Protocol Risk Mitigation Report

Version 1.2

Maroutis

June 12, 2024

PuppetToken Risk Mitigation Report

Maroutis

June, 2024

PuppetToken Audit Report

Prepared by: Maroutis

Table of Contents

- Low
  - [L-1] `PuppetToken::mint` Function allows minting more than 1% of total supply during first epoch
- Informational
  - [I-1] Incorrect variables names
  - [I-2] The successive minting of small amounts allow users to receive more tokens than allowed

Protocol Summary

PuppetToken is an ERC20 token that represents governance shares within a larger system. It includes a minting limitation feature, which restricts new token issuance to be proportional to the existing supply within each epoch. Initially, core contributors, the owner, or the protocol hold the majority of governance power. However, over time, this power is gradually transferred to regular users. The minting functions can only be executed by an authorized party.

Disclaimer

Maroutis makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of 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.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

Audit Details

The findings described in this document correspond the following commit hash:

```
11b2eafb74a877524582e86f01cd382b7e1b2736
```

Scope

```
src/token/
--- PuppetToken.sol
```

Protocol Summary

Roles

- Authorized: Is the only party who should be able to mint tokens.
- For this contract, only the authorized parties should be able to interact with the contract. This party is supposed to be somewhat trusted. The amounts that they can mint are predetermined depending on the revenue that each protocol brings.
- The authorized contract will be a reward distribution contract in which all core contributors can update the distribution for all contributors, this will be either automated or get called very often. This will help avoid any front-running issues in the `mint()` function.

Executive Summary

Issues found

Severity	Number of issues found
High	0
Medium	1
Low	1
Info	2
Gas Optimizations	0

Severity	Number of issues found
Total	4

Findings

Medium

[M-1] Users are able to mint more than 1% of total supply when `_timeElapsed` surpasses `_durationWindow`

Description:

The `PuppetToken::mint` function allows users to mint more than 1% of the total supply if the `_timeElapsed` since the last mint surpasses the `_durationWindow`. This occurs because `_decayRate` keeps increasing above the `_limitAmount`.

```
// https://github.com/GMX-Blueberry-Club/puppet-contracts/blob/11
   b2eafb74a877524582e86f01cd382b7e1b2736/src/token/PuppetToken.sol#L75
   -L76
       uint _timeElapsed = block.timestamp - lastMintTime;
       uint _decayRate = _limitAmount * _timeElapsed / config.
          durationWindow;
```

It only takes the amount to mint to be at least equal to `_decayRate` for the check `if ( emissionRate > _limitAmount)` to not be true. The amount to mint can actually be equal to a maximum `_decayRate + _limitAmount` if `emissionRate` is 0 when `PuppetToken::mint` is called. This occurs because of this line :

```
// https://github.com/GMX-Blueberry-Club/puppet-contracts/blob/11
   b2eafb74a877524582e86f01cd382b7e1b2736/src/token/PuppetToken.sol#
   L82C13-L82C26
       } else {
           emissionRate += (_amount - _decayRate);
```

Impact:

This vulnerability enables users to mint significantly more tokens than the intended 1% of the total supply depending on the time elapsed since last mint.

Proof of Concept:

You can add the following `testCanMintMoreThanLimitAmount` test in the file `PUppetToken.t.sol`:

```
            function testCanMintMoreThanLimitAmount() public {
                skip(2 hours); // Wait for 2 hours
                puppetToken.mint(users.alice, 2 * puppetToken.
                    getLimitAmount()); // Mint 2 times the limit amount
            }
```

Recommended Mitigation:

The root cause of this issue is that `_decayRate` can go above `_limitAmount` which is the limit. The `_decayRate` is then substracted from the amount to mint to determine if `emissionRate` is above `_limitAmount`. This allows the user to choose an amount bigger than `_limitAmount` and still be in range. A potential mitigation would be to set `_decayRate` in a way to not surpass `_limitAmount`. While also making sure to not substract it when calculation the new `emissionRate` amount in the else condition.

Fix:

Fixed in GMX-Blueberry-Club/puppet-contracts@89ec28f

Low

[L-1] `PuppetToken::mint` Function allows minting more than 1% of total supply during first epoch

Description:

The `PuppetToken::mint` function allows users to mint more than the configured 1% of the total supply within the first hour after deployment. This occurs because `_decayRate` is equal to 0 at first and the emission rate can increase up to `_limitAmount` even if no time has passed.

```
// https://github.com/GMX-Blueberry-Club/puppet-contracts/blob/11
    b2eafb74a877524582e86f01cd382b7e1b2736/src/token/PuppetToken.sol#L85
    -L88

            if (emissionRate > _limitAmount) {
                revert PuppetToken__ExceededRateLimit(_limitAmount,
                    emissionRate);
            }
        }
```

Impact:

Users can mint more tokens than the configured limit within the first hour.

Proof of Concept:

You can add the following `testCanMintMoreThan1PercentDuringFirstHour` test in the file `PUppetToken.t.sol`:

```
    function testCanMintMoreThan1PercentDuringFirstHour() public {

        uint256 currentLimit = puppetToken.getLimitAmount();
        assertEq(currentLimit, 1000e18);

        // Mints max mintable amount at first then the max amount for
           each period
        puppetToken.mint(users.alice, puppetToken.getLimitAmount());

        skip(uint(1 hours / 4));
        puppetToken.mint(users.alice, puppetToken.getLimitAmount() / 4)
           ;

        skip(uint(1 hours / 4));
        puppetToken.mint(users.alice, puppetToken.getLimitAmount()/4);

        skip(uint(1 hours / 4));
        puppetToken.mint(users.alice, puppetToken.getLimitAmount()/4);

        skip(uint(1 hours / 4));
        puppetToken.mint(users.alice, puppetToken.getLimitAmount()/4);

        console.log(puppetToken.getLimitAmount());
        assertEq(puppetToken.balanceOf(users.alice),
           2013793816445312500000);

        // 2013793816445312500000 minted in 1 hour while the limit
           should be 1000000000000000000000. In other words more than
           2% of the initial supply was minted
    }
```

Recommended Mitigation:

To resolve this issue, you can consider tracking the amount of tokens minted within the rate limit window and ensuring it does not exceed the configured limit.

Fix:

Fixed in GMX-Blueberry-Club/puppet-contracts@f79d9cd

Informational

[I-1] Incorrect variables names

Description:

Some variables need to be corrected to better reflect the executed operations.

Recommended Mitigation:

```diff
-       uint _totalMinedAmount = totalSupply() - mintedCoreAmount - GENESIS_MINT_AMOUNT;
-       uint _maxMintableAmount = Precision.applyFactor(getCoreShare(_lastMintTime), _totalMinedAmount);

+       uint _totalMintedAmount = totalSupply() - mintedCoreAmount - GENESIS_MINT_AMOUNT;
+       uint _maxMintableAmount = Precision.applyFactor(getCoreShare(_lastMintTime), _totalMintedAmount);
```

Fix:

Fixed in GMX-Blueberry-Club/puppet-contracts@89ec28f

[I-2] The successive minting of small amounts allow users to receive more tokens than allowed

Description:

The `PuppetToken::mint` function allows users to mint tokens in smaller amounts continuously, leading to the accumulation of more tokens than if the tokens were minted in one shot. This occurs because the emission rate and decay calculations uses the `totalSupply`, which increases after each mint, for the `_limitAmount` calculation, and do not account for the compounded effect of multiple small mints within the rate limit window.

```
// https://github.com/GMX-Blueberry-Club/puppet-contracts/blob/11b2eafb74a877524582e86f01cd382b7e1b2736/src/token/PuppetToken.sol#L64

    function getLimitAmount() public view returns (uint) {
        return Precision.applyFactor(config.limitFactor, totalSupply());
    }
```

```
// https://github.com/GMX-Blueberry-Club/puppet-contracts/blob/11b2eafb74a877524582e86f01cd382b7e1b2736/src/token/PuppetToken.sol#L85-L91

            if (emissionRate > _limitAmount) {
                revert PuppetToken__ExceededRateLimit(_limitAmount,
                    emissionRate);
            }
        }

        // Add the requested mint amount to the window's mint count
        _mint(_receiver, _amount);
```

Impact:

This vulnerability enables users to mint more tokens than the configured rate limit by dividing their mints into smaller increments.

Proof of Concept:

You can add the following `testMintSmallAmountContinuouslyGivesMoreTokens` test in the file `PUppetToken.t.sol`:

```
    function testMintSmallAmountContinuouslyGivesMoreTokens() public {

        assertEq(puppetToken.getLimitAmount(), 1000e18); // Max amount
            that can be minted in one shot at time 0

        // Alice notices that by dividing the buys into smaller ones
            she can earn more tokens.
        puppetToken.mint(users.alice, puppetToken.getLimitAmount()/5);
        puppetToken.mint(users.alice, puppetToken.getLimitAmount()/5);
        puppetToken.mint(users.alice, puppetToken.getLimitAmount()/5);
        puppetToken.mint(users.alice, puppetToken.getLimitAmount()/5);
        puppetToken.mint(users.alice, puppetToken.getLimitAmount()/5);


        assertEq(puppetToken.balanceOf(users.alice),
            1004008008003200000000); // 4 tokens more than what Alice
            should be able to mint so about a 0,4% increase. The more
            the getLimitAmount() increases, the more this method will
            earn Alice more.
    }
```

Recommended Mitigation:

- One way to correct this would be to fixate `_limitAmount` to be piecewise constant function. Example : for epoch 0 (first hour after deployement), `getLimitAmount()` would returns a constant `1000e18` during the first hour. Then, the `getLimitAmount()` would only be recalculated after 1 epoch.
- The other mitigation would be to track the amount minted and then revert if a user attempts to mint more.

Fix:

According to dev, it is acceptable to have a slighlty higher limit during the 1 hour window. The `1,04%` minting fault is negligeable.