TempleGold Audit Report

Version 1.0

Shikhar Agarwal

August 1, 2024

TempleGold Audit Report

Shikhar Agarwal

Aug 1, 2024

Prepared by: Shikhar Agarwal (https://www.codehawks.com/profile/clk3yh639002emf08ywok1hzf)

Lead Auditors: - Shikhar Agarwal

Table of Contents

About the Project

Temple Gold is a claim on future airdrops. Airdrops are farmed token allocations from protocols like Ethena or Origami. We introduce Temple Gold as a non-tradable claim on farmed token allocations.

Using auction markets and staking contracts, users can get some emission of TGLD to claim farmed token allocations that temple creates in the future. Users can claim allocations from specialized auctions called Spice auctions.

See more contest details here - https://codehawks.cyfrin.io/c/2024-07-templegold/

Disclaimer

As the sole auditor all efforts have been made to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

Timeline

Jul 4th, 2024 - Jul 11th, 2024

Sponsor

TempleDAO

```
Commit Hash: 57a3e597e9199f9e9e0c26aab2123332eb19cc28
```

Scope

```
contracts/
#-- templegold
    #-- AuctionBase.sol
    #-- DaiGoldAuction.sol
    #-- EpochLib.sol
    #-- SpiceAuction.sol
    #-- SpiceAuctionFactory.sol
    #-- TempleGold.sol
    #-- TempleGoldAdmin.sol
    #-- TempleGoldStaking.sol
    #-- TempleTeleporter.sol
```

Roles

Elevated Access: Gnosis multisig address of Temple DAO. Not set up yet but ideally a 3/4 multisig address. Elevated access controls staking and TGLD parameters. There is centralization risks if 3 addresses get compromised.

Staker: Account staking Temple for TGLD rewards.

DAO Executor: Temple DAO governance executor contract. Controls spice auction configuration and parameters through governance.

Distribution starter: A bot if set to non-zero address. Distributes TGLD for the next reward epoch.

Issues found

Severity	Count of Findings
High	1
Medium	1
Low	1

Findings

High Risk Findings

H-01. Users staking in TempleGoldStaking will have more than expected claimable rewards and create a DoS as for a user the rewardPerToken also contains the rewardPerToken from starting due to unupdated `userRewardPerTokenPaid` on staking

Medium Risk Findings

M-01. Tokens cannot be recovered for `SpiceAuction` even when the auction is in cooldown

Low Risk Findings

L-01. `SpiceAuction::removeAuctionConfig` cannot be called for starting auction as it reverts due to startTime check

High Risk Findings

H-01. Users staking in TempleGoldStaking will have more than expected claimable rewards and create a DoS as for a user the rewardPerToken also contains the rewardPerToken from starting due to unupdated `userRewardPerTokenPaid` on staking

Relevant Github Links

https://github.com/Cyfrin/2024-07-templegold/blob/main/protocol/contracts/templegold/TempleGoldStaking.sol#L495

https://github.com/Cyfrin/2024-07-templegold/blob/main/protocol/contracts/templegold/TempleGoldStaking.sol#L463

Summary

The Staking keeps a track of global `rewardPerTokenStored` and for a particular user the `userRewardPerTokenPaid`, which ensures that the user will get their reward subtracting their previously claimed reward.

When a user opens a stake in future within the current reward distribution period they will have their `userRewardPerTokenPaid` for that stake as 0 which was expected to be updated to

`rewardPerTokenStored` so that they will be excluded from all the previous reward accumulated and have a start with current reward distribution, but due to this missing implementation the user will be included in the rewards distribution from the starting and have their claimable rewards more than expected and will make (Total Claimable Rewards of all Users > Total TGLD avaiable for claims for current distribution period) and thus creates a DoS for withdrawal and other stuffs.

Vulnerability Details

The vulnerability is present in the `TempleGoldStaking::_applyStake` function where it lags the implementation to update the `userRewardPerTokenPaid` which makes the users participating to have claimable rewards from the starting value of `rewardPerTokenStored` which will thus increase their claimable rewards by a large amount and creates a Denial of Service for claiming of rewards, as the total claimable rewards are more than TGLD claim avaiable for a particular distribution period, thus some users will not be able to claim.

When a user stakes, the `rewardPerTokenStored` is updated to calculate it for the current timestamp by including the previous supply of stakes before the user has staked so the user will be expected to receive their rewards from when they have deposited for an ongoing reward distribution period, but due to the missing updation of `userRewardPerTokenPaid` when a user stakes, it makes it calcuate their rewards by considering the previous rewards.

Impact

- As the claimable rewards for users will be much larger than they actually should get, will make the total claimable rewards of user much more than the actual TGLD amount available for claim, thus some or all users will not be able to perform their claim.
- If minted TGLD is supplied to `TempleGoldStaking`, so after the reward for a period is consumed then this TGLD which was for next distribution will be consumed and thus creating DoS for the whole Staking system.

PoC

Add the below coded PoC in the `TempleGoldStakingTest` contract present in file: `test/forge/templegold/TempleGoldStaking.t.sol`

Run the test:

```
forge test --mt
    test_ClaimableRewardsAreAllocated_MoreThanWhatWasExpected
```

```
function test_ClaimableRewardsAreAllocated_MoreThanWhatWasExpected()
    public {
     // send temple gold to Temple Gold Staking
     // to keep things simple, lets say the reward duration is of 2
        weeks and vesting period is of 1 week
     // let the tgld amount be such that reward rate is 1 tgld per
        second
     vm.startPrank(executor);
     staking.setVestingPeriod(1 weeks);
     staking.setRewardDuration(2 weeks);
     staking.setRewardDistributionCoolDown(0);
     vm.stopPrank();
     uint256 tgldTotalRewardAmount = 2 weeks * 1 ether;
     deal(address(templeGold), address(staking), tgldTotalRewardAmount);

     vm.prank(address(templeGold));
     staking.notifyDistribution(tgldTotalRewardAmount);


     deal(address(templeToken), address(this), 1000000 ether);
     templeToken.approve(address(staking), type(uint256).max);

     uint256 stakeAmt = 50 ether;
     // stake for alice
     staking.stakeFor(alice, stakeAmt);

     // start the distribution reward
     staking.distributeRewards();

     // now after 1 week bob stakes
     vm.warp(block.timestamp + 1 weeks);
     staking.stakeFor(bob, stakeAmt);

     // alice claims current reward
     uint256 aliceLatestStakeIdx = staking.getAccountLastStakeIndex(
        alice);
     staking.getReward(alice, aliceLatestStakeIdx);

     // now after 1 week, bob vesting period will be over
     vm.warp(block.timestamp + 1 weeks);

     // now alice and bob again claims their reward
     uint256 bobLatestStakeIdx = staking.getAccountLastStakeIndex(bob);

     uint256 aliceCurrentClaimable = staking.earned(alice,
        aliceLatestStakeIdx);
     uint256 bobCurrentEarnedClaimable = staking.earned(bob,
        bobLatestStakeIdx);
     uint256 totalCurrentClaimable = aliceCurrentClaimable +
        bobCurrentEarnedClaimable;
```

```
44
45      // As the reward for bob was calculated considering the previous
            previous reward per token also
46      // therefore the totalCurrentClaimable will be more than current
            avaiable rewards
47      // and one of them will not be able to claim their reward
48      // Because bob was allocated more rewards than they were actually
            expected to get
49
50      assert(totalCurrentClaimable > templeGold.balanceOf(address(staking
            )));
51
52      // alice claims
53      // even if Alice doesn't make a claim, the bob claim will always
            revert because
54      // the claimable amt calculated is higher than the total amount
            available for claim
55      staking.getReward(alice, aliceLatestStakeIdx);
56
57      // bob claim reverts
58      vm.expectRevert();
59      staking.getReward(bob, bobLatestStakeIdx);
60  }
```

Tools Used

Manual Review, Unit Test in Foundry

Recommendations

When a user peforms a stake, then it should be ensured that they are not included in the previous rewards, therefore in the `_applyStake` function update the `userRewardPerTokenPaid` for that user's stake to current `rewardPerTokenStored` so that while calculating their reward they will only be included in their current period and not in previous reward period from the starting.

```
1  function _applyStake(address _for, uint256 _amount, uint256 _index)
       internal updateReward(_for, _index) {
2      totalSupply += _amount;
3      _balances[_for] += _amount;
4      _stakeInfos[_for][_index] = StakeInfo(uint64(block.timestamp),
           uint64(block.timestamp + vestingPeriod), _amount);
5  +   userRewardPerTokenPaid[_for][_index] = uint256(rewardData.
       rewardPerTokenStored);
6      emit Staked(_for, _amount);
7  }
```

Now after applying the recommendation, the test fails due to a failing assert which shows that now the rewards are allocated correctly.

Medium Risk Findings

M-01. Tokens cannot be recovered for `SpiceAuction` even when the auction is in cooldown

Relevant Github Links

https://github.com/Cyfrin/2024-07-templegold/blob/main/protocol/contracts/templegold/SpiceAuction.sol#L250-L252

https://github.com/Cyfrin/2024-07-templegold/blob/main/protocol/contracts/templegold/SpiceAuction.sol#L119-L126

Summary

The tokens in in SpiceAuction is expected to be recovered by the `SpiceAuction::recoverToken` function for the last but not started auction. For the case when the `startAuction` is called for an auction and it is currently in cooldown, the function reverts with a message to call `RemoveAuctionConfig` function.

But `removeAuctionConfig` just removes the config of the auction and doesn't perform any recovery of the token, as a result of which calling `removeAuctionConfig` will remove config and tokens that were expected to be recovered will be stuck in the contract and as `_totalAuctionTokenAllocation` has the value of the token that was expected to be recovered, and the contract evaluate tokens in there as tokens that are allocated to the auction and now there is no possible way to perform recovery after calling `removeAuctionConfig`, thus resulting in stuck funds.

Vulnerability Details

The vulnerability is present in the `recoverToken` function of the `SpiceAuction` contract, where it reverts with a message to call the `removeAuctionConfig` for the case when an auction is in cooldown and yet to be started.

It was expected for the tokens allocated for an auction currently in cooldown to be recovered via `recoverToken`, but due to incorrect implementation there is no way to perform recover tokens

operation due to above discussed issue, as remove auction config just removes the config and does nothing else.

As `removeAuctionConfig` performs a reset operation on the epochs and auctionConfigs mapping but as `startAuction` function was already called so the funds were already allocated in _totalAuctionTokenAllocation mapping as a result of which there is no way for those tokens to be recovered and `removeAuctionConfig` doesn't perform any updations related to `_totalAuctionTokenAllocation`.

Impact

Tokens cannot be recovered for the case when the auction is in cooldown and is not started yet.

PoC

Add the below coded PoC in the `SpiceAuctionTest` contract in the file: `test/forge/templegold/SpiceAuction.t.sol`

Run the test:

```
forge test --mt
    test_AuctionTokenCannotBeRecovered_EvenIfCoolDownIsActive
```

```
function test_AuctionTokenCannotBeRecovered_EvenIfCoolDownIsActive()
    public {
    uint160 auctionTokenReward = 100e18;
    address recipient = makeAddr("recipient");

    // setting up auction config
    ISpiceAuction.SpiceAuctionConfig memory _config = ISpiceAuction.
        SpiceAuctionConfig({
        duration: 1 weeks,
        waitPeriod: 1,
        minimumDistributedAuctionToken: auctionTokenReward,
        starter: address(0),
        startCooldown: 1 hours,
        isTempleGoldAuctionToken: false,
        activationMode: ISpiceAuction.ActivationMode.
            AUCTION_TOKEN_BALANCE,
        recipient: recipient
    });

    vm.prank(daoExecutor);
    spice.setAuctionConfig(_config);

    vm.warp(block.timestamp + _config.waitPeriod);
```

```
21      deal(daiToken, address(spice), auctionTokenReward);
22      spice.startAuction();
23
24      // now the cooldown period is active for the current epoch
25      uint256 currentEpoch = spice.currentEpoch();
26      IAuctionBase.EpochInfo memory _epochInfo = spice.getEpochInfo(
            currentEpoch);
27
28      // epoch info set, and auction in cooldown
29      assert(_epochInfo.startTime > block.timestamp);
30
31      // Now there is a need to recover the reward token
32      // the recover token function have a revert statement with the
            message to call remove auction config
33      // when startAuction is called, but on cooldown for start
34
35      uint256 initRecipientBalance = IERC20(daiToken).balanceOf(recipient
            );
36
37      vm.startPrank(daoExecutor);
38      vm.expectRevert(ISpiceAuction.RemoveAuctionConfig.selector);
39      spice.recoverToken(daiToken, recipient, auctionTokenReward);
40
41      // now call the remove auction config as guided to recover the
            tokens
42      spice.removeAuctionConfig();
43      vm.stopPrank();
44
45      // but as we know remove auction config is just used to reset the
            config parameters.
46      // thus it is observed that the recoverToken function has
            implemented things in opposite manner
47      assertEq(IERC20(daiToken).balanceOf(recipient) -
            initRecipientBalance, 0);
48  }
```

Tools Used

Manual Review, Unit Test in Foundry

Recommendations

- Updation 1 Update the `recoverToken` function to perform the recovery of the tokens for the auction that is in cooldown and yet to be started. Instead of performing a revert for this case recover the tokens to the recipient. Perform the following recover operation in the `recoverToken` function for the case when auction is in cooldown:

```
+ uint256 amountToRecover = info.totalAuctionTokenAmount;
+ _totalAuctionTokenAllocation[token] -= amountToRecover;
+ IERC20(token).safeTransfer(to, amountToRecover);
+ delete auctionConfigs[id];
+ delete epochs[id];
```

- Updation 2 In `removeAuctionConfig` update the `_totalAuctionTokenAllocation` mapping to remove the tokens allocated for the auction that is being removed.

```
if (!configSetButAuctionStartNotCalled) {
    /// @dev unlikely because this is a DAO execution, but avoid
       deleting old ended auctions
    if (info.hasEnded()) { revert AuctionEnded(); }
    /// auction was started but cooldown has not passed yet
+   (, address auctionToken) = _getBidAndAuctionTokens(auctionConfigs[
    id]);
+   _totalAuctionTokenAllocation[auctionToken] -= info.
    totalAuctionTokenAmount;
    delete auctionConfigs[id];
    delete epochs[id];
    _currentEpochId = id - 1;
    emit AuctionConfigRemoved(id, id);
}
```

Low Risk Findings

L-01. `SpiceAuction::removeAuctionConfig` cannot be called for starting auction as it reverts due to startTime check

Relevant Github Links

https://github.com/Cyfrin/2024-07-templegold/blob/main/protocol/contracts/templegold/SpiceAuction.sol#L113

Summary

The `removeAuctionConfig` function is expected to reset an auction config either for an auction that is in cooldown phase or for an auction for which config is set but `startAuction` is not called.

Vulnerability Details

The vulnerability is present in `removeAuctionConfig` function at line 113 which reverts when startTime for current auction epoch info is 0, but it missed the scenario where for the very first auction, as when config is set then `startTime` will be 0, as epoch id is not incremented as a result of which config cannot be reset.

For the very first auction config set via `setAuctionConfig`, the config is updated for the next epoch id, i.e 1, the currentEpochId still stores 0, as it will be updated when `startAuction` is called.

Now, for the requirement to remove auction config for the very first auction will fail, as it checks for the startTime for the current epoch id to be `0` and as for the current epoch id which is not used, it will always be 0 and as a result of which for the very first auction for which only config is set, the `removeAuctionConfig` cannot be called.

Impact

`removeAuctionConfig` will revert for starting auction, and auction config cannot be reset.

Tools Used

Manual Review

Recommendations

Remove the below check from `removeAuctionConfig`:

```
1 - if (info.startTime == 0) { revert InvalidConfigOperation(); }
```