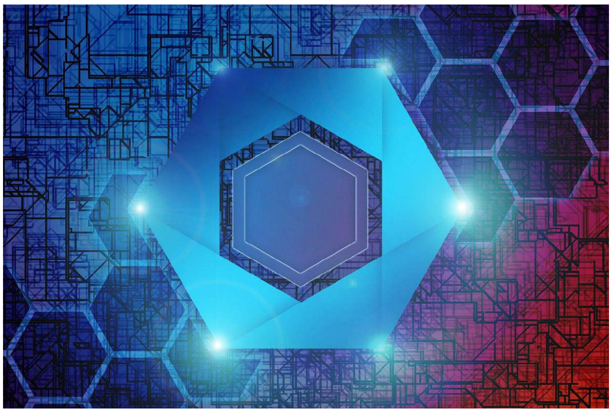

Protocol Audit Report

Version 1.0

Maroutis

June 2, 2024

PuppetToken Audit Report

Maroutis

May, 2024

PuppetToken Audit Report

Prepared by: Maroutis

Table of Contents

- Low
  - [L-1] For the same amount of minted tokens to users, `PuppetToken::mintCore` can mint fewer tokens depending on timing
- Informational
  - [I-1] `PuppetToken::getCoreShare` function can be improved for better readability, less precision loss and less gas cost
  - [I-2] Events missing indexed fields
  - [I-3] Use `uint256` instead of `uint` for better readability
- Gas
  - [G-1] The `PuppetToken::deployTimestamp` should be set as an immutable variable to save gas
  - [G-2] **`public`** functions not used internally could be marked `external`

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
Likelihood	High	H	H/M	M
	Medium	H/M	M	M/L
	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	1

Severity	Number of issues found
Low	1
Info	3
Gas Optimizations	2
Total	7

Findings

Medium

[M-1] Potential underflow issue in PuppetToken::mintCore calculation can cause the function to revert

Description:

The PuppetToken::mintCore function allows the authorized user to mint governance tokens to the _receiver. The initial intention is for the minted amount to be slightly less than the amount minted to users, with this rate decreasing over time to transfer governance power gradually to regular users. The issue arises in the calculation of the _mintable variable :

```
        uint _mintShare = getCoreShare();
        uint _maxMintable = Precision.applyFactor(_mintShare,
            mineMintCount);
        uint _mintable = _maxMintable - coreMintCount;
```

- _mintShare determines the rate at which the tokens should be minted.

The simplified formula for the getCoreShare() is : $1e30 \times \frac{1\text{ year}}{\Delta+1\text{ year}}$ where $Y = 31560000$ and Δ is the time between the current timestamp and deployTimestamp.

The _mintShare is always less or equal to $1 \times 1e30$ and decreasing over time. For example, for Δ equal to 1 year meaning that 1 year has passed since deploy, _mintShare will be equal to $0.5 \times 1e30$ or about half.

- mineMintCount represents the total amount of tokens minted to users since deployment.

After one year has passed, _maxMintable would be equal to half the amount mineMintCount.

- coreMintCount represents the amount of tokens that were minted through the function PuppetToken::mintCore to date.

If coreMintCount exceeds _maxMintable, which can happen since _maxMintable is only a portion of mineMintCount and not the full amount, the substraction _maxMintable - coreMintCount will cause an underflow and revert. This situation can occur if the tokens minted since the last PuppetToken::mintCore do not account for the time elapsed. Which is something difficult to achieve, since PuppetToken::mintCore can be called at the authorized user discretion meaning at anytime.

Moroever, The formula used to calculate the _mintable results in less tokens than it should be. Since the percentage or _mintShare is first applied to the total amount minted to users before substracting. While the correct formula should be to apply the share to the max mintable amount. The max mintable amount in this case being the difference between mineMintCount and coreMintCount.

Impact:

- The protocol can lose significant governance power very early. The minting function could become unusable until many tokens are minted to users, inadvertently increasing their governance power beyond intended limits.
- The formula used to calculate the _mintable results in less tokens minted than it should be.

Proof of Concept:

You can add the following test in the file PuppetToken.t.sol

```
    function testMintCoreUnderflows() public {

        // Initially tokens are minted to users
        puppetToken.mint(users.alice, 100e18);

        // Owner gets the equal portion of the governance tokens since no time has passed
        puppetToken.mintCore(users.owner);

        // Jump by 1 year
        vm.warp(31560000);
        vm.roll(1000000);

        // Mints some tokens to users
        puppetToken.mint(users.bob, 80e18);

        // Minting the same portion reverts because the function applies the rate before substracting
        vm.expectRevert(stdError.arithmeticError);
        puppetToken.mintCore(users.owner); // mints at 0,5 rate

    }
```

Recommended Mitigation:

Consider substracting `coreMintCount` from `mineMintCount` before applying the `Precision.applyFactor` function :

```
-    uint _maxMintable = Precision.applyFactor(_mintShare, mineMintCount);
-    uint _mintable = _maxMintable - coreMintCount;

+    uint _mintable = Precision.applyFactor(_mintShare, mineMintCount - coreMintCount);
```

Low

[L-1] For the same amount of minted tokens to users, `PuppetToken::mintCore` can mint fewer tokens depending on timing

Description:

When an authorized party mints a certain amount of tokens to users via `mint`, a corresponding amount is expected to be minted via `mintCore`. The current design only takes into account the current block.timestamp of when the function `mintCore` is run. However, the amount minted through `mintCore` depends on the current `block.timestamp` at the time `mintCore` is called. This can lead to discrepancies where the protocol or owner receives fewer tokens if there is a delay in calling `mintCore` after tokens have been minted to users. This is because the `getCoreShare()` function calculates the share based on the time elapsed since deployment using the current timestamp, rather than the timestamp of the last minting event.

```
    function getCoreShare() public view returns (uint) {
        uint _timeElapsed = block.timestamp - deployTimestamp;
```

Impact:

If the authorized party forgets or delays running the function `mintCore`, the protocol or core contributors may receive fewer tokens than intended. These discrepancies can accumulate over time and become important.

Proof of Concept:

```
    function testMintCoreAmountIslessThanExpected() public {

        // Mints 200 tokens to alice
        assertEq(puppetToken.mint(users.alice, 200e18), 200e18);

        // The code contributors should be getting 200 or slighly less tokens than that

        // Assume authorized party calls mintCore 10 days after
        vm.warp(864000); // jump 10 days
        assertApproxEqAbs(puppetToken.mintCore(users.bob), 194e18, 1e18);
        // Bob is only getting 194 tokens, 6 tokens less than expected

        // This issue can happen each time mintCore is called with core contributors getting less tokens than expected every single time. Over time, these discrepancies can accumulate and become important. Had getCoreShare used the exact time when mint was called to calculate the _timeElapsed, these discrepancies would be less important.
    }
```

Recommended Mitigation:

Introduce a new storage variable to keep track of the timestamp of the last `mint` call. Then use this timestamp in the calculate of the time delta in the function `getCoreShare()`, ensuring a more consistent and fair distribution of minted tokens regardless of timing delays since the last `mint` was run.

Informational

[I-1] `PuppetToken::getCoreShare` function can be improved for better readability, less precision loss and less gas cost

Description:

As of now, the function `PuppetToken::getCoreShare` makes two operations to calculate the share. It first determines the `_diminishFactor`, then returns the rate.

```
        uint _diminishFactor = Precision.toFactor(CORE_RELEASE_DURATION_DIVISOR + _timeElapsed, CORE_RELEASE_DURATION_DIVISOR);
        return Precision.toFactor(Precision.FLOAT_PRECISION, _diminishFactor);
```

This could be shortened into one simpler operation.

Impact:

Precision loss and more gas fees.

Recommended Mitigation:

Consider the following changes:

```
-       uint _diminishFactor = Precision.toFactor(CORE_RELEASE_DURATION_DIVISOR + _timeElapsed, CORE_RELEASE_DURATION_DIVISOR);
-       return Precision.toFactor(Precision.FLOAT_PRECISION, _diminishFactor);

+       return Precision.toFactor(CORE_RELEASE_DURATION_DIVISOR, CORE_RELEASE_DURATION_DIVISOR + _timeElapsed);
```

[I-2] Events missing indexed fields

Index event fields make the field more quickly accessible to off-chain tools that parse events. However, keep in mind that indexing a field costs extra gas.

```
        event PuppetToken__SetConfig(Config config);
```

```
        event PuppetToken__MintCore(address operator, address receiver,
            uint timestamp, uint amount, uint share);
```

[I-3] Use `uint256` instead of `uint` for better readability

In many instances of the contracts, `uint` is used to declare unsigned 32 bytes integer variables. `uint` is an alias of `uint256`. For better readability, it is better to specify the number of bits needed to store the variables.

Gas

[G-1] The `PuppetToken::deployTimestamp` should be set as an immutable variable to save gas

Description:

`deployTimestamp` is declared as a storage variable while its value never changes. To save gas, it should be declared as an immutable variable since it will be stored directly in the contract bytecode rather than storage. Which will save significant gas.

```
    uint public deployTimestamp = block.timestamp;
```

Recommended Mitigation:

```
    uint256 public constant deployTimestamp = block.timestamp;
```

[G-2] `public` functions not used internally could be marked `external`

Description:

The function `PuppetToken::getMarginAmount` is not used internally. It should be marked external to save gas.

```
    function getMarginAmount() public view returns (uint) {
```

Recommended Mitigation:

```
    function getMarginAmount() external view returns (uint) {
```