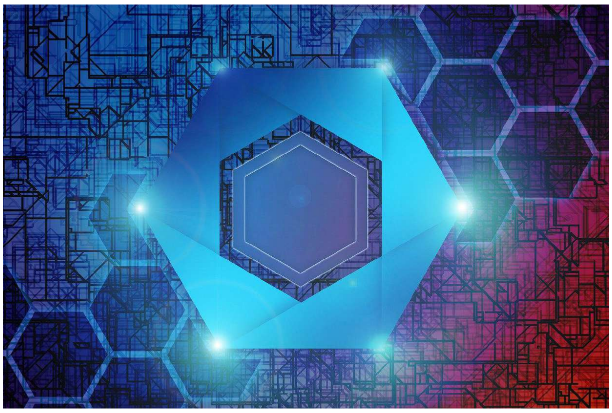

Protocol Risk Mitigation Report

Version 1.1

Maroutis

June 9, 2024

PuppetToken Risk Mitigation Report

Maroutis

June, 2024

PuppetToken Audit Report

Prepared by: Maroutis

Table of Contents

  - [M-2] The successive minting of small amounts allow users to receive more tokens than allowed
  - [M-3] Calling `PuppetToken::mint` resets `_decayRate` which prevent subsequent users from minting their full entitled amounts

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

- Informational
  - [I-1] Incorrect variables names

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

	Impact			
	Low	M	M/L	L

Audit Details

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

```
0f0c84fd629c013a62c952c1a20170dd3a49ca51
```

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.

Executive Summary

Issues found

Severity	Number of issues found
High	0
Medium	3
Low	1
Info	1
Gas Optimizations	0

Severity	Number of issues found
Total	5

Findings

Medium

[M-1] Front-runners can call `PuppetToken::mint` with 0 or low amount to reduce minting power of users

Description:

The `PuppetToken::mint` function can be abused by an attacker to front-run the minting process by calling the `mint` function with 0 (or a small amount of) tokens. This action updates the `lastMintTime`, effectively resetting the emission rate window and reducing the amount of tokens that can be minted by subsequent users within the intended rate limit window.

```
// https://github.com/GMX-Blueberry-Club/puppet-contracts/blob/11b2eafb74a877524582e86f01cd382b7e1b2736/src/token/PuppetToken.sol#L92C9-L92C40

lastMintTime = block.timestamp;
```

```
// https://github.com/GMX-Blueberry-Club/puppet-contracts/blob/11b2eafb74a877524582e86f01cd382b7e1b2736/src/token/PuppetToken.sol#L75-L76

        uint _timeElapsed = block.timestamp - lastMintTime;
        uint _decayRate = _limitAmount * _timeElapsed / config.durationWindow;
```

Impact:

This vulnerability enables an attacker to manipulate the minting process, reducing the amount of tokens that legitimate users and core minters can mint.

Proof of Concept:

This attack can be done to both `PuppetToken::mint` and `PuppetToken::mintCore` function to effectively reduce the amount of tokens that can be minted for both regular users and core users.

Here are the steps that can be followed to reduce tokens for regular protocols in the PuppetToken ::mint function :

- Assume some time have passed to where _decayRate variable is not 0
- Alice wants to mint the maximum amount available to her which is _limitAmount + _decayRate. If 4 hours has passed, the maximum will be 4 * _limitAmount. Alice executes the tx.
- An attacker/Mev sees Alice's tx in the mempool, front-runs it and executes the function PuppetToken::mint with amount variable equal to 0.
- _timeElapsed becomes 0, which means that _decayRate is also equal to 0. Alice's tx reverts. When she tries to mint again she realizes that she can only mint a maximum of _limitAmount losing about 3*_limitAmount tokens.

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

```
    function testCanFrontRunToReduceMintAmountForOtherUsers() public {

        // Assume 3 hours has passed, this should allow _decayRate to be equal to 3 * getLimitAmount()
        skip(3 hours);
        // However, an attacker (some authorized protocol) called Bob front-runs the call to the mint function
        puppetToken.mint(users.bob, 0); // This resets the lastMintTime which means that the call now should revert

        // Normally Alice should be Able to mint up to 4 * getLimitAmount(). but it doesnt work since _decayRate is now equal to 0. Alice can only mint a max equal to getLimitAmount() even after 3 epochs of nothing minted
        uint256 amountToMint = 2 * puppetToken.getLimitAmount();
        vm.expectRevert(abi.encodeWithSelector(PuppetToken.PuppetToken__ExceededRateLimit.selector,1000000000000000000000, 2000000000000000000000));
        puppetToken.mint(users.alice, amountToMint);


        uint256 maxAmountToMint = puppetToken.getLimitAmount();
        puppetToken.mint(users.alice, maxAmountToMint);

        // Alice can only mint getLimitAmount() effectively losing 3 * getLimitAmount()

    }
```

Here are the steps that can be followed to reduce tokens for core protocols in the PuppetToken:: mintCore function :

- Assume tokens have been minted using the `PuppetToken::mint` function
- Some time passes.
- A core minter decides to call the function `PuppetToken::mintCore`. He expected to receive tokens calculated with the timestamp of the last `mint` call.
- A front-runner sees his tx in the mempool. He front-runs it with a call to `PuppetToken::mint` with amount equal to `0`.
- Since `lastMintTime` is updated, the core minter gets less tokens than expected.

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

```
    function testCanFrontRunToReduceMintAmountForCoreMinters() public {

        // Assume max amount of Tokens have been minted
        puppetToken.mint(users.alice, puppetToken.getLimitAmount());

        skip( 24 hours);
        // The core minters should be able to mint 1000e18 tokens
        // However, an attacker (some authorized protocol) called Bob front-runs the call to the mintCore function and calls mint with 0 amount
        puppetToken.mint(users.alice, 0);
        assertEq(puppetToken.emissionRate(), 0); // This resets the lastMintTime to now

        uint256 balanceOwnerBefore = puppetToken.balanceOf(users.owner);
        puppetToken.mintCore(users.owner);
        uint256 balanceOwnerAfter = puppetToken.balanceOf(users.owner);

        uint256 change = balanceOwnerAfter - balanceOwnerBefore;

        assertEq(change, 997269831639617776429); // Core minters gets 3 tokens less than expected.
        // This attack can be done everytime the mintCore function is pending in the mempool to reduce governance power of core.

    }
```

Recommended Mitigation:

To resolve this issue, enforce a **minimum mint amount** to prevent front-running with a zero or low amount.

[M-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.

[M-3] Calling `PuppetToken::mint` resets `_decayRate` which prevent subsequent users from minting their full entitled amounts

Description:

The `PuppetToken::mint` function has a flaw where one user is unable to mint their full entitled amount within an epoch because another user mints before them. This occurs because the emission rate and decay calculations reset with each mint, preventing subsequent users from minting their full amount within the same epoch.

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

Impact:

This issue leads to unfair minting opportunities, where users cannot rely on being able to mint their full entitled amount if another user mints before them within the same epoch.

Proof of Concept:

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

```
        skip(3 hours); // Assume 3 epochs have passed
        // Max mintable amount = 4 * puppetToken.getLimitAmount()

        // There are two minters Bob and Alice, Bob can mint 25% and Alice 75%
        // This should give Bob a max of puppetToken.getLimitAmount()
        // While Alice should be able to mint 3 * puppetToken.getLimitAmount()

        // Bob decides to mint first his amount
        puppetToken.mint(users.bob, puppetToken.getLimitAmount());

        // Since _decayRate is now 0, Alice can only mint a maximum of puppetToken.getLimitAmount()
        uint256 amountToMint = 3 * puppetToken.getLimitAmount();
        vm.expectRevert(abi.encodeWithSelector(PuppetToken.PuppetToken__ExceededRateLimit.selector,101000000000000000000, 303000000000000000000));
        puppetToken.mint(users.alice, amountToMint);


        puppetToken.mint(users.alice, puppetToken.getLimitAmount()); // Alice lost 2 * puppetToken.getLimitAmount()
```

Recommended Mitigation:

When a user mints, it means that other users cannot mint their full amount in the same epoch. This can lead to a race condition where each user tries to execute their transaction first. The `_decayRate` variable should only resets when the max amount that should be minted has been reached.

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/11b2eafb74a877524582e86f01cd382b7e1b2736/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.

Informational

[I-1] Incorrect variables names

Description:

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

Recommended Mitigation:

```
-       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);
```