Badger Citadel contest Findings & Analysis Report

2022-07-08

Table of contents

- G-06 `Funding.getRemainingFundable()` : L236 should be unchecked due to L235
- G-07 `Funding.claimAssetToTreasury()` : `asset` should get cached
- G-08 `KnightingRound.initialize()` : should use memory instead of storage variable
- G-09 `KnightingRound.buy()` : `saleStart` , `totalTokenIn` and `guestlist` should get cached
- G-10 `KnightingRound.getTokenInLimitLeft()` : `totalTokenIn` and `tokenInLimit` should get cached + L250 should be unchecked due to L249
- G-11 `StakedCitadel.deposit()` : Use `calldata` instead of `memory`
- G-12 `StakedCitadel.depositAll()` : Use `calldata` instead of `memory`
- G-13 `StakedCitadel.setStrategy()` : `strategy` should get cached
- G-14 `StakedCitadel.earn()` : `strategy` should get cached
- G-15 `StakedCitadel._depositFor()` : `token` should get cached
- G-16 `StakedCitadel._depositFor()` : L776 should be unchecked due to L773-L775
- G-17 `StakedCitadel._depositForWithAuthorization()` : `guestList` should get cached
- G-18 `StakedCitadel._withdraw()` : `token` and `vesting` should get cached
- G-19 `StakedCitadel._withdraw()` : L817 should be unchecked due to L816
- G-20 `StakedCitadelLocker.sol` : state variables can be tightly packed to save 1 storage slot
- G-21 `StakedCitadelLocker.totalSupplyAtEpoch()` : Use a storage variable's reference instead of repeatedly fetching it (`epochs[i]`)
- G-22 `StakedCitadel._withdraw()` : `maximumStake` , `minimumStake` and `stakingProxy` should get cached
- G-23 `StakedCitadelVester.claimableBalance()` : Help the optimizer by saving a storage variable's reference instead of repeatedly fetching it

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 Citadel smart contract system written in Solidity. The audit contest took place between April 14—April 20 2022.

Wardens

79 Wardens contributed reports to the Badger Citadel contest:

1. IllIllI
2. georgypetrov
3. cmichel
4. cccz
5. VAD37
6. 0xDjango
7. danb
8. hyh
9. berndartmueller
10. reassor
11. TrungOre
12. rayn
13. minhquanym
14. wuwe1
15. Ruhum
16. shenwilly
17. kyliek
18. gs8nrv
19. gzeon
20. m9800

21. 0xBug
22. pedroais
23. Dravee
24. CertoraInc (egjlmn1, OriDabush, ItayG, and shakedwinder)
25. horsefacts
26. scaraven
27. sorrynotsorry
28. MaratCerby
29. joestakey
30. TomFrenchBlockchain
31. remora
32. ilan
33. csanuragjain
34. defsec
35. rfa
36. TerrierLover
37. fatherOfBlocks
38. 0xkatana
39. robee
40. ellahi
41. 0x1f8b
42. kenta
43. securerodd
44. tchkvsky
45. Funen
46. kebabsec (okkothejawa and FlameHorizon)
47. SolidityScan (cyberboy and zombie)
48. teryanarmen
49. z3s

50. Ov3rf1Ow
51. jah
52. oyc_109
53. delfin454000
54. Hawkeye (0xwags and 0xmint)
55. hubble (ksk2345 and shri4net)
56. AmitN
57. dipp
58. p_crypt0
59. peritoflores
60. Picodes
61. Jujic
62. Yiko
63. Tomio
64. saian
65. 0xAsm0d3us
66. 0xNazgul
67. joshie
68. slywaters
69. Cityscape
70. simon135
71. bae11
72. nahnah

This contest was judged by Jack the Pug.

Final report assembled by itsmetechjay.

Summary

The C4 analysis yielded an aggregated total of 8 unique vulnerabilities. Of these vulnerabilities, 3 received a risk rating in the category of HIGH severity and 5 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 58 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 48 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 Citadel contest repository, and is composed of 8 smart contracts written in the Solidity programming language and includes 2,339 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 (3)

[H-01] StakedCitadel doesn't use correct balance for internal accounting

Submitted by Ruhum, also found by cccz, wuwe1, VAD37, TrungOre, shenwilly, minhquanym, kyliek, danb, gs8nrv, and rayn

https://github.com/code-423n4/2022-04-badger-citadel/blob/main/src/StakedCitadel.sol#L291-L295

https://github.com/code-423n4/2022-04-badger-citadel/blob/main/src/StakedCitadel.sol#L772-L776

https://github.com/code-423n4/2022-04-badger-citadel/blob/main/src/StakedCitadel.sol#L881-L893

Impact

The StakedCitadel contract's `balance()` function is supposed to return the balance of the vault + the balance of the strategy. But, it only returns the balance of the vault. The balance is used to determine the number of shares that should be minted when depositing funds into the vault and the number of shares that should be burned when withdrawing funds from it.

Since most of the funds will be located in the strategy, the vault's balance will be very low. Some of the issues that arise from this:

You can't deposit to a vault that already minted shares but has no balance of the underlying token:

1. fresh vault with 0 funds and 0 shares
2. Alice deposits 10 tokens. She receives 10 shares back (https://github.com/code-423n4/2022-04-badger-citadel/blob/main/src/StakedCitadel.sol#L887-L888)
3. Vault's tokens are deposited into the strategy (now `balance == 0` and `totalSupply == 10`)
4. Bob tries to deposit but the transaction fails because the contract tries to divide by zero: https://github.com/code-423n4/2022-04-badger-citadel/blob/main/src/StakedCitadel.sol#L890 (`pool == balance()`)

You get more shares than you should

1. fresh vault with 0 funds and 0 shares
2. Alice deposits 10 tokens. She receives 10 shares back (https://github.com/code-423n4/2022-04-badger-citadel/blob/main/src/StakedCitadel.sol#L887-L888)
3. Vault's tokens are deposited into the strategy (now `balance == 0` and `totalSupply == 10`)
4. Bob now first transfers 1 token to the vault so that the balance is now `1` instead of `0`.
5. Bob deposits 5 tokens. He receives `5 * 10 / 1 == 50` shares: https://github.com/code-423n4/2022-04-badger-citadel/blob/main/src/StakedCitadel.sol#L890

Now, the vault received 15 tokens. 10 from Alice and 5 from Bob. But Alice only has 10 shares while Bob has 50. Thus, Bob can withdraw more tokens than he should be able to.

It simply breaks the whole accounting of the vault.

Proof of Concept

The comment says that it should be vault's + strategy's balance: https://github.com/code-423n4/2022-04-badger-citadel/blob/main/src/StakedCitadel.sol#L291-L295

Here's another vault from the badger team where the function is implemented correctly: https://github.com/Badger-Finance/badger-vaults-1.5/blob/main/contracts/Vault.sol#L262

Recommended Mitigation Steps

Add the strategy's balance to the return value of the `balance()` function like here.

GalloDaSballo (BadgerDAO) confirmed and commented:

> Agree balance must have been changed by mistake or perhaps earn should not transfer to a strategy either would work

[H-02] StakedCitadel: wrong setupVesting function name

Submitted by cccz, also found by TrungOre, wuwe1, reassor, 0xBug, georgypetrov, 0xDjango, scaraven, horsefacts, berndartmueller, CertoraInc, rayn, m9800, pedroais, and VAD37

In the `\_withdraw` function of the StakedCitadel contract, the setupVesting function of vesting is called, while in the StakedCitadelVester contract, the function name is vest, which will cause the _withdraw function to fail, so that the user cannot withdraw the tokens.

```
            IVesting(vesting).setupVesting(msg.sender, _amount, bloc
            token.safeTransfer(vesting, _amount);
            ...
        function vest(
            address recipient,
            uint256 _amount,
            uint256 _unlockBegin
        ) external {
            require(msg.sender == vault, "StakedCitadelVester: only
            require(_amount > 0, "StakedCitadelVester: cannot vest (

            vesting[recipient].lockedAmounts =
                vesting[recipient].lockedAmounts +
                _amount;
            vesting[recipient].unlockBegin = _unlockBegin;
            vesting[recipient].unlockEnd = _unlockBegin + vestingDur

            emit Vest(
                recipient,
                vesting[recipient].lockedAmounts,
                _unlockBegin,
                vesting[recipient].unlockEnd
            );
        }
```

Proof of Concept

https://github.com/code-423n4/2022-04-badger-citadel/blob/main/src/StakedCitadel.sol#L830 https://github.com/code-423n4/2022-04-badger-citadel/blob/main/src/interfaces/citadel/IVesting.sol#L5

Recommended Mitigation Steps

Use the correct function name

```
    interface IVesting {
        function vest(
            address recipient,
            uint256 _amount,
            uint256 _unlockBegin
        ) external;
    }
    ...
    IVesting(vesting).vest(msg.sender, _amount, block.timestamp);
    token.safeTransfer(vesting, _amount);
```

dapp-whisperer (BadgerDAO) confirmed and resolved

[H-03] StakedCitadel depositors can be attacked by the first depositor with depressing of vault token denomination

Submitted by hyh, also found by VAD37, cmichel, 0xDjango, berndartmueller, and danb

https://github.com/code-423n4/2022-04-badger-citadel/blob/main/src/StakedCitadel.sol#L881-L892

https://github.com/code-423n4/2022-04-badger-citadel/blob/main/src/StakedCitadel.sol#L293-L295

Impact

An attacker can become the first depositor for a recently created StakedCitadel contract, providing a tiny amount of Citadel tokens by calling `deposit(1)` (raw values here, `1` is `1 wei`, `1e18` is `1 Citadel` as it has 18 decimals). Then the attacker can directly transfer, for example, `10^6*1e18 - 1` Citadel to

StakedCitadel, effectively setting the cost of `1` of the vault token to be `10^6 * 1e18` Citadel. The attacker will still own 100% of the StakedCitadel's pool being the only depositor.

All subsequent depositors will have their Citadel token investments rounded to `10^6 * 1e18`, due to the lack of precision which initial tiny deposit caused, with the remainder divided between all current depositors, i.e. the subsequent depositors lose value to the attacker.

For example, if the second depositor brings in `1.9*10^6 * 1e18` Citadel, only `1` of new vault to be issued as `1.9*10^6 * 1e18` divided by `10^6 * 1e18` will yield just `1`, which means that `2.9*10^6 * 1e18` total Citadel pool will be divided 50/50 between the second depositor and the attacker, as each have 1 wei of the total 2 wei of vault tokens, i.e. the depositor lost and the attacker gained `0.45*10^6 * 1e18` Citadel tokens.

As there are no penalties to exit with StakedCitadel.withdraw(), the attacker can remain staked for an arbitrary time, gathering the share of all new deposits' remainder amounts.

Placing severity to be high as this is principal funds loss scenario for many users (most of depositors), easily executable, albeit only for the new StakedCitadel contract.

Proof of Concept

deposit() -> _depositFor() -> _mintSharesFor() call doesn't require minimum amount and mints according to the provided amount:

deposit:

https://github.com/code-423n4/2022-04-badger-citadel/blob/main/src/StakedCitadel.sol#L309-L311

_depositFor:

https://github.com/code-423n4/2022-04-badger-citadel/blob/main/src/StakedCitadel.sol#L764-L777 _mintSharesFor:

https://github.com/code-423n4/2022-04-badger-citadel/blob/main/src/StakedCitadel.sol#L881-L892

When StakedCitadel is new the `_pool = balance()` is just initially empty contract balance:

https://github.com/code-423n4/2022-04-badger-citadel/blob/main/src/StakedCitadel.sol#L293-L295

Any deposit lower than total attacker's stake will be fully stolen from the depositor as `0` vault tokens will be issued in this case.

References

The issue is similar to the `TOB-YEARN-003` one of the Trail of Bits audit of Yearn Finance:

https://github.com/yearn/yearn-security/tree/master/audits/20210719_ToB_yearn_vaultsv2

Recommended Mitigation Steps

A minimum for deposit value can drastically reduce the economic viability of the attack. I.e. `deposit() -> ...` can require each amount to surpass the threshold, and then an attacker would have to provide too big direct investment to capture any meaningful share of the subsequent deposits.

An alternative is to require only the first depositor to freeze big enough initial amount of liquidity. This approach has been used long enough by various projects, for example in Uniswap V2:

https://github.com/Uniswap/v2-core/blob/master/contracts/UniswapV2Pair.sol#L119-L121

GalloDaSballo (BadgerDAO) acknowledged, disagreed with severity and commented:

> Disagree with the dramatic effect the warden is implying.

> Agree with the finding as this is a property of vault based systems

> Also worth noting that anyone else can still get more deposits in and get their fair share, it's just that the first deposit would now require a deposit of at least `vault.balanceOf` in order to get the fair amount of shares (which at this point would be rebased to be 1 = `prevBalanceOf`)

[jack-the-pug (judge) commented]:

> I believe this is a valid `High` even though the precondition of this attack is quite strict (the attacker has to be the 1st depositor).

> The impact is not just a regular precision loss, but with the pricePerShare of the vault being manipulated to an extreme value, all regular users will lose up to the pricePerShare of the deposited amount due to huge precision loss.

Medium Risk Findings (5)

[M-01] Guaranteed citadel profit

Submitted by georgypetrov

User can sandwich `mintAndDistribute` function if mintable is high enough

- Deposit before
- Withdraw after
- Take after 21 days citadels

Proof of Concept

`mintAndDistribute` increase a price of staking share, that allows to withdraw more than deposited. user takes part of distributed citadels, so different users have smaller profit from distribution

Recommended Mitigation Steps

Call `mintAndDistribute` through flashbots

[GalloDaSballo (BadgerDAO) confirmed, disagreed with severity and commented](https://github.com/code-423n4/2022-04-badger-citadel-findings/issues/):

> My interpretation of the finding is that there's no linear vesting in the way more rewards are distributed so they can be frontrun.

> I have to disagree in that taking 21 days of exposure to a random token in order to gain a small sub 1% gain is probably not what I'd call a smart move.

> That said, I believe the front-running finding to be valid, and while I disagree with High I believe the finding to have validity

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

> Downgrading to `Medium` as this attack vector is not economically profitable in practice (because of the 21 days vesting).

[M-02] Funding.deposit() doesn't work if there is no discount set

Submitted by Ruhum, also found by TrungOre, MaratCerby, 0xBug, minhquanym, shenwilly, 0xDjango, remora, danb, IllIIII, pedroais, m9800, and hyh

https://github.com/code-423n4/2022-04-badger-citadel/blob/main/src/Funding.sol#L177

https://github.com/code-423n4/2022-04-badger-citadel/blob/main/src/Funding.sol#L202

https://github.com/code-423n4/2022-04-badger-citadel/blob/main/src/Funding.sol#L184

https://github.com/code-423n4/2022-04-badger-citadel/blob/main/src/StakedCitadel.sol#L769

Impact

The Funding contract's `deposit()` function uses the `getAmountOut()` function to determine how many citadel tokens the user should receive for their deposit. But, if no discount is set, the function always returns 0. Now the `deposit()` function tries to deposit 0 tokens for the user through the StakedCitadel contract. But, that function requires the number of tokens to be `!= 0`. The transaction reverts.

This means, that no deposits are possible. Unless there is a discount.

Proof of Concept

`Funding.deposit()` calls `getAmountOut()`: https://github.com/code-423n4/2022-04-badger-citadel/blob/main/src/Funding.sol#L177

Here's the `getAmountOut()` function:

```
    function getAmountOut(uint256 _assetAmountIn)
        public
        view
        returns (uint256 citadelAmount_)
    {
        uint256 citadelAmountWithoutDiscount = _assetAmountIn *

        if (funding.discount > 0) {
            citadelAmount_ =
                (citadelAmountWithoutDiscount * MAX_BPS) /
                (MAX_BPS - funding.discount);
        }

        // unless the above if block is executed, `citadelAmount
        // 0 = 0 / x
        citadelAmount_ = citadelAmount_ / assetDecimalsNormaliza
    }
```

Call to `StakedCitadel.depositFor()`: https://github.com/code-423n4/2022-04-badger-citadel/blob/main/src/Funding.sol#L184

require statement that makes the whole transaction revert: https://github.com/code-423n4/2022-04-badger-citadel/blob/main/src/StakedCitadel.sol#L769

Recommended Mitigation Steps

Change the `getAmountOut()` function to:

```
    function getAmountOut(uint256 _assetAmountIn)
        public
        view
        returns (uint256 citadelAmount_)
    {

        uint256 citadelAmount_ = _assetAmountIn * citadelPriceIr

        if (funding.discount > 0) {
            citadelAmount_ =
                (citadelAmount_ * MAX_BPS) /
                (MAX_BPS - funding.discount);
        }

        citadelAmount_ = citadelAmount_ / assetDecimalsNormaliza
    }
```

shuklaayush (BadgerDAO) confirmed

[M-03] KnightingRound tokenOutPrice changes

Submitted by reassor, also found by cccz and cmichel

`Function.buy` buys the tokens for whatever price is set as `tokenOutPrice`. This might lead to accidental collisions or front-running attacks when user is trying to buy the tokens and his transaction is being included after the transaction of changing the price of the token via `setTokenOutPrice`.

Scenario:

1. User wants to `buy` tokens and can see price `tokenOutPrice`
2. User likes the price and issues a transaction to `buy` tokens
3. At the same time `CONTRACT_GOVERNANCE_ROLE` account is increasing `tokenOutPrice` through `setTokenOutPrice`

4. `setTokenOutPrice` transaction is included before user's `buy` transaction
5. User buys tokens with the price he was not aware of

Another variation of this attack can be performed using front-running.

Proof of Concept

- https://github.com/code-423n4/2022-04-badger-citadel/blob/18f8c392b6fc303fe95602eba6303725023e53da/src/KnightingRound.sol#L162-L204

Tools Used

Manual Review / VSCode

Recommended Mitigation Steps

It is recommended to add additional parameter `uint256 believedPrice` to `KnightingRound.buy` function and check if `believedPrice` is equal to `tokenOutPrice`.

[GalloDaSballo (BadgerDAO) confirmed]()

[M-04] New vest reset `unlockBegin` of existing vest without removing vested amount

Submitted by gzeon, also found by cccz, TrungOre, minhquanym, cmichel, 0xDjango, and rayn

https://github.com/code-423n4/2022-04-badger-citadel/blob/18f8c392b6fc303fe95602eba6303725023e53da/src/StakedCitadelVester.sol#L143

https://github.com/code-423n4/2022-04-badger-citadel/blob/18f8c392b6fc303fe95602eba6303725023e53da/src/StakedCitadelVester.sol#L109

Impact

When `vest` is called by xCTDL vault, the previous amount will re-lock according to the new vesting timeline. While this is as described in L127, `claimableBalance` might revert due to underflow if `vesting[recipient].claimedAmounts` > 0 because the user will need to vest the `claimedAmounts` again which should not be an expected behavior as it is already vested.

Proof of Concept

https://github.com/code-423n4/2022-04-badger-citadel/blob/18f8c392b6fc303fe95602eba6303725023e53da/src/StakedCitadelVester.sol#L143

```
            vesting[recipient].lockedAmounts =
                vesting[recipient].lockedAmounts +
                _amount;
            vesting[recipient].unlockBegin = _unlockBegin;
            vesting[recipient].unlockEnd = _unlockBegin + vestingDu
```

https://github.com/code-423n4/2022-04-badger-citadel/blob/18f8c392b6fc303fe95602eba6303725023e53da/src/StakedCitadelVester.sol#L109

```
            uint256 locked = vesting[recipient].lockedAmounts;
            uint256 claimed = vesting[recipient].claimedAmounts;
            if (block.timestamp >= vesting[recipient].unlockEnd) {
                return locked - claimed;
            }
            return
                ((locked * (block.timestamp - vesting[recipient].un
                    (vesting[recipient].unlockEnd -
                        vesting[recipient].unlockBegin)) - claimed;
```

Recommended Mitigation Steps

Reset claimedAmounts on new vest

```
            vesting[recipient].lockedAmounts =
                vesting[recipient].lockedAmounts -
                vesting[recipient].claimedAmounts +
                _amount;
            vesting[recipient].claimedAmounts = 0
            vesting[recipient].unlockBegin = _unlockBegin;
            vesting[recipient].unlockEnd = _unlockBegin + vestingDur
```

shuklaayush (BadgerDAO) confirmed and commented:

> I think this is valid and was fixed in https://github.com/Citadel-DAO/citadel-contracts/pull/44

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

> I'm downgrading this to Medium as there are no funds directly at risk, but a malfunction and leak of value. The user will have to wait for a longer than expected time to claim their vested funds.

[M-05] Stale price used when `citadelPriceFlag` is cleared

Submitted by IllIllI

During the video it was explained that the policy operations team was meant to be a nimble group that could change protocol values considered to be safe. Further, it was explained that since pricing comes from an oracle, and there would have to be unusual coordination between the two to affect outcomes, the group was given the ability to clear the pricing flag to get things moving again once the price was determined to be valid

Impact

If an oracle price falls out of the valid min/max range, the `citadelPriceFlag` is set to true, but the out-of-bounds value is not stored. If the policy operations team calls `clearCitadelPriceFlag()`, the stale price from before the flag will be used. Not only is it an issue because of stale prices, but this means the policy op team now has a way to affect pricing not under the control of the oracle (i.e. no unusual coordination required to affect an outcome). Incorrect pricing leads to incorrect asset valuations, and loss of funds.

Proof of Concept

The flag is set but the price is not stored File: src/Funding.sol (lines 427-437)

```
        if (
            _citadelPriceInAsset < minCitadelPriceInAsset ||
            _citadelPriceInAsset > maxCitadelPriceInAsset
        ) {
            citadelPriceFlag = true;
            emit CitadelPriceFlag(
                _citadelPriceInAsset,
                minCitadelPriceInAsset,
                maxCitadelPriceInAsset
            );
        } else {
```

Tools Used

Code inspection

Recommended Mitigation Steps

Always set the `citadelPriceInAsset`

shuklaayush (BadgerDAO) confirmed and commented:

> Makes sense. It's best to update the price even when it's flagged

jack-the-pug (judge) commented:

> This is a very good catch! `citadelPriceInAsset` is not updated when `citadelPriceFlag` is set, therefore clearing the flag will not approve the out of range price but continues with a stale price before the out of range price.

Low Risk and Non-Critical Issues

For this contest, 58 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: hyh, sorrynotsorry, berndartmueller, csanuragjain, ilan, kyliek, Ruhum, joestakey, defsec, reassor, TerrierLover, TomFrenchBlockchain, CertoraInc, ellahi, robee, shenwilly, TrungOre, danb, Dravee, fatherOfBlocks, hubble, AmitN, Funen, horsefacts, kebabsec, kenta, scaraven, securerodd, tchkvsky, 0xkatana, rayn, SolidityScan, 0x1f8b, 0xDjango, cmichel, delfin454000, dipp, gs8nrv, gzeon, Hawkeye, jah, m9800, minhquanym, oyc_109, p_crypt0, peritoflores, Picodes, remora, teryanarmen, VAD37, z3s, 0v3rf10w, georgypetrov, Jujic, MaratCerby, rfa, *and* Yiko.

[L-01] New min/max values should be checked against the current stored value

If `citadelPriceInAsset` is above the new max or below the new min, the next update will likely have a similar value and immediately cause problems. The code should require that the current value falls within the new range

1. File: src/Funding.sol (lines 402-403)

```
            minCitadelPriceInAsset = _minPrice;
            maxCitadelPriceInAsset = _maxPrice;
```

[L-02] Loss of precision

If `tokenOutPrice` is less than `tokenInNormalizationValue`, then the amount will be zero for some amounts. The caller of `getAmountOut()` should revert if `tokenOutAmount` ends up being zero

1. File: src/KnightingRound.sol (lines 239-241)

```
            tokenOutAmount_ =
                (_tokenInAmount * tokenOutPrice) /
                tokenInNormalizationValue;
```

[L-03] Unsafe calls to optional ERC20 functions

`decimals()`, `name()` and `symbol()` are optional parts of the ERC20 specification, so there are tokens that do not implement them. It's not safe to cast arbitrary token addresses in order to call these functions. If `IERC20Metadata` is to be relied on, that should be the variable type of the token variable, rather than it being `address`, so the compiler can verify that types correctly match, rather than this being a runtime failure. See this prior instance of this issue which was marked as Low risk. Do this to resolve the issue.

1. File: src/interfaces/erc20/IERC20.sol (lines 14-18)

```
    function name() external view returns (string memory);

    function symbol() external view returns (string memory);

    function decimals() external view returns (uint256);
```

2. File: src/KnightingRound.sol (line 148)

```
        tokenInNormalizationValue = 10**tokenIn.decimals();
```

3. File: src/StakedCitadel.sol (line 218)

```
                abi.encodePacked(_defaultNamePrefix, namedToken.
```

4. File: src/StakedCitadel.sol (line 226)

```
                abi.encodePacked(_symbolSymbolPrefix, namedToker
```

[L-04] Missing checks for `address(0x0)` when assigning values to `address` state variables

1. File: external/StakedCitadelLocker.sol (line 186)

```
        stakingProxy = _staking;
```

2. File: src/lib/SettAccessControl.sol (line 39)

```
        strategist = _strategist;
```

3. File: src/lib/SettAccessControl.sol (line 46)

```
        keeper = _keeper;
```

4. File: src/lib/SettAccessControl.sol (line 53)

```
        governance = _governance;
```

[L-05] `initialize` functions can be front-run

See this finding from a prior badger-dao contest for details

1. File: src/CitadelMinter.sol (line 109)

```
    function initialize(
```

2. File: src/KnightingRound.sol (line 119)

```
    ) external initializer {
```

3. File: src/Funding.sol (line 112)

```
    ) external initializer {
```

4. File: src/StakedCitadel.sol (line 179)

```
    ) public initializer whenNotPaused {
```

[L-06] `safeApprove()` is deprecated

Deprecated in favor of `safeIncreaseAllowance()` and `safeDecreaseAllowance()`

1. File: src/CitadelMinter.sol (line 133)

```
        IERC20Upgradeable(_citadelToken).safeApprove(_xCitadel,
```

2. File: src/CitadelMinter.sol (line 136)

```
        IERC20Upgradeable(_xCitadel).safeApprove(_xCitadelLocker
```

3. File: src/Funding.sol (line 142)

```
        IERC20(_citadel).safeApprove(address(_xCitadel), type(ui
```

[L-07] 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.

1. File: external/StakedCitadelLocker.sol (line 26)

```
contract StakedCitadelLocker is Initializable, ReentrancyGuardUp
```

2. File: src/CitadelMinter.sol (lines 23-25)

```
contract CitadelMinter is
    GlobalAccessControlManaged,
    ReentrancyGuardUpgradeable
```

3. File: src/CitadelToken.sol (line 8)

```
contract CitadelToken is GlobalAccessControlManaged, ERC20Upgrac
```

4. File: src/Funding.sol (line 17)

```
contract Funding is GlobalAccessControlManaged, ReentrancyGuardU
```

5. File: src/GlobalAccessControl.sol (lines 19-21)

```
contract GlobalAccessControl is
    AccessControlEnumerableUpgradeable,
    PausableUpgradeable
```

6. File: src/KnightingRound.sol (line 16)

```
contract KnightingRound is GlobalAccessControlManaged, Reentranc
```

7. File: src/lib/GlobalAccessControlManaged.sol (line 12)

```
contract GlobalAccessControlManaged is PausableUpgradeable {
```

8. File: src/StakedCitadel.sol (lines 59-63)

```
contract StakedCitadel is
    ERC20Upgradeable,
    SettAccessControl,
    PausableUpgradeable,
    ReentrancyGuardUpgradeable
```

9. File: src/StakedCitadelVester.sol (lines 14-16)

```
    contract StakedCitadelVester is
        GlobalAccessControlManaged,
        ReentrancyGuardUpgradeable
```

[L-08] Unbounded loop

If there are too many pools, the function may run out of gas while returning them. It’s best to allow for a start offset and maximum length, so data can be returned in batches that don’t run out of gas

1. File: src/CitadelMinter.sol (lines 143-147)

```
        function getFundingPoolWeights()
            external
            view
            returns (address[] memory pools, uint256[] memory weight
        {
```

[N-01] Open TODOs

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

1. File: src/Funding.sol (line 15)

```
     * TODO: Better revert strings
```

2. File: src/Funding.sol (line 61)

```
    // TODO: we should conform to some interface here
```

3. File: src/Funding.sol (line 183)

```
        // TODO: Check gas costs. How does this relate to market
```

4. File: src/GlobalAccessControl.sol (line 106)

```
    /// TODO: Add string -> hash EnumerableSet to a new RoleRegi
```

5. File: src/KnightingRound.sol (line 14)

```
 * TODO: Better revert strings
```

6. File: src/SupplySchedule.sol (line 159)

```
        // TODO: Require this epoch is in the future. What happe
```

[N-02] Misleading comment

The value of `transferFromDisabled` is never updated, let alone in an `initialize()` function. I don't see any bugs related to this, but this comment makes it seem as though something was overlooked when branching.

1. File: src/GlobalAccessControl.sol (line 51)

```
    bool public transferFromDisabled; // Set to true in initiali
```

[N-03] Multiple definitions of an interface

These are the only two differences between `IEmptyStrategy` and `IStrategy`. `IEmptyStrategy` should be changed to be `is IStrategy` and remove the duplicate functions

1. File: src/interfaces/badger/IEmptyStrategy.sol (lines 12-14)

```
    function initialize(address vault, address want) external;

    function getName() external view returns (string memory);
```

[N-04] Unused file

1. File: src/interfaces/convex/BoringMath.sol (line 1)

```
// SPDX-License-Identifier: MIT
```

[N-05] Contract header not updated after branching

1. File: src/GlobalAccessControl.sol (lines 12-17)

```
/**
 * @title Badger Geyser
 @dev Tracks stakes and pledged tokens to be distributed, for us
 @dev BadgerTree merkle distribution system. An arbitrary number
 distribute can be specified.
 */
```

[N-06] Comment not moved when function was moved

1. File: src/SupplySchedule.sol (lines 52-53)

```
    // @dev duplicate of getMintable() with debug print added
    // @dev this function is out of scope for reviews and audits
```

[N-07] Comments not updated after branching

There are a lot of references to the old owner-related code. The comments should be updated to talk about the new RBAC system

1. File: src/KnightingRound.sol

```
$ grep owner src/KnightingRound.sol
     * @notice Finalize the sale after sale duration. Can only b
     * @notice Update the sale start time. Can only be called by
     * @notice Update sale duration. Can only be called by owner
     * @notice Modify the tokenOut price in. Can only be called
     * @notice Update the `tokenIn` receipient address. Can only
     * @notice Update the guestlist address. Can only be called
     * @notice Modify the max tokenIn that this contract can tak
     * @notice Transfers out any tokens accidentally sent to the
```

The price calulation seems inverted since this comment was first written, so it should be updated to reflect the new calculation: 2. File: src/KnightingRound.sol (line 43)

```
    /// eg. 1 WBTC (8 decimals) = 40,000 CTDL ==> price = 10^8 /
```

[N-08] Remove `include` for ds-test

Test code should not be mixed in with production code. The production version should be extended and have its functions overridden for testing purposes

1. File: src/SupplySchedule.sol (line 4)

```
import "ds-test/test.sol";
```

[N-09] The `nonReentrant` `modifier` should occur before all other modifiers

This is a best-practice to protect against reentrancy in other modifiers

1. File: src/CitadelMinter.sol (line 173)

```
            nonReentrant
```

2. File: src/CitadelMinter.sol (line 254)

```
            nonReentrant
```

3. File: src/CitadelMinter.sol (line 298)

```
        ) external onlyRole(POLICY_OPERATIONS_ROLE) gacPausable nonR
```

4. File: src/Funding.sol (line 167)

```
            nonReentrant
```

5. File: src/Funding.sol (line 318)

```
            nonReentrant
```

6. File: src/KnightingRound.sol (line 402)

```
        function sweep(address _token) external gacPausable nonReent
```

[N-10] Solidity versions greater than the current version should not be included in the pragma range

The below pragmas should be `<` 0.9.0, not `<=`

```
$ grep '<= 0.9.0' src/*/*/*
src/interfaces/badger/IBadgerGuestlist.sol:pragma solidity >= 0.
```

```
src/interfaces/badger/IBadgerVipGuestlist.sol:pragma solidity >=
src/interfaces/badger/IEmptyStrategy.sol:pragma solidity >= 0.5.
src/interfaces/badger/IStrategy.sol:pragma solidity >= 0.5.0 <=
src/interfaces/badger/IVault.sol:pragma solidity >= 0.5.0 <= 0.9
src/interfaces/citadel/ICitadelToken.sol:pragma solidity >= 0.5.
src/interfaces/citadel/IGac.sol:pragma solidity >= 0.5.0 <= 0.9.
src/interfaces/citadel/IStakedCitadelLocker.sol:pragma solidity
src/interfaces/citadel/ISupplySchedule.sol:pragma solidity >= 0.
src/interfaces/citadel/IVesting.sol:pragma solidity >= 0.5.0 <=
src/interfaces/convex/BoringMath.sol:pragma solidity >= 0.5.0 <=
src/interfaces/convex/IRewardStaking.sol:pragma solidity >= 0.5.
src/interfaces/convex/IStakingProxy.sol:pragma solidity >= 0.5.0
src/interfaces/convex/MathUtil.sol:pragma solidity >= 0.5.0 <= 0
src/interfaces/erc20/IERC20.sol:pragma solidity >= 0.5.0 <= 0.9.
```

[N-11] Adding a `return` statement when the function defines a named return variable, is redundant

1. File: external/StakedCitadelLocker.sol (line 272)

```
        return userRewards;
```

2. File: external/StakedCitadelLocker.sol (line 311)

```
        return amount;
```

3. File: external/StakedCitadelLocker.sol (line 338)

```
        return amount;
```

4. File: external/StakedCitadelLocker.sol (line 399)

```
        return supply;
```

5. File: external/StakedCitadelLocker.sol (line 417)

```
return supply;
```

[N-12] `require()` / `revert()` statements should have descriptive reason strings

1. File: external/MedianOracle.sol (line 68)

```
require(reportExpirationTimeSec_ <= MAX_REPORT_EXPIRATI(
```

2. File: external/MedianOracle.sol (line 69)

```
require(minimumProviders_ > 0);
```

3. File: external/MedianOracle.sol (line 84)

```
require(reportExpirationTimeSec_ <= MAX_REPORT_EXPIRATI(
```

4. File: external/MedianOracle.sol (line 109)

```
require(minimumProviders_ > 0);
```

5. File: external/MedianOracle.sol (line 123)

```
require(timestamps[0] > 0);
```

6. File: external/MedianOracle.sol (line 129)

```
require(timestamps[index_recent].add(reportDelaySec) <=
```

7. File: external/MedianOracle.sol (line 143)

```
        require (providerReports[providerAddress][0].timestamp >
```

8. File: external/MedianOracle.sol (line 211)

```
        require(providerReports[provider][0].timestamp == 0);
```

9. File: external/StakedCitadelLocker.sol (line 126)

```
        require(_stakingToken != address(0)); // dev: _stakingTo
```

10. File: external/StakedCitadelLocker.sol (line 163)

```
        require(rewardData[_rewardsToken].lastUpdateTime == 0);
```

11. File: external/StakedCitadelLocker.sol (line 178)

```
        require(rewardData[_rewardsToken].lastUpdateTime > 0);
```

12. File: external/StakedCitadelLocker.sol (line 812)

```
        require(rewardDistributors[_rewardsToken][msg.sender]);
```

13. File: src/lib/GlobalAccessControlManaged.sol (line 81)

```
        require(gac.hasRole(PAUSER_ROLE, msg.sender));
```

14. File: src/lib/GlobalAccessControlManaged.sol (line 86)

```
        require(gac.hasRole(UNPAUSER_ROLE, msg.sender));
```

15. File: src/StakedCitadel.sol (line 180)

```
        require(_token != address(0)); // dev: _token address sh
```

16. File: src/StakedCitadel.sol (line 181)

```
        require(_governance != address(0)); // dev: _governance
```

17. File: src/StakedCitadel.sol (line 182)

```
        require(_keeper != address(0)); // dev: _keeper address
```

18. File: src/StakedCitadel.sol (line 183)

```
        require(_guardian != address(0)); // dev: _guardian addr
```

19. File: src/StakedCitadel.sol (line 184)

```
        require(_treasury != address(0)); // dev: _treasury addr
```

20. File: src/StakedCitadel.sol (line 185)

```
        require(_strategist != address(0)); // dev: _strategist
```

21. File: src/StakedCitadel.sol (line 186)

```
        require(_badgerTree != address(0)); // dev: _badgerTree
```

22. File: src/StakedCitadel.sol (line 187)

```
        require(_vesting != address(0)); // dev: _vesting addres
```

[N-13] `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`.

1. File: external/StakedCitadelLocker.sol (lines 121-125)

```
    function initialize(
        address _stakingToken,
        string calldata name,
        string calldata symbol
    ) public initializer {
```

2. File: external/StakedCitadelLocker.sol (line 142)

```
    function decimals() public view returns (uint8) {
```

3. File: external/StakedCitadelLocker.sol (line 145)

```
    function name() public view returns (string memory) {
```

4. File: external/StakedCitadelLocker.sol (line 148)

```
    function symbol() public view returns (string memory) {
```

5. File: external/StakedCitadelLocker.sol (line 151)

```
    function version() public view returns(uint256){
```

6. File: external/StakedCitadelLocker.sol (lines 158-162)

```
function addReward(
    address _rewardsToken,
    address _distributor,
    bool _useBoost
) public onlyOwner {
```

7. File: external/StakedCitadelLocker.sol (line 250)

```
function lastTimeRewardApplicable(address _rewardsToken) puk
```

8. File: src/CitadelToken.sol (lines 22-26)

```
function initialize(
    string memory _name,
    string memory _symbol,
    address _gac
) public initializer {
```

9. File: src/Funding.sol (line 223)

```
function getStakedCitadelAmountOut(uint256 _assetAmountIn) p
```

10. File: src/lib/GlobalAccessControlManaged.sol (lines 27-29)

```
function __GlobalAccessControlManaged_init(address _globalAc
    public
    onlyInitializing
```

11. File: src/lib/SettAccessControl.sol (line 51)

```
function setGovernance(address _governance) public {
```

12. File: src/StakedCitadel.sol (lines 167-179)

```
function initialize(
    address _token,
    address _governance,
    address _keeper,
    address _guardian,
    address _treasury,
    address _strategist,
    address _badgerTree,
    address _vesting,
    string memory _name,
    string memory _symbol,
    uint256[4] memory _feeConfig
) public initializer whenNotPaused {
```

13. File: src/StakedCitadel.sol (line 284)

```
function getPricePerFullShare() public view returns (uint25
```

14. File: src/SupplySchedule.sol (line 43)

```
function initialize(address _gac) public initializer {
```

15. File: src/SupplySchedule.sol (line 79)

```
function getEmissionsForCurrentEpoch() public view returns
```

[N-14] `constant` s should be defined rather than using magic numbers

1. File: external/StakedCitadelLocker.sol (line 131)

```
    _decimals = 18;
```

2. File: external/StakedCitadelLocker.sol (line 201)

```
        require(_max < 1500, "over max payment"); //max 15%
```

3. File: external/StakedCitadelLocker.sol (line 202)

```
        require(_rate < 30000, "over max rate"); //max 3x
```

4. File: external/StakedCitadelLocker.sol (line 211)

```
        require(_rate <= 500, "over max rate"); //max 5% per epc
```

5. File: external/StakedCitadelLocker.sol (line 232)

```
                rewardData[_rewardsToken].rewardRate).mul(1e18).
```

6. File: external/StakedCitadelLocker.sol (line 243)

```
        ).div(1e18).add(rewards[_user][_rewardsToken]);
```

7. File: external/StakedCitadelLocker.sol (line 428)

```
        for (uint256 i = 0; i < 128; i++) {
```

8. File: src/CitadelMinter.sol (line 272)

```
            require(_weight <= 10000, "exceed max funding pool w
```

9. File: src/StakedCitadel.sol (line 178)

```
        uint256[4] memory _feeConfig
```

10. File: src/StakedCitadel.sol (line 203)

```
            _feeConfig[3] <= MANAGEMENT_FEE_HARD_CAP,
```

11. File: src/StakedCitadel.sol (line 250)

```
        managementFee = _feeConfig[3];
```

12. File: src/StakedCitadel.sol (line 255)

```
        toEarnBps = 9_500; // initial value of toEarnBps // 95%
```

13. File: src/SupplySchedule.sol (line 170)

```
        epochRate[0] = 593962000000000000000000 / epochLength;
```

14. File: src/SupplySchedule.sol (line 171)

```
        epochRate[1] = 591445000000000000000000 / epochLength;
```

15. File: src/SupplySchedule.sol (line 172)

```
        epochRate[2] = 585021000000000000000000 / epochLength;
```

16. File: src/SupplySchedule.sol (line 173)

```
        epochRate[3] = 574138000000000000000000 / epochLength;
```

17. File: src/SupplySchedule.sol (line 173)

```
            epochRate[3] = 574138000000000000000000 / epochLength;
```

18. File: src/SupplySchedule.sol (line 174)

```
            epochRate[4] = 558275000000000000000000 / epochLength;
```

19. File: src/SupplySchedule.sol (line 174)

```
            epochRate[4] = 558275000000000000000000 / epochLength;
```

20. File: src/SupplySchedule.sol (line 175)

```
            epochRate[5] = 536986000000000000000000 / epochLength;
```

21. File: src/SupplySchedule.sol (line 175)

```
            epochRate[5] = 536986000000000000000000 / epochLength;
```

[N-15] Numeric values having to do with time should use time units for readability

There are units for seconds, minutes, hours, days, and weeks

1. File: external/StakedCitadelLocker.sol (line 70)

```
    uint256 public constant rewardsDuration = 86400; // 1 day
```

2. File: external/StakedCitadelLocker.sol (line 70)

```
uint256 public constant rewardsDuration = 86400; // 1 day
```

3. File: src/StakedCitadelVester.sol (line 34)

```
uint256 public constant INITIAL_VESTING_DURATION = 86400 * 2
```

4. File: src/StakedCitadelVester.sol (line 34)

```
uint256 public constant INITIAL_VESTING_DURATION = 86400 * 2
```

[N-16] Constant redefined elsewhere

Consider defining in only one contract so that values cannot become out of sync when only one location is updated

1. File: src/Funding.sol (lines 21-22)

```
bytes32 public constant CONTRACT_GOVERNANCE_ROLE =
    keccak256("CONTRACT_GOVERNANCE_ROLE");
```

seen in src/CitadelMinter.sol

2. File: src/Funding.sol (lines 23-24)

```
bytes32 public constant POLICY_OPERATIONS_ROLE =
    keccak256("POLICY_OPERATIONS_ROLE");
```

seen in src/CitadelMinter.sol

3. File: src/GlobalAccessControl.sol (lines 25-26)

```
bytes32 public constant CONTRACT_GOVERNANCE_ROLE =
    keccak256("CONTRACT_GOVERNANCE_ROLE");
```

seen in src/Funding.sol

4. File: src/GlobalAccessControl.sol (lines 32-33)

```
bytes32 public constant POLICY_OPERATIONS_ROLE =
    keccak256("POLICY_OPERATIONS_ROLE");
```

seen in src/Funding.sol

5. File: src/GlobalAccessControl.sol (lines 34-35)

```
bytes32 public constant TREASURY_OPERATIONS_ROLE =
    keccak256("TREASURY_OPERATIONS_ROLE");
```

seen in src/Funding.sol

6. File: src/GlobalAccessControl.sol (line 37)

```
bytes32 public constant KEEPER_ROLE = keccak256("KEEPER_ROLE
```

seen in src/Funding.sol

7. File: src/GlobalAccessControl.sol (lines 46-47)

```
bytes32 public constant CITADEL_MINTER_ROLE =
    keccak256("CITADEL_MINTER_ROLE");
```

seen in src/CitadelToken.sol

8. File: src/KnightingRound.sol (lines 19-20)

```
    bytes32 public constant CONTRACT_GOVERNANCE_ROLE =
        keccak256("CONTRACT_GOVERNANCE_ROLE");
```

seen in src/GlobalAccessControl.sol

9. File: src/KnightingRound.sol (lines 21-22)

```
    bytes32 public constant TREASURY_GOVERNANCE_ROLE =
        keccak256("TREASURY_GOVERNANCE_ROLE");
```

seen in src/GlobalAccessControl.sol

10. File: src/KnightingRound.sol (lines 24-25)

```
    bytes32 public constant TECH_OPERATIONS_ROLE =
        keccak256("TECH_OPERATIONS_ROLE");
```

seen in src/GlobalAccessControl.sol

11. File: src/KnightingRound.sol (lines 26-27)

```
    bytes32 public constant TREASURY_OPERATIONS_ROLE =
        keccak256("TREASURY_OPERATIONS_ROLE");
```

seen in src/GlobalAccessControl.sol

12. File: src/lib/GlobalAccessControlManaged.sol (line 15)

```
    bytes32 public constant PAUSER_ROLE = keccak256("PAUSER_ROL
```

seen in src/GlobalAccessControl.sol

13. File: src/lib/GlobalAccessControlManaged.sol (line 16)

```
bytes32 public constant UNPAUSER_ROLE = keccak256("UNPAUSER_
```

seen in src/GlobalAccessControl.sol

14. File: src/StakedCitadel.sol (line 112)

```
uint256 public constant MAX_BPS = 10_000;
```

seen in src/Funding.sol

15. File: src/StakedCitadelVester.sol (lines 20-21)

```
bytes32 public constant CONTRACT_GOVERNANCE_ROLE =
    keccak256("CONTRACT_GOVERNANCE_ROLE");
```

seen in src/KnightingRound.sol

16. File: src/SupplySchedule.sol (lines 22-23)

```
bytes32 public constant CONTRACT_GOVERNANCE_ROLE =
    keccak256("CONTRACT_GOVERNANCE_ROLE");
```

seen in src/StakedCitadelVester.sol

[N-17] Non-library/interface files should use fixed compiler versions, not floating ones

1. File: src/CitadelToken.sol (line 2)

```
pragma solidity ^0.8.0;
```

2. File: src/GlobalAccessControl.sol (line 3)

```
pragma solidity ^0.8.0;
```

3. File: src/lib/GlobalAccessControlManaged.sol (line 3)

```
pragma solidity ^0.8.12;
```

[N-18] Typos

1. File: external/StakedCitadelLocker.sol (line 300)

```
                    //stop now as no futher checks are needed
```

futher

2. File: src/CitadelMinter.sol (line 102)

```
        * @dev this contract is intended to be the only way citadel
```

expection

3. File: src/Funding.sol (line 289)

```
        * @param _assetCap New max cumulatiive amountIn
```

cumulatiive

4. File: src/Funding.sol (line 333)

```
       /// @dev We let assets accumulate and batch transfer to trea
```

deposi)t

5. File: src/KnightingRound.sol (line 342)

```
         * @notice Update the `tokenIn` receipient address. Can only
```

receipient

6. File: src/lib/GlobalAccessControlManaged.sol (line 24)

```
         * @dev this is assumed to be used in the initializer of the
```

inhereiting

7. File: src/lib/GlobalAccessControlManaged.sol (line 60)

```
        // @dev used to faciliate extra contract-specific permission
```

faciliate

8. File: src/StakedCitadel.sol (line 81)

```
        address public badgerTree; // Address we send tokens too via
```

too -> to

[N-19] File does not contain an SPDX Identifier

1. File: external/MedianOracle.sol (line 0)

```
    pragma solidity 0.4.24;
```

[N-20] File is missing NatSpec

1. File: external/StakedCitadelLocker.sol (line 0)

```
// SPDX-License-Identifier: MIT
```

2. File: src/interfaces/badger/IBadgerGuestlist.sol (line 0)

```
// SPDX-License-Identifier: MIT
```

3. File: src/interfaces/badger/IBadgerVipGuestlist.sol (line 0)

```
// SPDX-License-Identifier: MIT
```

4. File: src/interfaces/badger/IEmptyStrategy.sol (line 0)

```
// SPDX-License-Identifier: MIT
```

5. File: src/interfaces/badger/IStrategy.sol (line 0)

```
// SPDX-License-Identifier: MIT
```

6. File: src/interfaces/badger/IVault.sol (line 0)

```
// SPDX-License-Identifier: MIT
```

7. File: src/interfaces/citadel/ICitadelToken.sol (line 0)

```
// SPDX-License-Identifier: MIT
```

8. File: src/interfaces/citadel/IGac.sol (line 0)

```
/// SPDX-License-Identifier: MIT
```

9. File: src/interfaces/citadel/IMedianOracle.sol (line 0)

```
/// SPDX-License-Identifier: MIT
```

10. File: src/interfaces/citadel/IStakedCitadelLocker.sol (line 0)

```
// SPDX-License-Identifier: MIT
```

11. File: src/interfaces/citadel/ISupplySchedule.sol (line 0)

```
// SPDX-License-Identifier: MIT
```

12. File: src/interfaces/citadel/IVesting.sol (line 0)

```
// SPDX-License-Identifier: MIT
```

13. File: src/interfaces/convex/IRewardStaking.sol (line 0)

```
// SPDX-License-Identifier: MIT
```

14. File: src/interfaces/convex/IStakingProxy.sol (line 0)

```
// SPDX-License-Identifier: MIT
```

[N-21] NatSpec is incorrect

Wrong parameter description

1. File: src/Funding.sol (line 160)

```
     * @param _minCitadelOut ID of DAO to vote for
```

[N-22] NatSpec is incomplete

1. File: src/Funding.sol (lines [95-112](#))

```
    /**
     * @notice Initializer.
     * @param _gac Global access control
     * @param _citadel The token this contract will return in a
     * @param _asset The token this contract will receive in a t
     * @param _xCitadel Staked citadel, citadel will be granted
     * @param _saleRecipient The address receiving the proceeds
     * @param _assetCap The max asset that the contract can take
     */
    function initialize(
        address _gac,
        address _citadel,
        address _asset,
        address _xCitadel,
        address _saleRecipient,
        address _citadelPriceInAssetOracle,
        uint256 _assetCap
    ) external initializer {
```

Missing: `@param _citadelPriceInAssetOracle`

2. File: src/KnightingRound.sol (lines [98-119](#))

```
    /**
     * @notice Initializer.
     * @param _tokenOut The token this contract will return in a
     * @param _tokenIn The token this contract will receive in a
     * @param _saleStart The time when tokens can be first purch
     * @param _saleDuration The duration of the token sale
     * @param _tokenOutPrice The tokenOut per tokenIn price
     * @param _saleRecipient The address receiving the proceeds
     * @param _guestlist Address that will manage auction approv
     * @param _tokenInLimit The max tokenIn that the contract ca
     */
    function initialize(
        address _globalAccessControl,
        address _tokenOut,
        address _tokenIn,
        uint256 _saleStart,
        uint256 _saleDuration,
        uint256 _tokenOutPrice,
        address _saleRecipient,
        address _guestlist,
        uint256 _tokenInLimit
    ) external initializer {
```

Missing: `@param _globalAccessControl`

3. File: src/StakedCitadel.sol (lines 154-179)

```
    /// @notice Initializes the Sett. Can only be called once, i
    /// @param _token Address of the token that can be depositec
    /// @param _governance Address authorized as governance.
    /// @param _keeper Address authorized as keeper.
    /// @param _guardian Address authorized as guardian.
    /// @param _treasury Address to distribute governance fees/r
    /// @param _strategist Address authorized as strategist.
    /// @param _badgerTree Address of badgerTree used for emissi
    /// @param _name Specify a custom sett name. Leave empty for
    /// @param _symbol Specify a custom sett symbol. Leave empty
    /// @param _feeConfig Values for the 4 different types of fe
    ///           [performanceFeeGovernance, performanceFeeStrateg
    ///           Each fee should be less than the constant hard-c
    function initialize(
        address _token,
        address _governance,
        address _keeper,
        address _guardian,
        address _treasury,
        address _strategist,
        address _badgerTree,
        address _vesting,
        string memory _name,
        string memory _symbol,
        uint256[4] memory _feeConfig
    ) public initializer whenNotPaused {
```

Missing: `@param _vesting`

4. File: src/StakedCitadel.sol (lines 357-367)

```
/// @notice Deposits `_amount` tokens, issuing shares to `re
///         Checks the guestlist to verify that `recipient`
///         Note that deposits are not accepted when the Set
/// @dev See `_depositForWithAuthorization` for details on g
/// @param _recipient Address to issue the Sett shares to.
/// @param _amount Quantity of tokens to deposit.
function depositFor(
    address _recipient,
    uint256 _amount,
    bytes32[] memory proof
) external whenNotPaused {
```

Missing: `@param proof`

[N-23] Event is missing `indexed` fields

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

See original submission for details.

[N-24] Non-exploitable reentrancies

```
Reentrancy in CitadelMinter.mintAndDistribute() (src/CitadelMint
External calls:
- citadelToken.mint(address(this),mintable) (src/CitadelMinter.s
- IVault(cachedXCitadel).deposit(lockingAmount) (src/CitadelMint
- xCitadelLocker.notifyRewardAmount(address(cachedXCitadel),xCit
- IERC20Upgradeable(address(citadelToken)).safeTransfer(address(
- _transferToFundingPools(fundingAmount) (src/CitadelMinter.sol#
- returndata = address(token).functionCall(data,SafeERC20: low-l
- (success,returndata) = target.call{value: value}(data) (node_m
- IERC20Upgradeable(address(citadelToken)).safeTransfer(pool,amo
External calls sending eth:
- _transferToFundingPools(fundingAmount) (src/CitadelMinter.sol#
- (success,returndata) = target.call{value: value}(data) (node_m
State variables written after the call(s):
- lastMintTimestamp = block.timestamp (src/CitadelMinter.sol#24(



Reentrancy in StakedCitadel._withdraw(uint256) (src/StakedCitade
External calls:
- IStrategy(strategy).withdraw(_toWithdraw) (src/StakedCitadel.s
- IVesting(vesting).setupVesting(msg.sender,_amount,block.timest
- token.safeTransfer(vesting,_amount) (src/StakedCitadel.sol#831
State variables written after the call(s):
- _mintSharesFor(treasury,_fee,balance() - _fee) (src/StakedCita
- _balances[account] += amount (node_modules/@openzeppelin/contr
- _mintSharesFor(treasury,_fee,balance() - _fee) (src/StakedCita
- _totalSupply += amount (node_modules/@openzeppelin/contracts-u



Reentrancy in StakedCitadel._depositFor(address,uint256) (src/St
External calls:
- token.safeTransferFrom(msg.sender,address(this),_amount) (src/
State variables written after the call(s):
- _mintSharesFor(_recipient,_after - _before,_pool) (src/StakedC
- _balances[account] += amount (node_modules/@openzeppelin/contr
- _mintSharesFor(_recipient,_after - _before,_pool) (src/StakedC
- _totalSupply += amount (node_modules/@openzeppelin/contracts-u
Reentrancy in Funding.updateCitadelPriceInAsset() (src/Funding.s
External calls:
- (_citadelPriceInAsset,_valid) = IMedianOracle(citadelPriceInAs
State variables written after the call(s):
- citadelPriceFlag = true (src/Funding.sol#431-432)
- citadelPriceInAsset = _citadelPriceInAsset (src/Funding.sol#43
```

[N-25] `now` is deprecated

Use `block.timestamp` instead

1. File: external/MedianOracle.sol (line 129)

```
        require(timestamps[index_recent].add(reportDelaySec) <=
```

2. File: external/MedianOracle.sol (line 131)

```
            reports[index_past].timestamp = now;
```

3. File: external/MedianOracle.sol (line 134)

```
            emit ProviderReportPushed(providerAddress, payload, now)
```

4. File: external/MedianOracle.sol (line 161)

```
            uint256 minValidTimestamp =  now.sub(reportExpirationTim
```

5. File: external/MedianOracle.sol (line 162)

```
            uint256 maxValidTimestamp =  now.sub(reportDelaySec);
```

Gas Optimizations

For this contest, 48 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **Dravee** received the top score from the judge.

The following wardens also submitted reports: IllIllI, joestakey, TomFrenchBlockchain, defsec, rfa, Tomio, 0xkatana, fatherOfBlocks, saian, sorrynotsorry, TerrierLover, TrungOre, CertoraInc, 0x1f8b, 0xAsm0d3us, 0xNazgul, gzeon, joshie, kenta, robee, horsefacts, ilan, securerodd, slywaters, tchkvsky, 0v3rf10w, berndartmueller, Cityscape, ellahi, gs8nrv, simon135, SolidityScan, teryanarmen, z3s, 0xBug, 0xDjango, Funen, jah, kebabsec, MaratCerby, oyc_109, bae11, csanuragjain, delfin454000, Hawkeye, nahnah, *and* rayn.

Table of Contents

See original submission.

[G-01] `CitadelMinter.mintAndDistribute()` : L199 should be unchecked due to L193-L197

Solidity version 0.8+ comes with implicit overflow and underflow checks on unsigned integers. When an overflow or an underflow isn't possible (as an example, when a comparison is made before the arithmetic operation), some gas can be saved by using an `unchecked` block:
https://docs.soliditylang.org/en/v0.8.10/control-structures.html#checked-or-unchecked-arithmetic

I suggest wrapping with an `unchecked` block here (see `@audit` tag):

```
File: CitadelMinter.sol
169:     function mintAndDistribute()
...
193:             uint256 beforeAmount = cachedXCitadel.balan
194:
195:             IVault(cachedXCitadel).deposit(lockingAmoun
196:
197:             uint256 afterAmount = cachedXCitadel.balanc
198:
199:             uint256 xCitadelToLockers = afterAmount - b
```

[G-02] `Funding.sol` : state variables can be tightly packed to save 1 storage slot

From (see `@audit` tags):

```
File: Funding.sol
38:     uint256 public maxCitadelPriceInAsset;  //@audit gas: 32
39:     bool public citadelPriceFlag; //@audit gas: 1 byte size,
40:
41:     uint256 public assetDecimalsNormalizationValue; //@audit
42:
43:     address public citadelPriceInAssetOracle; //@audit gas:
44:     address public saleRecipient; //@audit gas: 20 bytes siz
45:
```

to:

```
File: Funding.sol
      uint256 public maxCitadelPriceInAsset;  //@audit gas: 32
      bool public citadelPriceFlag; //@audit gas: 1 byte size,

      address public citadelPriceInAssetOracle; //@audit gas: 2
      address public saleRecipient; //@audit gas: 20 bytes size

      uint256 public assetDecimalsNormalizationValue; //@audit
```

[G-03] `Funding.initialize()` : should use memory instead of storage variable

See `@audit` tag:

```
File: Funding.sol
104:     function initialize(
...
127:         asset = IERC20(_asset);
...
134:         assetDecimalsNormalizationValue = 10**asset.decima
```

[G-04] `Funding.onlyWhenPriceNotFlagged()` : boolean comparison 147

Comparing to a constant (`true` or `false`) is a bit more expensive than directly checking the returned boolean value. I suggest using `if(directValue)` instead of `if(directValue == true)` and `if(!directValue)` instead of `if(directValue == false)` here (see `@audit` tag):

```
File: Funding.sol
145:     modifier onlyWhenPriceNotFlagged() {
146:         require(
147:             citadelPriceFlag == false, //@audit gas: inste
148:             "Funding: citadel price from oracle flagged an
149:         );
150:         _;
151:     }
```

[G-05] `Funding.deposit()` : `funding.assetCumulativeFunded + _assetAmountIn` should get cached

See `@audit` tags:

```
File: Funding.sol
163:     function deposit(uint256 _assetAmountIn, uint256 _minCi
164:         external
165:         onlyWhenPriceNotFlagged
166:         gacPausable
167:         nonReentrant
168:         returns (uint256 citadelAmount_)
169:     {
170:         require(_assetAmountIn > 0, "_assetAmountIn must nc
171:         require(
172:             funding.assetCumulativeFunded + _assetAmountIn
173:             "asset funding cap exceeded"
174:         );
175:         funding.assetCumulativeFunded = funding.assetCumula
```

[G-06] `Funding.getRemainingFundable()` : L236 should be unchecked due to L235

See `@audit` tag:

```
    File: Funding.sol
    232:     function getRemainingFundable() external view retur
    233:         uint256 assetCumulativeFunded = funding.assetCu
    234:         uint256 assetCap = funding.assetCap;
    235:         if (assetCumulativeFunded < assetCap) {
235 236:             limitLeft_ = assetCap - assetCumulativeFund
236 237:         }
237 238:     }
```

[G-07] `Funding.claimAssetToTreasury()` : `asset` should get cached

See `@audit` tag:

```
File: Funding.sol
334:     function claimAssetToTreasury()
335:         external
336:         gacPausable
337:         onlyRole(TREASURY_OPERATIONS_ROLE)
338:     {
339:         uint256 amount = asset.balanceOf(address(this)); //
340:         require(amount > 0, "nothing to claim");
341:         asset.safeTransfer(saleRecipient, amount);//@audit
342: 
343:         emit ClaimToTreasury(address(asset), amount);//@auc
344:     }
```

[G-08] `KnightingRound.initialize()` : should use memory instead of storage variable

See `@audit` tag:

```
File: KnightingRound.sol
109:     function initialize(
...
140:         tokenIn = ERC20Upgradeable(_tokenIn);
...
148:         tokenInNormalizationValue = 10**tokenIn.decimals();
```

[G-09] `KnightingRound.buy()` : `saleStart` , `totalTokenIn` and `guestlist` should get cached

See `@audit` tags:

```
File: KnightingRound.sol
162:     function buy(
...
167:         require(saleStart <= block.timestamp, "KnightingRou
168:         require(
169:             block.timestamp < saleStart + saleDuration, //@
```

```
...
173:         require(
174:             totalTokenIn + _tokenInAmount <= tokenInLimit,
...
178:         if (address(guestlist) != address(0)) { //@audit ga
179:             require(guestlist.authorized(msg.sender, _proof
180:         }
...
198:         totalTokenIn = totalTokenIn + _tokenInAmount;  //@a
```

[G-10] `KnightingRound.getTokenInLimitLeft()` : `totalTokenIn` and `tokenInLimit` should get cached + L250 should be unchecked due to L249

See `@audit` tags:

```
File: KnightingRound.sol
248:     function getTokenInLimitLeft() external view returns (u
249:         if (totalTokenIn < tokenInLimit) { //@audit gas: sh
250:             limitLeft_ = tokenInLimit - totalTokenIn; //@au
251:         }
252:     }
```

[G-11] `StakedCitadel.deposit()` : Use `calldata` instead of `memory`

When arguments are read-only on external functions, the data location should be `calldata` :

```
File: StakedCitadel.sol
319:     function deposit(uint256 _amount, bytes32[] memory proo
320:         external
321:         whenNotPaused
322:     {
323:         _depositWithAuthorization(_amount, proof);
324:     }
```

[G-12] `StakedCitadel.depositAll()`: Use `calldata` instead of `memory`

See `@audit` tag:

```
File: StakedCitadel.sol
319:     function deposit(uint256 _amount, bytes32[] memory proc
320:         external
321:         whenNotPaused
322:     {
323:         _depositWithAuthorization(_amount, proof);
324:     }
```

[G-13] `StakedCitadel.setStrategy()`: `strategy` should get cached

See `@audit` tags:

```
File: StakedCitadel.sol
500:     function setStrategy(address _strategy) external whenNc
...
505:         if (strategy != address(0)) { //@audit gas: should
506:             require(
507:                 IStrategy(strategy).balanceOf() == 0, //@au
```

[G-14] `StakedCitadel.earn()`: `strategy` should get cached

See `@audit` tags:

```
File: StakedCitadel.sol
717:     function earn() external {
...
722:         token.safeTransfer(strategy, _bal); //@audit gas: s
723:         IStrategy(strategy).earn();//@audit gas: should use
724:     }
```

[G-15] `StakedCitadel._depositFor()`: `token` should get cached

See `@audit` tags:

```
File: StakedCitadel.sol
764:     function _depositFor(address _recipient, uint256 _amour
...
773:         uint256 _before = token.balanceOf(address(this)); /
774:         token.safeTransferFrom(msg.sender, address(this), _
775:         uint256 _after = token.balanceOf(address(this));//@
```

[G-16] `StakedCitadel._depositFor()`: L776 should be unchecked due to L773-L775

See `@audit` tags:

```
    File: StakedCitadel.sol
    764:     function _depositFor(address _recipient, uint256 _ar
    ...
    773:         uint256 _before = token.balanceOf(address(this)
    774:         token.safeTransferFrom(msg.sender, address(this
    775:         uint256 _after = token.balanceOf(address(this))
775 776:         _mintSharesFor(_recipient, _after - _before, _p
776 777:     }
```

[G-17] `StakedCitadel._depositForWithAuthorization()`: `guestList` should get cached

See `@audit` tags:

```
File: StakedCitadel.sol
788:     function _depositForWithAuthorization(
...
793:         if (address(guestList) != address(0)) {//@audit gas
794:             require(
795:                 guestList.authorized(_recipient, _amount, p
```

[G-18] `StakedCitadel._withdraw()` : `token` and `vesting` should get cached

See `@audit` tags:

```
File: StakedCitadel.sol
808:     function _withdraw(uint256 _shares) internal nonReentra
...
815:         uint256 b = token.balanceOf(address(this)); //@audi
...
819:             uint256 _after = token.balanceOf(address(this))
...
830:         IVesting(vesting).setupVesting(msg.sender, _amount,
831:         token.safeTransfer(vesting, _amount);  //@audit gas
```

[G-19] `StakedCitadel._withdraw()` : L817 should be unchecked due to L816

See `@audit` tag:

```
    File: StakedCitadel.sol
    808:     function _withdraw(uint256 _shares) internal nonRee
    ...
    816:         if (b < r) {
816 817:             uint256 _toWithdraw = r - b;
```

[G-20] `StakedCitadelLocker.sol` : state variables can be tightly packed to save 1 storage slot

From (see `@audit` tags):

```
File: StakedCitadelLocker.sol
109:     uint256 public kickRewardEpochDelay = 4;
110:
111:     //shutdown
112:     bool public isShutdown = false; //@audit gas: can be ti
113:
    //erc20-like interface
    string private _name;
    string private _symbol;
    uint8 private _decimals;
```

to:

```
    uint256 public kickRewardEpochDelay = 4;

    //erc20-like interface
    string private _name;
    string private _symbol;
    uint8 private _decimals;

    //shutdown
    bool public isShutdown = false;
```

[G-21] `StakedCitadelLocker.totalSupplyAtEpoch()` : Use a storage variable's reference instead of repeatedly fetching it (`epochs[i]`)

See `@audit` tag:

```
File: StakedCitadelLocker.sol
    function totalSupplyAtEpoch(uint256 _epoch) view exterr
...
        for (uint i = _epoch; i + 1 != 0; i--) {
            Epoch storage e = epochs[i];
            if (uint256(e.date) <= cutoffEpoch) {
                break;
            }
            supply = supply.add(epochs[i].supply); //@audit
```

[G-22] `StakedCitadel._withdraw()` : `maximumStake` , `minimumStake` and `stakingProxy` should get cached

See `@audit` tags:

```
File: StakedCitadelLocker.sol
    function updateStakeRatio(uint256 _offset) internal {
...
        uint256 mean = maximumStake.add(minimumStake).div(2
        uint256 max = maximumStake.add(_offset); //@audit g
        uint256 min = MathUpgradeable.min(minimumStake, mir
        if (ratio > max) {
...
        } else if (ratio < min) {
...
            stakingToken.safeTransfer(stakingProxy, increas
            IStakingProxy(stakingProxy).stake(); //@audit g
        }
    }
```

[G-23] `StakedCitadelVester.claimableBalance()` : Help the optimizer by saving a storage variable's reference instead of repeatedly fetching it (`vesting[recipient]`)

To help the optimizer, declare a `storage` type variable and use it instead of repeatedly fetching the reference in a map or an array.

The effect can be quite significant.

Here, instead of repeatedly calling `vesting[recipient]` , save its reference like this: `VestingParams storage _vestingParams = vesting[recipient]` and use it.

Impacted lines (see `@audit` tags):

```
File: StakedCitadelVester.sol
    function claimableBalance(address recipient) public vie
        uint256 locked = vesting[recipient].lockedAmounts;
        uint256 claimed = vesting[recipient].claimedAmounts
        if (block.timestamp >= vesting[recipient].unlockEnc
            return locked - claimed;
        }
        return
            ((locked * (block.timestamp - vesting[recipient
                (vesting[recipient].unlockEnd - //@audit ga
                                 vesting[recipient].unlockBegin)) - clai
    }
```

[G-24] `StakedCitadelVester.vest()` : Help the optimizer by saving a storage variable's reference instead of repeatedly fetching it (`vesting[recipient]`)

Just like in `StakedCitadelVester.claimableBalance()` above:

```
File: StakedCitadelVester.sol
    function vest(
...
        vesting[recipient].lockedAmounts = //@audit gas: he
            vesting[recipient].lockedAmounts + //@audit gas
            _amount;
        vesting[recipient].unlockBegin = _unlockBegin; //@a
        vesting[recipient].unlockEnd = _unlockBegin + vesti

        emit Vest(
            recipient,
            vesting[recipient].lockedAmounts, //@audit gas:
            _unlockBegin,
            vesting[recipient].unlockEnd //@audit gas: use
        );
    }
```

[G-25] `SupplySchedule.getEpochAtTimestamp()` : `globalStartTimestamp` should get cached

See `@audit` tags:

```
File: SupplySchedule.sol
    function getEpochAtTimestamp(uint256 _timestamp)
...
        require(
            globalStartTimestamp > 0, //@audit gas: should c
...
        return (_timestamp - globalStartTimestamp) / epochLe
```

[G-26] `SupplySchedule.getMintable()`: L105-L110 should be unchecked due to L95 and L99-L101

See `@audit` tags:

```
File: SupplySchedule.sol
94:         require(
95:             block.timestamp > lastMintTimestamp,
96:             "SupplySchedule: already minted up to curren
97:         );
...
099:         if (lastMintTimestamp < cachedGlobalStartTimest
100:             lastMintTimestamp = cachedGlobalStartTimest
101:         }
...
101 105:         uint256 startingEpoch = (lastMintTimestamp - ca
102 106:             epochLength;
103 107:
101 108:         uint256 endingEpoch = (block.timestamp - cached
102 109:             epochLength;
103 110:
```

[G-27] `SupplySchedule.getMintableDebug()`: `globalStartTimestamp` should get cached

See `@audit` tags:

```
File: SupplySchedule.sol
178:     function getMintableDebug(uint256 lastMintTimestamp) ex
179:         require(
180:             globalStartTimestamp > 0, //@audit gas: should
...
183:         require(
184:             lastMintTimestamp > globalStartTimestamp, //@au
...
197:         emit log_named_uint("globalStartTimestamp", globalS
...
200:         uint256 startingEpoch = (lastMintTimestamp - global
201:             epochLength;
...
```

```
204:          uint256 endingEpoch = (block.timestamp - globalStar
...
208:          for (uint256 i = startingEpoch; i <= endingEpoch; i
...
211:              uint256 epochStartTime = globalStartTimestamp +
212:              uint256 epochEndTime = globalStartTimestamp +
```

[G-28] `SupplySchedule.getMintableDebug()` : L200-L205 should be unchecked due to L184 and L188

```
    File: SupplySchedule.sol
    178:     function getMintableDebug(uint256 lastMintTimestamp
    ...
    183:         require(
    184:             lastMintTimestamp > globalStartTimestamp, /
    185:             "SupplySchedule: attempting to mint before
    186:         );
    187:         require(
    188:             block.timestamp > lastMintTimestamp,
    189:             "SupplySchedule: already minted up to curre
    190:         );
    ...
184 200:         uint256 startingEpoch = (lastMintTimestamp - gl
185 201:             epochLength;
186 202:         emit log_named_uint("startingEpoch", startingEp
187 203:
188 204:         uint256 endingEpoch = (block.timestamp - global
189 205:             epochLength;
```

[G-29] No need to explicitly initialize variables with default values

If a variable is not set/initialized, it is assumed to have the default value (`0` for `uint`, `false` for `bool`, `address(0)` for address...). Explicitly initializing it with its default value is an anti-pattern and wastes gas.

As an example: `for (uint256 i = 0; i < numIterations; ++i) {` should be replaced with `for (uint256 i; i < numIterations; ++i) {`

Instances include:

```
lib/GlobalAccessControlManaged.sol:47:        bool validRoleFour
lib/GlobalAccessControlManaged.sol:48:        for (uint256 i = (
CitadelMinter.sol:152:        for (uint256 i = 0; i < numPools;
CitadelMinter.sol:180:        uint256 lockingAmount = 0;
CitadelMinter.sol:181:        uint256 stakingAmount = 0;
CitadelMinter.sol:182:        uint256 fundingAmount = 0;
Funding.sol:283:        citadelPriceFlag = false;
MedianOracle.sol:160:        uint256 size = 0;
MedianOracle.sol:164:        for (uint256 i = 0; i < reportsCour
MedianOracle.sol:226:        for (uint256 i = 0; i < providers.l
StakedCitadelLocker.sol:93:    address public boostPayment = adc
StakedCitadelLocker.sol:94:    uint256 public maximumBoostPaymer
StakedCitadelLocker.sol:96:    uint256 public nextMaximumBoostPa
StakedCitadelLocker.sol:104:    address public stakingProxy = ac
StakedCitadelLocker.sol:112:    bool public isShutdown = false;
StakedCitadelLocker.sol:267:        for (uint256 i = 0; i < user
StakedCitadelLocker.sol:423:        uint256 min = 0;
StakedCitadelLocker.sol:428:        for (uint256 i = 0; i < 128;
StakedCitadelLocker.sol:634:        uint256 reward = 0;
StakedCitadelLocker.sol:838:            for (uint i = 0; i < rew
SupplySchedule.sol:103:        uint256 mintable = 0;
SupplySchedule.sol:192:        uint256 mintable = 0;
```

I suggest removing explicit initializations for default values.

[G-30] `> 0` is less efficient than `!= 0` for unsigned integers (with proof)

`!= 0` costs less gas compared to `> 0` for unsigned integers in `require` statements with the optimizer enabled (6 gas)

Proof: While it may seem that `> 0` is cheaper than `!=`, this is only true without the optimizer enabled and outside a require statement. If you enable the optimizer at 10k AND you're in a `require` statement, this will save gas. You can see this tweet for more proofs: https://twitter.com/gzeon/status/1485428085885640706

I suggest changing `> 0` with `!= 0` here:

```
interfaces/convex/BoringMath.sol:20:        require(b > 0, "Bori
interfaces/convex/BoringMath.sol:102:        require(b > 0, "Boı
interfaces/convex/BoringMath.sol:122:        require(b > 0, "Boı
interfaces/convex/BoringMath.sol:142:        require(b > 0, "Boı
CitadelMinter.sol:343:        require(length > 0, "CitadelMinteı
Funding.sol:170:        require(_assetAmountIn > 0, "_assetAmouı
Funding.sol:322:        require(amount > 0, "nothing to sweep");
Funding.sol:340:        require(amount > 0, "nothing to claim");
Funding.sol:424:        require(_citadelPriceInAsset > 0, "citac
Funding.sol:452:        require(_citadelPriceInAsset > 0, "citac
KnightingRound.sol:125:            _saleDuration > 0,
KnightingRound.sol:129:            _tokenOutPrice > 0,
KnightingRound.sol:172:        require(_tokenInAmount > 0, "_tok
KnightingRound.sol:215:        require(tokenOutAmount_ > 0, "not
KnightingRound.sol:313:            _saleDuration > 0,
KnightingRound.sol:332:            _tokenOutPrice > 0,
KnightingRound.sol:411:        require(amount > 0, "nothing to s
MedianOracle.sol:69:        require(minimumProviders_ > 0);
MedianOracle.sol:109:        require(minimumProviders_ > 0);
MedianOracle.sol:123:        require(timestamps[0] > 0);
MedianOracle.sol:143:        require (providerReports[providerAc
StakedCitadelLocker.sol:178:        require(rewardData[_rewardsT
StakedCitadelLocker.sol:526:        require(_amount > 0, "Cannot
StakedCitadelLocker.sol:681:        require(locked > 0, "no exp
StakedCitadelLocker.sol:813:        require(_reward > 0, "No rew
StakedCitadelVester.sol:138:        require(_amount > 0, "Stakec
SupplySchedule.sol:61:            globalStartTimestamp > 0,
SupplySchedule.sol:91:            cachedGlobalStartTimestamp > (
SupplySchedule.sol:180:            globalStartTimestamp > 0,
```

Also, please enable the Optimizer.

[G-31] `>=` is cheaper than `>`

Strict inequalities (`>`) are more expensive than non-strict ones (`>=`). This is due to some supplementary checks (ISZERO, 3 gas)

I suggest using `>=` instead of `>` to avoid some opcodes here:

```
interfaces/convex/MathUtil.sol:12:        return a < b ? a : b;
```

[G-32] Shift Right instead of Dividing by 2

A division by 2 can be calculated by shifting one to the right.

While the `DIV` opcode uses 5 gas, the `SHR` opcode only uses 3 gas. Furthermore, Solidity's division operation also includes a division-by-0 prevention which is bypassed using shifting.

I suggest replacing `/ 2` with `>> 1` here:

```
StakedCitadelLocker.sol:431:            uint256 mid = (min + max
```

[G-33] An array's length should be cached to save gas in for-loops

Reading array length at each iteration of the loop takes 6 gas (3 for mload and 3 to place memory_offset) in the stack.

Caching the array length in the stack saves around 3 gas per iteration.

Here, I suggest storing the array's length in a variable before the for-loop, and use it instead:

```
lib/GlobalAccessControlManaged.sol:48:          for (uint256 i = (
StakedCitadelLocker.sol:267:          for (uint256 i = 0; i < user
StakedCitadelLocker.sol:459:          for (uint i = nextUnlockInde
StakedCitadelLocker.sol:777:          for (uint i; i < rewardToken
StakedCitadelLocker.sol:838:              for (uint i = 0; i < rew
```

[G-34] `++i` costs less gas compared to `i++` or `i += 1`

`++i` costs less gas compared to `i++` or `i += 1` for unsigned integer, as pre-increment is cheaper (about 5 gas per iteration). This statement is true even with the optimizer enabled.

`i++` increments `i` and returns the initial value of `i`. Which means:

```
uint i = 1;
i++; // == 1 but i == 2
```

But `++i` returns the actual incremented value:

```
uint i = 1;
++i; // == 2 and i == 2 too, so no need for a temporary variable
```

In the first case, the compiler has to create a temporary variable (when used) for returning `1` instead of `2`

Instances include:

```
lib/GlobalAccessControlManaged.sol:48:        for (uint256 i = (
CitadelMinter.sol:152:        for (uint256 i = 0; i < numPools;
MedianOracle.sol:164:        for (uint256 i = 0; i < reportsCour
MedianOracle.sol:226:        for (uint256 i = 0; i < providers.l
StakedCitadelLocker.sol:267:        for (uint256 i = 0; i < user
StakedCitadelLocker.sol:296:        for (uint i = nextUnlockInde
StakedCitadelLocker.sol:428:        for (uint256 i = 0; i < 128;
StakedCitadelLocker.sol:459:        for (uint i = nextUnlockInde
StakedCitadelLocker.sol:465:                idx++;
StakedCitadelLocker.sol:659:            for (uint i = nextUnlock
StakedCitadelLocker.sol:676:                nextUnlockIndex++;
StakedCitadelLocker.sol:777:        for (uint i; i < rewardToker
StakedCitadelLocker.sol:838:            for (uint i = 0; i < rew
SupplySchedule.sol:208:        for (uint256 i = startingEpoch; i
```

I suggest using `++i` instead of `i++` to increment the value of an uint variable.

This is already done here:

```
CitadelMinter.sol:344:        for (uint256 i; i < length; ++i) {
```

[G-35] Increments can be unchecked

In Solidity 0.8+, there's a default overflow check on unsigned integers. It's possible to uncheck this in for-loops and save some gas at each iteration, but at the cost of some code readability, as this uncheck cannot be made inline.

ethereum/solidity#10695

Instances include:

```
lib/GlobalAccessControlManaged.sol:48:        for (uint256 i = (
CitadelMinter.sol:152:        for (uint256 i = 0; i < numPools;
CitadelMinter.sol:344:        for (uint256 i; i < length; ++i) {
SupplySchedule.sol:208:        for (uint256 i = startingEpoch; i
```

The code would go from:

```
for (uint256 i; i < numIterations; i++) {
 // ...
}
```

to:

```
for (uint256 i; i < numIterations;) {
 // ...
 unchecked { ++i; }
}
```

The risk of overflow is inexistant for a `uint256` here.

This is already done here:

```
SupplySchedule.sol:122:            unchecked { ++i; }
```

[G-36] Consider making some constants as non-public to save gas

Reducing from `public` to `private` or `internal` can save gas when a constant isn't used outside of its contract. I suggest changing the visibility from `public` to `internal` or `private` here:

```
lib/GlobalAccessControlManaged.sol:15:    bytes32 public constar
lib/GlobalAccessControlManaged.sol:16:    bytes32 public constar
CitadelMinter.sol:30:    bytes32 public constant CONTRACT_GOVERN
CitadelMinter.sol:32:    bytes32 public constant POLICY_OPERATIO
CitadelToken.sol:9:    bytes32 public constant CITADEL_MINTER_RO
Funding.sol:21:    bytes32 public constant CONTRACT_GOVERNANCE_R
Funding.sol:23:    bytes32 public constant POLICY_OPERATIONS_ROI
Funding.sol:25:    bytes32 public constant TREASURY_OPERATIONS_R
Funding.sol:26:    bytes32 public constant TREASURY_VAULT_ROLE =
Funding.sol:28:    bytes32 public constant KEEPER_ROLE = keccak2
Funding.sol:30:    uint256 public constant MAX_BPS = 10000;
GlobalAccessControl.sol:25:    bytes32 public constant CONTRACT_
GlobalAccessControl.sol:27:    bytes32 public constant TREASURY_
GlobalAccessControl.sol:30:    bytes32 public constant TECH_OPER
GlobalAccessControl.sol:32:    bytes32 public constant POLICY_OF
GlobalAccessControl.sol:34:    bytes32 public constant TREASURY_
GlobalAccessControl.sol:37:    bytes32 public constant KEEPER_RO
GlobalAccessControl.sol:39:    bytes32 public constant PAUSER_RO
GlobalAccessControl.sol:40:    bytes32 public constant UNPAUSER_
GlobalAccessControl.sol:42:    bytes32 public constant BLOCKLIST
GlobalAccessControl.sol:44:    bytes32 public constant BLOCKLIST
GlobalAccessControl.sol:46:    bytes32 public constant CITADEL_M
KnightingRound.sol:19:    bytes32 public constant CONTRACT_GOVER
KnightingRound.sol:21:    bytes32 public constant TREASURY_GOVER
KnightingRound.sol:24:    bytes32 public constant TECH_OPERATION
KnightingRound.sol:26:    bytes32 public constant TREASURY_OPERA
MedianOracle.sol:53:    uint256 private constant MAX_REPORT_EXPI
StakedCitadel.sol:112:    uint256 public constant MAX_BPS = 10_0
StakedCitadel.sol:113:    uint256 public constant SECS_PER_YEAR
StakedCitadel.sol:115:    uint256 public constant WITHDRAWAL_FEE
StakedCitadel.sol:116:    uint256 public constant PERFORMANCE_FE
StakedCitadel.sol:117:    uint256 public constant MANAGEMENT_FEE
StakedCitadelLocker.sol:70:    uint256 public constant rewardsDu
StakedCitadelLocker.sol:73:    uint256 public constant lockDurat
StakedCitadelLocker.sol:98:    uint256 public constant denominat
StakedCitadelLocker.sol:105:    uint256 public constant stakeOff
StakedCitadelVester.sol:20:    bytes32 public constant CONTRACT_
StakedCitadelVester.sol:34:    uint256 public constant INITIAL_V
SupplySchedule.sol:22:    bytes32 public constant CONTRACT_GOVER
SupplySchedule.sol:25:    uint256 public constant epochLength =
```

[G-37] Reduce the size of error messages (Long revert Strings)

Shortening revert strings to fit in 32 bytes will decrease deployment time gas and will decrease runtime gas when the revert condition is met.

Revert strings that are longer than 32 bytes require at least one additional mstore, along with additional overhead for computing memory offset, etc.

Revert strings > 32 bytes:

```
lib/GlobalAccessControlManaged.sol:64:            "GAC: invalid-
lib/SafeERC20.sol:57:            "SafeERC20: approve from non-ze
lib/SafeERC20.sol:78:            require(oldAllowance >= value,
lib/SafeERC20.sol:98:            require(abi.decode(returndata,
CitadelMinter.sol:301:            "CitadelMinter: Sum of propval
CitadelMinter.sol:321:            "CitadelMinter: last mint time
CitadelMinter.sol:328:            "CitadelMinter: supply schedul
CitadelMinter.sol:370:            "CitadelMinter: funding pool c
CitadelMinter.sol:377:            "CitadelMinter: funding pool a
Funding.sol:148:            "Funding: citadel price from oracle
Funding.sol:298:            "cannot decrease cap below global su
Funding.sol:325:            "cannot sweep funding asset, use cla
Funding.sol:390:            "Funding: sale recipient should not
GlobalAccessControl.sol:118:            "Role string and role dc
KnightingRound.sol:122:            "KnightingRound: start date m
KnightingRound.sol:126:            "KnightingRound: the sale dur
KnightingRound.sol:130:            "KnightingRound: the price mu
KnightingRound.sol:134:            "KnightingRound: sale recipie
KnightingRound.sol:273:        require(!finalized, "KnightingRou
KnightingRound.sol:277:            "KnightingRound: not enough b
KnightingRound.sol:295:            "KnightingRound: start date m
KnightingRound.sol:297:        require(!finalized, "KnightingRou
KnightingRound.sol:314:            "KnightingRound: the sale dur
KnightingRound.sol:316:        require(!finalized, "KnightingRou
KnightingRound.sol:333:            "KnightingRound: the price mu
KnightingRound.sol:351:            "KnightingRound: sale recipie
KnightingRound.sol:384:        require(!finalized, "KnightingRou
StakedCitadel.sol:192:            "performanceFeeGovernance too
StakedCitadel.sol:196:            "performanceFeeStrategist too
```

```
    StakedCitadel.sol:508:                       "Please withdrawToVault be
    StakedCitadel.sol:537:                   "performanceFeeStrategist too
    StakedCitadel.sol:632:                   "Excessive strategist performa
    StakedCitadel.sol:652:                   "Excessive governance performa
    StakedCitadelVester.sol:137:               require(msg.sender == vault,
    StakedCitadelVester.sol:138:               require(_amount > 0, "Stakec
    SupplySchedule.sol:62:                   "SupplySchedule: minting not s
    SupplySchedule.sol:92:                   "SupplySchedule: minting not s
    SupplySchedule.sol:96:                   "SupplySchedule: already minte
    SupplySchedule.sol:139:                   "SupplySchedule: minting alre
    SupplySchedule.sol:143:                   "SupplySchedule: minting must
    SupplySchedule.sol:157:                   "SupplySchedule: rate already
    SupplySchedule.sol:181:                   "SupplySchedule: minting not
    SupplySchedule.sol:185:                   "SupplySchedule: attempting t
    SupplySchedule.sol:189:                   "SupplySchedule: already mint
    SupplySchedule.sol:227:                       "total mintable after thi
```

I suggest shortening the revert strings to fit in 32 bytes, or using custom errors as described next.

[G-38] Use Custom Errors instead of Revert Strings to save Gas

Custom errors from Solidity 0.8.4 are cheaper than revert strings (cheaper deployment cost and runtime cost when the revert condition is met)

Source: https://blog.soliditylang.org/2021/04/21/custom-errors/:

> Starting from [Solidity v0.8.4](https://github.com/ethereum/solidity/releases/tag/v0.8.4), there is a convenient and gas-efficient way to explain to users why an operation failed through the use of custom errors. Until now, you could already use strings to give more information about failures (e.g., `revert("Insufficient funds.");`), but they are rather expensive, especially when it comes to deploy cost, and it is difficult to use dynamic information in them.

Custom errors are defined using the `error` statement, which can be used inside and outside of contracts (including interfaces and libraries).

Instances include:

```
interfaces/convex/BoringMath.sol:8:        require((c = a + b) >
interfaces/convex/BoringMath.sol:12:        require((c = a - b)
interfaces/convex/BoringMath.sol:16:        require(b == 0 || (c
interfaces/convex/BoringMath.sol:20:        require(b > 0, "Bori
interfaces/convex/BoringMath.sol:25:        require(a <= type(ui
interfaces/convex/BoringMath.sol:30:        require(a <= type(ui
interfaces/convex/BoringMath.sol:35:        require(a <= type(ui
interfaces/convex/BoringMath.sol:40:        require(a <= type(ui
interfaces/convex/BoringMath.sol:45:        require(a <= type(ui
interfaces/convex/BoringMath.sol:50:        require(a <= type(ui
interfaces/convex/BoringMath.sol:55:        require(a <= type(ui
interfaces/convex/BoringMath.sol:60:        require(a <= type(ui
interfaces/convex/BoringMath.sol:68:        require((c = a + b)
interfaces/convex/BoringMath.sol:72:        require((c = a - b)
interfaces/convex/BoringMath.sol:79:        require((c = a + b)
interfaces/convex/BoringMath.sol:83:        require((c = a - b)
interfaces/convex/BoringMath.sol:90:        require((c = a + b)
interfaces/convex/BoringMath.sol:94:        require((c = a - b)
interfaces/convex/BoringMath.sol:98:        require(b == 0 || (c
interfaces/convex/BoringMath.sol:102:        require(b > 0, "Bor
interfaces/convex/BoringMath.sol:110:        require((c = a + b)
interfaces/convex/BoringMath.sol:114:        require((c = a - b)
interfaces/convex/BoringMath.sol:118:        require(b == 0 || (
interfaces/convex/BoringMath.sol:122:        require(b > 0, "Bor
interfaces/convex/BoringMath.sol:130:        require((c = a + b)
interfaces/convex/BoringMath.sol:134:        require((c = a - b)
interfaces/convex/BoringMath.sol:138:        require(b == 0 || (
interfaces/convex/BoringMath.sol:142:        require(b > 0, "Bor
lib/GlobalAccessControlManaged.sol:41:        require(gac.hasRol
lib/GlobalAccessControlManaged.sol:55:        require(validRoleP
lib/GlobalAccessControlManaged.sol:62:        require(
lib/GlobalAccessControlManaged.sol:71:        require(!gac.pause
lib/GlobalAccessControlManaged.sol:72:        require(!paused(),
lib/GlobalAccessControlManaged.sol:81:        require(gac.hasRol
lib/GlobalAccessControlManaged.sol:86:        require(gac.hasRol
lib/SafeERC20.sol:55:        require(
lib/SafeERC20.sol:78:            require(oldAllowance >= value,
lib/SafeERC20.sol:98:            require(abi.decode(returndata,
lib/SettAccessControl.sol:16:        require(msg.sender == gover
lib/SettAccessControl.sol:20:        require(
lib/SettAccessControl.sol:27:        require(
CitadelMinter.sol:116:        require(_gac != address(0), "addre
CitadelMinter.sol:117:        require(_citadelToken != address(0
CitadelMinter.sol:118:        require(_xCitadel != address(0), "
CitadelMinter.sol:119:        require(_xCitadelLocker != address
CitadelMinter.sol:120:        require(_supplySchedule != address
CitadelMinter.sol:256:        require(
CitadelMinter.sol:272:            require(_weight <= 10000, "exc
CitadelMinter.sol:299:        require(
CitadelMinter.sol:319:        require(
CitadelMinter.sol:326:        require(
CitadelMinter.sol:343:        require(length > 0, "CitadelMinter
CitadelMinter.sol:368:        require(
CitadelMinter.sol:375:        require(
Funding.sol:80:        require(
Funding.sol:113:        require(
Funding.sol:117:        require(
Funding.sol:146:        require(
Funding.sol:170:        require(_assetAmountIn > 0, "_assetAmour
Funding.sol:171:        require(
Funding.sol:178:        require(citadelAmount_ >= _minCitadelOut
Funding.sol:270:        require(_discount >= funding.minDiscount
Funding.sol:271:        require(_discount <= funding.maxDiscount
Funding.sol:296:        require(
Funding.sol:322:        require(amount > 0, "nothing to sweep");
Funding.sol:323:        require(
Funding.sol:340:        require(amount > 0, "nothing to claim");
Funding.sol:361:        require(_maxDiscount < MAX_BPS , "maxDis
Funding.sol:388:        require(
Funding.sol:424:        require(_citadelPriceInAsset > 0, "citac
Funding.sol:425:        require(_valid, "oracle data must be val
Funding.sol:452:        require(_citadelPriceInAsset > 0, "citac
GlobalAccessControl.sol:95:        require(hasRole(PAUSER_ROLE,
GlobalAccessControl.sol<img class="emoji-icon" alt="emoji-100" c
GlobalAccessControl.sol:112:        require(
GlobalAccessControl.sol:116:        require(
KnightingRound.sol:120:        require(
KnightingRound.sol:124:        require(
KnightingRound.sol:128:        require(
KnightingRound.sol:132:        require(
KnightingRound.sol:167:        require(saleStart <= block.timest
KnightingRound.sol:168:        require(
KnightingRound.sol:172:        require(_tokenInAmount > 0, "_tok
KnightingRound.sol:173:        require(
KnightingRound.sol:179:            require(guestlist.authorized(
KnightingRound.sol:185:            require(
KnightingRound.sol:210:        require(finalized, "sale not fina
KnightingRound.sol:211:        require(!hasClaimed[msg.sender],
KnightingRound.sol:215:        require(tokenOutAmount_ > 0, "not
KnightingRound.sol:273:        require(!finalized, "KnightingRou
KnightingRound.sol:274:        require(saleEnded(), "KnightingRo
KnightingRound.sol:275:        require(
KnightingRound.sol:293:        require(
KnightingRound.sol:297:        require(!finalized, "KnightingRou
KnightingRound.sol:312:        require(
KnightingRound.sol:316:        require(!finalized, "KnightingRou
KnightingRound.sol:331:        require(
KnightingRound.sol:349:        require(
KnightingRound.sol:384:        require(!finalized, "KnightingRou
KnightingRound.sol:411:        require(amount > 0, "nothing to s
StakedCitadel.sol:180:        require(_token != address(0)); //
StakedCitadel.sol:181:        require(_governance != address(0))
StakedCitadel.sol:182:        require(_keeper != address(0)); //
StakedCitadel.sol:183:        require(_guardian != address(0));
StakedCitadel.sol:184:        require(_treasury != address(0));
StakedCitadel.sol:185:        require(_strategist != address(0))
StakedCitadel.sol:186:        require(_badgerTree != address(0))
StakedCitadel.sol:187:        require(_vesting != address(0)); /
StakedCitadel.sol:190:        require(
StakedCitadel.sol:194:        require(
StakedCitadel.sol:198:        require(
StakedCitadel.sol:202:        require(
StakedCitadel.sol:262:        require(
StakedCitadel.sol:270:        require(msg.sender == strategy, "c
StakedCitadel.sol:441:        require(address(token) != _token,
StakedCitadel.sol:487:        require(_treasury != address(0), '
StakedCitadel.sol:502:        require(_strategy != address(0), '
StakedCitadel.sol:506:            require(
StakedCitadel.sol:523:        require(_fees <= WITHDRAWAL_FEE_HA
StakedCitadel.sol:535:        require(
StakedCitadel.sol:550:        require(_fees <= MANAGEMENT_FEE_HA
StakedCitadel.sol:562:        require(_guardian != address(0), '
StakedCitadel.sol:574:        require(_vesting != address(0), "A
StakedCitadel.sol:588:        require(_newToEarnBps <= MAX_BPS,
StakedCitadel.sol:613:        require(_withdrawalFee <= maxWithd
StakedCitadel.sol:630:        require(
StakedCitadel.sol:650:        require(
StakedCitadel.sol:666:        require(_fees <= maxManagementFee,
StakedCitadel.sol:700:        require(address(token) != _token,
StakedCitadel.sol:718:        require(!pausedDeposit, "pausedDep
StakedCitadel.sol:768:        require(_recipient != address(0),
StakedCitadel.sol:769:        require(_amount != 0, "Amount 0");
StakedCitadel.sol:770:        require(!pausedDeposit, "pausedDep
StakedCitadel.sol:794:            require(
StakedCitadel.sol:809:        require(_shares != 0, "0 Shares");
StakedCitadelVester.sol:64:        require(_vestingToken != addr
StakedCitadelVester.sol:65:        require(_vault != address(0),
StakedCitadelVester.sol:137:        require(msg.sender == vault,
StakedCitadelVester.sol:138:        require(_amount > 0, "Stakec
SupplySchedule.sol:60:        require(
SupplySchedule.sol:90:        require(
SupplySchedule.sol:94:        require(
SupplySchedule.sol:137:        require(
SupplySchedule.sol:141:        require(
SupplySchedule.sol:155:        require(
SupplySchedule.sol:179:        require(
SupplySchedule.sol:183:        require(
SupplySchedule.sol:187:        require(
```

I suggest replacing revert strings with custom errors.

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.