TempleGold Audit Report

Prepared by Cyfrin

Version 2.1

Lead Auditors

Hans

June 18, 2024

Contents

1 About Cyfrin **2**

2 Disclaimer **2**

3 Risk Classification **2**

4 Protocol Summary **2**

5 Audit Scope **2**

6 Executive Summary **2**

7 Findings **5**

7.1 High Risk . . . 5

7.1.1 A malicious staker can delay the increase of any delegator's voteWeight as much as he wants 5

7.1.2 Malicious users can double their voting power. . . . 5

7.1.3 The delegator resetting self-delegation causes multiple issues in the protocol . . . 6

7.2 Medium Risk . . . 8

7.2.1 Misconfiguration of minting TGLD leads to max supply amount being minted too fast . . . 8

7.2.2 When recoverToken in DaiGoldAction contract is called, it does not update the status . . . 8

7.2.3 In staking contract, rewards distribution can be delayed by front-running distributeGold function 8

7.2.4 Configuring compose option will prevent users from receiving TGLD on the destination chain 9

7.2.5 In staking contract, resetting self delegation will reset vote weight to zero . . . 9

7.2.6 Incorrect end time setting for spice auction . . . 9

7.2.7 No token recover mechanism when the auction ends without any bid . . . 10

7.2.8 `TempleGoldStaking.getVoteWeight()` calculates vote weight of an account incorrectly . . . 10

7.2.9 Rewards are calculated as distributed even if there are no stakers, locking the rewards forever 11

7.2.10 While changing vote delegator from himself to another user, `_delegateBalances` should not be decreased . . . 12

7.3 Low Risk . . . 14

7.3.1 In rewards distribution, dust amount is left and stuck in the contract . . . 14

7.3.2 DAI-GOLD auction might not get started for last mint of gold tokens . . . 14

7.3.3 Fee-on-transfer tokens are not supported in spice auction . . . 14

7.3.4 Current design of TempleGold prevents distribution of tokens . . . 14

7.4 Informational . . . 15

7.4.1 Zero addresses not checked in contracts constructors . . . 15

7.4.2 Redundant logic of `send` function of `TempleGold` contract . . . 15

7.4.3 Over-complicated design of delegation . . . 15

7.4.4 TempleGold incompatibility with some chains . . . 15

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse 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.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

TempleDAO is a DAO created to be a safe haven in a sea of risky assets, to facilitate long-term, stable wealth creation with a community-first mindset, and to elevate the overall DeFi experience for investors.

TempleDAO introduces a new cross-chain ERC20 token TempleGold which will be distributed to users through various auctions and Temple staking, as well as allocating a portion to the team. TempleGold inherits from LayerZero's OFT standard to make cross-chain available while the token is only minted on Arbitrum chain.

5 Audit Scope

The scope includes contracts that are related to TempleGold and its distribution, that are located in `templegold` directory in contracts.

6 Executive Summary

Over the course of 10 days, the Cyfrin team conducted an audit on the TempleGold smart contracts provided by TempleDAO. In this period, a total of 21 issues were found.

Summary of the audit findings and any additional executive comments.

Summary

Project Name	TempleGold
Repository	temple
Commit	a5862b9e01f1...
Audit Timeline	May 20th - May 31th
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	3
Medium Risk	10
Low Risk	4
Informational	4
Gas Optimizations	0
Total Issues	21

Summary of Findings

[H-1] A malicious staker can delay the increase of any delegator's voteWeight as much as he wants	Resolved
[H-2] Malicious users can double their voting power.	Resolved
[H-3] The delegator resetting self-delegation causes multiple issues in the protocol	Resolved
[M-1] Misconfiguration of minting TGLD leads to max supply amount being minted too fast	Resolved
[M-2] When recoverToken in DaiGoldAction contract is called, it does not update the status	Resolved
[M-3] In staking contract, rewards distribution can be delayed by front-running distributeGold function	Resolved
[M-4] Configuring compose option will prevent users from receiving TGLD on the destination chain	Resolved
[M-5] In staking contract, resetting self delegation will reset vote weight to zero	Resolved
[M-6] Incorrect end time setting for spice auction	Resolved
[M-7] No token recover mechanism when the auction ends without any bid	Resolved
[M-8] `TempleGoldStaking.getVoteWeight()` calculates vote weight of an account incorrectly	Resolved
[M-9] Rewards are calculated as distributed even if there are no stakers, locking the rewards forever	Resolved

[M-10] While changing vote delegator from himself to another user, `_delegateBalances` should not be decreased	Resolved
[L-1] In rewards distribution, dust amount is left and stuck in the contract	Resolved
[L-2] DAI-GOLD auction might not get started for last mint of gold tokens	Acknowledged
[L-3] Fee-on-transfer tokens are not supported in spice auction	Resolved
[L-4] Current design of TempleGold prevents distribution of tokens	Acknowledged
[I-1] Zero addresses not checked in contracts constructors	Acknowledged
[I-2] Redundant logic of `send` function of `TempleGold` contract	Acknowledged
[I-3] Over-complicated design of delegation	Resolved
[I-4] TempleGold incompatibility with some chains	Acknowledged

7 Findings

7.1 High Risk

7.1.1 A malicious staker can delay the increase of any delegator's voteWeight as much as he wants

Description: When a staker delegates his balance to a delegator, the time duration variable `_weights[delegate].stakeTime` is decreased. However, when undelegating only the balance is decreased. This makes it possible a malicious user can decrease `_weights[delegate].stakeTime`. He can expand the impact of this attack by repeating delegating and undelegating.

Impact: A malicious user can delay the increase of any delegator's `voteWeight` as much as he wants.

Proof of Concept: Here's an example scenario:

Assume that `HalfTime` is 7days.

1. Alice delegates `200` to Bob(a victim).
2. `14` days passed. `t` is `14`. Bob's `votePower` is `200*14/(14+7)=133`.
3. Charlie(a malicious user) delegates `100` to Bob. `t` decreases from `14` to `200*14*7/(300*(14+7)-200*14)=5.6` Bob's `votePower` is `300*5.6/(5.6+7)=133`.
4. Charlie undelegates `100` from Bob. `t` is still `5.6`. However, `balance` is decreased from `300` to `200`. Bob's `votePower` is `200*5.6/(5.6+7)=88`.

A malicious user can decrease the `t` as much as he wants by repeating step 3 and 4 in one transaction. In this way, he can delay the increase of `votePower` as much as he wants. In the worst case, it is possible to limit the `votePower` of any user almost 0 forever.

Note: Two combining ways of delegation (step 3) and undelegation (step 4) are possible. One is the combination of `setUserVoteDelegate()` and `unsetUserVoteDelegate()`. Another is the combination of `stake()` and `withdraw()`.

Recommended Mitigation: The mechanism for adjusting the stakingTime between delegation and undelegation should be improved.

TempleDAO: Fixed in PR 1041

Cyfrin: Verified

7.1.2 Malicious users can double their voting power.

Description: When calculating The `voteWeight` of a delegatee, its own `ownVoteWeight` is added. This makes it possible for some users to double their voting power by constructing a cycle. For example, Alice delegates Bob, Bob delegates Alice.

```
    function getDelegatedVoteWeight(address _delegate) external view returns (uint256) {
        // check address is delegate
        uint256 ownVoteWeight = _voteWeight(_delegate, _balances[_delegate]);
        address callerDelegate = userDelegates[_delegate];
        if (!delegates[_delegate] || callerDelegate == msg.sender) { return ownVoteWeight; }
@>      return ownVoteWeight + _voteWeight(_delegate, _delegateBalances[_delegate]);
    }
```

Impact: Malicious users can double their voting power.

Proof of Concept: Here's an example scenario: 1. Alice delegated his voting weight 100 to Bob. 2. Bob delegated his voting weight 200 to Alice. 3. The total sum of _delegateBalances of Alice and Bob is (100 + 200) + (200 + 100) = 600.

Recommended Mitigation: There might be several ways to mitigate the issue, here's a few:

-The first way

Delegating itself should not be allowed.

```
-   function unsetUserVoteDelegate() external override {
+   function unsetUserVoteDelegate() public override {

    function setSelfAsDelegate(bool _approve) external override {
            [...]
+       if (_approve && !prevStatusTrue) unsetUserVoteDelegate();
    }
```

And, any delegatee should not be allowed to delegate.

```
    function setUserVoteDelegate(address _delegate) external override {
        if (_delegate == address(0)) { revert CommonEventsAndErrors.InvalidAddress(); }
        if (!delegates[_delegate]) {  revert InvalidDelegate(); }
+       if (userDelegates[msg.sender] == msg.sender) { revert(); }
            [...]
    }
```

-The second way

`ownVoteWeight` should not be add, unless if a user delegate himself. This is the typical way for calculationg woting power.

```
    function getDelegatedVoteWeight(address _delegate) external view returns (uint256) {
        // check address is delegate
-       uint256 ownVoteWeight = _voteWeight(_delegate, _balances[_delegate]);
-       address callerDelegate = userDelegates[_delegate];
-       if (!delegates[_delegate] || callerDelegate == msg.sender) { return ownVoteWeight; }
-       return ownVoteWeight + _voteWeight(_delegate, _delegateBalances[_delegate]);
        return _voteWeight(_delegate, _delegateBalances[_delegate]);
    }
```

TempleDAO: Fixed in PR 1040

Cyfrin: Verified

7.1.3 The delegator resetting self-delegation causes multiple issues in the protocol

Description: Whenever the vote power of delegators are changed, the validity of self-delegation of the delegator is not checked, which results in issues in multiple parts of the protocol.

- `unsetUserVoteDelegate`: It does not allow stakers to unset delegation through the function because it tries to subtract delegated balance from zero.
- `_withdrawFor`: It does not allow stakers to withdraw their assets because it tries to subtract delegated balance from zero.
- `_stakeFor`: It allows malicious stakers to get infinite voting power by repeating staking & modifying delegator & withdrawal process.

Impact: It allows attackers to repeat issues in `_stakeFor` to accumulate voting power, also it causes reverts in withdrawals.

Proof of Concept: Here's a scenario where a malicious attacker can get infinite voting power by repeating the following process:

1. Staker delegates to Bob.
2. Bob resets self-delegation.
3. The staker stakes the token.
4. The staker changes the delegation to another party.

Recommended Mitigation: In 3 functions above, it should validate if the delegator has self-delegation enabled at the time of function call.

TempleDAO: Fixed in PR 1034

Cyfrin: Verified

7.2 Medium Risk

7.2.1 Misconfiguration of minting TGLD leads to max supply amount being minted too fast

Description: When minting TGLD through TempleGold contract, the minting amount should be larger than the minimum amount which is 10,000 TGLD. Also if it's the first time that mint happens, the mint amount is always SUPPLY * n/d, which is the amount for one second.

The math above shows that the minting amount per second should be bigger than 10,000 TGLD, which also means that the max supply(1e9 TGLD) gets minted in 1e5 seconds ~ 1day 4hrs.

Impact: Either max supply tokens are minted in 1day 4hrs, or tokens can't be minted for the first minting.

Recommended Mitigation: When TGLD contract is created or setVestingFactor is called for the first time, it should initialize `_lastMintTimestamp`.

TempleDAO: Fixed in PR 1026

Cyfrin: Verified

7.2.2 When recoverToken in DaiGoldAction contract is called, it does not update the status

Description: recoverToken function moves specific amount of TGLD from the auction contract to another and delete last epoch so that it can be reset and restart. The tokens get moved but the status nextAuctionGoldAmount is not subtracted, and this gives incorrect amount of TLGD tokens to the next epoch, and as a result, the bidders won't be able to claim their TGLD.

Impact: The next auction has incorrect number as auction amount which results in users not being able to claim tokens after the auction ends.

Recommended Mitigation: The nextAuctionGoldAmount should be subtracted when tokens are moved in recoverToken function.

TempleDAO: Fixed in PR 1027

Cyfrin: Verified

7.2.3 In staking contract, rewards distribution can be delayed by front-running distributeGold function

Description: In `TempleGoldStaking` contract, the reward distributer calls `distributeRewards` to by minting available TLGD from the token contract and updates reward rate based on the minted amount. However, the distribution works only after the distribution cooldown period is passed:

```
if (lastRewardNotificationTimestamp + rewardDistributionCoolDown > block.timestamp)
    { revert CannotDistribute(); }
```

It uses `lastRewardNotificationTimestamp` as latest updated timestamp, which can be also updated by anyone who calls a public function `distributeGold`. By front-running this function, it prevents the distributor from updating reward rate.

More seriously, if `distributeGold` is called regularly before the distribution cooldown, rewards distribution can be prevented forever.

Impact: Reward distribution can be delayed.

Recommended Mitigation: Either making `distributeGold` permissioned or introduce other state variable that represents the latest timestamp of rewards distribution, which is only updated in `distributeRewards` function.

TempleDAO: Fixed in PR 1028

Cyfrin: Verified

7.2.4 Configuring compose option will prevent users from receiving TGLD on the destination chain

Description: Users are able to send their TGLD by calling `send` function of TempleGold contract. Since `SendParam` struct is passed from users, they can add any field including compose data into the message. However on the destination chain, in `_lzReceive` function, it blocks messages with compose options in it:

```
if (_message.isComposed()) { revert CannotCompose(); }
```

This means that users who call `send` with compose option, they will not receive TGLD tokens on the destination chain.

Impact: Token transfer does not work based on the parameter and causes loss of funds for users.

Recommended Mitigation:

1. In `send` function, if the `_sendParam` includes compose option, it should revert.
2. Even better, `send` function only inputs mandatory fields from users like the amount to send, and it constructs `SendParam` in it rather than receiving it as a whole from the user.

TempleDAO: Fixed in PR 1029

Cyfrin: Verified

7.2.5 In staking contract, resetting self delegation will reset vote weight to zero

Description: In `TempleGoldStaking` contract, when a delegate resets self delegation, the vote weights delegated to the delegate will be removed by calling `_updateAccountWeight`. Since zero is passed as new balance parameter, the `stakingTime` of vote weight is reset to zero:

```
// TempleGoldStaking.sol:L580
if (_newBalance == 0) {
    t = 0;
    _lastShares = 0;
}
```

This means that the delegate's vote weight starts accumulating from zero even though the delegate has their own balance, which should not be decreased.

Impact: The delegate loses voting power, even becoming zero if resetting self delegation happens right before an epoch ends.

Recommended Mitigation: When resetting self delegation, it should not decrease stakeTime so that delegate's voting power remains based on their own balance

TempleDAO: Fixed in PR 1043

Cyfrin: Verified

7.2.6 Incorrect end time setting for spice auction

Description: In `startAuction` function of `SpiceAuction` contract, the end time is set wrong.

```
uint128 startTime = info.startTime = uint128(block.timestamp) + config.startCooldown;
uint128 endTime = info.endTime = uint128(block.timestamp) + config.duration;
```

Currently, `endTime` is set using `block.timestamp`, not `startTime` calculated above.

Impact: The auction end time becomes shorter than expected.

Recommended Mitigation: `endTime` should be calculated using `startTime` and `duration`

```
uint128 endTime = info.endTime = startTime + config.duration;
```

TempleDAO: Fixed in PR 1032

Cyfrin: Verified

7.2.7 No token recover mechanism when the auction ends without any bid

Description: In `SpiceAuction` contract, when an auction ends with no bids, the auction tokens are locked into the contract and can't be recovered. `recoverToken` function doesn't help recover the auction token because it limits recoverable amount to `balance - totalAllocation`

Impact: The auction tokens can be stuck in the contract and can't be recovered, although this would be rare case.

Recommended Mitigation: There has to be a mechanism implemented to recover tokens from auctions when it ends without any bids.

TempleDAO: Fixed in PR 1033

Cyfrin: Verified

7.2.8 `TempleGoldStaking.getVoteWeight()` calculates vote weight of an account incorrectly

Description: The `TempleGoldStaking.sol` contract, the function is expected to return the vote power of account, but sometimes it is calculated incorrectly. It uses current balance of account from L379.

```
File: contracts\templegold\TempleGoldStaking.sol
378:     function getVoteWeight(address account) external view returns (uint256) {
379:         return _voteWeight(account, _balances[account]);
380:     }
```

However, in the `_voteWeight()` function, it uses previous vote weight when `week > currentWeek` from L556. Then, it uses current balance and previous vote weight to calculate the vote power from L563.

```
File: contracts\templegold\TempleGoldStaking.sol
549:     function _voteWeight(address _account, uint256 _balance) private view returns (uint256) {
550:         /// @dev Vote weights are always evaluated at the end of last week
551:         /// Borrowed from st-yETH staking contract
↪   (https://etherscan.io/address/0x583019fF0f430721aDa9cfb4fac8F06cA104d0B4#code)
552:         uint256 currentWeek = (block.timestamp / WEEK_LENGTH) - 1;
553:         AccountWeightParams storage weight = _weights[_account];
554:         uint256 week = uint256(weight.weekNumber);
555:         if (week > currentWeek) {
556:             weight = _prevWeights[_account];
557:         }
558:         uint256 t = weight.stakeTime;
559:         uint256 updated = weight.updateTime;
560:         if (week > 0) {
561:             t += (block.timestamp / WEEK_LENGTH * WEEK_LENGTH) - updated;
562:         }
563:         return _balance * t / (t + halfTime);
564:     }
```

As a result, it calculates the vote power incorrectly.

Impact: TempleGoldStaking.getVoteWeight() calculates vote weight of an account incorrectly.

Proof of Concept: Let's suppose `halfTime` is 7 days.

- Alice stakes `100 ether` on May 3.
- Alice stakes `400 ether` again on May 17.
- She calls the `getVoteWeight()` function on May 18. The function returns `500 * 13 / (13 + 7)` instead of `100 * 13 / (13 + 7)`, which is 5 times greater than real vote power.

Recommended Mitigation: It is recommended to use previous balance of an user when previous vote weight is used in the function.

TempleDAO: Fixed in PR 1043

Cyfrin: Verified

7.2.9 Rewards are calculated as distributed even if there are no stakers, locking the rewards forever

Description: The `TempleGoldStaking.sol` contract is a fork of the Synthetix rewards distribution contract, with slight modifications. The code special-cases the scenario where there are no users, by not updating the cumulative rate when the _totalSupply is zero, but it does not include such a condition for the tracking of the timestamp from L476.

```
File: contracts\templegold\TempleGoldStaking.sol
475:     function _rewardPerToken() internal view returns (uint256) {
476:         if (totalSupply == 0) {
477:             return rewardData.rewardPerTokenStored;
478:         }
479:
480:         return
481:             rewardData.rewardPerTokenStored +
482:             (((_lastTimeRewardApplicable(rewardData.periodFinish) -
483:                 rewardData.lastUpdateTime) *
484:                 rewardData.rewardRate * 1e18)
485:                 / totalSupply);
486:     }
```

Because of this, even when there are no users staking, the accounting logic still thinks funds were being dispersed during that timeframe (because the starting timestamp is updated),

As a result, if the distributeRewards() function is called prior to there being any users staking, the funds that should have gone to the first stakers will instead accrue to nobody, and be locked in the contract forever.

Impact: Non distributed rewards are stuck in the contract.

Proof of Concept: Here's an example scenario: Alice is `distributionStarter` and Bob is a person who wants to stake `Temple`.

- Alice calls the `distributeRewards()` function to mint TGLD for this contract. Let's suppose the minted TGLD is `7*86400 ether` to calculate simply. Then `rewardRate` becomes `1 ether`.
- After 24 hours, Bob stakes `10000` TGLD into the contract.
- After 6 days, Bob withdraw all staked TGLD and claim rewards. Then he gets `6*86400 ether`.

As a result, `86400 ether` is locked in the contract.

Recommended Mitigation: In the function `distributeRewards()`, check if there are enough reward tokens already in the contract.

```
File: contracts\templegold\TempleGoldStaking.sol
245:     function distributeRewards() updateReward(address(0)) external {
246:         if (distributionStarter != address(0) && msg.sender != distributionStarter)
247:             { revert CommonEventsAndErrors.InvalidAccess(); }
248:         // Mint and distribute TGLD if no cooldown set
249:         if (lastRewardNotificationTimestamp + rewardDistributionCoolDown > block.timestamp)
```

```
250:                 { revert CannotDistribute(); }
251:         _distributeGold();
252:         uint256 rewardAmount = nextRewardAmount;
253:         if (rewardAmount == 0 ) { revert CommonEventsAndErrors.ExpectedNonZero(); }
254:         nextRewardAmount = 0;
+            if (totalSupply == 0) { revert CommonEventsAndErrors.NoStaker(); }
255:         _notifyReward(rewardAmount);
256:     }
```

```
File: contracts\common\CommonEventsAndErrors.sol
06: library CommonEventsAndErrors {
07:     error InsufficientBalance(address token, uint256 required, uint256 balance);
08:     error InvalidParam();
09:     error InvalidAddress();
10:     error InvalidAccess();
11:     error InvalidAmount(address token, uint256 amount);
12:     error ExpectedNonZero();
13:     error Unimplemented();
+       error NoStaker();
14:     event TokenRecovered(address indexed to, address indexed token, uint256 amount);
15: }
```

TempleDAO: Fixed in PR 1037

Cyfrin: Verified

7.2.10 While changing vote delegator from himself to another user, `_delegateBalances` should not be decreased

Description: In `setUserVoteDelegate` function, when a user set vote delegator to himeself, his `_delegateBalances` value doesn't contain his `userBalance` from L149. When he changes vote delegator from himself to another user, it subtracts `userBalance` from `_delegateBalances` from L145. But it shouldn't subtract `userBalance`.

```
File: contracts\templegold\TempleGoldStaking.sol
136:         if (userBalance > 0) {
137:             uint256 _prevBalance;
138:             uint256 _newDelegateBalance;
139:             if (removed) {
140:                 // update vote weight of old delegate
141:                 _prevBalance = _delegateBalances[_userDelegate];
142:                 /// @dev skip for a previous delegate with 0 users delegated and 0 own vote weight
143:                 if (_prevBalance > 0) {
144:                     /// @dev `_prevBalance > 0` because when a user sets delegate, vote weight and
↪  `_delegateBalance` are updated for delegate
145:                     _newDelegateBalance = _delegateBalances[_userDelegate] = _prevBalance -
↪  userBalance;
146:                     _updateAccountWeight(_userDelegate, _prevBalance, _newDelegateBalance, false);
147:                 }
148:             }
149:             if (msg.sender != _delegate) { // @audit msg.sender == _delegate ?
150:                 /// @dev Reuse variables
151:                 _prevBalance = _delegateBalances[_delegate];
152:                 _newDelegateBalance = _delegateBalances[_delegate] = _prevBalance + userBalance;
153:                 _updateAccountWeight(_delegate, _prevBalance, _newDelegateBalance, true);
154:             }
155:         }
```

Impact: `_delegateBalances` is not calculated correctly.

Recommended Mitigation: It is recommended to check if previous delegated user is not self.

```
File: contracts\templegold\TempleGoldStaking.sol
        if (removed) {
            // update vote weight of old delegate
            _prevBalance = _delegateBalances[_userDelegate];
            /// @dev skip for a previous delegate with 0 users delegated and 0 own vote weight
-           if (_prevBalance > 0) {
+           if (_prevBalance > 0 && _userDelegate != msg.sender) {
                /// @dev `_prevBalance > 0` because when a user sets delegate, vote weight and
                ↪  `_delegateBalance` are updated for delegate
                _newDelegateBalance = _delegateBalances[_userDelegate] = _prevBalance - userBalance;
                _updateAccountWeight(_userDelegate, _prevBalance, _newDelegateBalance, false);
            }
        }
```

TempleDAO: Fixed in PR 1038

Cyfrin: Verified

7.3 Low Risk

7.3.1 In rewards distribution, dust amount is left and stuck in the contract

Description: When rewards distribution happens, it's distributed over 7 days(604800 seconds), the reward rate is calculated per second, thus every time, the remainder which is less than 604800 wei of tokens are left in the contract.

Recommended Mitigation: The left amount of rewards should be added to next round of distribution.

Temple DAO: Fixed in PR 1046

Cyfrin: Verified

7.3.2 DAI-GOLD auction might not get started for last mint of gold tokens

Description: Usually minting gold happens when the available amount is bigger than 1e4, which will configure the minimum distribution amount of DaiGoldAuction. However when it reaches the max supply, the minting amount can be smaller than minimum amount, thus DaiGoldAuction might not get started because the amount will be smaller than the minimum distribution amount.

Recommended Mitigation: When it reached max supply, it allows to start auction with remaining amount.

Temple DAO: `config.auctionMinimumDistributedGold` is not necessarily equal to the minimum gold mint amount. Also, for the last mint (max supply), `config.auctionMinimumDistributedGold` will be updated.

Cyfrin: Acknowledged

7.3.3 Fee-on-transfer tokens are not supported in spice auction

Description: Fee-on-transfer tokens are not supported in spice auction contract, which might result in incorrect calculation in auction tokens.

Recommended Mitigation: Use pre/post balance difference to calculate actually moved bid token amount.

Temple DAO: Fixed in PR 1046

Cyfrin: Verified

7.3.4 Current design of TempleGold prevents distribution of tokens

Description: With current TempleGold token design, tokens only can be transferred to/from whitelisted addresses. This prevents further distribution of TempleGold tokens where they could be listed on DEXes, put on lending protocols as collateral and so on, which would be impossible because the protocol can not whitelist all the addresses based on user demands.

Recommended Mitigation: There should be a whitelist flag, when it is set to true, only whitelisted from/to addresses are accepted and when it is set to false, it should be open.

Temple DAO: Acknowledged, but TempleGold is not traded or used on lending platforms

Cyfrin: Acknowledged

7.4 Informational

7.4.1 Zero addresses not checked in contracts constructors

Description: In constructors of `DaiGoldAuction`, `TempleGold`, and `TempleGoldStaking` contracts, zero addresses are not validated.

7.4.2 Redundant logic of `send` function of `TempleGold` contract

Description: The `send` function is a copy of `send` in `OFTCore`, only destination address is validated. This can be updated using `super.send`.

7.4.3 Over-complicated design of delegation

Description: In staking contract, the logic around delegation is over-complicated, where it has concepts of `delegation`, `self-delegation`, and `delegation to self`, of these logic mixed, which could be simplified.

7.4.4 TempleGold incompatibility with some chains

Description: Because of `PUSH0` not supported in 0.8.19 or lower versions of solidity compiler, TempleGold will be incompatible with chains like Linea where it only supports solidity compiler 0.8.19 or lower.