Paladin - Warden Pledges contest Findings & Analysis Report

2023-01-06

Table of contents

Overview

About C4

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

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

During the audit contest outlined in this document, C4 conducted an analysis of the Paladin - Warden Pledges smart contract system written in Solidity. The audit contest took place between October 27—October 30 2022.

Wardens

97 Wardens contributed reports to the Paladin - Warden Pledges contest:

1. 0x007
2. 0x1f8b
3. 0x52
4. 0xDecorativePineapple
5. 0xDjango
6. 0xNazgul
7. 0xRoxas
8. 0xSmartContract
9. 0xbepresent
10. 0xhunter
11. 8olidity
12. Amithuddar
13. Awesome

14. Aymen0909
15. B2
16. BnkeOxO
17. Chom
18. Diana
19. Dravee
20. JTJabba
21. Jeiwan
22. Josiah
23. KingNFT
24. KoKo
25. Lambda
26. Mathieu
27. Nyx
28. Picodes
29. RaoulSchaffranek
30. RaymondFam
31. RedOneN
32. ReyAdmirado
33. RockingMiles (robee and pants)
34. Rolezn
35. Ruhum
36. SadBase
37. Sm4rty
38. SooYa
39. Tricko
40. Trust
41. Waze
42. __141345__

43. a12jmx
44. adriro
45. ajtra
46. ballx
47. bin2chen
48. brgltd
49. c3phas
50. carlitox477
51. cccz
52. ch0bu
53. chaduke
54. chrisdior4
55. codexploder
56. corerouter
57. cryptonue
58. csanuragjain
59. ctf_sec
60. cylzxje
61. delfin454000
62. dic0de
63. djxploit
64. durianSausage
65. emrekocak
66. erictee
67. gogo
68. halden
69. hansfriese
70. horsefacts
71. hxzy

72. imare
73. indijanc
74. jayphbee
75. jwood
76. karanctf
77. ktg
78. ladboy233
79. leosathya
80. lukris02
81. minhtrng
82. neko_nyaa
83. oyc_109
84. pashov
85. peiw
86. peritoflores
87. rbserver
88. robee
89. rvierdiiev
90. sakman
91. shark
92. skyle
93. subtle77
94. tnevler
95. wagmi
96. yixxas

This contest was judged by kirk-baird.

Final report assembled by itsmetechjay.

Summary

The C4 analysis yielded an aggregated total of 8 unique vulnerabilities. Of these vulnerabilities, 0 received a risk rating in the category of HIGH severity and 8 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 68 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 49 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 Paladin - Warden Pledges contest repository, and is composed of 1 smart contract written in the Solidity programming language and includes 317 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.

Medium Risk Findings (8)

[M-01] Due to loss of precision, targetVotes may not reach

Submitted by [cccz]

In the `\_pledge` function, require delegationBoost.adjusted*balance*of(pledgeParams.receiver) + amount <= pledgeParams.targetVotes.

In reality, when the user pledges the amount of votes, the actual votes received by the receiver are the bias in the following calculation. And the bias will be less than amount due to the loss of precision.

```
uint256 slope = amount / boostDuration;
uint256 bias = slope * boostDuration;
```

This means that the balance of receiver may not reach targetVotes

```
point = self._checkpoint_read(_user, False)
amount += (point.bias - point.slope * (block.timestamp - poi
return amount
```

Proof of Concept

https://github.com/code-423n4/2022-10-paladin/blob/d6d0c0e57ad80f15e9691086c9c7270d4ccfe0e6/contracts/WardenPledge.sol#L245-L246

https://github.com/curvefi/curve-veBoost/blob/master/contracts/BoostV2.vy#L192-L209

https://github.com/curvefi/curve-veBoost/blob/master/contracts/BoostV2.vy#L175

Recommended Mitigation Steps

Use bias instead of amount in the check below:

```
uint256 slope = amount / boostDuration;
uint256 bias = slope * boostDuration;
```

```
        if(delegationBoost.adjusted_balance_of(pledgeParams.rece
        delegationBoost.boost(
            pledgeParams.receiver,
            amount,
            endTimestamp,
            user
        );
```

Kogaroshi (Paladin) acknowledged and commented:

> The current check is made that way to prevent any unnecessary call to the BoostV2 contract (and save gas by not creating the Boost) in the case of a `targetVotes` overflow.

[M-02] Owner can transfer all ERC20 reward token out using function recoverERC20

Submitted by ladboy233, *also found by* yixxas, JTJabba, rbserver, Aymen0909, horsefacts, minhtrng, 0xhunter, Trust, peritoflores, 0xDecorativePineapple, Dravee, hansfriese, imare, Jeiwan, wagmi, 0x52, Picodes, cryptonue, pashov, BnkeOx0, Lambda, Nyx, cccz, dic0de, csanuragjain, *and* rvierdiiev

The function recoverERC20 is very privileged. It means to recover any token that is accidently sent to the contract.

```
    function recoverERC20(address token) external onlyOwner returns(
        if(minAmountRewardToken[token] != 0) revert Errors.Canno

        uint256 amount = IERC20(token).balanceOf(address(this));
        if(amount == 0) revert Errors.NullValue();
        IERC20(token).safeTransfer(owner(), amount);

        return true;
    }
```

However, admin / owner can use this function to transfer all the reserved reward tokens, which result in fund loss of the pledge creator and the loss of reward for users that want to delegate the veToken.

Also, the recovered token is sent to owner directly instead of sending to a recipient address.

The safeguard

```
if(minAmountRewardToken[token] != 0)
```

cannot stop owner transferring funds because if the owner is compromised or misbehaves, he can adjust the whitelist easily.

Proof of Concept

The admin can set minAmountRewardToken[token] to 0 first by calling updateRewardToken:

```
function updateRewardToken(address token, uint256 minRewardPerSe
```

By doing this the admin removes the token from the whitelist, then the token can call recoverERC20 to transfer all the token into the owner wallet.

```
function recoverERC20(address token) external onlyOwner returns(
```

Recommended Mitigation Steps

We recommend that the project uses a multisig wallet to safeguard the owner's wallet.

We can also keep track of the reserved amount for rewarding token and only transfer the remaining amount of token out.

```
pledgeAvailableRewardAmounts[pledgeId] += totalRewardAmount;
reservedReward[token] += totalRewardAmount;
```

Then we can change the implementation to:

```
function recoverERC20(address token, address recipient) external
```

```
        uint256 amount = IERC20(token).balanceOf(address(this));
        if(amount == 0) revert Errors.NullValue();

        if(minAmountRewardToken[token] == 0) {
         // if it is not whitelisted, we assume it is mistakenly
           // we transfer the token to recipient
         IERC20(token).safeTransfer(recipient, amount);
        } else {
        // revert if the owner over transfer
        if(amount >  reservedReward[token]) revert rewardReserve
          IERC20(token).safeTransfer(recipient, amount - reserve
        }

        return true;

    }
```

Kogaroshi (Paladin) confirmed

Kogaroshi (Paladin) commented:

> Interesting proposed Mitigation to be noted.

[M-03] Pledge may be out of reward due to the decay in veCRV balance. targetVotes is never reached.

Submitted by Chom, *also found by* Jeiwan, Picodes, *and* KingNFT

https://github.com/code-423n4/2022-10-paladin/blob/d6d0c0e57ad80f15e9691086c9c7270d4ccfe0e6/contracts/WardenPledge.sol#L325-L335

https://github.com/code-423n4/2022-10-paladin/blob/d6d0c0e57ad80f15e9691086c9c7270d4ccfe0e6/contracts/WardenPledge.sol#L259-L268

Impact

Pledge may be out of reward due to the decay in veCRV balance. The receiver may lose his reward given to boosters but get nothing in return since her targetVotes is never reached.

Proof of Concept

According to Curve documentation at https://curve.readthedocs.io/dao-vecrv.html

```
A user's veCRV balance decays linearly as the remaining time unt
```

On creation, targetVotes = 100, balance = 20 -> votesDifference = 80 -> reward is allocated for 80 votes

```
// Get the missing votes for the given receiver to reach
// We ignore any delegated boost here because they might
// (we can have a future version of this contract using
vars.votesDifference = targetVotes - votingEscrow.balanc

vars.totalRewardAmount = (rewardPerVote * vars.votesDiff
vars.feeAmount = (vars.totalRewardAmount * protocalFeeRa
if(vars.totalRewardAmount > maxTotalRewardAmount) revert
if(vars.feeAmount > maxFeeAmount) revert Errors.Incorrec

// Pull all the rewards in this contract
IERC20(rewardToken).safeTransferFrom(creator, address(th
// And transfer the fees from the Pledge creator to the
IERC20(rewardToken).safeTransferFrom(creator, chestAddre
```

Then 1 week passed, receiver's balance decay to 10

On creation, targetVotes = 100, balance = 10 but votesDifference stays 80, and reward has only allocated for 80 votes.

```
// Rewards are set in the Pledge as reward/veToken/sec
// To find the total amount of veToken delegated through
// based on the Boost bias & the Boost duration, to take
// each second of the Boost duration
uint256 totalDelegatedAmount = ((bias * boostDuration) +
// Then we can calculate the total amount of rewards for
```

```
uint256 rewardAmount = (totalDelegatedAmount * pledgePar

if(rewardAmount > pledgeAvailableRewardAmounts[pledgeId]
pledgeAvailableRewardAmounts[pledgeId] -= rewardAmount;
```

A booster boosts 80 votes and takes all rewards in the pool. However, only 80 (From booster) + 10 (From receiver) = 90 votes is active. Not 100 votes that receiver promise in the targetVotes.

Then, if another booster tries to boost 10 votes, it will be reverted with RewardsBalanceTooLow since the first booster has taken all reward that is allocated for only 80 votes.

Recommended Mitigation Steps

You should provide a way for the creator to provide additional rewards after the pledge creation. Or provide some reward refreshment function that recalculates votesDifference and transfers the required additional reward.

Kogaroshi (Paladin) confirmed, resolved, and commented:

> Changed the logic in PR 2, commit Now the whole amount of votes needed for each second of the Pledge duration is calculated, taking in account the receiver potential veCRV balance, and the veCRV decay.

> This should allow to add only the exact amount of reward needed to the Pledge reward pool, and have always the correct amount of rewards to achieve the vote target of the Pledge at all times.

Kogaroshi (Paladin) commented:

> If possible, a feedback on the new calculation and logic would be appreciated.

[M-04] Pledges that contain delisted tokens can be extended to continue using delisted reward tokens

Submitted by 0x52, also found by bin2chen

Delisted reward tokens can continue to be used by extending current pledges that already use it.

Proof of Concept

```
if(pledgeId >= pledgesIndex()) revert Errors.InvalidPledgeID();
address creator = pledgeOwner[pledgeId];
if(msg.sender != creator) revert Errors.NotPledgeCreator();


Pledge storage pledgeParams = pledges[pledgeId];
if(pledgeParams.closed) revert Errors.PledgeClosed();
if(pledgeParams.endTimestamp <= block.timestamp) revert Errors.E
if(newEndTimestamp == 0) revert Errors.NullEndTimestamp();
uint256 oldEndTimestamp = pledgeParams.endTimestamp;
if(newEndTimestamp != _getRoundedTimestamp(newEndTimestamp) || n


uint256 addedDuration = newEndTimestamp - oldEndTimestamp;
if(addedDuration < minDelegationTime) revert Errors.DurationTooS
uint256 totalRewardAmount = (pledgeParams.rewardPerVote * pledge
uint256 feeAmount = (totalRewardAmount * protocalFeeRatio) / MAX
if(totalRewardAmount > maxTotalRewardAmount) revert Errors.Incor
if(feeAmount > maxFeeAmount) revert Errors.IncorrectMaxFeeAmount
```

During the input validation checks, it's never checked that reward token of the pledge being extended is still a valid reward token. This would allow creators using delisted tokens to continue using them as long as they wanted, by simply extending their currently active pledges.

Recommended Mitigation Steps

Add the following check during the input validation block:

```
+    if(minAmountRewardToken[rewardToken] == 0) revert Errors.Tok
```

Kogaroshi (Paladin) confirmed, resolved, and commented:

> Fixed in PR 2, commit.

kirk-baird (judge) commented:

> I consider this a valid Medium risk as pledges can be extended indefinitely. It bypasses the whitelisting which may be damaging if a token is found to be malicious and removed.

[M-05] WardenPledge accidentally inherits Ownable instead of Owner which removes an important safeguard without sponsor knowledge

Submitted by 0x52, *also found by* pashov *and* indijanc

Owner may accidentally transfer ownership to inoperable address due to perceived safeguard that doesn't exist.

Proof of Concept

```
contract WardenPledge is Ownable, Pausable, ReentrancyGuard {
```

WardenPledge inherits from Ownable rather than Owner, which is the intended contract. Owner overwrites the critical Ownable#transferOwnership function to make the ownership transfer process a two step process. This adds important safeguards because in the event that the target is unable to accept for any reason (input typo, incompatible multisig/contract, etc.) the ownership transfer process will fail because the pending owner will not be able to accept the transfer. To make matters worse, since it only overwrites the transferOwnership function the WardenPledge contract will otherwise function as intended just without this safeguard. It is likely that the owner won't even realize until it's too late and the safeguard has failed. A perceived safeguard where there isn't one is more damaging than not having any safeguard at all.

Recommended Mitigation Steps

```
-   contract WardenPledge is Ownable, Pausable, ReentrancyGuard
+   contract WardenPledge is Owner, Pausable, ReentrancyGuard {
```

Kogaroshi (Paladin) confirmed

[M-06] Reward can be over- or undercounted in `extendPledge` and `increasePledgeRewardPerVote`

Submitted by Jeiwan, *also found by* Aymen0909, Trust, 0xDjango, Chom, Lambda, *and* Ruhum

https://github.com/code-423n4/2022-10-paladin/blob/d6d0c0e57ad80f15e9691086c9c7270d4ccfe0e6/contracts/WardenPledge.sol#L387

https://github.com/code-423n4/2022-10-paladin/blob/d6d0c0e57ad80f15e9691086c9c7270d4ccfe0e6/contracts/WardenPledge.sol#L432

Impact

Total reward amount in `extendPledge` and `increasePledgeRewardPerVote` can be calculated incorrectly due to cached `pledgeParams.votesDifference`, which can lead to two outcomes:

1. total reward amount is higher, thus a portion of it won't be claimable;
2. total reward amount is lower, thus the pledge target won't be reached.

Proof of Concept

When a pledge is created, the creator chooses the target—the total amount of votes they want to reach with the pledge. Based on a target, the number of missing votes is calculated, which is then used to calculated the total reward amount (WardenPledge.sol#L325-L327):

```
    function createPledge(
        address receiver,
        address rewardToken,
        uint256 targetVotes,
        uint256 rewardPerVote, // reward/veToken/second
        uint256 endTimestamp,
        uint256 maxTotalRewardAmount,
```

```
        uint256 maxFeeAmount
    ) external whenNotPaused nonReentrant returns(uint256){
        ...
        // Get the missing votes for the given receiver to reach the
        // We ignore any delegated boost here because they might exp
        // (we can have a future version of this contract using adju
        vars.votesDifference = targetVotes - votingEscrow.balanceOf(

        vars.totalRewardAmount = (rewardPerVote * vars.votesDifferen
        ...
      }
```

When extending a pledge or increasing a pledge reward per vote, current veToken balance of the pledge's receiver (`votingEscrow.balanceOf(receiver)`) can be different from the one it had when the pledge was created (e.g. the receiver managed to lock more CRV or some of locked tokens have expired). However `pledgeParams.votesDifference` is not recalculated (WardenPledge.sol#L387, WardenPledge.sol#L432):

```
    function extendPledge(
        uint256 pledgeId,
        uint256 newEndTimestamp,
        uint256 maxTotalRewardAmount,
        uint256 maxFeeAmount
    ) external whenNotPaused nonReentrant {
        ...
        Pledge storage pledgeParams = pledges[pledgeId];
        ...
        uint256 totalRewardAmount = (pledgeParams.rewardPerVote * pl
        ...
    }

    function increasePledgeRewardPerVote(
        uint256 pledgeId,
        uint256 newRewardPerVote,
        uint256 maxTotalRewardAmount,
        uint256 maxFeeAmount
    ) external whenNotPaused nonReentrant {
        ...
        Pledge storage pledgeParams = pledges[pledgeId];
        ...
        uint256 totalRewardAmount = (rewardPerVoteDiff * pledgeParam
        ...
```

```
    }
```

This can lead to two consequences:

1. When receiver's veToken balance has increased (i.e. `votesDifference` got in fact smaller), pledge creator will overpay for pledge extension and pledge reward per vote increase. This extra reward cannot be received by pledgers because a receiver cannot get more votes than `pledgeParams.targetVotes` (which is not updated when modifying a pledge):

```
function _pledge(uint256 pledgeId, address user, uint256 amount, uint25
    ...
    // Check that this will not go over the Pledge target of votes
    if(delegationBoost.adjusted_balance_of(pledgeParams.receiver) + amo
    ...
}
```

2. When receiver's veToken balance has decreased (i.e. `votesDifference` got in fact bigger), the pledge target cannot be reached because the reward amount was underpaid in `extendPledge` / `increasePledgeRewardPerVote`.

Recommended Mitigation Steps

Consider updating `votesDifference` when extending a pledge or increasing a pledge reward per vote.

Kogaroshi (Paladin) confirmed and commented:

> As stated in #91, new method for needed votes & needed reward calculations is introduced in this commit, allowing to get the exact amount of reward token the Pledge creator should pay when extending the Pledge or increasing the `rewardPerVote`.

[M-07] Fees charged from entire theoretical pledge amount instead of actual pledge amount

Submitted by Trust, also found by 0x52

Paladin receives a 5% cut from Boost purchases, as documented on the website:

“Warden takes a 5% fee on Boost purchases, and 5% on Quest incentives. However, there are various pricing tiers for Quest creators. Contact the Paladin team for more info.”

Here’s how fee calculation looks at `createPledge` function:

```
vars.totalRewardAmount = (rewardPerVote * vars.votesDifference *
vars.feeAmount = (vars.totalRewardAmount * protocalFeeRatio) / M
if(vars.totalRewardAmount > maxTotalRewardAmount) revert Errors.
if(vars.feeAmount > maxFeeAmount) revert Errors.IncorrectMaxFeeA
// Pull all the rewards in this contract
IERC20(rewardToken).safeTransferFrom(creator, address(this), var
// And transfer the fees from the Pledge creator to the Chest co
IERC20(rewardToken).safeTransferFrom(creator, chestAddress, vars
```

The issue is that the fee is taken up front, assuming `totalRewardAmount` will actually be rewarded by the pledge. In practice, the rewards actually utilized can be anywhere from zero to `totalRewardAmount`. Indeed, reward will only be `totalRewardAmount` if, in the entire period from pledge creation to pledge expiry, the desired targetVotes will be fulfilled, which is extremly unlikely.

As a result, if pledge expires with no pledgers, protocol will still take 5%. This behavior is both unfair and against the docs, as it’s not “Paladin receives a 5% cut from Boost purchases”.

Impact

Paladin fee collection assumes pledges will be matched immediately and fully, which is not realistic. Therefore far too many fees are collected at user’s expense.

Proof of Concept

1. Bob creates a pledge, with target = 200, current balance = 100, rewardVotes = 10, remaining time = 1 week.
2. Protocol collects (200 - 100) * 10 * WEEK_SECONDS * 5% fees
3. A week passed, rewards were not attractive enough to bring pledgers.

4. After expiry, Bob calls `retrievePledgeRewards()` and gets 100 * 10 * WEEK_SECONDS back, but 5% of the fees still went to chestAddress.

Recommended Mitigation Steps

Fee collection should be done after the pledge completes, in one of the close functions or in a newly created pull function for owner to collect fees. Otherwise, it is a completely unfair system.

Kogaroshi (Paladin) acknowledged and commented:

> The issue is acknowledged, and we do calculate fee on the basis of all rewards, and not only the one that are gonna be used to reward users.

> The fee ratio is gonna be of 1% to start with (might change before deploy based on market estimations), and the Core team will be able to change the ratio quickly to adapt it to market and Pledge creators needs (with also considering the Paladin DAO revenues). The Paladin team will also considers Pledge creators that are in specific cases and overpay fees (because they already have delegated boost that will last through the whole Pledge and more), and will be able to refund a part of those fees to the creator if the DAO agrees.

> And if this system does not fit in the current market, and is a blocker to potential Pledge creators, we will be able to modify the way fees are handled, and deploy a new iteration of Pledge pretty fast to answer the issue.

[M-08] Pausing `WardenPledge` contract, which takes effect immediately, by its owner can unexpectedly block pledge creator from calling `closePledge` or `retrievePledgeRewards` function

Submitted by rbserver, *also found by* 0x1f8b, 0xSmartContract, Trust, hansfriese, ctf_sec, cccz, *and* codexploder

https://github.com/code-423n4/2022-10-paladin/blob/main/contracts/WardenPledge.sol#L636-L638

https://github.com/code-423n4/2022-10-paladin/blob/main/contracts/WardenPledge.sol#L488-L515

https://github.com/code-423n4/2022-10-paladin/blob/main/contracts/WardenPledge.sol#L456-L480

Impact

The owner of the `WardenPledge` contract is able to call the `pause` function to pause this contract. When the `WardenPledge` contract is paused, calling the `closePledge` or `retrievePledgeRewards` function that uses the `whenNotPaused` modifier reverts, and the pledge creator is not able to get back any of the reward token amount, which was deposited by the creator previously. Because calling the `pause` function takes effect immediately, it can be unexpected to the creator for suddenly not being able to call the `closePledge` or `retrievePledgeRewards` function. For instance, when an emergency occurs that requires an increase of cash flow, the creator wants to close the pledge early so she or he can use the remaining deposited reward token amount. However, just before the creator's `closePledge` transaction is executed, the `pause` transaction has been sent by the owner of the `WardenPledge` contract for some reason and executed. Without knowing in advance that the `WardenPledge` contract would be paused, the creator anticipates receiving the remaining deposited reward token amount but this is not the case since calling the `closePledge` function reverts. Because the creator unexpectedly fails to receive such amount and might fail to deal with the emergency, disputes with the protocol can occur, and the user experience becomes degraded.

https://github.com/code-423n4/2022-10-paladin/blob/main/contracts/WardenPledge.sol#L636-L638

```
function pause() external onlyOwner {
    _pause();
}
```

https://github.com/code-423n4/2022-10-paladin/blob/main/contracts/WardenPledge.sol#L488-L515

```
function closePledge(uint256 pledgeId, address receiver) ext
```

```
        ...

        // Get the current remaining amount of rewards not distr
        uint256 remainingAmount = pledgeAvailableRewardAmounts[p

        if(remainingAmount > 0) {
            // Transfer the non used rewards and reset storage
            pledgeAvailableRewardAmounts[pledgeId] = 0;

            IERC20(pledgeParams.rewardToken).safeTransfer(receiv

            ...

        }

        ...
    }
```

https://github.com/code-423n4/2022-10-paladin/blob/main/contracts/WardenPledge.sol#L456-L480

```
    function retrievePledgeRewards(uint256 pledgeId, address rec
        ...

        // Get the current remaining amount of rewards not distr
        uint256 remainingAmount = pledgeAvailableRewardAmounts[p

        ...

        if(remainingAmount > 0) {
            // Transfer the non used rewards and reset storage
            pledgeAvailableRewardAmounts[pledgeId] = 0;

            IERC20(pledgeParams.rewardToken).safeTransfer(receiv

            ...

        }
    }
```

Proof of Concept

Please append the following test in the `pause & unpause describe` block in `test\wardenPledge.test.ts`. This test will pass to demonstrate the described scenario.

```
        it.only('Pausing WardenPledge contract, which takes effe
            // before calling the createPledge function, the war
            const rewardToken1BalanceWardenPledgeBefore = await
            expect(rewardToken1BalanceWardenPledgeBefore).to.be.

            const rewardToken1BalanceCreatorBefore = await rewar

            // creator calls the createPledge function
            await wardenPledge.connect(creator).createPledge(
                receiver.address,
                rewardToken1.address,
                target_votes,
                reward_per_vote,
                end_timestamp,
                max_total_reward_amount,
                max_fee_amount
            )

            // after one week, admin, who is the owner of the wa
            await advanceTime(WEEK.toNumber())
            await wardenPledge.connect(admin).pause()

            // Since an emergency that requires an increase of c
            // Without knowing in advance that the wardenPledge
            //   creator calls the closePledge function and anti
            // Unfortunately, admin's pause transaction has been
            await expect(
                wardenPledge.connect(creator).closePledge(pledge
            ).to.be.revertedWith("Pausable: paused")

            // after creator's closePledge transaction reverts,
            const rewardToken1BalanceCreatorAfter = await reward
            expect(rewardToken1BalanceCreatorAfter).to.be.lt(rew

            // meanwhile, the wardenPledge contract still holds
            const rewardToken1BalanceWardenPledgeAfter = await r
            expect(rewardToken1BalanceWardenPledgeAfter).to.be.g
        });
```

Tools Used

VSCode

Recommended Mitigation Steps

The `pause` function can be updated to be time-delayed so the pledge creator can have more time to react. One way would be making this function only callable by a timelock governance contract.

Kogaroshi (Paladin) confirmed

Low Risk and Non-Critical Issues

For this contest, 67 reports were submitted by wardens detailing low risk and non-critical issues. The **report highlighted below** by **robee** received the top score from the judge.

The following wardens also submitted reports: ajtra, rbserver, adriro, peritoflores, Josiah, tnevler, horsefacts, brgltd, djxploit, minhtrng, Dravee, B2, Trust, lukris02, delfin454000, c3phas, RaoulSchaffranek, Waze, Tricko, JTJabba, a12jmx, Sm4rty, 0xSmartContract, imare, 0xNazgul, Jeiwan, 0x52, Diana, shark, __141345__, carlitox477, ktg, Awesome, Picodes, corerouter, 0x007, RedOneN, cryptonue, jayphbee, 0xDjango, Ruhum, pashov, cylzxje, Chom, ReyAdmirado, Rolezn, Lambda, ctf_sec, ladboy233, 8olidity, ch0bu, jwood, cccz, oyc_109, yixxas, dic0de, chaduke, csanuragjain, chrisdior4, neko_nyaa, Bnke0x0, 0x1f8b, rvierdiiev, leosathya, RaymondFam, *and* Mathieu.

[1] Missing fee parameter validation

Some fee parameters of functions are not checked for invalid values. Validate the parameters:

Code instances:

```
WardenPledge.increasePledgeRewardPerVote (maxFeeAmount)
WardenPledge.extendPledge (maxFeeAmount)
WardenPledge.createPledge (maxFeeAmount)
```

```
WardenPledge.updatePlatformFee (newFee)
```

[2] safeApprove of openZeppelin is deprecated

You use safeApprove of openZeppelin although it's deprecated. (see here).

You should change it to increase/decrease Allowance as OpenZeppilin says.

Code instances:

```
Deprecated safeApprove in SafeERC20.sol line 64: _callOption
Deprecated safeApprove in SafeERC20.sol line 76: _callOption
Deprecated safeApprove in SafeERC20.sol line 55: _callOption
```

[3] Not verified input

External / public functions parameters should be validated to make sure the address is not 0.

Otherwise if not given the right input it can mistakenly lead to loss of user funds.

Code instance:

```
WardenPledge.sol.recoverERC20 token
```

[4] Solidity compiler versions mismatch

The project is compiled with different versions of Solidity, which is not recommended because it can lead to undefined behaviors.

[5] Not verified owner

```
owner param should be validated to make sure the owner addre
Otherwise if not given the right input all only owner access
```

Code instance:

```
Ownable.sol.transferOwnership newOwner
```

[6] Two Steps Verification before Transferring Ownership

The following contracts have a function that allows them an admin to change it to a different address. If the admin accidentally uses an invalid address for which they do not have the private key, then the system gets locked.

It is important to have two steps admin change where the first is announcing a pending new admin and the new address should then claim its ownership.

A similar issue was reported in a previous contest and was assigned a severity of Medium: [code-423n4/2021-06-realitycards-findings#105]

Code instance:

```
Ownable.sol
```

[7] Missing non reentrancy modifier

The following functions are missing reentrancy modifier although some other pulbic/external functions does use reentrancy modifer. Even though I did not find a way to exploit it, it seems like those functions should have the nonReentrant modifier as the other functions have it as well..

Code instance:

```
WardenPledge.sol, recoverERC20 is missing a reentrancy modif
```

[8] In the following public update functions no value is returned

In the following functions no value is returned, due to which by default value of return will be 0.

We assumed that after the update you return the latest new value. (similar issue here: https://github.com/code-423n4/2021-10-badgerdao-findings/issues/85).

Code instances:

```
WardenPledge.sol, updateChest
WardenPledge.sol, updateMinTargetVotes
WardenPledge.sol, updatePlatformFee
WardenPledge.sol, updateRewardToken
```

[9] Check transfer receiver is not 0 to avoid burned money

Transferring tokens to the zero address is usually prohibited to accidentally avoid “burning” tokens by sending them to an unrecoverable zero address.

Code instances:

```
658     https:
472     https:
271     https:
438     https:
333     https:
394     https:
505     https:
```

[10] Missing commenting

```
The following functions are missing commenting as describe b
```

Code instance:

```
Pausable.sol, paused (public), @return is missing
```

[11] Unsafe Cast

Use openzeppilin's safeCast in:

Code instance:

```
https://github.com/code-423n4/2022-10-paladin/tree/main/cont
```

[12] Div by 0

Division by 0 can lead to accidentally revert, (An example of a similar issue - https://github.com/code-423n4/2021-10-defiprotocol-findings/issues/84)

Code instance:

```
https://github.com/code-423n4/2022-10-paladin/tree/main/cont
```

[13] Tokens with fee on transfer are not supported

There are ERC20 tokens that charge fee for every transfer() / transferFrom().

Vault.sol#addValue() assumes that the received amount is the same as the transfer amount, and uses it to calculate attributions, balance amounts, etc.

But, the actual transferred amount can be lower for those tokens. Therefore it's recommended to use the balance change before and after the transfer instead of the amount.

This way you also support the tokens with transfer fee - that are popular.

Code instances:

```
438     https:
333     https:
394     https:
```

Gas Optimizations

For this contest, 48 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **c3phas** received the top score from the judge.

The following wardens also submitted reports: sakman, tnevler, ajtra, adriro, lukris02, horsefacts, B2, peiw, djxploit, KoKo, Dravee, indijanc, gogo, RockingMiles, Waze, 0xSmartContract, SooYa, 0xRoxas, Amithuddar, imare, SadBase, 0xNazgul, neko_nyaa, halden, shark, __141345__, carlitox477, Picodes, karanctf, emrekocak, RedOneN, erictee, 0xbepresent, cylzxje, ReyAdmirado, Ruhum, Mathieu, ch0bu, durianSausage, oyc_109, Awesome, skyle, BnkeOx0, 0x1f8b, ballx, leosathya, *and* RaymondFam.

Findings

NB: Some functions have been truncated where neccessary to just show affected parts of the code.

Throughout the report, some places might be denoted with audit tags to show the actual place affected.

[G-01] Using immutable on variables that are only set in the constructor and never after

Use immutable if you want to assign a permanent value at construction. Use constants if you already know the permanent value. Both get directly embedded in bytecode, saving SLOAD. Variables only set in the constructor and never edited afterwards should be marked as immutable, as it would avoid the expensive storage-writing operation in the constructor (around 20 000 gas per variable) and replace the expensive storage-reading operations (around 2100 gas per reading) to a less expensive value reading (3 gas)

https://github.com/code-423n4/2022-10-paladin/blob/d6d0c0e57ad80f15e9691086c9c7270d4ccfe0e6/contracts/WardenPledge.sol#L60

```
File: /contracts/WardenPledge.sol
60:     IVotingEscrow public votingEscrow;
```

https://github.com/code-423n4/2022-10-paladin/blob/d6d0c0e57ad80f15e9691086c9c7270d4ccfe0e6/contracts/WardenPledge.sol#L62

```
File: /contracts/WardenPledge.sol
62:     IBoostV2 public delegationBoost;
```

[G-02] Use constants for variables whose value is known beforehand and is never changed

https://github.com/code-423n4/2022-10-paladin/blob/d6d0c0e57ad80f15e9691086c9c7270d4ccfe0e6/contracts/WardenPledge.sol#L79

```
File: /contracts/WardenPledge.sol
79:     uint256 public minDelegationTime = 1 weeks;
```

```
diff --git a/contracts/WardenPledge.sol b/contracts/WardenPledge
index beb990d..642a848 100644
--- a/contracts/WardenPledge.sol
+++ b/contracts/WardenPledge.sol
@@ -76,7 +76,7 @@ contract WardenPledge is Ownable, Pausable, Re
     uint256 public minTargetVotes;

     /** @notice Minimum delegation time, taken from veBoost con
-    uint256 public minDelegationTime = 1 weeks;
+    uint256 public constant minDelegationTime = 1 weeks;
```

[G-03] Cache storage values in memory to minimize SLOADs

The code can be optimized by minimizing the number of SLOADs.

SLOADs are expensive (100 gas after the 1st one) compared to MLOADs/MSTOREs (3 gas each). Storage values read multiple times should instead be cached in memory the first time (costing 1 SLOAD) and then read from this cache to avoid multiple SLOADs.

NB: *Some functions have been truncated where neccessary to just show affected parts of the code*

https://github.com/code-423n4/2022-10-paladin/blob/d6d0c0e57ad80f15e9691086c9c7270d4ccfe0e6/contracts/WardenPledge.sol#L222-L274

WardenPledge.sol._pledge(): delegationBoost should be cached(Saves 4 SLOADs ~394 gas)

```
File: /contracts/WardenPledge.sol
222:     function _pledge(uint256 pledgeId, address user, uint256

240:         delegationBoost.checkpoint_user(user); //@audit: 1st
241:         if(delegationBoost.allowance(user, address(this)) < 
242:         if(delegationBoost.delegable_balance(user) < amount)

245:         if(delegationBoost.adjusted_balance_of(pledgeParams.

248:         delegationBoost.boost( //@audit: 5th SLOAD
             pledgeParams.receiver,
             amount,
             endTimestamp,
             user
         );
```

```
diff --git a/contracts/WardenPledge.sol b/contracts/WardenPledge
index beb990d..5b3d1bd 100644
--- a/contracts/WardenPledge.sol
+++ b/contracts/WardenPledge.sol
@@ -237,15 +237,16 @@ contract WardenPledge is Ownable, Pausable
         uint256 boostDuration = endTimestamp - block.timestamp;
```

```
        // Check that the user has enough boost delegation avai
-       delegationBoost.checkpoint_user(user);
-       if(delegationBoost.allowance(user, address(this)) < amo
-       if(delegationBoost.delegable_balance(user) < amount) re
+        IBoostV2 _delegationBoost = delegationBoost;
+       _delegationBoost.checkpoint_user(user);
+       if(_delegationBoost.allowance(user, address(this)) < am
+       if(_delegationBoost.delegable_balance(user) < amount) r

        // Check that this will not go over the Pledge target o
-       if(delegationBoost.adjusted_balance_of(pledgeParams.rec
+       if(_delegationBoost.adjusted_balance_of(pledgeParams.re

        // Creates the DelegationBoost
-       delegationBoost.boost(
+       _delegationBoost.boost(
            pledgeParams.receiver,
            amount,
            endTimestamp,
```

https://github.com/code-423n4/2022-10-paladin/blob/d6d0c0e57ad80f15e9691086c9c7270d4ccfe0e6/contracts/WardenPledge.sol#L222-L274

WardenPledge.sol._pledge(): pledgeAvailableRewardAmounts[pledgeId] should be cached(saves 1 SLOAD ~97 gas)

```
File: /contracts/WardenPledge.sol
    function _pledge(uint256 pledgeId, address user, uint256 amo


267:        if(rewardAmount > pledgeAvailableRewardAmounts[pledg
268:        pledgeAvailableRewardAmounts[pledgeId] -= rewardAmou
```

```
diff --git a/contracts/WardenPledge.sol b/contracts/WardenPledge
index beb990d..2bb2cd2 100644
--- a/contracts/WardenPledge.sol
+++ b/contracts/WardenPledge.sol
@@ -263,9 +263,10 @@ contract WardenPledge is Ownable, Pausable,
        uint256 totalDelegatedAmount = ((bias * boostDuration)
```

```
        // Then we can calculate the total amount of rewards fo
        uint256 rewardAmount = (totalDelegatedAmount * pledgePa
+       uint _pledgeAvailableRewardAmounts = pledgeAvailableRew

-       if(rewardAmount > pledgeAvailableRewardAmounts[pledgeId
-       pledgeAvailableRewardAmounts[pledgeId] -= rewardAmount;
+       if(rewardAmount > _pledgeAvailableRewardAmounts) revert
+       pledgeAvailableRewardAmounts[pledgeId] = _pledgeAvailab

        // Send the rewards to the user
        IERC20(pledgeParams.rewardToken).safeTransfer(user, rew
```

https://github.com/code-423n4/2022-10-paladin/blob/d6d0c0e57ad80f15e9691086c9c7270d4ccfe0e6/contracts/WardenPledge.sol#L299-L358

WardenPledge.sol.createPledge(): minAmountRewardToken[rewardToken] should be cached(saves 1 SLOAD ~97 gas) - happy path

```
File: /contracts/WardenPledge.sol
299:    function createPledge(

312:        if(minAmountRewardToken[rewardToken] == 0) revert Er
313:        if(rewardPerVote < minAmountRewardToken[rewardToken]
```

```
diff --git a/contracts/WardenPledge.sol b/contracts/WardenPledge
index beb990d..247e5f8 100644
--- a/contracts/WardenPledge.sol
+++ b/contracts/WardenPledge.sol
@@ -309,8 +309,9 @@ contract WardenPledge is Ownable, Pausable,

        if(receiver == address(0) || rewardToken == address(0))
        if(targetVotes < minTargetVotes) revert Errors.TargetVo
-       if(minAmountRewardToken[rewardToken] == 0) revert Error
-       if(rewardPerVote < minAmountRewardToken[rewardToken]) r
+       uint256 _minAmountRewardToken = minAmountRewardToken[re
+       if(_minAmountRewardToken == 0) revert Errors.TokenNotWh
+       if(rewardPerVote < _minAmountRewardToken) revert Errors

        if(endTimestamp == 0) revert Errors.NullEndTimestamp();
        if(endTimestamp != _getRoundedTimestamp(endTimestamp))
```

[G-04] `require()` or `revert()` statements that check input arguments should be at the top of the function

Checks that involve constants should come before checks that involve state variables, function calls, and calculations. By doing these checks first, the function is able to revert before wasting a Gcoldsload (2100 gas) in a function that may ultimately revert in the unhappy case.

https://github.com/code-423n4/2022-10-paladin/blob/d6d0c0e57ad80f15e9691086c9c7270d4ccfe0e6/contracts/WardenPledge.sol#L222-L274

```
File: /contracts/WardenPledge.sol
222:     function _pledge(uint256 pledgeId, address user, uint256
223:         if(pledgeId >= pledgesIndex()) revert Errors.Invalid
224:         if(amount == 0) revert Errors.NullValue();
```

```
diff --git a/contracts/WardenPledge.sol b/contracts/WardenPledge
index beb990d..dfd3ff4 100644
--- a/contracts/WardenPledge.sol
+++ b/contracts/WardenPledge.sol
@@ -220,8 +220,9 @@ contract WardenPledge is Ownable, Pausable, 
      * @param endTimestamp End of delegation
      */
     function _pledge(uint256 pledgeId, address user, uint256 am
-        if(pledgeId >= pledgesIndex()) revert Errors.InvalidPle
         if(amount == 0) revert Errors.NullValue();
+        if(pledgeId >= pledgesIndex()) revert Errors.InvalidPle

         // Load Pledge parameters & check the Pledge is still a
         Pledge memory pledgeParams = pledges[pledgeId];
```

https://github.com/code-423n4/2022-10-paladin/blob/d6d0c0e57ad80f15e9691086c9c7270d4ccfe0e6/contracts/WardenPledge.sol#L456-L480

```
File: /contracts/WardenPledge.sol
456:     function retrievePledgeRewards(uint256 pledgeId, address
457:         if(pledgeId >= pledgesIndex()) revert Errors.Invalid
```

```
458:         address creator = pledgeOwner[pledgeId];
459:         if(msg.sender != creator) revert Errors.NotPledgeCrea
460:         if(receiver == address(0)) revert Errors.ZeroAddress
```

```
diff --git a/contracts/WardenPledge.sol b/contracts/WardenPledge
index beb990d..9c82ad9 100644
--- a/contracts/WardenPledge.sol
+++ b/contracts/WardenPledge.sol
@@ -454,10 +454,11 @@ contract WardenPledge is Ownable, Pausable
      * @param receiver Address to receive the remaining rewards
      */
     function retrievePledgeRewards(uint256 pledgeId, address re
+        if(receiver == address(0)) revert Errors.ZeroAddress();
         if(pledgeId >= pledgesIndex()) revert Errors.InvalidPle
         address creator = pledgeOwner[pledgeId];
         if(msg.sender != creator) revert Errors.NotPledgeCreato
-        if(receiver == address(0)) revert Errors.ZeroAddress();
+
 
         Pledge storage pledgeParams = pledges[pledgeId];
         if(pledgeParams.endTimestamp > block.timestamp) revert
```

https://github.com/code-423n4/2022-10-paladin/blob/d6d0c0e57ad80f15e9691086c9c7270d4ccfe0e6/contracts/WardenPledge.sol#L488-L515

```
File: /contracts/WardenPledge.sol
488:     function closePledge(uint256 pledgeId, address receiver)
489:         if(pledgeId >= pledgesIndex()) revert Errors.Invalid
490:         address creator = pledgeOwner[pledgeId];
491:         if(msg.sender != creator) revert Errors.NotPledgeCrea
492:         if(receiver == address(0)) revert Errors.ZeroAddress
```

```
diff --git a/contracts/WardenPledge.sol b/contracts/WardenPledge
index beb990d..c06f2ee 100644
--- a/contracts/WardenPledge.sol
+++ b/contracts/WardenPledge.sol
@@ -486,10 +486,11 @@ contract WardenPledge is Ownable, Pausable
      * @param receiver Address to receive the remaining rewards
      */
     function closePledge(uint256 pledgeId, address receiver) ex
```

```
+         if(receiver == address(0)) revert Errors.ZeroAddress();
          if(pledgeId >= pledgesIndex()) revert Errors.InvalidPle
          address creator = pledgeOwner[pledgeId];
          if(msg.sender != creator) revert Errors.NotPledgeCreato
-         if(receiver == address(0)) revert Errors.ZeroAddress();
```

https://github.com/code-423n4/2022-10-paladin/blob/d6d0c0e57ad80f15e9691086c9c7270d4ccfe0e6/contracts/WardenPledge.sol#L525-L533

```
File: /contracts/WardenPledge.sol
525:     function _addRewardToken(address token, uint256 minRewar
526:         if(minAmountRewardToken[token] != 0) revert Errors.A
527:         if(token == address(0)) revert Errors.ZeroAddress();
528:         if(minRewardPerSecond == 0) revert Errors.NullValue(
```

```
diff --git a/contracts/WardenPledge.sol b/contracts/WardenPledge
index beb990d..71d0087 100644
--- a/contracts/WardenPledge.sol
+++ b/contracts/WardenPledge.sol
@@ -523,10 +523,10 @@ contract WardenPledge is Ownable, Pausable
      * @param minRewardPerSecond Minmum amount of reward per vot
      */
     function _addRewardToken(address token, uint256 minRewardPe
-         if(minAmountRewardToken[token] != 0) revert Errors.Alre
          if(token == address(0)) revert Errors.ZeroAddress();
          if(minRewardPerSecond == 0) revert Errors.NullValue();

+         if(minAmountRewardToken[token] != 0) revert Errors.Alre
+
          minAmountRewardToken[token] = minRewardPerSecond;

          emit NewRewardToken(token, minRewardPerSecond);
```

[G-05] Using storage instead of memory for structs/arrays saves gas

When fetching data from a storage location, assigning the data to a memory variable causes all fields of the struct/array to be read from storage, which incurs a Gcoldsload (2100 gas) for each field of the struct/array. If the fields are read from the new

memory variable, they incur an additional MLOAD rather than a cheap stack read. Instead of declearing the variable with the memory keyword, declaring the variable with the storage keyword and caching any fields that need to be re-read in stack variables, will be much cheaper, only incuring the Gcoldsload for the fields actually read. The only time it makes sense to read the whole struct/array into a memory variable, is if the full struct/array is being returned by the function, is being passed to a function that requires memory, or if the array/struct is being read from another memory array/struct

https://github.com/code-423n4/2022-10-paladin/blob/d6d0c0e57ad80f15e9691086c9c7270d4ccfe0e6/contracts/WardenPledge.sol#L227

```
File: /contracts/WardenPledge.sol
227:         Pledge memory pledgeParams = pledges[pledgeId];
```

[G-06] Using unchecked blocks to save gas

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 see resource

https://github.com/code-423n4/2022-10-paladin/blob/d6d0c0e57ad80f15e9691086c9c7270d4ccfe0e6/contracts/WardenPledge.sol#L268

```
File: /contracts/WardenPledge.sol
268:         pledgeAvailableRewardAmounts[pledgeId] -= rewardAmou
```

The operation `pledgeAvailableRewardAmounts[pledgeId] -= rewardAmount;` cannot underflow due to the check on Line 267 that ensures that `pledgeAvailableRewardAmounts[pledgeId]` is greater than `rewardAmount` before perfoming the arithmetic operation.

Disclosures

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

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

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

Top