SHERLOCK

SHERLOCK SECURITY REVIEW FOR

SHERLOCK

Prepared for:	3D FrankenPunks
Prepared by:	Sherlock
Lead Security Expert:	WATCHPUG
Dates Audited:	November 9 - November 14, 2022
Prepared on:	November 23, 2022

Introduction

The Franken DAO is a fully on chain governance system for FrankenPunks holders, which overcomes the "1 token, 1 vote" problem by rewarding long term stakers and active participants in the DAO.

Scope

The following contracts are in scope:

- `src/Governance.sol` (337 nSLOC)
- `src/Staking.sol` (333 nSLOC)
- `src/Executor.sol` (44 nSLOC)
- `src/proxy/GovernanceProxy.sol` (27 nSLOC)
- `src/utils/Admin.sol` (56 nSLOC)
- `src/utils/Refundable.sol` (22 nSLOC)

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Issues found

Medium	High
7	4

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

0x52
hansfriese
Trumpero
cccz
curiousapple
Haruxe
Tomo
neumo
rvierdiiev
WATCHPUG
John
koxuan
bin2chen
Nyx
Bnke0x0
saian

Issue H-1: The total community voting power is updated incorrectly when a user delegates.

Source: https://github.com/sherlock-audit/2022-11-frankendao-judging/issues/91

Found by

Trumpero, curiousapple, hansfriese

Summary

When a user delegates their voting power from staked tokens, the total community voting power should be updated. But the update logic is not correct, the the total community voting power could be wrong values.

Vulnerability Detail

```
tokenVotingPower[currentDelegate] -= amount;
tokenVotingPower[_delegatee] += amount;

// If a user is delegating back to themselves, they regain their community voting
↪  power, so adjust totals up
if (_delegator == _delegatee) {
  _updateTotalCommunityVotingPower(_delegator, true);

// If a user delegates away their votes, they forfeit their community voting
↪  power, so adjust totals down
} else if (currentDelegate == _delegator) {
  _updateTotalCommunityVotingPower(_delegator, false);
}
```

When the total community voting power is increased in the first if statement, `_delegator`'s token voting power might be positive already and community voting power might be added to total community voting power before.

Also, `currentDelegate`'s token voting power might be still positive after delegation so we shouldn't remove the communitiy voting power this time.

Impact

The total community voting power can be incorrect.

Code Snippet

https://github.com/sherlock-audit/2022-11-frankendao/blob/main/src/Staking.sol#L293-L313

Tool used

Manual Review

Recommendation

Add more conditions to check if the msg.sender delegated or not.

```
if (_delegator == _delegatee) {
    if(tokenVotingPower[_delegatee] == amount) {
        _updateTotalCommunityVotingPower(_delegator, true);
    }
    if(tokenVotingPower[currentDelegate] == 0) {
        _updateTotalCommunityVotingPower(currentDelegate, false);
    }
} else if (currentDelegate == _delegator) {
    if(tokenVotingPower[_delegatee] == amount) {
        _updateTotalCommunityVotingPower(_delegatee, true);
    }
    if(tokenVotingPower[_delegator] == 0) {
        _updateTotalCommunityVotingPower(_delegator, false);
    }
}
```

Discussion

zobront

Fixed: https://github.com/Solidity-Guild/FrankenDAO/pull/15

Note for JTP: Please double check this one, as I'm 99% confident but would love a second set of eyes on it.

Issue H-2: `Staking.unstake()` doesn't decrease the original voting power that was used in `Staking.stake()`.

Source: https://github.com/sherlock-audit/2022-11-frankendao-judging/issues/70

Found by

Haruxe, 0x52, hansfriese

Summary

`Staking.unstake()` doesn't decrease the original voting power that was used in `Staking.stake()`.

Vulnerability Detail

When users stake/unstake the underlying NFTs, it calculates the token voting power using getTokenVotingPower() and increases/decreases their voting power accordingly.

```
function getTokenVotingPower(uint _tokenId) public override view returns (uint) {
  if (ownerOf(_tokenId) == address(0)) revert NonExistentToken();

  // If tokenId < 10000, it's a FrankenPunk, so 100/100 = a multiplier of 1
  uint multiplier = _tokenId < 10_000 ? PERCENT : monsterMultiplier;

  // evilBonus will return 0 for all FrankenMonsters, as they are not eligible
 ↪  for the evil bonus
  return ((baseVotes * multiplier) / PERCENT) + stakedTimeBonus[_tokenId] +
 ↪  evilBonus(_tokenId);
}
```

But `getTokenVotingPower()` uses some parameters like `monsterMultiplier` and `baseVotes` and the output would be changed for the same `tokenId` after the admin changed these settings.

Currently, `_stake()` and `_unstake()` calculates the token voting power independently and the below scenario would be possible.

- At the first time, `baseVotes=20,monsterMultiplier=50`.
- A user staked a `FrankenMonsters` and his voting power = 10 here.
- After that, the admin changed `monsterMultiplier=60`.
- When a user tries to unstake the NFT, the token voting power will be `20*60/100=12` here.

- So it will revert with uint underflow here.
- After all, he can't unstake the NFT.

Impact

`votesFromOwnedTokens` might be updated wrongly or users can't unstake for the worst case because it doesn't decrease the same token voting power while unstaking.

Code Snippet

https://github.com/sherlock-audit/2022-11-frankendao/blob/main/src/Staking.sol#L427-L440

Tool used

Manual Review

Recommendation

I think we should add a mapping like `tokenVotingPower` to save an original token voting power when users stake the token and decrease the same amount when they unstake.

Discussion

zobront

Fixed: https://github.com/Solidity-Guild/FrankenDAO/pull/17

Issue H-3: Unbounded `_unlockTime` allows the attacker to get a huge `stakedTimeBonus` and dominate the voting

Source: https://github.com/sherlock-audit/2022-11-frankendao-judging/issues/53

Found by

Trumpero, WATCHPUG, neumo, bin2chen, curiousapple, koxuan, John, hansfriese

Summary

`stakingSettings.maxStakeBonusTime` is not enforced, allowing the attacker to gain a huge `stakedTimeBonus` by using a huge value for `_unlockTime`.

Vulnerability Detail

There is no max `_unlockTime` check in `_stakeToken()` to enforce the `stakingSettings.maxStakeBonusTime`.

As a result, an attacker can set a huge value for `_unlockTime` and get an enormous `stakedTimeBonus`.

Impact

The attacker can get a huge amount of votes and dominate the voting.

Code Snippet

https://github.com/sherlock-audit/2022-11-frankendao/blob/main/src/Staking.sol#L389-L394

https://github.com/sherlock-audit/2022-11-frankendao/blob/main/src/Staking.sol#L356-L384

Tool used

Manual Review

Recommendation

Change to:

```
function _stakeToken(uint _tokenId, uint _unlockTime) internal returns (uint) {
  if (_unlockTime > 0) {
    unlockTime[_tokenId] = _unlockTime;
    uint time = _unlockTime - block.timestamp;
```

```
    uint maxtime = stakingSettings.maxStakeBonusTime;
    uint maxBonus = stakingSettings.maxStakeBonusAmount;
    if (time < stakingSettings.maxStakeBonusTime){
      uint fullStakedTimeBonus = (time * maxBonus) / maxtime;
    }else{
      uint fullStakedTimeBonus = maxBonus;
    }
    stakedTimeBonus[_tokenId] = _tokenId < 10000 ? fullStakedTimeBonus :
↪   fullStakedTimeBonus / 2;
  }
```

Discussion

zobront

Fixed: https://github.com/Solidity-Guild/FrankenDAO/pull/13

Issue H-4: Staking#_unstake removes votes from wrong person if msg.sender != owner

Source: https://github.com/sherlock-audit/2022-11-frankendao-judging/issues/30

Found by

0x52, cccz

Summary

Staking#_unstake allows any msg.sender to unstake tokens for any owner that has approved them. The issue is that even when msg.sender != owner the votes are removed from msg.sender instead of owner. The result is that the owner keeps their votes and msg.sender loses theirs. This could be abused to hijack or damage voting.

Vulnerability Detail

```
address owner = ownerOf(_tokenId);
if (msg.sender != owner && !isApprovedForAll[owner][msg.sender] && msg.sender !=
↪  getApproved[_tokenId]) revert NotAuthorized();
```

Staking#_unstake allows any msg.sender to unstake tokens for any owner that has approved them.

```
uint lostVotingPower;
for (uint i = 0; i < numTokens; i++) {
    lostVotingPower += _unstakeToken(_tokenIds[i], _to);
}

votesFromOwnedTokens[msg.sender] -= lostVotingPower;
// Since the delegate currently has the voting power, it must be removed from
↪  their balance
// If the user doesn't delegate, delegates(msg.sender) will return self
tokenVotingPower[getDelegate(msg.sender)] -= lostVotingPower;
totalTokenVotingPower -= lostVotingPower;
```

After looping through _unstakeToken all accumulated votes are removed from msg.sender. The problem with this is that msg.sender is allowed to unstake tokens for users other than themselves and in these cases they will lose votes rather than the user who owns the token.

Example: User A and User B both stake tokens and have 10 votes each. User A approves User B to unstake their tokens. User B calls unstake for User A. User B is msg.sender and User A is owner. The votes should be removed from owner but

instead are removed from msg.sender. The result is that after unstaking User B has a vote balance of 0 while still having their locked token and User B has a vote balance of 10 and their token back. Now User B is unable to unstake their token because their votes will underflow on unstake, permanently trapping their NFT.

Impact

Votes are removed incorrectly if msg.sender != owner. By extension this would forever trap msg.sender tokens in the contract.

Code Snippet

https://github.com/sherlock-audit/2022-11-frankendao/blob/main/src/Staking.sol#L427-L458

Tool used

Manual Review

Recommendation

Remove the ability for users to unstake for other users

Discussion

zobront

Fixed: https://github.com/Solidity-Guild/FrankenDAO/pull/14

Issue M-1: [Tomo-M3] Use safeMint instead of mint for ERC721

Source: https://github.com/sherlock-audit/2022-11-frankendao-judging/issues/65

Found by

Tomo

Summary

Use safeMint instead of mint for ERC721

Vulnerability Detail

The `msg.sender` will be minted as a proof of staking NFT when `_stakeToken()` is called.

However, if `msg.sender` is a contract address that does not support ERC721, the NFT can be frozen in the contract.

As per the documentation of EIP-721:

> A wallet/broker/auction application MUST implement the wallet interface if it will accept safe transfers.

Ref: https://eips.ethereum.org/EIPS/eip-721

As per the documentation of ERC721.sol by Openzeppelin

Ref: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/ERC721.sol#L274-L285

```
/**
 * @dev Mints `tokenId` and transfers it to `to`.
 *
 * WARNING: Usage of this method is discouraged, use {_safeMint} whenever
 ↪  possible
 *
 * Requirements:
 *
 * - `tokenId` must not exist.
 * - `to` cannot be the zero address.
 *
 * Emits a {Transfer} event.
 */
function _mint(address to, uint256 tokenId) internal virtual {
```

Impact

Users possibly lose their NFTs

Code Snippet

https://github.com/sherlock-audit/2022-11-frankendao/blob/main/src/Staking.sol#L411

```
_mint(msg.sender, _tokenId);
```

Tool used

Manual Review

Recommendation

Use `safeMint` instead of `mint` to check received address support for ERC721 implementation.

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/ERC721.sol#L262

Discussion

zobront

I might consider this a duplicate of #55 but not sure how this is usually judged. We will be changing this function based on other issues to not allow "approved" spenders, so msg.sender will be the owner of the FrankenPunk, which ensures they are able to hold NFTs.

zobront

Fixed: https://github.com/Solidity-Guild/FrankenDAO/pull/14

I didn't need to add safeMint, as I made a change for another issue that removed the ability to non holder to unstake, which means they have the ability to hold NFTs.

Issue M-2: [Medium-1] Hardcoded `monsterMultiplier` in case of `stakedTimeBonus` disregards the updates done to `monsterMultiplier` through `setMonsterMultiplier()`

Source: https://github.com/sherlock-audit/2022-11-frankendao-judging/issues/56

Found by

curiousapple, hansfriese

Summary

[Medium-1] Hardcoded monsterMultiplier in case of `stakedTimeBonus` disregards the updates done to `monsterMultiplier` through `setMonsterMultiplier()`

Vulnerability Detail

FrankenDAO allows users to stake two types of NFTs, `Frankenpunks` and `Frankenmonsters` , one of which is considered more valuable, ie: `Frankenpunks`,

This is achieved by reducing votes applicable for `Frankenmonsters` by `monsterMultiplier`.

```
function getTokenVotingPower(uint _tokenId) public override view returns (uint) {
      if (ownerOf(_tokenId) == address(0)) revert NonExistentToken();

      // If tokenId < 10000, it's a FrankenPunk, so 100/100 = a multiplier of 1
      uint multiplier = _tokenId < 10_000 ? PERCENT : monsterMultiplier;

      // evilBonus will return 0 for all FrankenMonsters, as they are not
↪   eligible for the evil bonus
      return ((baseVotes * multiplier) / PERCENT) + stakedTimeBonus[_tokenId] +
↪   evilBonus(_tokenId);
    }
```

This `monsterMultiplier` is initially set as 50 and could be changed by governance proposal.

```
function setMonsterMultiplier(uint _monsterMultiplier) external onlyExecutor {
    emit MonsterMultiplierChanged(monsterMultiplier = _monsterMultiplier);
  }
```

However, one piece of code inside the FrakenDAO staking contract doesn't consider this and has a monster multiplier hardcoded.

```
function stake(uint[] calldata _tokenIds, uint _unlockTime)
----
function _stakeToken(uint _tokenId, uint _unlockTime) internal returns (uint) {
   if (_unlockTime > 0) {
     --------
      stakedTimeBonus[_tokenId] = _tokenId < 10000 ? **fullStakedTimeBonus :
↪  fullStakedTimeBonus / 2;**
    }
--------
```

Hence any update done to `monsterMultiplier` would not reflect in the calculation of `stakedTimeBonus`, and thereby votes.

Impact : Medium

Any update done to monsterMultiplier would not be reflected in stakedTimeBonus; it would always remain as /2 or 50%.

Likelihood: Medium

One needs to pass a governance proposal to change the monster multiplier, so this is definitely not a high likelihood; it's not low as well, as there is a clear provision in spec regarding this.

Code Snippet

https://github.com/sherlock-audit/2022-11-frankendao/blob/main/src/Staking.sol#L393

Tool used

Manual Review

Recommendation

Consider replacing the hardcoded value with monsterMultiplier

Discussion

zobront

Fixed: https://github.com/Solidity-Guild/FrankenDAO/pull/12

Issue M-3: Using `ERC721.transferFrom()` instead of `safeTransferFrom()` may cause the user's NFT to be frozen in a contract that does not support ERC721

Source: https://github.com/sherlock-audit/2022-11-frankendao-judging/issues/55

Found by

saian, rvierdiiev, WATCHPUG, Tomo, Bnke0x0, Nyx

Summary

There are certain smart contracts that do not support ERC721, using `transferFrom()` may result in the NFT being sent to such contracts.

Vulnerability Detail

In `unstake()`, `_to` is param from user's input.

However, if `_to` is a contract address that does not support ERC721, the NFT can be frozen in that contract.

As per the documentation of EIP-721:

> A wallet/broker/auction application MUST implement the wallet interface if it will accept safe transfers.

Ref: https://eips.ethereum.org/EIPS/eip-721

Impact

The NFT may get stuck in the contract that does support ERC721.

Code Snippet

https://github.com/sherlock-audit/2022-11-frankendao/blob/main/src/Staking.sol#L463-L489

Tool used

Manual Review

Recommendation

Consider using `safeTransferFrom()` instead of `transferFrom()`.

Discussion

zobront

Fixed: https://github.com/Solidity-Guild/FrankenDAO/pull/10

Issue M-4: `queue()` should increase `proposalsPassed` instead of `proposalsCreated`

Source: https://github.com/sherlock-audit/2022-11-frankendao-judging/issues/54

Found by

Trumpero, rvierdiiev, 0x52, WATCHPUG, neumo, cccz, John, hansfriese

Summary

`proposalsCreated` will be increased in `verifyProposal()`, `queue()` should increase `proposalsPassed`.

Vulnerability Detail

Based on the context, `queue()` should increase `userCommunityScoreData[proposal.proposer].proposalsPassed` and `totalCommunityScoreData.proposalsPassed` instead.

Impact

Due to the presence of `setProposalsCreatedMultiplier()` and `setProposalsPassedMultiplier()`, the multiplier of both scores can be different, when that's the case, the proposer's voting power bonuses will be wrongly calculated because `proposalsPassed` is not correctly increased in `queue()`.

Code Snippet

https://github.com/sherlock-audit/2022-11-frankendao/blob/main/src/Governance.sol#L467-L495

https://github.com/sherlock-audit/2022-11-frankendao/blob/main/src/Staking.sol#L592-L601

Tool used

Manual Review

Recommendation

Change to:

```
    function queue(uint256 _proposalId) external {
        // Succeeded means we're past the endTime, yes votes outweigh no votes,
↪   and quorum threshold is met
```

```
        if(state(_proposalId) != ProposalState.Succeeded) revert InvalidStatus();

        Proposal storage proposal = proposals[_proposalId];

        // Set the ETA (time for execution) to the soonest time based on the
↪   Executor's delay
        uint256 eta = block.timestamp + executor.DELAY();
        proposal.eta = eta.toUint32();

        // Queue separate transactions for each action in the proposal
        uint numTargets = proposal.targets.length;
        for (uint256 i = 0; i < numTargets; i++) {
            executor.queueTransaction(proposal.targets[i], proposal.values[i],
↪   proposal.signatures[i], proposal.calldatas[i], eta);
        }

        // If a proposal is queued, we are ready to award the community voting
↪   power bonuses to the proposer
-         ++userCommunityScoreData[proposal.proposer].proposalsCreated;
+         ++userCommunityScoreData[proposal.proposer].proposalsPassed;

        // We don't need to check whether the proposer is accruing community
↪   voting power because
        // they needed that voting power to propose, and once they have an Active
↪   Proposal, their
        // tokens are locked from delegating and unstaking.
-         ++totalCommunityScoreData.proposalsCreated;
+         ++totalCommunityScoreData.proposalsPassed;

        // Remove the proposal from the Active Proposals array
        _removeFromActiveProposals(_proposalId);

        emit ProposalQueued(_proposalId, eta);
    }
```

Discussion

ZakkMan

Fixed: https://github.com/Solidity-Guild/FrankenDAO/pull/20

Issue M-5: castVote can be called by anyone even those without votes

Source: https://github.com/sherlock-audit/2022-11-frankendao-judging/issues/25

Found by

Trumpero, 0x52, hansfriese

Summary

Governance#castVote can be called by anyone, even users that don't have any votes. Since the voting refund is per address, an adversary could use a large number of addresses to vote with zero votes to drain the vault.

Vulnerability Detail

```
function _castVote(address _voter, uint256 _proposalId, uint8 _support) internal
↪  returns (uint) {
    // Only Active proposals can be voted on
    if (state(_proposalId) != ProposalState.Active) revert InvalidStatus();

    // Only valid values for _support are 0 (against), 1 (for), and 2 (abstain)
    if (_support > 2) revert InvalidInput();

    Proposal storage proposal = proposals[_proposalId];

    // If the voter has already voted, revert
    Receipt storage receipt = proposal.receipts[_voter];
    if (receipt.hasVoted) revert AlreadyVoted();

    // Calculate the number of votes a user is able to cast
    // This takes into account delegation and community voting power
    uint24 votes = (staking.getVotes(_voter)).toUint24();

    // Update the proposal's total voting records based on the votes
    if (_support == 0) {
        proposal.againstVotes = proposal.againstVotes + votes;
    } else if (_support == 1) {
        proposal.forVotes = proposal.forVotes + votes;
    } else if (_support == 2) {
        proposal.abstainVotes = proposal.abstainVotes + votes;
    }

    // Update the user's receipt for this proposal
```

```
    receipt.hasVoted = true;
    receipt.support = _support;
    receipt.votes = votes;

    // Make these updates after the vote so it doesn't impact voting power for
    ↪  this vote.
    ++totalCommunityScoreData.votes;

    // We can update the total community voting power with no check because if
    ↪  you can vote,
    // it means you have votes so you haven't delegated.
    ++userCommunityScoreData[_voter].votes;

    return votes;
}
```

Nowhere in the flow of voting does the function revert if the user calling it doesn't actually have any votes. staking#getVotes won't revert under any circumstances. Governance#_castVote only reverts if 1) the proposal isn't active 2) support > 2 or 3) if the user has already voted. The result is that any user can vote even if they don't have any votes, allowing users to maliciously burn vault funds by voting and claiming the vote refund.

Impact

Vault can be drained maliciously by users with no votes

Code Snippet

https://github.com/sherlock-audit/2022-11-frankendao/blob/main/src/Governance.sol#L607-L646

Tool used

Manual Review

Recommendation

Governance#_castVote should revert if msg.sender doesn't have any votes:

```
    // Calculate the number of votes a user is able to cast
    // This takes into account delegation and community voting power
    uint24 votes = (staking.getVotes(_voter)).toUint24();

+   if (votes == 0) revert NoVotes();
```

```
    // Update the proposal's total voting records based on the votes
    if (_support == 0) {
        proposal.againstVotes = proposal.againstVotes + votes;
    } else if (_support == 1) {
        proposal.forVotes = proposal.forVotes + votes;
    } else if (_support == 2) {
        proposal.abstainVotes = proposal.abstainVotes + votes;
    }
```

Discussion

zobront

This is great and will fix, but an adversary being able to burn a small pool put aside for gas refunds for no personal benefit seems like a Medium, not a High.

ZakkMan

Fixed: https://github.com/Solidity-Guild/FrankenDAO/pull/21

Issue M-6: Delegate can keep can keep delegatee trapped indefinitely

Source: https://github.com/sherlock-audit/2022-11-frankendao-judging/issues/23

Found by

rvierdiiev, 0x52

Summary

Users are allowed to delegate their votes to other users. Since staking does not implement checkpoints, users are not allowed to delegate or unstake during an active proposal if their delegate has already voted. A malicious delegate can abuse this by creating proposals so that there is always an active proposal and their delegatees are always locked to them.

Vulnerability Detail

```
modifier lockedWhileVotesCast() {
  uint[] memory activeProposals = governance.getActiveProposals();
  for (uint i = 0; i < activeProposals.length; i++) {
    if (governance.getReceipt(activeProposals[i],
    ↪  getDelegate(msg.sender)).hasVoted) revert TokenLocked();
    (, address proposer,) = governance.getProposalData(activeProposals[i]);
    if (proposer == getDelegate(msg.sender)) revert TokenLocked();
  }
  _;
}
```

The above modifier is applied when unstaking or delegating. This reverts if the delegate of msg.sender either has voted or currently has an open proposal. The result is that under those conditions, the delgatee cannot unstake or delegate. A malicious delegate can abuse these conditions to keep their delegatees forever delegated to them. They would keep opening proposals so that delegatees could never unstake or delegate. A single users can only have a one proposal opened at the same time so they would use a secondary account to alternate and always keep an active proposal.

Impact

Delegatees can never unstake or delegate to anyone else

Code Snippet

https://github.com/sherlock-audit/2022-11-frankendao/blob/main/src/Staking.sol#L166-L174

Tool used

Manual Review

Recommendation

There should be a function to emergency eject the token from staking. To prevent abuse a token that has been emergency ejected should be blacklisted from staking again for a certain cooldown period, such as the length of current voting period.

Discussion

zobront

In the case that a user did this, Admins would create a contract that all "stuck" users approve, and they would veto and unstake all tokens in one transaction. It would be inconvenient, but no long term harm would be caused.

I still think it's valid as a Medium, as this is obviously a situation we'd like to avoid being possible.

zobront

Fixed: https://github.com/Solidity-Guild/FrankenDAO/pull/18

Issue M-7: Veto function should decrease proposalsPassed (and possibly proposalsCreated)

Source: https://github.com/sherlock-audit/2022-11-frankendao-judging/issues/9

Found by

Trumpero, neumo

Summary

When a proposal that has passed is vetoed, the proposer still has the bonus of `proposalsCreated` and `proposalsPassed` related to the proposal. Both should be decremented because `veto` is intended to be used for malicious proposals.

Vulnerability Detail

The function `getCommunityVotingPower` returns a bonus to users for voting, creating proposals and having them passed. https://github.com/sherlock-audit/2022-11-frankendao/blob/main/src/Staking.sol#L520-L541 The `veto` function is supposed to be used against malicious proposals (see comment in line 522): https://github.com/sherlock-audit/2022-11-frankendao/blob/main/src/Governance.sol#L520-L534 So the malicius proposer should not keep a bonus based in this proposal's creation and/or passing, and the function should decrease both the values of `proposalsCreated` and `proposalsPassed`.

Impact

Malicious proposer keeps bonus after his proposal is vetoed.

Code Snippet

N/A

Tool used

Manual Review

Recommendation

Put a check in the `veto` function that decreases the values of both `proposalsCreated` and `proposalsPassed`.

```
if(proposal.verified){
    --userCommunityScoreData[proposal.proposer].proposalsCreated;
    --totalCommunityScoreData.proposalsCreated;
}
if (state(_proposalId) == ProposalState.Queued){
    --userCommunityScoreData[proposal.proposer].proposalsPassed;
    --totalCommunityScoreData.proposalsPassed;
}
```

Discussion

ZakkMan

Fixed: https://github.com/Solidity-Guild/FrankenDAO/pull/22