(token);

        uint256 toSend = IERC20Upgradeable(token).balanceOf(addres
        _handleRewardTransfer(token, toSend);
    }

    function _handleRewardTransfer(address token, uint256 amount)
        // NOTE: BADGER is emitted through the tree
        if (token == BADGER) {
            _sendBadgerToTree(amount);
        } else {
            // NOTE: All other tokens are sent to bribes processor
            _sendTokenToBribesProcessor(token, amount);
        }
    }

    function _sendTokenToBribesProcessor(address token, uint256 am
        // TODO: Too many SLOADs
        IERC20Upgradeable(token).safeTransfer(address(bribesProces
        emit RewardsCollected(token, amount);
    }
```

As you can see calling `sweepRewardToken()` eventually (`sweepRewardToken() -> _handleRewardTransfer() -> _sendTokenToBribesProcessor()`) would transfer reward funds to `bribesProcessor` and there is no check that `bribesProcessor!=0x0` in execution follow. so attacker can call `sweepRewardToken()` when `bribesProcessor` is `0x0` and contract will lose all reward tokens.

Tools Used

VIM

Recommended Mitigation Steps

Check the value of `bribesProcessor` in `_sendTokenToBribesProcessor()` .

Alex the Entreprenerd (BadgerDAO) confirmed and commented:

> A transfer to address 0 would cause a loss, we should have a check or add a safe default (governance for example).

> Mitigated by adding a 0 check.

jack-the-pug (judge) validated

[H-02] auraBAL can be stuck into the Strategy contract

Submitted by PumpkingWok, also found by kirk-baird, rfa, tabish, and unforgiven https://github.com/Badger-Finance/vested-aura/blob/v0.0.2/contracts/MyStrategy.sol#L220-L228

https://github.com/Badger-Finance/vested-aura/blob/v0.0.2/contracts/MyStrategy.sol#L288

Impact

The internal `_harvest()` function defined is responsible to claim auraBAL from the aura locker and within the function it swaps them to auraBAL -> BAL/ETH BPT -> WETH -> AURA, finally it locks AURA to the locker to increase the position. For claiming auraBAL it calls `LOCKER.getReward(address(this))` and it calculates the tokes earned, checking the balance before and after the claiming.

The function to get the rewards is public and any address can call it for the strategy address, and it will transfer all rewards tokens to the strategy, but in this scenario the auraBAL will remain in stuck into the contract, because they won't be counted as auraBAL earned during the next `_harvest()` . Also they could not sweep because auraBAL is a protected token.

Also, the aura Locker will be able to add other token as reward apart of auraBAL, but the harvest function won't be able to manage them, so they will need to be sweep every time.

The same scenario can happen during the `claimBribesFromHiddenHand()` call, the `IRewardDistributor.Claim[] calldata _claims` pass as input parameters could be frontrunned, and another address can call the `hiddenHandDistributor.claim(_claims)` (except for ETH rewards) for the strategy address, and like during the `_harvest()` only the tokens received during the call will be counted as earned. However every token, except auraBAL can be sweep, but the `_notifyBribesProcessor()` may never be called.

Proof of Concept

At every `_harvest()` it checks the balance before the claim and after, to calculate the auraBAL earned, so every auraBAL transferred to the strategy address not during this call, won't be swapped to AURA.

Recommended Mitigation Steps

Instead of calculating the balance before and after the claim, for both `harvest≠ and` claimBribesFromHiddenHand()`, the whole balance could be taken, directly after the claim.

[Alex the Entreprenerd (BadgerDAO) confirmed and commented]:

> Mitigated by refactoring from a delta of balance to absolute balances

Medium Risk Findings (3)

[M-01] `_harvest` has no slippage protection when swapping `auraBAL` for `AURA`

*Submitted by Picodes, also found by 0x1f8b, 0x52, berndartmueller, cccz, Chom, defsec, georgypetrov, GimelSec, hyh, IllIllI, kenzo, minhquanym, oyc*109, scaraven, and unforgiven_

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L249

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L275

Impact

Single swaps of `_harvest` contains no slippage or deadline, which makes it vulnerable to sandwich attacks, MEV exploits and may lead to significant loss of yield.

Proof of Concept

When using `BALANCER_VAULT.swap` here and here, there is no slippage protection. Therefore a call to `_harvest` generating swaps could be exploited for sandwich attacks or other MEV exploits such as JIT.

The scenario would be: A authorized actor calls `harvest`, leading to a swap of say x `auraBAL` to `BAL/ETH BPT` and then y `WETH` to `BAL`.

Then while the transaction is in the mempool, it is exploited for example like in https://medium.com/coinmonks/defi-sandwich-attack-explain-776f6f43b2fd

Recommended Mitigation Steps

The easiest mitigation would be to pass a minimum amount of `AURA` that the swap is supposed to get in `harvest`. It should not add security issues as callers of `harvest` are trusted.

Another solution would be to do like here to use Cowswap for example, or any other aggregator.

Alex the Entreprenerd (BadgerDAO) commented:

> I love how the warden linked my code to integrate cowswap XD

jack-the-pug (judge) validated

Alex the Entreprenerd (BadgerDAO) confirmed and commented:

> Confirmed and mitigated in 2 ways:
>
> - We do use Private Transactions to Harvest (reduce change of front-run can still be sandwiched).
> - We Refactored to have a slippage tollerance

[M-02] Badger rewards from Hidden Hand can permanently prevent Strategy from receiving bribes

Submitted by scaraven, also found by berndartmueller, cccz, dipp, GimelSec, kenzo, kirk-baird, and unforgiven

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrate gy.sol#L428-L430

https://github.com/Badger-Finance/badger-vaults-1.5/blob/3c96bd83e9400671256b235422f63644f1ae3d2a/contracts/BaseStrategy.sol#L351

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L407-L408

Impact

If the contract receives rewards from the hidden hand marketplace in BADGER then the contract tries to transfer the same amount of tokens twice to two different accounts, once with `_sendBadgerToTree()` in `MyStrategy` and again with `_processExtraToken()` in the `BasicStrategy` contract. As it is very likely that the strategy will not start with any BADGER tokens, the second transfer will revert (as we are using safeTransfer). This means that `claimBribesFromHiddenHand()` will always revert preventing any other bribes from being received.

Proof of Concept

1. `claimBribesFromHiddenHand()` is called by strategist
2. Multiple bribes are sent to the strategy including BADGER. For example lets say 50 USDT And 50 BADGER
3. Strategy receives BADGER and calls `_handleRewardTransfer()` which calls `_sendBadgerToTree()`. 50 BADGER is sent to the Badger Tree so balance has dropped to 0.
4. 50 Badger is then again sent to Vault however balance is 0 so the command fails and reverts
5. No more tokens can be claimed anymore

Tools Used

VS Code

Recommended Mitigation Steps

`_processExtraToken()` eventually sends the badger to the badger tree through the `Vault` contract. Change

```
        function _sendBadgerToTree(uint256 amount) internal {
            IERC20Upgradeable(BADGER).safeTransfer(BADGER_TREE, amou
            _processExtraToken(address(BADGER), amount);
        }
```

to

```
        function _sendBadgerToTree(uint256 amount) internal {
            _processExtraToken(address(BADGER), amount);
        }
```

[Alex the Entreprenerd (BadgerDAO) confirmed and commented]():

> Developer oversight yeah.

[shuklaayush (BadgerDAO) commented]():

> Yeah, badger bribes can't be claimed. Not sure if I'll call it high risk but definitely an oversight.

[jack-the-pug (judge) validated and decreased severity to Medium]()

[Alex the Entreprenerd (BadgerDAO) commented]():

> We mitigated by fixing the mistake.

[M-03] Withdrawing all funds at once to vault can be DoS attacked by frontrunning and locking dust

Submitted by berndartmueller, also found by minhquanym

All funds can be migrated (withdrawn) at once to the caller vault by using the `BaseStrategy.withdrawToVault` function which internally calls

`MyStrategy._withdrawAll`.

The latter function has the following check in place:

[MyStrategy.sol#L184-L187]

```
    require(
        balanceOfPool() == 0 && LOCKER.balanceOf(address(this)) == (
        "You have to wait for unlock or have to manually rebalance c
    );
```

Funds can only be withdrawn (migrated) if the balance in `LOCKER` is fully unlocked.

By locking a small amount of want tokens via `AuraLocker.lock` with the `strategy` address, a malicious individual can cause DoS and prevent withdrawing and migrating funds to the vault.

Proof of Concept

The following test case will replicate the DoS attack by locking "dust" want tokens for the `strategy` address. This causes `vault.withdrawToVault` to revert.

```
    def test_frontrun_migration(locker, deployer, vault, strategy, w
        # Setup
        randomUser = accounts[6]
        snap = SnapshotManager(vault, strategy, "StrategySnapshot")

        startingBalance = want.balanceOf(deployer)
        depositAmount = startingBalance // 2
        assert startingBalance >= depositAmount
        # End Setup

        # Deposit
        want.approve(vault, MaxUint256, {"from": deployer})
        snap.settDeposit(depositAmount, {"from": deployer})

        chain.sleep(15)
        chain.mine()

        vault.earn({"from": keeper})
        chain.snapshot()

        # Test no harvests
        chain.sleep(86400 * 250)  ## Wait 250 days so we can withdra
        chain.mine()

        before = {"settWant": want.balanceOf(vault), "stratWant": st

        strategy.prepareWithdrawAll({"from": governance})

        want.approve(locker, 1, {"from": deployer})
        locker.lock(strategy, 1, { "from": deployer }) # Donate "dus

        vault.withdrawToVault({"from": governance}) # @audit-info re

        after = {"settWant": want.balanceOf(vault), "stratWant": str

        assert after["settWant"] > before["settWant"]
        assert after["stratWant"] < before["stratWant"]
        assert after["stratWant"] == 0
```

Recommended Mitigation Steps

Call `LOCKER.processExpiredLocks(false);` in `MyStrategy._withdrawAll` directly and remove the check which enforces unlocking all want tokens on L184-L187.

Alex the Entreprenerd (BadgerDAO) confirmed and commented:

> I have to agree with the evidence that `_withdrawAll` will be ineffective.

> The implications are that no strategy migration is possible for this set of Vault <-> Strategy as even 1 wei would cause the `setStrategy` to fail.

> In terms of impact, ultimately the warden didn't show how withdrawals would be denied nor broken, end users can always withdraw via `_withdraw` meaning that the vault would still allow user withdrawals but governance would be unable to move tokens away from the strategy.

Alex the Entreprenerd (BadgerDAO) disagreed with severity and commented:

> I have to correct myself, we actually have a way to send all tokens back to the vault in `manualSendAuraToVault`

> I want to commend the warden for finding an interesting find, however I believe impact is further reduced as long as we accept that the strategy will not be changeable.

jack-the-pug (judge) validated and decreased severity to Medium

Alex the Entreprenerd (BadgerDAO) commented:

> Acknowledged, I believe this effectively means that we won't be able to replace the locking strategy. In the future we may end up using lockingProxies (separate contract just for locking) although that may create further trust issues.

> End users can still withdraw their tokens at any time, however the finding confirms that if we ever want to do a "bveAURA V2", we'll need to deploy a new Vault.

Low Risk and Non-Critical Issues

For this contest, 35 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by IllIllI received the top score from the judge.

The following wardens also submitted reports: sorrynotsorry, Tadashi, unforgiven, 0xNazgul, defsec, reassor, oyc_109, Chom, Meera, robee, joestakey, codexploder, minhquanym, 0xFar5eer, 0xNineDec, a12jmx, hyh, saian, zzzitron, 242, asutorufos, cryptphi, Czar102, Funen, GimelSec, gzeon, Picodes, Sm4rty, TerrierLover, Waze, _Adam, 0v3rf10w, 0x1f8b, *and* 0xDjango.

Summary

Low Risk Issues

	Issue	Instances
L-	`require()` should be used instead of `assert()`	1

	Issue	Instances
01		
L-02	Upgradeable contract is missing a `__gap[50]` storage variable to allow for new storage variables in later versions	1

Total: 2 instances over 2 issues

Non-critical Issues

	Issue	Instances
N-01	`safeApprove()` is deprecated	3
N-02	Open TODOs	2
N-03	Using vulnerable version of OpenZeppelin	1
N-04	Missing `initializer` modifier on constructor	1
N-05	`public` functions not called by the contract should be declared `external` instead	1
N-06	`constant`s should be defined rather than using magic numbers	1
N-07	Redundant cast	1
N-08	Missing event and timelock for critical parameter change	3
N-09	Use a more recent version of solidity	1
N-10	Inconsistent spacing in comments	4
N-11	Typos	4
N-12	Event is missing `indexed` fields	1

Total: 23 instances over 12 issues

[L-01] `require()` should be used instead of `assert()`

Prior to solidity version 0.8.0, hitting an assert consumes the **remainder of the transaction's available gas** rather than returning it, as `require()` / `revert()` do. `assert()` should be avoided even past solidity version 0.8.0 as its documentation states that "The assert function creates an error of type Panic(uint256). ... Properly functioning code should never create a Panic, not even on invalid external input. If this happens, then there is a bug in your contract which you should fix".

There is 1 instance of this issue:

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

57:            assert(IVault(_vault).token() == address(AURA));
```

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L57

[L-02] Upgradeable contract is missing a `__gap[50]` storage variable to allow for new storage variables in later versions

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

There is 1 instance of this issue:

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

20:   contract MyStrategy is BaseStrategy, ReentrancyGuardUpgrac
```

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L20

[N-01] `safeApprove()` is deprecated

Deprecated in favor of `safeIncreaseAllowance()` and `safeDecreaseAllowance()`. If only setting the initial allowance to the value that means infinite, `safeIncreaseAllowance()` can be used instead

There are 3 instances of this issue:

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

65:            AURA.safeApprove(address(LOCKER), type(uint256).ma
```

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L65

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

67:            AURABAL.safeApprove(address(BALANCER_VAULT), type(
```

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L67

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

68:            WETH.safeApprove(address(BALANCER_VAULT), type(uir
```

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L68

[N-02] Open TODOs

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

There are 2 instances of this issue:

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

284:      // TODO: Hardcode claim.account = address(this)?
```

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L284

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

422:          // TODO: Too many SLOADs
```

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L422

[N-03] Using vulnerable version of OpenZeppelin

The brownie configuration file says that the project is using 3.4.0 of OpenZeppelin which [has a vulnerability](#) in initializers that call external contracts, which this code does. You're protecting against it by having the comment stating to change all state at the end, but it would be better to upgrade and use the `onlyInitializing` modifier

There is 1 instance of this issue:

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

55:      /// @dev add any extra changeable variable at end of ir
```

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L55

[N-04] Missing `initializer` modifier on constructor

OpenZeppelin [recommends](#) that the `initializer` modifier be applied to constructors

There is 1 instance of this issue:

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

20:   contract MyStrategy is BaseStrategy, ReentrancyGuardUpgrac
```

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L20

[N-05] `public` functions not called by the contract should be declared `external` instead

Contracts [are allowed](#) to override their parents' functions and change the visibility from `external` to `public`.

There is 1 instance of this issue:

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

56:       function initialize(address _vault) public initializer
```

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L56

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

Even [assembly](#) can benefit from using readable constants instead of hex/numeric literals

There is 1 instance of this issue:

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

/// @audit 9_980
205:                require(max >= _amount.mul(9_980).div(MAX_BPS)
```

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L205

[N-07] Redundant cast

The type of the variable is the same as the type to which the variable is being cast

There is 1 instance of this issue:

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

/// @audit address(BADGER)
430:            _processExtraToken(address(BADGER), amount);
```

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L430

[N-08] Missing event and timelock for critical parameter change

Events help non-contract tools to track changes, and events prevent users from being surprised by changes

There are 3 instances of this issue:

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

86        function setWithdrawalSafetyCheck(bool newWithdrawalSa
87            _onlyGovernance();
88            withdrawalSafetyCheck = newWithdrawalSafetyCheck;
89:        }
```

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L86-L89

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

92         function setProcessLocksOnReinvest(bool newProcessLock
93             _onlyGovernance();
94             processLocksOnReinvest = newProcessLocksOnReinvest
95:        }
```

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L92-L95

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

98         function setBribesProcessor(IBribesProcessor newBribes
99             _onlyGovernance();
100            bribesProcessor = newBribesProcessor;
101:       }
```

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L98-L101

[N-09] Use a more recent version of solidity

Use a solidity version of at least 0.8.13 to get the ability to use `using for` with a list of free functions

There is 1 instance of this issue:

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

```
3:    pragma solidity 0.6.12;
```

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L3

[N-10] Inconsistent spacing in comments

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

There are 4 instances of this issue:

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

85:        ///@dev Should we check if the amount requested is mor
```

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L85

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

91:        ///@dev Should we processExpiredLocks during reinvest?
```

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L91

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

97:         ///@dev Change the contract that handles bribes
```

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrate [gy.sol#L97](gy.sol#L97)

```
File: contracts/MyStrategy.sol   #4

183:          //NOTE: This probably will always fail unless we h
```

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L183

[N-11] Typos

There are 4 instances of this issue:

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

/// @audit hardcoded
105:      /// @notice because token paths are hardcoded, this fu
```

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L105

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

/// @audit sweeped
160:      /// @notice this provides security guarantees to the c
```

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L160

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

/// @audit compunded
218:      ///      after claiming rewards or swapping are auto-c
```

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L218

```
File: contracts/MyStrategy.sol   #4

/// @audit Hardcode
284:        // TODO: Hardcode claim.account = address(this)?
```

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L284

[N-12] Event is missing `indexed` fields

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

There is 1 instance of this issue:

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

51:         event RewardsCollected(address token, uint256 amount);
```

https://github.com/Badger-Finance/vested-aura/blob/d504684e4f9b56660a9e6c6dfb839dcebac3c174/contracts/MyStrategy.sol#L51

Alex the Entreprenerd (BadgerDAO) acknowledged

> (Note: See original submission for sponsor's full commentary.)

IllIllI000 (warden) commented:

> @Alex the Entreprenerd Ser, there is no constructor defined in this contract therefore the default one is used, where the initializer modified is not being used

Alex the Entreprenerd (BadgerDAO) commented:

> @IIIIIIIOOO I've looked into it and had I agree with the finding, would recommend rephrasing to: Implementation contract may not be initialized. Per OZs Post implementation contract should be initialized to avoid potential griefs or exploits.

> Personally our UpgradeableProxy doesn't risk being self-destructed, that said if the finding is contextualized in that way I agree with it.

> In terms of the Proxy, we deploy + initialize in the same transaction via `constructor(admin, logic, data)` or similar, meaning initialization will not be front-run on the side of the proxy.

> I have changed my mind about `assert` we should have used `require` and we have changed the code.

jack-the-pug (judge) validated and commented:

> Overall, this is an excellent QA report with top-notch formatting. I love how you put a short and clear description for each issue, with all the instances listed.

Gas Optimizations

For this contest, 28 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **0xKitsune** received the top score from the judge.

The following wardens also submitted reports: c3phas, IllIIII, reassor, robee, 0xkatana, defsec, fatherOfBlocks, simon135, TerrierLover, 0xFar5eer, Meera, joestakey, MiloTruck, rfa, TomJ, _Adam, 0xNazgul, Waze, JC, Picodes, 0x1f8b, Chom, codexploder, gzeon, Funen, oyc_109, *and* sach1r0.

Summary

The following sections detail the gas optimizations found throughout the codebase. Each optimization is documented with the setup, an explainer for the optimization, a gas report and line identifiers for each optimization across the codebase.

For each section's gas report, the optimizer was turned on and set to 10000 runs. You can replicate any tests/gas reports by heading to 0xKitsune/gas-lab and cloning the repo. Then, simply copy/paste the contract examples from any section and run `forge test --gas-report`. You can also easily update the optimizer runs in the `foundry.toml`.

[G-01] Use assembly to write storage values

```
contract GasTest is DSTest {
    Contract0 c0;
    Contract1 c1;

    function setUp() public {
        c0 = new Contract0();
        c1 = new Contract1();
    }

    function testGas() public {
        c0.updateOwner(0x158B28A1b1CB1BE12C6bD8f5a646a0e3B202473
        c1.assemblyUpdateOwner(0x158B28A1b1CB1BE12C6bD8f5a646a0e
    }
}

contract Contract0 {
    address owner = 0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84;

    function updateOwner(address newOwner) public {
        owner = newOwner;
    }
}

contract Contract1 {
    address owner = 0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84;

    function assemblyUpdateOwner(address newOwner) public {
        assembly {
            sstore(owner.slot, newOwner)
        }
    }
}
```

Gas Report

```
╭─────────────────────┬──────────────────┬──────┬────────┬──────┬
| Contract0 contract  ┆                  ┆      ┆        ┆      
╞═════════════════════╪══════════════════╪══════╪════════╪══════╪
| Deployment Cost     ┆ Deployment Size  ┆      ┆        ┆      
├─────────────────────┼──────────────────┼──────┼────────┼──────┼
| 60623               ┆ 261              ┆      ┆        ┆      
├─────────────────────┼──────────────────┼──────┼────────┼──────┼
| Function Name       ┆ min              ┆ avg  ┆ median ┆ max  
├─────────────────────┼──────────────────┼──────┼────────┼──────┼
| updateOwner         ┆ 5302             ┆ 5302 ┆ 5302   ┆ 5302 
╰─────────────────────┴──────────────────┴──────┴────────┴──────┴

╭─────────────────────┬──────────────────┬──────┬────────┬──────┬
| Contract1 contract  ┆                  ┆      ┆        ┆      
╞═════════════════════╪══════════════════╪══════╪════════╪══════╪
| Deployment Cost     ┆ Deployment Size  ┆      ┆        ┆      
├─────────────────────┼──────────────────┼──────┼────────┼──────┼
| 54823               ┆ 232              ┆      ┆        ┆      
├─────────────────────┼──────────────────┼──────┼────────┼──────┼
| Function Name       ┆ min              ┆ avg  ┆ median ┆ max  
├─────────────────────┼──────────────────┼──────┼────────┼──────┼
| assemblyUpdateOwner ┆ 5236             ┆ 5236 ┆ 5236   ┆ 5236 
╰─────────────────────┴──────────────────┴──────┴────────┴──────┴
```

Lines

- MyStrategy.sol:71
- MyStrategy.sol:88
- MyStrategy.sol:94
- MyStrategy.sol:310
- MyStrategy.sol:312

[G-02] Use assembly when getting a contract's balance of ETH.

You can use `selfbalance()` instead of `address(this).balance` when getting your contract's balance of ETH to save gas. Additionally, you can use `balance(address)` instead of `address.balance()` when getting an external contract's balance of ETH.

```
contract GasTest is DSTest {
    Contract0 c0;
    Contract1 c1;
    Contract2 c2;
    Contract3 c3;

    function setUp() public {
        c0 = new Contract0();
        c1 = new Contract1();
        c2 = new Contract2();
        c3 = new Contract3();
    }

    function testGas() public {
        c0.addressInternalBalance();
        c1.assemblyInternalBalance();
        c2.addressExternalBalance(address(this));
        c3.assemblyExternalBalance(address(this));
    }
}

contract Contract0 {
    function addressInternalBalance() public returns (uint256) {
        return address(this).balance;
    }
}

contract Contract1 {
    function assemblyInternalBalance() public returns (uint256)
        assembly {
            let c := selfbalance()
            mstore(0x00, c)
            return(0x00, 0x20)
        }
    }
}

contract Contract2 {
    function addressExternalBalance(address addr) public {
        uint256 bal = address(addr).balance;
        bal++;
    }
}

contract Contract3 {
    function assemblyExternalBalance(address addr) public {
        uint256 bal;
        assembly {
            bal := balance(addr)
        }
        bal++;
    }
}
```

Gas Report

```
╭──────────────────────────┬─────────────────┬─────┬────────┬────
│ Contract0 contract       ┆                 ┆     ┆        ┆
╞══════════════════════════╪═════════════════╪═════╪════════╪════
│ Deployment Cost          ┆ Deployment Size ┆     ┆        ┆
├╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌┼╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌┼╌╌╌╌╌┼╌╌╌╌╌╌╌╌┼╌╌╌╌
│ 23675                    ┆ 147             ┆     ┆        ┆
├╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌┼╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌┼╌╌╌╌╌┼╌╌╌╌╌╌╌╌┼╌╌╌╌
│ Function Name            ┆ min             ┆ avg ┆ median ┆ m
├╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌┼╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌┼╌╌╌╌╌┼╌╌╌╌╌╌╌╌┼╌╌╌╌
│ addressInternalBalance   ┆ 148             ┆ 148 ┆ 148    ┆ 1
╰──────────────────────────┴─────────────────┴─────┴────────┴────

╭───────────────────────────┬─────────────────┬─────┬────────┬───
│ Contract1 contract        ┆                 ┆     ┆        ┆
╞═══════════════════════════╪═════════════════╪═════╪════════╪═══
│ Deployment Cost           ┆ Deployment Size ┆     ┆        ┆
├╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌┼╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌┼╌╌╌╌╌┼╌╌╌╌╌╌╌╌┼╌╌╌
│ 27081                     ┆ 165             ┆     ┆        ┆
├╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌┼╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌┼╌╌╌╌╌┼╌╌╌╌╌╌╌╌┼╌╌╌
│ Function Name             ┆ min             ┆ avg ┆ median ┆
├╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌┼╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌┼╌╌╌╌╌┼╌╌╌╌╌╌╌╌┼╌╌╌
│ assemblyInternalBalance   ┆ 133             ┆ 133 ┆ 133    ┆
╰───────────────────────────┴─────────────────┴─────┴────────┴───

╭──────────────────────────┬─────────────────┬─────┬────────┬────
│ Contract2 contract       ┆                 ┆     ┆        ┆
╞══════════════════════════╪═════════════════╪═════╪════════╪════
│ Deployment Cost          ┆ Deployment Size ┆     ┆        ┆
├╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌┼╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌┼╌╌╌╌╌┼╌╌╌╌╌╌╌╌┼╌╌╌╌
│ 61511                    ┆ 339             ┆     ┆        ┆
├╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌┼╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌┼╌╌╌╌╌┼╌╌╌╌╌╌╌╌┼╌╌╌╌
│ Function Name            ┆ min             ┆ avg ┆ median ┆ m
├╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌┼╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌╌┼╌╌╌╌╌┼╌╌╌╌╌╌╌╌┼╌╌╌╌
│ addressExternalBalance   ┆ 417             ┆ 417 ┆ 417    ┆ 4
```

```
╰───────────────────────────┴──────────────────┴──────┴────────┴───────

╭───────────────────────────┬──────────────────┬──────┬────────┬───────
│ Contract3 contract        ┆                  ┆      ┆        ┆
╞═══════════════════════════╪══════════════════╪══════╪════════╪═══════
│ Deployment Cost           ┆ Deployment Size  ┆      ┆        ┆
├───────────────────────────┼──────────────────┼──────┼────────┼───────
│ 57105                     ┆ 317              ┆      ┆        ┆
├───────────────────────────┼──────────────────┼──────┼────────┼───────
│ Function Name             ┆ min              ┆ avg  ┆ median ┆
├───────────────────────────┼──────────────────┼──────┼────────┼───────
│ assemblyExternalBalance   ┆ 411              ┆ 411  ┆ 411    ┆
╰───────────────────────────┴──────────────────┴──────┴────────┴───────
```

Lines

- MyStrategy.sol:303
- MyStrategy.sol:322

[G-03] `unchecked{++i}` instead of `i++` (or use assembly when applicable)

Use `++i` instead of `i++`. This is especially useful in for loops but this optimization can be used anywhere in your code. You can also use `unchecked{++i;}` for even more gas savings but this will not check to see if `i` overflows. For extra safety if you are worried about this, you can add a require statement after the loop checking if `i` is equal to the final incremented value. For best gas savings, use inline assembly, however this limits the functionality you can achieve. For example you cant use Solidity syntax to internally call your own contract within an assembly block and external calls must be done with the `call()` or `delegatecall()` instruction. However when applicable, inline assembly will save much more gas.

```
contract GasTest is DSTest {
    Contract0 c0;
    Contract1 c1;
    Contract2 c2;
    Contract3 c3;
    Contract4 c4;

    function setUp() public {
        c0 = new Contract0();
        c1 = new Contract1();
        c2 = new Contract2();
        c3 = new Contract3();
        c4 = new Contract4();
    }

    function testGas() public {
        c0.iPlusPlus();
        c1.plusPlusI();
        c2.uncheckedPlusPlusI();
        c3.safeUncheckedPlusPlusI();
        c4.inlineAssemblyLoop();
    }
}

contract Contract0 {
    //loop with i++
    function iPlusPlus() public pure {
        uint256 j = 0;
        for (uint256 i; i < 10; i++) {
            j++;
        }
    }
}

contract Contract1 {
    //loop with ++i
    function plusPlusI() public pure {
        uint256 j = 0;
        for (uint256 i; i < 10; ++i) {
            j++;
        }
    }
}

contract Contract2 {
    //loop with unchecked{++i}
    function uncheckedPlusPlusI() public pure {
        uint256 j = 0;
        for (uint256 i; i < 10; ) {
            j++;

            unchecked {
                ++i;
            }
        }
    }
}

contract Contract3 {
    //loop with unchecked{++i} with additional overflow check
    function safeUncheckedPlusPlusI() public pure {
        uint256 j = 0;
        uint256 i = 0;
        for (i; i < 10; ) {
            j++;

            unchecked {
                ++i;
            }
        }

        //check for overflow
        assembly {
            if lt(i, 10) {
                mstore(0x00, "loop overflow")
                revert(0x00, 0x20)
            }
        }
    }
}

contract Contract4 {
    //loop with inline assembly
    function inlineAssemblyLoop() public pure {
        assembly {
            let j := 0

            for {
                let i := 0
            } lt(i, 10) {
                i := add(i, 0x01)
            } {
                j := add(j, 0x01)
            }
        }
    }
}
```

Gas Report

```
╭─────────────────────┬─────────────────┬──────┬────────┬──────
│ Contract0 contract  ┆                 ┆      ┆        ┆
╞═════════════════════╪═════════════════╪══════╪════════╪══════
│ Deployment Cost     ┆ Deployment Size ┆      ┆        ┆
├─────────────────────┼─────────────────┼──────┼────────┼──────
│ 37687               ┆ 219             ┆      ┆        ┆
├─────────────────────┼─────────────────┼──────┼────────┼──────
│ Function Name       ┆ min             ┆ avg  ┆ median ┆ max
├─────────────────────┼─────────────────┼──────┼────────┼──────
│ iPlusPlus           ┆ 2039            ┆ 2039 ┆ 2039   ┆ 2039
╰─────────────────────┴─────────────────┴──────┴────────┴──────

╭─────────────────────┬─────────────────┬──────┬────────┬──────
│ Contract1 contract  ┆                 ┆      ┆        ┆
╞═════════════════════╪═════════════════╪══════╪════════╪══════
│ Deployment Cost     ┆ Deployment Size ┆      ┆        ┆
├─────────────────────┼─────────────────┼──────┼────────┼──────
│ 37287               ┆ 217             ┆      ┆        ┆
├─────────────────────┼─────────────────┼──────┼────────┼──────
│ Function Name       ┆ min             ┆ avg  ┆ median ┆ max
├─────────────────────┼─────────────────┼──────┼────────┼──────
│ plusPlusI           ┆ 1989            ┆ 1989 ┆ 1989   ┆ 1989
╰─────────────────────┴─────────────────┴──────┴────────┴──────

╭─────────────────────────┬─────────────────┬──────┬────────┬──
│ Contract3 contract      ┆                 ┆      ┆        ┆
╞═════════════════════════╪═════════════════╪══════╪════════╪══
│ Deployment Cost         ┆ Deployment Size ┆      ┆        ┆
├─────────────────────────┼─────────────────┼──────┼────────┼──
│ 42693                   ┆ 244             ┆      ┆        ┆
├─────────────────────────┼─────────────────┼──────┼────────┼──
│ Function Name           ┆ min             ┆ avg  ┆ median ┆
├─────────────────────────┼─────────────────┼──────┼────────┼──
│ safeUncheckedPlusPlusI  ┆ 1355            ┆ 1355 ┆ 1355   ┆
╰─────────────────────────┴─────────────────┴──────┴────────┴──

╭─────────────────────┬─────────────────┬──────┬────────┬──────
│ Contract2 contract  ┆                 ┆      ┆        ┆
╞═════════════════════╪═════════════════╪══════╪════════╪══════
│ Deployment Cost     ┆ Deployment Size ┆      ┆        ┆
├─────────────────────┼─────────────────┼──────┼────────┼──────
│ 35887               ┆ 210             ┆      ┆        ┆
├─────────────────────┼─────────────────┼──────┼────────┼──────
│ Function Name       ┆ min             ┆ avg  ┆ median ┆ max
├─────────────────────┼─────────────────┼──────┼────────┼──────
│ uncheckedPlusPlusI  ┆ 1329            ┆ 1329 ┆ 1329   ┆ 1329
```

```
╰────────────────────┴──────────────────┴──────┴────────┴──────┴

╭────────────────────┬──────────────────┬──────┬────────┬──────┬
│ Contract4 contract ┆                  ┆      ┆        ┆      ┆
╞════════════════════╪══════════════════╪══════╪════════╪══════╪
│ Deployment Cost    ┆ Deployment Size  ┆      ┆        ┆      ┆
├────────────────────┼──────────────────┼──────┼────────┼──────┼
│ 26881              ┆ 164              ┆      ┆        ┆      ┆
├────────────────────┼──────────────────┼──────┼────────┼──────┼
│ Function Name      ┆ min              ┆ avg  ┆ median ┆ max  ┆
├────────────────────┼──────────────────┼──────┼────────┼──────┼
│ inlineAssemblyLoop ┆ 709              ┆ 709  ┆ 709    ┆ 709  ┆
╰────────────────────┴──────────────────┴──────┴────────┴──────┴
```

Lines

- MyStrategy.sol:118
- MyStrategy.sol:153
- MyStrategy.sol:300
- MyStrategy.sol:317

[G-04] Use multiple `require()` statments insted of `require(expression && expression && ...)`

```
contract GasTest is DSTest {
    Contract0 c0;
    Contract1 c1;

    function setUp() public {
        c0 = new Contract0();
        c1 = new Contract1();
    }

    function testGas() public {
        c0.singleRequire(3);
        c1.multipleRequire(3);
    }
}

contract Contract0 {
    function singleRequire(uint256 num) public {
        require(num > 1 && num < 10 && num == 3);
    }
}

contract Contract1 {
    function multipleRequire(uint256 num) public {
        require(num > 1);
        require(num < 10);
        require(num == 3);
    }
}
```

Gas Report

```
╭────────────────────┬─────────────────┬─────┬────────┬─────┬─────╮
| Contract0 contract ┆                 ┆     ┆        ┆     ┆     |
╞════════════════════╪═════════════════╪═════╪════════╪═════╪═════╡
| Deployment Cost    ┆ Deployment Size ┆     ┆        ┆     ┆     |
|--------------------+-----------------+-----+--------+-----+-----|
| 35487              ┆ 208             ┆     ┆        ┆     ┆     |
|--------------------+-----------------+-----+--------+-----+-----|
| Function Name      ┆ min             ┆ avg ┆ median ┆ max ┆     |
|--------------------+-----------------+-----+--------+-----+-----|
| singleRequire      ┆ 286             ┆ 286 ┆ 286    ┆ 286 ┆     |
╰────────────────────┴─────────────────┴─────┴────────┴─────┴─────╯
╭────────────────────┬─────────────────┬─────┬────────┬─────┬─────╮
| Contract1 contract ┆                 ┆     ┆        ┆     ┆     |
╞════════════════════╪═════════════════╪═════╪════════╪═════╪═════╡
| Deployment Cost    ┆ Deployment Size ┆     ┆        ┆     ┆     |
|--------------------+-----------------+-----+--------+-----+-----|
| 35887              ┆ 210             ┆     ┆        ┆     ┆     |
|--------------------+-----------------+-----+--------+-----+-----|
| Function Name      ┆ min             ┆ avg ┆ median ┆ max ┆     |
|--------------------+-----------------+-----+--------+-----+-----|
| multipleRequire    ┆ 270             ┆ 270 ┆ 270    ┆ 270 ┆     |
╰────────────────────┴─────────────────┴─────┴────────┴─────┴─────╯
```

Lines

- MyStrategy.sol:185

[G-05] Use assembly to check for address(0)

```
contract GasTest is DSTest {
    Contract0 c0;
    Contract1 c1;

    function setUp() public {
        c0 = new Contract0();
        c1 = new Contract1();
    }

    function testGas() public view {
        c0.ownerNotZero(address(this));
        c1.assemblyOwnerNotZero(address(this));
    }
}

contract Contract0 {
    function ownerNotZero(address _addr) public pure {
        require(_addr != address(0), "zero address)");
    }
}

contract Contract1 {
    function assemblyOwnerNotZero(address _addr) public pure {
        assembly {
            if iszero(_addr) {
                mstore(0x00, "zero address")
                revert(0x00, 0x20)
            }
        }
    }
}
```

Gas Report

```
╭─────────────────────┬──────────────────┬──────┬─────────┬──────┬──────
│ Contract0 contract  ┆                  ┆      ┆         ┆      ┆
╞═════════════════════╪══════════════════╪══════╪═════════╪══════╪══════
│ Deployment Cost     ┆ Deployment Size  ┆      ┆         ┆      ┆
├─────────────────────────────────────┼─────────────────────────────────
│ 61311               ┆ 338              ┆      ┆         ┆      ┆
```

```
├──────────────────────────────────────┼─────────────────────────────────
│ Function Name          ┆ min                 ┆ avg ┆ median ┆ max ┆
├──────────────────────────────────────┼─────────────────────────────────
│ ownerNotZero           ┆ 258                 ┆ 258 ┆ 258    ┆ 258 ┆
╰────────────────────────┴───────────────────┴──────┴─────────┴──────┴──

╭─────────────────────────┬───────────────────┬──────┬─────────┬──────┬──
│ Contract1 contract      ┆                   ┆      ┆         ┆
╞═════════════════════════╪═══════════════════╪══════╪═════════╪══════╪══
│ Deployment Cost         ┆ Deployment Size   ┆      ┆         ┆
├─────────────────────────────────────────────┼───────────────────────────
│ 44893                   ┆ 255               ┆      ┆         ┆
├─────────────────────────────────────────────┼───────────────────────────
│ Function Name           ┆ min               ┆ avg  ┆ median  ┆ max
├─────────────────────────────────────────────┼───────────────────────────
│ assemblyOwnerNotZero    ┆ 252               ┆ 252  ┆ 252     ┆ 252
╰─────────────────────────┴───────────────────┴──────┴─────────┴──────┴──
```

Lines

- MyStrategy.sol:290

Alex the Entreprenerd (BadgerDAO) confirmed and commented:

> Thank you for the thoughtful submission

> However note that saving a couple of storage loads would net us way more gas savings than most of these

jack-the-pug (judge) validated

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.

Top