VotingEscrow contract analysis report

Version 1.0

Maroutis

July 31, 2024

VotingEscrow contract analysis report

Maroutis

31/07/2024

Overview

- Users can lock their PUPPET tokens for a specified duration (up to 2 years) and receive vePUPPET tokens in return.
- The longer the lock duration, the higher the conversion rate of PUPPET to vePUPPET.

Stake Option:

- Users can stake their PUPPET tokens to earn vePUPPET tokens, which provide governance rights and long-term rewards.
- Users who opt for a 2-year lock-in receive a 100% conversion rate, while a 1-year lock-in translates to a 50% conversion rate.

Exit Option:

- Users can claim one-third of their generated revenue in PUPPET tokens for immediate returns, forgoing long-term rewards.

This should encourage long-term commitment while enhancing governance participation by rewarding users with vePUPPET tokens.

Security considerations :

Need to check for allowed Duration values

https://github.com/GMX-Blueberry-Club/puppet-contracts/src/token/VotingEscrow.sol §L86

```
uint _nextDuration = (_lockedAmount * _lock.duration + _amount
    * _duration) / _nextBalance;
```

- If I understand correctly, only 2 durations are available for locking : the 1 year and 2 years duration. For this reason, there should be checks to only allow locking for these periods.

Issues with the locking logic

https://github.com/GMX-Blueberry-Club/puppet-contracts/src/token/VotingEscrow.sol §L90

```
router.transfer(token, _depositor, address(this), _amount);
_mint(_user, _amount);
```

- The `lock()` function mints the same amount regardless of the locking duration (for the exact same amount transfered from the user to this contract) –> Reduced incentives for users to lock for bigger durations.

Issues with release logic

https://github.com/GMX-Blueberry-Club/puppet-contracts/src/token/VotingEscrow.sol §L110

```
accrued: _release.accrued + ((block.timestamp - _release.
    lastSyncTime) * _emissionRate),
```

- During the first release call, when the authorized account/contract calls `release()` function, the variable `_release.lastSyncTime` will be 0 which implies `(block.timestamp - _release.lastSyncTime)== block.timestamp`. Basically, the accrued interest will be `accrued = block.timestamp * _emissionRate which is >> amount`. The users can withdraw way more than what was locked without even waiting.

https://github.com/GMX-Blueberry-Club/puppet-contracts/src/token/VotingEscrow.sol §L110

```
accrued: _release.accrued + ((block.timestamp - _release.
    lastSyncTime) * _emissionRate),
```

- Assuming subsequent releases, accrued interests can become huge if `block.timestamp >> _release.lastSyncTime`. Basically there is no limit even if docs implies that max locking period is max 2 years.

Issues with withdrawal logic

https://github.com/GMX-Blueberry-Club/puppet-contracts/src/token/VotingEscrow.sol §L121

```
1     function withdraw(address _user, address _receiver, uint _amount)
          external auth {
2         if (_amount == 0) revert VotingEscrow__ZeroAmount();
```

- `amount` variable is not used and can be removed. This function withdraws tokens depending on time passed since release was called.

https://github.com/GMX-Blueberry-Club/puppet-contracts/src/token/VotingEscrow.sol §L127

```
1             amount: _release.amount,
```

- When the authorized account withdraws, the amount/notional is not updated. Basically he can keep withdrawing and `_releaseRate` will be calculated on the initial amount.

https://github.com/GMX-Blueberry-Club/puppet-contracts/src/token/VotingEscrow.sol §L133

```
1         token.transfer(_receiver, _release.accrued + ((block.timestamp
              - _release.lastSyncTime) * _releaseRate));
```

- A user can withdraw basically an infinite amount depending on time passed. `If block.timestamp = 10 * _release.lastSyncTime`, the amount transfered will be = `10 * the amount locked`.
- The release calculation does not take into account the time passed since locking the tokens but only the time after the release. Basically they have to release everything the moment they locked the amount. Which is not a good system because they burn all their vePUPPET tokens while also having their PUPPET tokens locked before withdrawing. In short they lose the incentives.

locking tokens can be circumvented

https://github.com/GMX-Blueberry-Club/puppet-contracts/src/token/VotingEscrow.sol §L136-139

```
1     function stake(address _depositor, address _user, uint _amount)
          external auth {
2         router.transfer(token, _depositor, address(this), _amount);
3         _mint(_user, _amount);
4     }
```

- The logic of the `stake` and `lock` functions can be circumvented :

- An entity can lock a small amount of tokens (assume 1 wei) then call `release()` function to initiate their `Release` struct. This entity will then call `stake` on a substantial amount of PUPPET that they bought or had before. Then they can call `release()` on this same amount of PUPPET tokens without ever locking them.

Use safeTransfer library instead of transfer to avoid locked tokens

https://github.com/GMX-Blueberry-Club/puppet-contracts/src/token/VotingEscrow.sol §L142

```
1  token.transfer(_user, _amount);
```

- Some tokens like USDT do not return a bool on ERC20 methods. It would be better to use safeTransfer to avoid cases where funds are locked in the contract while users lose their vePUPPET.

Other security considerations :

- Need some unit and fuzz tests
- Add some natspec