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:	December 1, 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
10	4

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

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

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

curiousapple, hansfriese, Trumpero

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.

jack-the-pug

Fix confirmed

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

hansfriese, 0x52, Haruxe

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.

SHERLOCK

- 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

jack-the-pug

Fix confirmed

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

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

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;
```


SHERLOCK

```
    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

jack-the-pug

Fix confirmed

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

cccz, 0x52

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

jack-the-pug

Fix confirmed

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 {
```


SHERLOCK

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.

jack-the-pug

Fix confirmed

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.

SHERLOCK

```
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

jack-the-pug

Fix confirmed

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

WATCHPUG, Nyx, saian, rvierdiiev, Bnke0x0, Tomo

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

jack-the-pug

Fix confirmed

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

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

Found by

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

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
```


SHERLOCK

```
        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

jack-the-pug

Fix confirmed

Issue M-5: `getCommunityVotingPower` doesn't calculate voting Power correctly due to precision loss

Source: https://github.com/sherlock-audit/2022-11-frankendao-judging/issues/48

Found by

ElKu

Summary

In `Staking.sol`, the getCommunityVotingPower function, doesn't calculate the votes correctly due to precision loss.

Vulnerability Detail

In getCommunityVotingPower function, the `return` statement is where the mistake lies in:

```
return
    (votes * cpMultipliers.votes / PERCENT) +
    (proposalsCreated * cpMultipliers.proposalsCreated / PERCENT) +
    (proposalsPassed * cpMultipliers.proposalsPassed / PERCENT);
```

Here, after each multiplication by the `Multipliers`, we immediately divide it by `PERCENT`. Every time we do a division, there is a certain amount of precision loss. And when its done thrice, the loss just accumulates. So instead, the division by `PERCENT` should be done after all 3 terms are added together.

Note that this loss is not there, if the `Multipliers` are a multiple of `PERCENT`. But these values can be changed through governance later. So its better to be careful assuming that they may not always be a multiple of `PERCENT`.

Impact

The community voting power of the user is calculated wrongly.

Code Snippet

The `getCommunityVotingPower` function:

```
function getCommunityVotingPower(address _voter) public override view returns
↪  (uint) {
  uint64 votes;
  uint64 proposalsCreated;
```

```
  uint64 proposalsPassed;

  // We allow this function to be called with the max uint value to get the total
↪   community voting power
  if (_voter == address(type(uint160).max)) {
    (votes, proposalsCreated, proposalsPassed) =
↪   governance.totalCommunityScoreData();
  } else {
    // This is only the case if they are delegated or unstaked, both of which
↪   should zero out the result
    if (tokenVotingPower[_voter] == 0) return 0;

    (votes, proposalsCreated, proposalsPassed) =
↪   governance.userCommunityScoreData(_voter);
  }

  CommunityPowerMultipliers memory cpMultipliers = communityPowerMultipliers;

  return
    (votes * cpMultipliers.votes / PERCENT) +
    (proposalsCreated * cpMultipliers.proposalsCreated / PERCENT) +
    (proposalsPassed * cpMultipliers.proposalsPassed / PERCENT);
}
```

Tool used

VSCode, Manual Analysis

Recommendation

Do the division once after all terms are added together:

```
  return
    ( (votes * cpMultipliers.votes) +
    (proposalsCreated * cpMultipliers.proposalsCreated) +
    (proposalsPassed * cpMultipliers.proposalsPassed) ) / PERCENT;
}
```

Discussion

zobront

This is a good suggestion but don't think it warrants a Medium. All these values are intended to be > 100. In the event that votes is lowered more, it will always be the case that proposalsCreated & proposalsPassed will be greater than 100, so combining them doesn't improve the accuracy.

El-Ku

Escalate for 5 USDC

If the multipliers are not a multiple of 100, say for example 125, 150 and 175 or something like that, then the precision will be still lost. But as I myself mentioned in the report, if all the multipliers are a multiplier of 100, then these equations dont cause an issue.

Otherwise we should do all the additions before doing the division. Even though the loss is still there it can be minimized by this step.

I think it deserves a medium as this calculation controls many Governance functions.

sherlock-admin

> Escalate for 5 USDC
>
> If the multipliers are not a multiple of 100, say for example 125, 150 and 175 or something like that, then the precision will be still lost. But as I myself mentioned in the report, if all the multipliers are a multiplier of 100, then these equations dont cause an issue.
>
> Otherwise we should do all the additions before doing the division. Even though the loss is still there it can be minimized by this step.
>
> I think it deserves a medium as this calculation controls many Governance functions.

You've created a valid escalation for 5 USDC!

To remove the escalation from consideration: Delete your comment. To change the amount you've staked on this escalation: Edit your comment **(do not create a new comment)**.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

zobront

I think this makes sense and I was wrong to dispute it. It's a change we would like to make and agree there's the potential for some values to calculated incorrectly.

Evert0x

Escalation accepted

sherlock-admin

> Escalation accepted

This issue's escalations have been accepted!

Contestants' payouts and scores will be updated according to the changes made on this issue.

zobront

Fixed: https://github.com/Solidity-Guild/FrankenDAO/pull/24

jack-the-pug

Fix confirmed

Issue M-6: Staking#changeStakeTime and changeStakeAmount are problematic given current staking design

Source: https://github.com/sherlock-audit/2022-11-frankendao-judging/issues/31

Found by

0x52

Summary

Staking#changeStakeTime and changeStakeAmount allow the locking bonus to be modified. Any change to this value will cause voting imbalance in the system. If changes result in a higher total bonus then existing stakers will be given a permanent advantage over new stakers. If the bonus is increased then existing stakers will be at a disadvantage because they will be locked and unable to realize the new staking bonus.

Vulnerability Detail

```
function _stakeToken(uint _tokenId, uint _unlockTime) internal returns (uint) {
  if (_unlockTime > 0) {
    unlockTime[_tokenId] = _unlockTime;
    uint fullStakedTimeBonus = ((_unlockTime - block.timestamp) *
    ↪  stakingSettings.maxStakeBonusAmount) / stakingSettings.maxStakeBonusTime;
    stakedTimeBonus[_tokenId] = _tokenId < 10000 ? fullStakedTimeBonus :
    ↪  fullStakedTimeBonus / 2;
  }
```

When a token is staked their stakeTimeBonus is stored. This means that any changes to stakingSettings.maxStakeBonusAmount or stakingSettings.maxStakeBonusTime won't affect tokens that are already stored. Storing the value is essential to prevent changes to the values causing major damage to the voting, but it leads to other more subtle issue when it is changed that will put either existing or new stakers at a disadvantage.

Example: User A stake when maxStakeBonusAmount = 10 and stake long enough to get the entire bonus. Now maxStakeBonusAmount is changed to 20. User A is unable to unstake their token right away because it is locked. They are now at a disadvantage because other users can now stake and get a bonus of 20 while they are stuck with only a bonus of 10. Now maxStakeBonusAmount is changed to 5. User A now has an advantage because other users can now only stake for a bonus of 5. If User A never unstakes then they will forever have that advantage over new users.

Impact

Voting power becomes skewed for users when Staking#changeStakeTime and changeStakeAmount are used

Code Snippet

https://github.com/sherlock-audit/2022-11-frankendao/blob/main/src/Staking.sol#L389-L415

Tool used

Manual Review

Recommendation

I recommend implementing a poke function that can be called by any user on any user. This function should loop through all tokens (or the tokens specified) and recalculate their voting power based on current multipliers, allowing all users to be normalized to prevent any abuse.

Discussion

zobront

This is the intended behavior. Staking windows will be relatively short (~1 month) and bonuses will change only by governance vote. We accept that there may be short periods where a user is locked in a suboptimal spot, but they can unstake and restake when the period is over.

0x00052

Escalate for 1 USDC

I think this should be considered valid. It won't just be for a small amount of time if the staking amount is lowered. In this case, all users who staked beforehand will have a permanent advantage over other users. Due to the permanent imbalance lowering it would cause in the voting power of users, I think that medium is appropriate.

sherlock-admin

> Escalate for 1 USDC
>
> I think this should be considered valid. It won't just be for a small amount of time if the staking amount is lowered. In this case, all users who staked beforehand will have a permanent advantage over other users. Due to the permanent imbalance lowering it would cause in the voting power of users, I think that medium is appropriate.

You've created a valid escalation for 1 USDC!

To remove the escalation from consideration: Delete your comment. To change the amount you've staked on this escalation: Edit your comment **(do not create a new comment)**.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

zobront

I can see the argument here. We don't want to change it and believe it's fine as is, but it may be a valid Medium.

Evert0x

Escalation accepted

sherlock-admin

> Escalation accepted

This issue's escalations have been accepted!

Contestants' payouts and scores will be updated according to the changes made on this issue.

Issue M-7: castVote can be called by anyone even those without votes

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

Found by

hansfriese, 0x52, Trumpero

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

jack-the-pug

Fix confirmed

Issue M-8: Adversary can abuse delegating to lower quorum

Source: https://github.com/sherlock-audit/2022-11-frankendao-judging/issues/24

Found by

0x52

Summary

When a user delegates to another user they surrender their community voting power. The quorum threshold for a vote is determined when it is created. Users can artificially lower quorum by delegating to other users then creating a proposal. After it's created they can self delegate and regain all their community voting power to reach quorum easier.

Vulnerability Detail

```
// 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 a user delegates to user other than themselves, they forfeit their community votes and lowers the total number of votes. When they self delegate again they will recover all their community voting power.

```
newProposal.id = newProposalId.toUint96();
newProposal.proposer = msg.sender;
newProposal.targets = _targets;
newProposal.values = _values;
newProposal.signatures = _signatures;
newProposal.calldatas = _calldatas;

//@audit quorum votes locked at creation

newProposal.quorumVotes = quorumVotes().toUint24();
```

```
newProposal.startTime = (block.timestamp + votingDelay).toUint32();
newProposal.endTime = (block.timestamp + votingDelay + votingPeriod).toUint32();
```

When a proposal is created the quorum is locked at the time at which it's created. Users can combine these two quirks to abuse the voting.

Example:

Assume there is 1000 total votes and quorum is 20%. Assume 5 users each have 35 votes, 10 base votes and 25 community votes. In this scenario quorum is 200 votes which they can't achieve. Each user delegates to other users, reducing each of their votes by 25 and reducing the total number of votes of 875. Now they can create a proposal and quorum will now be 175 votes (875*20%). They all self delegate and recover their community votes. Now they can reach quorum and pass their proposal.

Impact

Users can collude to lower quorum and pass proposal easier

Code Snippet

https://github.com/sherlock-audit/2022-11-frankendao/blob/main/src/Staking.sol#L286-L316

Tool used

Manual Review

Recommendation

One solution would be to add a vote cooldown to users after they delegate, long enough to make sure all active proposals have expired before they're able to vote. The other option would be to implement checkpoints.

Discussion

zobront

This is clever but the impact that a small number of users can have on quorum is relatively small, and we aren't concerned.

0x00052

Escalate for 1 USDC

This is a valid issue. Even if they don't want to fix, it still has an impact on quorum which impacts proposals passing or failing. The impact is small at low quorum thresholds but at higher quorum thresholds the impact is larger.

As an example if the quorum threshold is 10% then using this technique to drop overall votes by 100 would lower quorum by 10 votes but if the threshold is 50% then it would lower quorum by 50 votes.

This decrease in quorum could allow borderline proposals to pass that normally couldn't meet quorum. Since the impact depends on the current quorum threshold (which is adjustable), medium seems appropriate to me.

sherlock-admin

> Escalate for 1 USDC
>
> This is a valid issue. Even if they don't want to fix, it still has an impact on quorum which impacts proposals passing or failing. The impact is small at low quorum thresholds but at higher quorum thresholds the impact is larger.
>
> As an example if the quorum threshold is 10% then using this technique to drop overall votes by 100 would lower quorum by 10 votes but if the threshold is 50% then it would lower quorum by 50 votes.
>
> This decrease in quorum could allow borderline proposals to pass that normally couldn't meet quorum. Since the impact depends on the current quorum threshold (which is adjustable), medium seems appropriate to me.

You've created a valid escalation for 1 USDC!

To remove the escalation from consideration: Delete your comment. To change the amount you've staked on this escalation: Edit your comment **(do not create a new comment)**.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

zobront

I start by that this wouldn't have any substantial impact. First off, if a user delegates, they won't have any votes and therefore won't meet the threshold to create a proposal. So it would need to be a coordinated effort. And in the event that this happened, the impact would be relatively small. Seems unlikely and not impactful enough that a Medium feels like a stretch.

Evert0x

Escalation accepted. Assigning medium severity as the impact is not that significant + it requires a lot of effort to execute.

The quorum can be manipulated which can lead to unexpected behavior as illustrated.

sherlock-admin

> Escalation accepted. Assigning medium severity as the impact is not that significant + it requires a lot of effort to execute.
>
> The quorum can be manipulated which can lead to unexpected behavior as illustrated.

This issue's escalations have been accepted!

Contestants' payouts and scores will be updated according to the changes made on this issue.

Issue M-9: Delegate can keep can keep delegatee trapped indefinitely

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

Found by

rvierdiiev, curiousapple, 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

This fix addresses the risk laid out in the issue, that a delegate may repeatedly propose to keep votes locked.

We chose to not address the similar risk with voting because admins need to explicitly verify proposals before they can be voted on, so in the event this happens, admins would just hold off on verifying to give them a chance to undelegate.

We decided to go this direction because the cooldown period would add extra complexity and slightly increase gas fees on txs that we plan to refund a lot of.

jack-the-pug

Fix confirmed

Issue M-10: Veto function should decrease proposalsPassed (and possibly proposalsCreated)

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

Found by

neumo, Trumpero

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

jack-the-pug

Fix confirmed