Invariant Test Suite & Security Report

This report was produced for the M^ZERO Protocol by Prototech Labs

DRAFT

Mail: info@prototechlabs.dev
Twitter: @Prototech_Labs
Web: www.prototechlabs.dev

Contents

1. Executive Summary

This report and the accompanying invariant test suite was prepared for the M^ZERO team by Prototech Labs, a smart contract consultancy providing security, technical advisory, and code review services. Prototech Labs would like to thank the M^ZERO team for giving us the opportunity to review the current state of their protocol.

This document outlines the findings, limitations, and methodology of our review, which is broken down by issue and categorized by severity. It is our hope that this provides valuable findings and insights into the current implementation and that the invariant tooling can be continuously updated to inform the safety of future development.

2. Project Overview

This security review, unlike a traditional audit, focused on providing M^ZERO with a comprehensive test suite to define and test invariant assertions.

This involved identifying and documenting protocol invariants in order to develop a set of test cases to in-turn validate the identified invariants. This then enabled us to execute the developed test suite against the M^0 protocol, recording and analyzing the results of each run as well as any deviations from the expected invariant's behavior.

It is worth mentioning that this test suite, in addition to the already identified issues, is capable of continuously producing results if updated with new invariants. This will require ongoing analysis and will act as a continued "proof of work" i.e. meticulous analysis of current and future invariant behavior.

With this in mind, we have included a "Further Investigation" tag on certain issues that exhibit the potential for additional scrutiny beyond the scope of this engagement and may uncover additional insight and/or deviation from what is expected.

Project Details:

- **Security Researchers:**
    - Chris Mooney
    - Chris Smith
    - Brian McMichael
    - Derek Flossman
- **Timeline:** 2024-01-08 to 2024-02-09
- **Code Repository: https://github.com/MZero-Labs**
- **Commit:**

- **common -** 4a37119f2da946c6d8ad7b9a70dfdd219225115b
- **TTG -** a8127901fa1f24a2e821cf4d9854a1aa6ac8088c
- **Protocol -** 3499f50ff3382729f3e59565b19386ba61ef8e36

3. Introduction

The core M^0 protocol is a coordination layer for permissioned institutional actors to generate M, which is a fungible token generated by locking eligible collateral in a secure off-chain facility. The protocol enforces a common set of rules and safety procedures for the management of M.

4. Limitations and Report Use

It is worth highlighting that this security review is an invariant analysis and should be considered as complementary to a traditional audit, not as a replacement.

Disclaimer: No assessment can guarantee the absolute safety or security of a software-based system. Further, a system can become unsafe or insecure over time as it and/or its environment evolves. This assessment aimed to discover as many issues and make as many suggestions for improvement as possible within the specified timeframe. Undiscovered issues, even serious ones, may remain. Issues may also exist in components and dependencies not included in the assessment scope.

The software systems herein are emergent technologies and carry with them high levels of technical risk and uncertainty. This report and related analysis of projects to not constitute statements, representations or warranties of Prototech Labs in any respect, including regarding the security of the project, utility of the project, suitability of the project's business model, a project's regulatory or legal status or any other statements, representations or warranties about fitness of the project, including those related to its bug free status. You may not rely on the reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Our complete terms of service can be reviewed here.

Specifically, for the avoidance of doubt, any report published by Prototech Labs does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of any client or project, and is not a guarantee as to the absolute security of any project. Prototech Labs does not owe you any duty by virtue of publishing these reports.

5. Findings Overview

Below is an overview of the findings, split by severity, illustrating their status (Fixed/Acknowledged):

Critical Severity Findings	[3]
7.1 Any action that moves delegation to address(0) will cause that user's funds to be locked.	
7.2 PowerToken Balances can be double counted	
7.3 PowerToken: Delegation and transfer fails when actor.balance > actor.votes	

High Severity Findings	[4]
8.1 MToken.mint() can overflow totalNonEarningSupply and principalOfTotalEarningSupply	
8.2 ERC3009 validAfter and validBefore are incorrectly implemented as inclusive	
8.3 PowerToken: Inflation rounding creates deviation in account balances and total supply.	
8.4 PowerToken: User Balance is lost on each reset due to inflation rounding	

Medium Severity Findings	[5]
9.1 High Mint Ratio and High Collateral can cause Uint112 overflow in UpdateCollateral	
9.2 dynamic calculation of collateral expiry creates unintended consequences	
9.3 resetToTokenHolders() functions will brick the new vote token if the bootstrap token's pastTotalSupply(epoch) returns 0	
9.4 PowerToken: Account balances can exceed total supply.	
9.5 Invariant P_VD2 failure: Actor votes do not match delegated balance.	

Low Severity Findings	[9]

10.1 MToken updateIndex called multiple times in burn and mint	
10.2 cash token that doesn't return true on transfer	
10.3 updateCollateral potentially leaves the system in an undesirable state	
10.4 proposeMint allows type(uint240).max, but mintM only allows type(uint112).max	
10.5 TTG Setting Minter Rate too high will lead to updateIndex overflow	
10.6 TTG Set mintRatio() == 0 causes all positions to be undercollateralized	
10.7 Invariant Violation/Accounting reported incorrectly	
10.8 Users can accidentally lock their funds	
10.9 Inconsistent inflation due to rounding truncation	

Informational Findings	**[11]**
11.1 transferFrom with insufficient balance leads to Panic: over/underflow	
11.2 ERC20Extended: Insufficient allowance for transferFrom results in Panic underflow	
11.3 can freeze deactivated minter	
11.4 StandardGovernor.t.sol does not test setKey	
11.5 Allow public reading of proposalFees in StandardGovernor	
11.6 Reduce Duplicate Code to Prevent the Introduction of Bugs	
11.7 SignatureChecker.sol vulnerable to signature malleability	
11.8 Investigate MinterGateway and MToken updateIndex	
11.9 MinterGateway does not validate that all signatures are in ascending order	
11.10 MinterGateway verifyValidatorSignatures could bail early	
11.11 OnBehalf -> OnBehalfOf	

6. Findings Framework

Findings and recommendations are listed in the below section, grouped into broad categories. It is up to the team behind the code to ultimately decide whether the items listed here qualify as issues that need to be fixed, and whether any suggested changes are worth adopting. When a response from the team regarding a finding is available, it is provided.

Findings are given a severity rating based on their likelihood of causing harm in practice and the potential magnitude of their negative impact. Severity is only a rough guideline as to the risk an issue presents, and all issues should be carefully evaluated.

Severity Level Determination		Impact		
		High	Medium	Low
Likelihood	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low

Additionally;

- Issues that exhibit the potential for additional scrutiny beyond the scope of this engagement have an added Further Investigation tag.
- Issues that do not present any quantifiable risk are given a severity of Informational.

7. Critical Risks

7.1 Any action that moves delegation to address(0) will cause that user's funds to be locked.

Context:

EpochBasedVoteToken.sol#L184

Description:

Any delegation to `address(0)` causes the delegator's funds to be locked when attempting to re-delegate or transfer in the future. This is in contrast to the EIP-5805 specification which states that `Tokens that are delegated to address(0) should not be tracked. This allows users to optimize the gas cost of their token transfers by skipping the checkpoint update for their delegate.` This is further evidenced by suggested properties of EIP-5805: `For all timepoints t < clock, getVotes(address(0)) and getPastVotes(address(0), t) SHOULD return 0.` In addition, invariant testing and debugging suggested this property does not hold for either `ZeroToken` or `PowerToken`.

```
    function test_addressZeroFailure() external {
        _warpToNextTransferEpoch();

        _vote.mint(_alice, 1_000);
        _vote.mint(_bob, 900);
        _vote.mint(_carol, 800);

        vm.prank(_alice);
        _vote.delegate(address(0));

        // vm.prank(_alice);
        // _vote.delegate(_bob);

        vm.prank(_alice);
        _vote.transfer(_carol, 400);
    }
```

For Further Investigation

A transfer of funds to `address(0)` means those funds can never be accessed again, but this means even the transfer class of functions to `address(0)`, which delegate under the hood, will cause the funds and vote weight sent to `address(0)` to be locked.

It is suggested to prevent the transfer of funds to `address(0)` as this issue may introduce undefined behavior when combining multiple specifications or not following standard practices. For example, a movement of funds to `address(0)` may have no real negative implications other than bad UX for users, but one must carefully think through that implication when a transfer also moves voting weight to `address(0)`.

Recommendation:

The specification implies wanting to `allow users to optimize the gas cost of their token transfers by skipping the checkpoint update for their delegate` when delegating to `address(0)`. For this reason, you may still want to update a delegator's delegatee to `address(0)` to indicate this state, but not move the vote power to `address(0)`.

Suggested change:

```
diff --git a/src/abstract/EpochBasedVoteToken.sol
b/src/abstract/EpochBasedVoteToken.sol
index 8d857a9..ee2754c 100644
--- a/src/abstract/EpochBasedVoteToken.sol
+++ b/src/abstract/EpochBasedVoteToken.sol
@@ -181,7 +181,7 @@ abstract contract EpochBasedVoteToken is IEpochBasedVoteToken,
ERC5805, ERC20Ext
        address oldDelegatee_ = _setDelegatee(delegator_, newDelegatee_);
        uint240 votingPower_ = _getBalance(delegator_, _clock());

-        if (votingPower_ == 0) return;
+        if (votingPower_ == 0 || newDelegatee_ == address(0)) return;

        _removeVotingPower(oldDelegatee_, votingPower_);
        _addVotingPower(newDelegatee_, votingPower_);
```

Moderate testing of the suggested change has been done with no issues, but more testing is needed before having confidence in this fix.

M^ZERO:

Prototech:

7.2 PowerToken Balances can be double counted

Context: File.sol#L123

Description:

When a user has not synced their past balance and uses `transfer`, `transferFrom`, `delegateBySig` or `transferWithAuthorization` to delegate or transfer to/from themselves, their `pastBalanceOf` is double counted, once as the sender and once as the recipient giving them twice the tokens they should have. These regressions show this occurring and resulting in one user doubling their tokens:

test_regression_invariant_P_B1_abd6c842_failure()
test_regression_invariant_P_B1_5b9e6c92_failure()

test_regression_invariant_P_B1_97d35b59_failure()
test_regression_invariant_P_B1_b3dd037b_failure()
test_regression_invariant_P_B1_8f24f499_failure()

```
Balance:       11000
Total Supply:  10000
```

These regressions occur in the signature path and the regular paths for transfer, delegate and transferFrom.

Recommendation:

Use proper check-effects pattern to make sure the sender is fully synced before checking whether the recipient has been synced. Recommending Further Investigation for the direct source of this bug. We theorize that the use of `storage` for `voidSnaps_` is re-accessing the empty storage of voidSnaps_ when the to account and from account are the same in the same call.

M^ZERO:

Prototech:

7.3 PowerToken: Delegation and transfer fails when actor.balance > actor.votes

Context: PowerToken.sol

Description:

Identified a scenario where actor balance > actor votes. In this scenario, delegation and transfer above the vote amount but within the available balance fails with an overflow.

- Regression: test_regression_invariant_P_B3_52af74ce_failure()

```
actor:  0x0F8458E544c9D4C7C25A881240727209caae20B8
 delegatee:  0x7565Aef10F626C1972cd51B4A951b0601917940E
 actor votes:  2058
 actor balance:  4606
 delegatee votes:  0
 delegatee balance:  685
 Unhandled Error:
 0x4e487b7100000000000000000000000000000000000000000000000000000000000000011
```

- Output

Modifying the end of the regression with the following code demonstrates that transfers fail when the balance to be transferred is less than the available balance but more than the `getVotes()` total.

```
       ...

_powerTokenHandler.cancelAuthorizationWithVS(190053995411135797696459572131344865291035859170 40, 221376431755056998436218163688023 60, 220);
       _powerTokenHandler.markNextVotingEpochAsActive(1732991441);
       //_powerTokenHandler.delegate(190077957052806333731633 48, 1673);
       //_powerTokenHandler.delegate(2331204567, 52222);

       vm.prank(0x0F8458E544c9D4C7C25A881240727209caae20B8);
       powerToken.transfer(address(1), 3000);

       invariant_P_B3();

=====

   │   ├─ emit Transfer(sender: MockCashToken: [0x0F8458E544c9D4C7C25A881240727209caae20B8], recipient: 0x0000000000000000000000000000000000000001, amount: 3000)
   │   └─ ← panic: arithmetic underflow or overflow (0x11)
   └─ ← panic: arithmetic underflow or overflow (0x11)
```

Recommendation:

Analyze regression to determine conditions for the scenario where actor balance can be greater than votes. This leads to over/underflow in the vote transfer portion of `delegate()` and transfer operations. Recommending Further Investigation

M^ZERO:

Prototech:

8. High Risks

8.1 MToken.mint() can overflow totalNonEarningSupply and principalOfTotalEarningSupply

Context:

MToken.sol#L217

Description:

Calling MToken.mint() with a large enough value on either earners or non-earners will overflow `principalOfTotalEarningSupply` and `totalNonEarningSupply` as the accumulators are in `unchecked {}` blocks. While we would typically say this condition is a medium severity concern as minting that amount is unlikely given `MToken` being dollar denominated, there are a few additional considerations that lead us to bump this to a high severity risk:

1. The lower maximum limit of the `uint112` space over the lifespan of the protocol (150 years) assuming 10% to 20% inflation rates puts us marginally within reach of this limit around the 180 year mark. This alone isn't cause for concern unless one sees those parameters radically changing (think Argentinean inflation rates) in the medium term. However...
2. While the happy path of the minting protocol should not allow for accidental collateral magnitudes to be too large, or the governance set rates or minter ratios to be off, human errors do occur. A fat fingered decimal or magnitude in any one of these human controlled variables could lead to a large enough `MToken.mint()` that it triggers this overflow. Our team, having seen exactly this type of magnitude error in the past and building out extensive processes to catch it in the future, we can safely say the probability of this happening within a 10 year period is approaching 100%. Since the fallout of this happening is so extremely severe as to brick `MToken` for some users it would be prudent to follow a defense-in-depth strategy to try and prevent this, which is why...
3. We believe the purpose of the `OverflowsPrincipalOfTotalSupply()` check was intended to catch this case. Unfortunately, the overflow happens before this check can occur in both the earner and non-earner cases, and thus the account is credited with the larger balance while the `principalOfTotalEarningSupply` and `totalNonEarningSupply` are much smaller having previously overflowed.

To reproduce this overflow for `totalNonEarningSupply`, you can put the following regression test into `MTokenRegressionTests.t.sol`:

```
    function test_regression_invariant_M_B2_B3_B4_d1d15304_failure() external {
        _mTokenHandler.setMaxLeap(43200);
        _mTokenHandler.mint(633537332909532393609081341809671716 84,
1822344376649243943,
1157920892373161954235709850086879078532699846656405640394575840079131296399 33);

        invariant_M_B2_B3_B4();
```

```
    }
```

and run

```
make regression mt=test_regression_invariant_M_B2_B3_B4_d1d15304_failure
```

You can also reproduce this for the earning side with:

```
function test_regression_invariant_M_B2_B3_B4_880410c4_failure() external {
        _setMaxLeap(500000);

_mTokenHandler.updateIsEarnersListIgnored(48261550356638759991277929160587301488275
5448541786262084, true);
        _mTokenHandler.mint(2, 96506904651951523478815261949 95,
11579208923731619542357098500868790785326998466564056403945758400791312963 9932);
        _mTokenHandler.startEarning(67795994391464209998787765233700864096169072 35);
        _mTokenHandler.startEarning(989459965652266897858);
        _mTokenHandler.mint(0,
11579208923731619542357098500868790785326998466564056403945758400791312963 9934,
11579208923731619542357098500868790785326998466564056403945758400791312963 9935);

        invariant_M_B2_B3_B4();
    }
```

Output: https://gist.github.com/brianmcmichael/428b92ceb0a0db14619a5a121430bcac

Recommendation:

Either find a gas efficient way to front-load the `OverflowsPrincipalOfTotalSupply()` check, or simply remove the `unchecked {}` blocks from `_addEarningAmount()` and `_addNonEarningAmount()`. In testing, removing the `unchecked {}` blocks in both functions did resolve this regression.

Further, we would suggest diving deeper on `unchecked {}` blocks to ensure they are only used to save gas when over/under flow errors can be eliminated by other means.

M^ZERO:

Prototech:

8.2 ERC3009 validAfter and validBefore are incorrectly implemented as inclusive

Context: ERC3009.sol#L238-L239

Description:

The ERC3009 spec calls for signatures to not be valid until after `validAfter`, but before `validBefore`, non-inclusive. The way MZero has implemented this common library treats `validAfter` and `validBefore` as inclusive. This library is used for `MToken`, `PowerToken`, and `ZeroToken` and could allow for the incorrect execution time of a `transferWithAuthorization()` or `receiveWithAuthorization()` on any of those tokens.

This small difference could result in a user's transaction being executed outside (late or premature) of their expectations, and under the right condition, could result in the loss of funds for the sender or recipient.

Recommendation:

Simply making the following change should resolve this issue:

```
diff --git a/src/ERC3009.sol b/src/ERC3009.sol
index f988505..cbac1ec 100644
--- a/src/ERC3009.sol
+++ b/src/ERC3009.sol
@@ -235,8 +235,8 @@ abstract contract ERC3009 is IERC3009, StatefulERC712 {
         uint256 validBefore_,
         bytes32 nonce_
     ) internal {
-        if (block.timestamp < validAfter_) revert
AuthorizationNotYetValid(block.timestamp, validAfter_);
-        if (block.timestamp > validBefore_) revert
AuthorizationExpired(block.timestamp, validBefore_);
+        if (block.timestamp <= validAfter_) revert
AuthorizationNotYetValid(block.timestamp, validAfter_);
+        if (block.timestamp >= validBefore_) revert
AuthorizationExpired(block.timestamp, validBefore_);

         _revertIfAuthorizationAlreadyUsed(from_, nonce_);
```

There may be a chance this problem was introduced when converting the `require()` errors in the spec to custom error logic using `if()` conditionals. As custom error patterns with `if()` conditionals are the logical negation of the inequality in the `require()` and thus the:

```
       require(now > validAfter, "EIP3009: authorization is not yet valid");
        require(now < validBefore, "EIP3009: authorization is expired");
```

should become:

```
        if (now <= validAfter) revert AuthorizationNotYetValid();
        if (now >= validBefore) revert AuthorizationExpired();
```

It could be a common mistake in adapting other specifications or code to custom errors that the inequality was just reversed and the `=` was left out. For this reason, we have done, and recommend the M^ZERO team do a review of all custom error boundary conditions.

M^ZERO:

Prototech:

8.3 PowerToken: Inflation rounding creates deviation in account balances and total supply.

Context: PowerToken.sol

Description:

Several regressions were discovered that indicate that the total sum of user balances after inflation do not equal the total supply. This is likely due to the necessity of rounding down on the 10% epoch inflation.

Example regressions:

test_regression_invariant_P_B1_4d72c83e_failure()

```
Balance:  11099
Total Supply:  11100
```

test_regression_invariant_P_B1_bc6627bc_failure()

```
Balance:  13640
Total Supply:  13641
```

test_regression_invariant_P_B1_ba29ad51_failure()

```
Balance:  11099
Total Supply:  11100
```

Recommendation:

Consider the long-term effects of the sum of total balances not equaling the total supply after inflation, especially on smaller balances which can not be fully inflated by 10% and

must be rounded down. This deviation is likely to compound over many epochs and may have a material effect on voting thresholds. Recommending Further Investigation

M^ZERO:

Prototech:

8.4 PowerToken: User balance is lost on each reset due to inflation rounding

Context: PowerToken.sol

On each reset via a PowerBootstrapToken, some coins are lost in the scaling down of the inflated supply. In theory, the rapid inflation of PowerToken should dilute this discrepancy out of having a meaningful effect on the system.

Description:

```
./ttg/test/PowerToken.t.sol

function test_initial_balance_mismatch() external {
       uint256 _balance;

       for (uint256 i = 0; i < _initialAccounts.length; ++i) {
           _balance += _powerToken.balanceOf(_initialAccounts[i]);
       }

       // Balances == Total Supply
       assertEq(_balance, _powerToken.totalSupply());

       console.log("Balance of user accounts: ", _balance);
       console.log("PowerToken.totalSupply(): ", _powerToken.totalSupply());
   }
```

Result:

```
Running 1 test for test/PowerToken.t.sol:PowerTokenTests
[FAIL. Reason: assertion failed] test_initial_balance_mismatch() (gas: 133182)
Logs:
 Error: a == b not satisfied [uint]
       Left: 9998
      Right: 10000
 Balance of user accounts:  9998
 PowerToken.totalSupply():  10000
```

Further research has revealed that with an inflated supply of power token and a number of smaller token holders, the balances of all users can be rounded off, resulting in a `0` balance after a token reset. This can materially affect the protocol's ability to perpetuate governance after a reset if the token is owned by a large number of small holders. In fact, the total of user balances after a reset can equal zero if the token is sufficiently distributed.

```
./ttg/test/PowerToken.t.sol

 function test_bigSupplySmallHolders() external {
     MockBootstrapToken bootstrapToken2_ = new MockBootstrapToken();
     uint256 initialSupply2_ = 15_000_000 * 1e6;
     bootstrapToken2_.setTotalSupply(initialSupply2_);

     for (uint256 i; i < 20_000; ++i) {
         _bootstrapToken.setBalance(address(uint160(i + 1)), initialSupply2_ /
20_000);
     }

     PowerTokenHarness powerToken2_ = new
PowerTokenHarness(address(bootstrapToken2_), _standardGovernor,
address(_cashToken), _vault);

     uint256 userBalances_;
     for (uint256 i; i < 20_000; ++i) {
         userBalances_ = userBalances_ + powerToken2_.balanceOf(address(uint160(i +
1)));
     }

     assertEq(userBalances_, powerToken2_.totalSupply());
 }
```

Result:

```
[FAIL. Reason: assertion failed] test_bigSupplySmallHolders() (gas: 715354455)
Logs:
 Error: a == b not satisfied [uint]
       Left: 0
      Right: 10000
```

This test assumes 20,000 users. Other popular governance tokens, Maker (99,305 holders), Aave (165,852 holders), and Compound (218,296 holders) have far in excess of this, so a reset of these tokens would wipe most of these users holdings, and the proportions of holdings relative to reported total supply.

Recommendation:

M^ZERO:

Prototech:

9. Medium Risks

9.1 High Mint Ratio and High Collateral can cause Uint112 overflow in UpdateCollateral

Context:

- MinterGateway.sol#L714
- MinterGateway.sol#L606

Description:

If TTG sets the Mint Ratio to a larger than expected (per M^0's comment that it should be between 0 and 10_000) and a user has a large amount of collateral, then calculating the `principalOfMaxAllowedActiveOwedM` in the `_imposePenaltyIfUndercollateralized` function will overflow `Uint112` blocking the update. We tried Mint Ratios much larger but eventually saw the issue even with 18_500, so then restricted the handler to not go above 10_000. However, the `mintRatio()` function in MinterGateway allows it to return `(100 * uint32(10_000)` which will cause this overflow.

Recommendation:

Ensure this overflow on large number scales is desirable and clearly document to TTG the max limit of 10_000 for the Mint Ratio. Change the cap in `MinterGateway.mintRatio` so that it protects against this as it would block calls to `updateCollateral`.

Related: #73

M^ZERO:

Prototech:

9.2 dynamic calculation of collateral expiry creates unintended consequences

Context:

- MinterGateway.sol#L502-L503
- MinterGateway.sol#L536-L540
- MinterGateway.sol#L594-L596

Description:

Because the calculation for determining whether collateral is past its update timeframe is calculated "on demand" based on the TTG value, minters are not guaranteed a set period of time to update their collateral. This leads to two potentially dangerous scenarios and does not conform to the description in the whitepaper.

1. **TTG Rug Pull Scenario**

Changes to the TTG param for `UPDATE_COLLATERAL_INTERVAL` can result in a collateral "rug pull" for Minters. For instance, a minter calls updateCollateral when the `UPDATE_COLLATERAL_INTERVAL` is set to a relatively large number, e.g. `7 days`. In the next block the TTG changes the value to a very low number `1 second`. This automatically causes the minter to have a `collateralOf` balance of 0 and start incurring penalties.

2. **Inactive Minter Allowed to Mint without penalty potentially against invalid Collateral Scenario**

If a minter has missed a collateral update, but Power Token holders then change the UPDATE_INTERVAL to a large enough number such that `lastCollateralUpdateTimestamp + new UPDATE_INTERVAL > current timestamp`, that delinquent minter would be able to mint new M Token without penalty. A Minter in this situation would be incentivized to put forth a proposal to Power Token holders to increase the interval in their favor as long as the Proposal Fee < what they could mint and they believe they can get the proposal passed. This calculation becomes more straightforward if the Cash Token in TTG becomes M Token since they wouldn't have to factor in the relative price of WETH and M Token in their calculation. Further, if a Minter believed they could sway a Threshold of Power Token holders to vote for it, they could pass it as an Emergency Proposal without paying the fee.

This appears to contradict the Whitepaper. Section II.I.I Generation of M p5, states:

“If a Minter fails to call Update Collateral within **Update Collateral Time of the previous time they called it**, their on-chain Collateral Value

is assumed to be 0. *(emphasis added)”*

Recommendation:

Instead of storing `updateTimestamp` store it as `timestamp of update + current TTG interval`. This means that every time a minter updates their collateral, they know exactly how long they have before they have to call `updateCollateral` again.

M^ZERO:

Prototech:

9.3 resetToTokenHolders() functions will brick the new vote token if the bootstrap token's pastTotalSupply(epoch) returns 0

Context:

PowerToken.sol#L363-L364

Description:

On epoch 1, if a `resetToPowerHolders()` or `resetToZeroHolders()` is performed, it will succeed, but the new vote token will be looking back at either the prior `PowerToken` or `ZeroToken` as its bootstrap token to pull balances from. During this process, if `PriorToken.pastTotalSupply(epoch)` returns 0, it will brick the new vote token and everything that depends on it. In epoch 1, the look-back is to epoch 0 making `PriorToken.pastTotalSupply(epoch) == 0` true and causing a division-by-zero error on even simple `balanceOf()` and `pastBalanceOf()` checks of the new vote token.

To reproduce this, you need the `reset` PR, you have to unpatch `ZeroGovernorHandler` and run the following regression:

```
function test_regression_invariant_P_VD3_fcc87b5e_failure() external {
     _setMaxLeap(3600);
     _zeroGovernorHandler.resetToZeroHolders(165449107);

     invariant_P_VD3();
 }
```

You can run this on that branch with:

```
make regression mt=test_regression_invariant_P_VD3_fcc87b5e_failure
```

Which will give the following stack trace. If this is chased out, you will see that there is a division-by-zero in PowerToken.sol#L363-L364 as a result of getting `pastTotalSupply(0)`.

Recommendation:

A broader check on reset that the bootstrap token returns a `BootstrapToken.totalSupply() > 0` for the prior epoch is suggested before allowing a reset. Knowing this condition exists, it's also advisable to review what types of natural conditions (resetting inflation, etc.) could lead to the bootstrap token returning a `BootstrapToken.totalSupply() > 0` but if the guard is in place to never allow a reset under these conditions, the defect should be mitigated.

M^ZERO:

Prototech:

9.4 PowerToken: Account balances can exceed total supply.

Context: PowerToken.sol

Description:

It appears that `markParticipation` can cause user balances to exceed `totalSupply`. The following regression shows `Actor7` receives an extra `100` tokens (or 10%) after self delegating and getting a `markParticipation` call. It was not clear to us exactly where this happens or why and would strongly recommend further investigation.

test_regression_invariant_P_B1_5cd1d968_failure()

```
Balance:  11100
Total Supply:  11000
```

Recommendation:

Explore the effects of inflating the total supply in one epoch and then waiting multiple epochs before adjusting balances. Regressions indicate that balance inflation may occur beyond the expected supply inflation. Recommending Further Investigation

M^ZERO:

Prototech:

9.5 Invariant P_VD2 failure: Actor votes do not match delegated balance.

Context: test/invariant/PowerTokenInvariants.t.sol

Description:

Invariant testing found the following path indicating a violation of P_VD2, taken from the spec as:

- `POWER totalVotingPower(delegates) >= POWER totalSupply(holders), at Voting Epoch`

In this regression, we arrive at a state where the actor has 1000 tokens delegated to them but 0 voting power in the Voting Epoch.

```
function test_regression_invariant_P_VD2_dc8c60c1_failure() external {
       _setMaxLeap(500000);

_powerTokenHandler.receiveWithAuthorization(115792089237316195423570985008687907853
269984665640564039457584007913129639932, 3, 0,
253986705183879762055281264313715668777998804619367236751651335 45,
287291570320378927757552901276423655883425717256625866, 1,
115792089237316195423570985008687907853269984665640564039457584007913129639933);
       _powerTokenHandler.transferFrom(22213, 2434546984, 1702447259, 8260);

_powerTokenHandler.markNextVotingEpochAsActive(438802759611688308251264933659464847
473705653034007394227835954294284480 60990);

_powerTokenHandler.receiveWithAuthorization(115792089237316195423570985008687907853
269984665640564039457584007913129639933, 0, 49834230, 0, 424630258064581,
380586009783327831992825411860219834967406745103670131392013009246520094 8856,
139815080442061025309583034756701671410363211329023358648989029253274);

_powerTokenHandler.receiveWithAuthorizationWithVS(115792089237316195423570985008687
907853269984665640564039457584007913129639932,
199471554110287018978338904716975211 3056, 192443,
115792089237316195423570985008687907853269984665640564039457584007913129639932,
1491493340,
115792089237316195423570985008687907853269984665640564039457584007913129639932, 3);

_powerTokenHandler.receiveWithAuthorizationWithVS(811824123451705359251613629987707
971325940131480422301410313996184506492586 58, 2285641147, 2678319986, 3120,
860513737115125424259715838243655300661978674 1947, 17524, 962808020);
       _powerTokenHandler.permit(104941798318992048,
115792089237316195423570985008687907853269984665640564039457584007913129639935,
```

```
138163547134013850741991112958540145800041530563712984296599643316852362,
6935567968696137, 125224764112494289676856620912 2,
563632140391696797894471233576286382746254758 5);

_powerTokenHandler.markNextVotingEpochAsActive(115792089237316195423570985008687907
853269984665640564039457584007913129639932);

_powerTokenHandler.receiveWithAuthorizationWithSignature(993406476473724821303,
159781607658, 760621772264564080426596827777805184,
237167492934460516295381190104354166638442, 669429610,
115792089237316195423570985008687907853269984665640564039457584007913129639935,
992712411185438040671170853130937202605334673177682393628994984177463);
        _powerTokenHandler.markParticipation(1685781927, 11615);
        _powerTokenHandler.buy(35, 0,
163154436326094438575454595030815357256148917932890268311, 2);
        _powerTokenHandler.delegateBySig(16822, 1681616932, 2016020, 10820);

_powerTokenHandler.markNextVotingEpochAsActive(101654295392275046989449157414705073
27065631555795255612);

_powerTokenHandler.receiveWithAuthorization(115792089237316195423570985008687907853
269984665640564039457584007913129639933, 117283234093,
152575517326844531567016988 7, 2, 2,
115792089237316195423570985008687907853269984665640564039457584007913129639935,
115792089237316195423570985008687907853269984665640564039457584007913129639932);

_powerTokenHandler.approve(438802759611688308251264933659464847473705653034007394227
835954294284480610 1, 22214,
57896044618658097711785492504343953926418782139537452191302581570759080747167);

_powerTokenHandler.transferFrom(300495785111472157848088794504798162743572937124350
701877252055569332741079 4, 18446744073709478664, 23676,
310060342998169258142090471992827642853882285130000629075200 );
        _powerTokenHandler.delegate(2,
151614098927087407634265825587666678621700061471723619952908 );
        _powerTokenHandler.transferWithAuthorization(18446744073709400180, 24407,
16750511, 4694, 1865844395, 4006, 792985628);

        invariant_P_VD2();
    }
```

Regression output:
https://gist.github.com/brianmcmichael/be5e9703d91fc94c99ca1774e3a84084

Recommendation:

Review transaction log and evaluate the invariant specification. Recommending Further Investigation

M^ZERO:

Prototech:

10. Low Risk

10.1 MToken updateIndex called multiple times in burn and mint

Context:

- _burn - MToken.sol#L206
- _mint - MToken.sol#L223

Description:

The `mint` and `burn` functions on MToken are only callable by the `MinterGateway`. Inside `MinterGateway`, the functions that call `mint` and `burn` also call `MinterGateway.updateIndex()` which calls `MToken.updateIndex()`. Inside the MToken's `mint` and `burn` functions it checks if the `recipient` or `account` are earning and if they are it calls `MToken.updateIndex()`.

This at a minimum results in duplicate calls to `MToken.updateIndex()` and wasting gas. However, it could also represent unintended consequences since the `!isEarning` path of `mint` and `burn` do not "explicitly" call `MToken.updateIndex()`.

Recommendation:

1. Ensure there are no unintended consequences for having `MToken.updateIndex()` called when the `recipient` or `account` are not earning.
2. Remove the duplicate `updateIndex` call from MToken's function and let it be called by the MinterGateway's `updateIndex`.

M^ZERO:

Prototech:

10.2 cash token that doesn't return true on transfer

Context: PowerToken.sol#L119

Description:

PowerToken checks the return value of `transferFrom()` on CashTokens.

Tokens that do not return a value on the `transferFrom()` are ineligible to be cash tokens.

Not returning a `bool` on `transferFrom` is non-conformant to ERC-20 spec, but some popular tokens, like USDT, do not include the bool return and would not be suitable cashTokens for the protocol.

Recommendation:

- Add a test of a non-conformant mock.
- Document for token governors that certain tokens are ineligible for inclusion as cashTokens.
- Use a wrapper token for non-conformant cashtokens

M^ZERO:

Prototech:

10.3 updateCollateral potentially leaves the system in an undesirable state

Context:

- MinterGateway.sol#L192-L195
- MinterGateway.sol#L136-L179

Description:

Lines 192-195 imply that a Minter should never be able to propose more retrieval than the collateral they have. It was indicated that this should hold in your suggested invariants for minterGateway: `collateralOf >= totalPendingRetrievals`.

However, by calling `updateCollateral` with a new, lower number and not passing any RetreivalIds, the minter would be able to create that exact situation. This violates a potential invariant: Sum of PendingRetreivals <= Sum of MinterState Collateral values.

Recommendation:

To reconcile this inconsistency, either add a similar revert to `updateCollateral`, forcing the minter to process relevant retrievals before updating their collateral value to a smaller number. Alternatively, simplify the system by removing the revert in `proposeRetrieval` as it does not truly protect the system from entering into that state (pendingRetrievals are already removed from the minter collateral in `collateralOf` and therefore are factored in for undercollateralization purposes).

M^ZERO:

Prototech:

10.4 proposeMint allows type(uint240).max, but mintM only allows type(uint112).max

Context: ContinuousIndexing.sol#L100

Description:

If all other requirements are met, proposeMint allows a user to propose minting up to type(uint240).max amount. However, such a proposal would overflow on type(uint112).max once called because of `_getPresentAmountRoundedUp`

Recommendation:

Ensure the max amount in a `proposeMint` is `type(uint112).max`

M^ZERO:

Prototech:

10.5 TTG Setting Minter Rate too high will lead to updateIndex overflow

Context: ContinuousIndexing.sol#L60-L66

Description:

Setting a rate too high for too long will result in `multiplyIndices` overflowing `uint112` and lock the MinterGateway. As discussed you expect TTG to use rates between 0 and 40_000 for the minter rate, however, if governance deviates from this expectation or erroneously sets a very high rate, the system could be locked.

Recommendation:

Ensure that it is well documented for Governance what the limits on rates are.

M^ZERO:

Prototech:

10.6 TTG Set mintRatio() == 0 causes all positions to be undercollateralized

Context:

- MinterGateway.sol#L490

Description:

In our discussions you noted that `MINT_RATIO = [0, 10_000]` was a bounding that could be expected from TTG. However, setting `MINT_RATIO` to 0 will cause all positions to be reported as under-collateralized causing all minters to be penalized on `updateCollateral` and reverts in `proposeRetrieval`, `proposeMint`, and `mintM`.

Recommendation:

Do not set this value to 0 unless the desire is to shut off the system or have a fallback min for `MinterGateway.mintRatio()` to return if the ratio from TTG is set to 0.

Related to #90

M^ZERO:

Prototech:

10.7 Invariant Violation/Accounting reported incorrectly

Context:

- MinterGateway.sol#L565-L567
- MinterGateway.sol#L379-L380

Description:

When a Minter is deactivated, their balances are wiped and they can't become active in the system anymore. However, the `pendingCollateralRetrievalOf(minter, retreivalId)` function will still return a value of past proposed retrievals.

This means that the following invariant does not hold:

```
Sum of retrievalProposals == MinterState.totalPendingRetrievals
```

Recommendation:

We are not sure that there is a clean way of deleting the proposal from `_pendingCollateralRetrievals` since it would require iterating through retrievalIds. This could be possible by keeping an array in the MinterState, but adds to storage and processing. The easiest thing here would be to update `pendingCollateralRetrievalOf` so that it returns 0 for a deactivated minter.

M^ZERO:

Prototech:

10.8 Users can accidentally lock their funds

Context:
MToken.sol#L329

EpochBasedVoteToken.sol#L272

Description:

It's a common problem that users lock their funds by sending them to the token contract directly or to `address(0)`. For more context see this pull request in the Maker dss codebase on sending to the contract directly. While Maker chose not to prevent this behavior in DAI as it would have socialized the gas cost for this check to all users, they later regretted not putting the check in as it was a simple guard against extremely bad user experiences due to loss of funds.

Recommendation:

The suggestion we would make is to explicitly prevent this in the code as a require, which will add additional gas costs for all users. This socialization of gas costs, and the (ultimately false) assumption that UIs would prevent this, was what made Maker choose not to add this to DAI. But, given the already large cost of calculating earner balances and voting epochs for these users, we think a little extra gas cost here will be worth the improved UX.

There is also a hack that can be used where the balance of the contract address is set to `type(uint240).max` without adjusting `totalSupply`, but this violates certain invariants for the token (e.g. `totalSupply()` is the sum of all `user.balanceOf()`). As a result, we don't recommend this alternative approach, but would be happy to adjust invariant tests to account for it.

One last note, given governance's ability to mint(), it is conceivable that they could be pressured to violate the minting intentions of the protocol in order to fix a user's balance after such a mistake. This is the ultimate headache Maker had to deal with, and ultimately lead to most, if not all, engineers on the project changing their position over time.

M^ZERO:

Prototech:

10.9 Inconsistent inflation due to rounding truncation

Context: EpochBasedInflationaryVoteToken.sol

Description: Due to the small total supply and necessity to round down user balances across epochs, users with smaller balances will be unable to inflate past their initial balance, even when participating in an epoch.

The following passing test reveals the discrepancy in balance inflation where user `_alice` is unable to inflate their balance despite participation.

```
// ttg/test/EpochBasedInflationaryVoteToken.t.sol

   function test_InflationTruncation() external {

       _warpToNextTransferEpoch();

       _vote.mint(_alice, 4);
       _vote.mint(_bob, 9);
       _vote.mint(_carol, 10);

       assertEq(_vote.balanceOf(_alice), 4);
       assertEq(_vote.balanceOf(_bob), 9);
       assertEq(_vote.balanceOf(_carol), 10);

       _warpToNextVoteEpoch();

       _vote.markParticipation(_alice);
       _vote.markParticipation(_bob);
       _vote.markParticipation(_carol);

       _warpToNextTransferEpoch();

       // Balances inflate upon the end of the epoch
       assertEq(_vote.balanceOf(_alice), 4);
```

```
        assertEq(_vote.balanceOf(_bob), 10);
        assertEq(_vote.balanceOf(_carol), 12);
    }
```

Recommendation: Review rounding properties for users with small balances and corresponding effects on `totalSupply` when user inflation is truncated. Add tests of expected behavior for users who have had their inflation rounded against to document expected behavior.

M^ZERO:

Prototech:

11. Informational Findings

11.1 transferFrom with insufficient balance leads to Panic: over/underflow

Context: EpochBasedVoteToken.sol#L272

Description:

If the `from` address does not have a sufficient balance to transfer to the `to` address, the operation will fail for `Panic: over/underflow`

Recommendation:

Consider a custom error for this operation.

M^ZERO:

Prototech:

11.2 ERC20Extended: Insufficient allowance for transferFrom results in Panic underflow

Context: ERC20Extended.sol#L86

Description:

`PowerToken` having an insufficient allowance set for a `transferFrom` call will result in a `Panic` underflow error. This may apply to other tokens using `ERC20Extended`.

Recommendation:

This may be a common revert in practice. Consider checking that the amount being transferred has an approval and provide a custom `InsufficientAllowance()` error.

M^ZERO:

Prototech:

11.3 can freeze deactivated minter

Context: MinterGateway.sol#L338-L344

Description:

Unlike other functions that affect active minters, `freezeMinter` does not ensure that the minter has not already been deactivated. There does not seem to be an impact to this, other than the strange `minterState` where the minter is both `isDeactivated == true` and has a value for their `frozenUntilTimestamp`.

Recommendation:

Consider adding `onlyActiveMinter` modifier to the freeze function so it reverts if called on a deactivated minter.

M^ZERO:

Prototech:

11.4 StandardGovernor.t.sol does not test setKey

Context: StandardGovernor.t.sol#L717-L721

Description:

The StandardGovernor Unit test only checks that `onlySelf` test fails, it does not have a happy path test like the others. The setKey function is missing from the Mock as well.

Recommendation:

M^ZERO:

Prototech:

11.5 Allow public reading of proposalFees in StandardGovernor

Context: StandardGovernor.sol#L180-L182

Description:

In order for the DistributionVault to get CashToken's to distribute, someone needs to call `sendProposalFeeToVault`, but there is no way on-chain to tell if there is a fee to send since `_proposalFees` is internal and does not have a public accessor. This could be a limiting UX for keepers to be able to distribute fees and maintain the system.

Recommendation:

Add a `getFee(proposalId_)` function that returns `_proposalFees[proposalId_].fee`

M^ZERO:

Prototech:

11.6 Reduce Duplicate Code to Prevent the Introduction of Bugs

Context:
ERC3009.sol#L322

ERC3009.sol#L334-L336

Description:

The `AuthorizationAlreadyUsed()` check on line `322` is the same as the `_revertIfAuthorizationAlreadyUsed()` check and should be replaced with `_revertIfAuthorizationAlreadyUsed()` to reduce the chance bugs are introduced by changes happening in one location and not the other in the future. This change will marginally increase security, but also increase gas costs.

Recommendation:

```
diff --git a/src/ERC3009.sol b/src/ERC3009.sol
index f988505..af5a8e5 100644
--- a/src/ERC3009.sol
+++ b/src/ERC3009.sol
@@ -319,7 +319,7 @@ abstract contract ERC3009 is IERC3009, StatefulERC712 {
     * @param  nonce_      Nonce of the authorization.
     */
    function _cancelAuthorization(address authorizer_, bytes32 nonce_) internal {
```

```
-         if (authorizationState[authorizer_][nonce_]) revert
AuthorizationAlreadyUsed(authorizer_, nonce_);
+         _revertIfAuthorizationAlreadyUsed(authorizer_, nonce_);

          authorizationState[authorizer_][nonce_] = true;
```

M^ZERO:

Prototech:

11.7 SignatureChecker.sol vulnerable to signature malleability

Context:
SignatureChecker.sol#L76-L87

SignatureChecker.sol#L42-L49

Description:

The OZ libraries were found vulnerable to a signature malleability attack because they allowed valid signatures for the same signed data. The MZero `SignatureChecker.sol` appears to implement the same validation pattern as the vulnerable OZ contracts. The vulnerability does not seem to affect the code at this time.

This is the code diff of the OZ bug fix: OZ patch
The advisory: here

The researcher's PoC: here

Recommendation: Recommend updating signature_checker.sol to prevent signature malleability.

M^ZERO:

Prototech:

11.8 Investigate MinterGateway and MToken updateIndex

Context: protocol/MinterGateway.sol#L405-L410

Description:

Calling `updateCollateral` on MinterGateway calls `updateIndex` which updates the MinterGateway's `latestIndex`, **`minterRate`** and `latestUpdateTimestamp`. It also

calls `MToken.updateIndex` which updates its `latestIndex`, `latestUpdateTimestamp` and `earnerRate`.

Further, their note says:

```
// NOTE: Given the current implementation of the mToken transfers and its rate
model, while it is possible for
//       the above mint to already have updated the mToken index if M was minted to
an earning account, we want
//       to ensure the rate provided by the mToken's rate model is locked in.
```

Recommendation:

There is a potential that this is susceptible to read-only reentrancy where calls that depend on an updated rate can be made in such a way to expose a vulnerability. We recommend evaluating this path to reasonably ensure it is not a problem.

M^ZERO:

Prototech:

11.9 MinterGateway does not validate that all signatures are in ascending order

Context: protocol/MinterGateway.sol#L996

Description:

If the threshold is met before checking all the signatures submitted, then subsequent signatures will not be checked that the validators were sorted correctly.

For instance if the threshold is 3, then this would pass as long as the first 3 signatures were valid.

```
address[] validators = [
 address(0),
 address(1),
 address(2),
 address(2),
 address(1),
 address(0)
];
```

There are no known impacts of this, but if signature order is ever relied on outside of this function or in integrations, this is important (as well as that some signatures could be invalid since all this requires is that the min threshold are valid).

Recommendation:

M^ZERO:

Prototech:

11.10 MinterGateway verifyValidatorSignatures could bail early

Context: protocol/MinterGateway.sol#L987

Description:

`UpdateCollateral` already verifies that `validators_`, `timestamps_`, `signatures_` all have the same length. Verification could bail earlier by ensuring there are at least as many signatures as the required threshold.

Recommendation:

Add the following line after `threshold_` is retrieved.

```
if (signatures.length < threshold_) revert NotEnoughValidSignatures(0, threshold_)
```

M^ZERO:

Prototech:

11.11 OnBehalf -> OnBehalfOf

Context:

- protocol/MToken.sol#L25
- protocol/MToken.sol#L103-L112
- protocol/MToken.sol#L163-L165
- protocol/IMToken.sol#L19
- protocol/IMToken.sol#L46-L50
- IMToken.sol#L88-L92
- IMToken.sol#L125-L126

Description:

Names like allowEarningOnBehalf() don't match other nomenclature

Recommendation:

Potentially change this and others to `allowEarningOnBehalfOf()`

M^ZERO:

Prototech:

12. Appendix

12.1 UML Diagrams for Reference

- MToken, MinterGateway, Registrar

12.2 Slither Detection Report

- Slither Detection Report File

12.3 List of Invariants and Descriptions

The following is an itemized list of invariants from the invariant test suite:

Distribution Vault Invariants	
Invariant	**Description**
DV_M1	The zeroToken in the distributionVault contract is correctly set.
DV_M2	The name of the distributionVault contract is "DistributionVault".
DV_M3	The CLOCK_MODE in the distributionVault contract is set to "mode=epoch".
DV_M4	The clock in the distributionVault contract matches the currentEpoch from PureEpochs.
DV_G1	* Due to the nature of Foundry's testing suite and several spots where we have loops, it was necessary to ensure that we weren't stuck in an infinite loop. If our modifier reverts due to a gas violation, we increment this value in the appropriate handler.
DV_B1	After a successful claim or claimBySig we add the Distribution Vault's value for get claimable for the call we just made. This should always be zero.
DV_B2	Each successful claim/claimBySig call must mark all relevant Epochs as claimed.
DV_B3	There should never be successful claims greater than the total tokens in successful distribution calls.
Emergency Governor Invariants	
Invariant	**Description**

EG_M1	The zeroGovernor address matches the zeroGovernor within the emergencyGovernor contract.
EG_G1	See DV gas violation description

MToken Invariants

Invariant	Description
M_M1	The mToken contract's decimals are set to 6.
M_M2	The ttgRegistrar in the mToken contract matches the address of _registrar.
M_M3	The minterGateway in the mToken contract matches the address of _minterGateway.
M_G1	See DV gas violation description
M_B1	The total supply equals the sum of total earning and total non-earning supply in the mToken contract.
M_B2_B3_B4	Invariants B2, B3, B4 should always hold true, but are based on related values. This tests each in an efficient way.
M_B2	The sum of all user balances is greater than or equal to the total supply minus the sum of all earners but less than or equal to the total supply.
M_B3	The sum of balances for non-earners matches the total non-earning supply.
M_B4	The sum of all earner balances is greater than or equal to the total earning supply minus the sum of all earners but less than or equal to the total earning supply.
M_P1	No contract other than the minterGateway can have a successful mint call.
M_A1	No successful transfer can decrement the allowance if it was set as max beforehand.
M_A2	A successful transfer when max allowance is not set should always decrement the allowance by the amount transferred.
M_A3	All successful permit calls should properly increment the nonce per the EIP 2612 standard.
M_Z1	All relevant, successful allowance calls should properly increment the nonce per the EIP 3009 standard.
M_Z2	All relevant, successful allowance calls should only succeed if the valid period is respected per the EIP 3009 standard.
M_Z3	No successful, relevant allowance call should change the relevant allowance of the actors per the EIP 3009 standard.

MZero Invariants

Name	Description
MZ_T1	An Invariant harness sanity check which checks to ensure that timestamps across various models are in sync.

Minter Gateway Invariants	
Name	**Description**
MG_M1	The mToken associated with the minterGateway is equal to the expected mToken address.
MG_M2	The mint ratio of the minterGateway is less than or equal to 10,000% or 1,000,000 bps.
MG_B1	Regardless of updateCollateral expiration, the collateral of minters should sum to equal the total collateral tracked in successful updateCollateral calls.
MG_B2	The sum of all users' pending collateral retrievals should equal the total collateral successfully retrieved via the proposeRetrieval function.
MG_B3	Validates that the sum of user balances equals the total minted amount.
MG_B4	The total value from all proposeMint calls should equal the total of all successful mintM, cancelMint calls + the pending mint calls.
MG_B5	For each user, the sum of their pending Collateral retrievals should equal their total Pending collateral retrieval.
MG_B6	If a user does not have expired collateral their collateralOf should equal their tracked collateral - their pending retrievals. If their collateral has expired, their collateralOf should always be 0
MG_B7	The pending mint proposals equal the sum of mintProposalOf for each actor.
MG_B8	The minter gateway cannot burn more tokens than it minted.
MG_G1	See DV gas violation description
MG_B9	The sum of active Minter's rawOwedM equals principalOfTotalActiveOwedM.
MG_N1	The minter gateway's retrievalNonce should equal the number of successful propose Retrieval calls.
MG_N2	The minter gateway's mintNonce should equal the number of successful proposeMint calls.
MG_T1	The penalizedUntilTimestamp is less than or equal to updateTimestamp.
Power Bootstrap Token Invariants	
Name	**Description**
PB_B1	The balance is equal to the past total supply of the power bootstrap token at timestamp 0.
PB_G1	See DV gas violation description
Power Token Invariants	
Name	**Description**
P_M1	The decimals of the powerToken contract is equal to 0.
P_B1	During Transfer Epochs, the sum of each user's power token balance must equal the

	totalSupply.
P_B2	Power Token totalSupply never deflates.
P_B3	Each subsequent Epoch must have as much or more PowerToken totalSupply.
P_B4	For each Epoch that PowerToken inflates, it inflates by 10%.
P_Z1	All relevant, successful allowance calls should properly increment the nonce per the EIP 3009 standard.
P_Z2	All relevant, successful allowance calls should only succeed if the valid period is respected per the EIP 3009 standard.
P_Z3	No successful, relevant allowance call should change the relevant allowance of the actors per the EIP 3009 standard.
P_Z4	Only the StandardGovernor can take privileged actions
P_Z5	There are no violations related to functions executing outside the expected voting epoch.
P_VD1	For all timepoints t < clock, getVotes(address(0)) and getPastVotes(address(0), t) SHOULD return 0.
P_VD2	Votes for an account equal the sum of balances of all accounts that delegate to it.
P_VD3	POWER totalVotingPower(delegates) equals POWER totalSupply(holders) + amountToAuction, at Transfer Epoch
P_VD4	For all accounts a, getPastVotes(a, t) MUST be constant after t<clock is reached.
P_VD5	For all accounts a, pastBalanceOf(a, t) MUST be constant after t < clock is reached.
P_VD6	For all accounts a, pastDelegates(a, t) MUST be constant after t < clock is reached.
P_G1	See DV gas violation description

Protocol Invariants

Invariant	Description
PROT_T1	The current timestamp is equal to a variable or constant named currentTimestamp.
PROT_T2	updateIndex always updates the latestUpdateTimestamp and ensures distribution of excess Owed M
PROT_S1	MinterGateway.updateIndex always calls MToken.updateIndex

Registrar Invariants

Invariant	Description
R_M1	The zeroGovernor in the registrar is equal to the specified _zeroGovernor.addr.
R_M2	The consistency between the keys and values stored in a registrar contract.
R_Z1	No permissioned calls can succeed unless they are from the Standard or Zero governor
R_G1	See DV gas violation description

Standard Governor Invariants	
Invariant	**Description**
SG_M1	The emergencyGovernor in standardGovernor is equal to the specified _emergencyGovernor.addr.
SG_M2	The vault in standardGovernor is equal to the specified _distributionVault.addr.
SG_M3	The zeroGovernor in standardGovernor is equal to the specified _zeroGovernor.addr.
SG_M4	The zeroToken in standardGovernor is equal to the specified _zeroToken.addr.
SG_M5	That maxTotalZeroRewardPerActiveEpoch in standardGovernor is equal to the specified _maxTotalZeroRewardPerActiveEpoch.
SG_M6	The registrar in standardGovernor is equal to the specified _registrar.addr.
SG_M7	The voteToken in standardGovernor is equal to the specified _powerToken.addr.
SG_Z1	There are no violations related to the governor address being zero in the context of a Standard Governor.
SG_G1	See DV gas violation description
TTG Invariants	
Invariant	**Description**
TTG_T1	Invariant harness sanity check which checks to ensure that timestamps across various models are in sync.
Zero Governor Invariants	
Invariant	**Description**
ZG_M1	_cashToken1 is allowed by the zeroGovernor using the isAllowedCashToken function.
ZG_G1	See DV gas violation description
Zero Token Invariants	
Invariant	**Description**
ZT_M1	The zeroToken has 6 decimals.
ZT_B1	The sum of individual balances equals the total supply of zeroToken.
ZT_P1	Only the Standard Governor can call mint
ZT_A1	No successful transfer can decrement the allowance if it was set as max beforehand
ZT_A2	A successful transfer when max allowance is not set should always decrement the allowance by the amount transferred
ZT_A3	All successful permit calls should properly increment the nonce per the EIP 2612 standard.

ZT_Z1	All relevant, successful allowance calls should properly increment the nonce per the EIP 3009 standard.
ZT_Z2	All relevant, successful allowance calls should only succeed if the valid period is respected per the EIP 3009 standard.
ZT_Z3	No successful, relevant allowance call should change the relevant allowance of the actors per the EIP 3009 standard.
ZT_VD1	Votes for address(0) are 0 for all timepoints before the current epoch.
ZT_VD2	The votes for an account equal the sum of balances of all accounts that delegate to it.
ZT_VD3	For each account (a != 0) and timestamp (t < clock), the delegated voting power recorded by getPastVotes matches the sum of the historical balances of accounts that delegated their votes to account (a) when the clock overtook timestamp t.
ZT_VD4	The delegated voting power for all accounts remains constant after each timestamp until the current clock time is reached.
ZT_VD5	The past token balances for all accounts remain constant after each timestamp until the current clock time is reached.
ZT_VD6	Past delegates for all accounts remain constant after reaching the current epoch.
ZT_G1	See DV gas violation description