

Hyperware DAO

Security Review

Cantina Managed review by:

Gerard Persoon, Lead Security Researcher
Slowfi, Security Researcher

August 4, 2025

Contents

1 Introduction **3**
1.1 About Cantina . . . 3
1.2 Disclaimer . . . 3
1.3 Risk assessment . . . 3
1.3.1 Severity Classification . . . 3

2 Security Review Summary **4**

3 Findings **5**
3.1 High Risk . . . 5
3.1.1 `_validateSignature()` returns wrong value . . . 5
3.1.2 `ECDSA.recover()` reverts on invalid signatures . . . 5
3.1.3 Voting Power Delegation From Expired Lock Incorrectly Increases Delegatee's Power 5
3.2 Medium Risk . . . 7
3.2.1 NFT Transfer Can Fail and Block Seller Payment . . . 7
3.2.2 Incorrect calculation of `getUserUnlockStamp()` . . . 8
3.2.3 Detection of last month is not working . . . 8
3.2.4 Optimization in `_updateMultiplier()` not always correct . . . 9
3.2.5 Function `state()` doesn't cover all changes . . . 9
3.2.6 Last minute changes to TBA before end of auction . . . 10
3.2.7 Auctioned TBAs could still be managed . . . 10
3.2.8 Last member can be removed from multisig . . . 10
3.2.9 Rounding Error Prevented Voting Power Decay for Long Locks . . . 10
3.3 Low Risk . . . 11
3.3.1 When is lock expired? . . . 11
3.3.2 `updateDelegationMultipliers()` has no `nonReentrant` modifier . . . 11
3.3.3 Same value bids are possible . . . 11
3.3.4 Bids of 0 tokens allowed . . . 11
3.3.5 Auctions can be monopolized . . . 12
3.3.6 Two definitions for `auction has ended` . . . 12
3.3.7 Missing Check to Restrict Multisig Members to EOAs . . . 12
3.3.8 `manageLock()` doesn't always check `MIN_LOCK_DURATION` . . . 13
3.3.9 Logic for `DEFAULT_REGISTRATION_NAMEHASH` is missing . . . 13
3.3.10 `initialize(address validator)` required for `mint` . . . 13
3.3.11 `totalSupply()` doesn't use `Votes::_getTotalSupply()` . . . 13
3.4 Gas Optimization . . . 14
3.4.1 Unnecessary Inheritance from `ERC20Permit` in Non-Transferable Token . . . 14
3.4.2 Unused `AccesControl` Inheritance . . . 14
3.4.3 Redundant Check on `extensionTime > MAX_SINGLE_EXTENSION` . . . 14
3.4.4 `_revokeRole()` not necessary . . . 15
3.4.5 `hypermap` could be made `immutable` in `TLZAuction` . . . 15
3.4.6 Code can be simplified in `cancelAuction()` and `resetAuction()` . . . 15
3.4.7 Redundant check in `HyperAccountMultisig::initialize()` . . . 16
3.4.8 `ECDSA.recoverCalldata()` can be used . . . 16
3.4.9 `_getSender()` no longer necessary . . . 16
3.4.10 `ReentrancyGuardTransientUpgradeable` could be used . . . 17
3.4.11 Extra inheritance level in `HyperGridNamespaceMinter` . . . 17
3.4.12 Unused `nonce` in `HyperAccountMinterUpgradable` . . . 17
3.4.13 `TimelockController` could be used . . . 17
3.4.14 `struct Auction` can be packed more efficient . . . 17
3.5 Informational . . . 18
3.5.1 Redundant Declaration of `EIP1271_MAGIC_VALUE` . . . 18
3.5.2 Hardcoded Minimum Auction Bid Period . . . 18
3.5.3 Misleading Error Name: `AuctionInProgress` Used After Finalization . . . 18
3.5.4 `getRawVotes()` can be called with `address(0)` . . . 19
3.5.5 `updateDelegationMultipliers()` could use `_max()` . . . 19
3.5.6 Sybils can circumvent sublinear approach . . . 19
3.5.7 `_timepoint` should always be month start . . . 20
3.5.8 `_updateMultiplier()` could use `getMultiplier()` . . . 20

3.5.9 Function `_getRemainingDuration()` can be simplified . . . 20
3.5.10 Redundant check for `type(IAccessControl).interfaceId` . . . 20
3.5.11 `tokenRegistry` can be `immutable` . . . 21
3.5.12 Function `_update()` not necessary . . . 21
3.5.13 Same `nameHash` can't be auctioned twice . . . 21
3.5.14 Bid of `minPrice` is not accepted . . . 21
3.5.15 Old comment about `assembly` . . . 21
3.5.16 Repeated bids are inefficient . . . 22
3.5.17 Minting `gHYPR` Without Enforcing `HYPR` Max Supply Cap . . . 22
3.5.18 Unnecessary Exposure of `approve()` Function in Non-Transferable Token . . . 22
3.5.19 `HyperAccountMinterUpgradable` `initialize()` function . . . 23
3.5.20 `supportsInterface()` not implemented everywhere . . . 23
3.5.21 `HyperGridNamespaceMinter` has two parts . . . 24
3.5.22 Unused error `NotSupportedYet()` . . . 24
3.5.23 Voting Power Decay Does Not Trigger Until Day 56 for Max Duration Locks . . . 24

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Hyperware aims to provide a comprehensive suite of tools and infrastructure to simplify the creation and deployment of decentralized applications, including those leveraging AI.

From Jul 16th to Jul 23rd the Cantina team conducted a review of protocol on commit hash 08b88f24. The team identified a total of **60** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	3	3	0
Medium Risk	9	8	1
Low Risk	11	11	0
Gas Optimizations	14	12	2
Informational	23	21	2
Total	**60**	**55**	**5**

3 Findings

3.1 High Risk

3.1.1 `_validateSignature()` returns wrong value

Severity: High Risk

Context: HyperAccountMultisig.sol#L294-L311, Account.sol#L18-L20

Description: `_validateSignature()` returns 0 in case invalid parameters are passed. However this means `SIG_VALIDATION_SUCCEEDED`, which means the multisig can be circumvented.

Recommendation: Change the code to:

```
  if (signature.length == 0 || signature.length % SIGNATURE_LENGTH != 0) {
-     return 0;
+     return SIG_VALIDATION_FAILED;
  }
```

Hyperware: Fixed in commit 82b9f0c2.

Cantina Managed: Fix verified.

3.1.2 `ECDSA.recover()` reverts on invalid signatures

Severity: High Risk

Context: HyperAccountMinterUpgradable.sol#L120-L141, HyperAccountMultisig.sol#L218, HyperAccountMultisig.sol#L238-L242

Description: In `HyperAccountMultisig::extractSigners()`, the call to `ECDSA.recover()` reverts on invalid signatures and because only one of the hashes can be valid per signature, a revert will always occur. So execution will never be allowed.

The same issue is present in `HyperAccountMinterUpgradable::_validateSignature()`.

Recommendation: Change `ECDSA.recover()` to `ECDSA.tryRecover()` or `ECDSA.tryRecoverCalldata()` in both locations.

Hyperware: Fixed in commits:

- 00a91f81.
- a9143b39.

Cantina Managed: Fix verified.

3.1.3 Voting Power Delegation From Expired Lock Incorrectly Increases Delegatee's Power

Severity: High Risk

Context: *(No context files were provided by the reviewer)*

Summary: Delegation from an expired lock improperly inflates voting power on the delegatee account due to stale multipliers not being updated.

Finding Description: When a user with a fully decayed or expired lock delegates their voting power to another user, the voting power transferred does not properly reflect the lock's decay. Instead, the `TokenRegistry` allows delegation of `gHYPR` tokens using outdated or non-zero multipliers, effectively granting the recipient more voting power than they should receive.

This breaks a key invariant: voting power must reflect locked duration. Allowing expired locks to still influence governance, even via delegation, violates the intent of the voting weight mechanism and enables disproportionate control.

The bug stems from early returns in the delegation logic that skip updating multipliers when the delegator's lock duration is zero.

Impact Explanation: This issue allows expired or soon-to-expire tokens to be used to artificially increase another user's voting power. In a low-participation or tightly contested governance vote, this could be

exploited to tip the result. It undermines the principle that governance influence should decrease over time and expire entirely when tokens become liquid.

This breaks the governance accounting guarantee that:

$$\text{votingPower}_{\text{delegatee}} \leq \sum \text{normalized power}_{\text{delegators}}$$

Likelihood Explanation: High likelihood in production environments, especially with long-term lockers or DAOs relying on periodic delegation. The bug was confirmed through a unit test showing a mismatch in the total system voting power before and after delegation from an expired lock.

Proof of Concept: The following test demonstrates that delegating voting power from an expired lock incorrectly increases the total system voting power. It compares two scenarios:

1. Two users each lock 10 ETH: user1 locks for `MIN_LOCK_DURATION`, and user2 locks for `MAX_LOCK_DURATION`.
2. Voting power for both users is measured.
3. A snapshot is created, and time advances by one year. Voting power is measured again after the lock decay.
4. The snapshot is restored, and this time user1 delegates their voting power to user2.
5. Voting power is measured again (before and after advancing one year).
6. The test then compares the total voting power in both scenarios to ensure they are consistent.

Despite no changes in lock duration or amount, the final assertion fails:

```
[0] VM::assertEq(269999998650000000000 [2.699e20], 269999998650000000000 [2.699e20]) [staticcall]
  ← [Return]
[0] VM::assertEq(249999998750000000000 [2.499e20], 259999998700000000000 [2.599e20]) [staticcall]
  ← [Revert] assertion failed: 249999998750000000000 != 259999998700000000000
← [Revert] assertion failed: 249999998750000000000 != 259999998700000000000
```

Code of the test:

```
function test_votingPowerTwoAccounts_OneYearPass_DelegatingVoting() public {
    vm.warp(block.timestamp + 365 days); // Avoid timestamp 0 edge case

    // Scenario setup
    vm.startPrank(user1);
    hyper.approve(address(registry), type(uint256).max);
    registry.manageLock(10 ether, MIN_LOCK_DURATION);
    vm.stopPrank();

    vm.startPrank(user2);
    hyper.approve(address(registry), type(uint256).max);
    registry.manageLock(10 ether, MAX_LOCK_DURATION);
    vm.stopPrank();

    // Scenario 1 - no delegation
    uint256 votesUser1_before = gHyper.getVotes(user1);
    uint256 votesUser2_before = gHyper.getVotes(user2);

    uint256 snapshotId = vm.snapshot();
    vm.warp(block.timestamp + 365 days); // Advance one year

    uint256 votesUser1_after = gHyper.getVotes(user1);
    uint256 votesUser2_after = gHyper.getVotes(user2);

    // Scenario 2 - with delegation
    vm.revertTo(snapshotId);

    vm.startPrank(user1);
    gHyper.delegate(user2);
    vm.stopPrank();

    uint256 votesUser1_before_scenario2 = gHyper.getVotes(user1);
    uint256 votesUser2_before_scenario2 = gHyper.getVotes(user2);

    vm.warp(block.timestamp + 365 days);
```

```
    uint256 votesUser1_after_scenario2 = gHyper.getVotes(user1);
    uint256 votesUser2_after_scenario2 = gHyper.getVotes(user2);

    assertEq(
        votesUser1_before + votesUser2_before,
        votesUser1_before_scenario2 + votesUser2_before_scenario2
    );

    assertEq(
        votesUser1_after + votesUser2_after,
        votesUser1_after_scenario2 + votesUser2_after_scenario2
    );
}
```

This test fails because after one year, user1's lock has expired and their voting power should be zero. Delegating voting power from that expired state should not increase user2's power. However, due to the `TokenRegistry` not properly updating or invalidating expired voting multipliers during delegation, the system incorrectly allows user2 to gain more voting power than was present in scenario 1. This reveals a flaw in how the registry updates normalized weight during delegation, breaking the conservation of total voting power.

Recommendation: Ensure that delegation always updates the multiplier, even for expired locks. Also consider:

- Validating that expired locks yield zero voting power, even if delegated.
- Adding an invariant test to ensure conservation of voting power across delegation.

Hyperware: Fixed in commit db3d810a. To trace this bug it was added some logging in a not-for-merge branch on PR 105

The issue was that we were returning early and not updating all multipliers when delegating. The total token amount changes (i.e. the actual gHYPR tokens) but we were not renormalizing anything after duration == 0, so those values were larger than expected.

Cantina Managed: Fix verified.

3.2 Medium Risk

3.2.1 NFT Transfer Can Fail and Block Seller Payment

Severity: Medium Risk

Context: TLZAuction.sol#L564-L568

Description: In `TLZAuction.sol` at line 564, the auction finalization logic attempts to deliver the NFT to the auction winner using `safeTransferFrom`:

```
IERC721(address(_hypermap)).safeTransferFrom(address(this), winner, mintedNode);
```

This call will revert if the `winner` address is a contract that does **not implement** `IERC721Receiver.onERC721Received()`, such as a minimal proxy or non-upgradeable logic contract. When this happens:

- The NFT remains stuck in the auction contract.
- The seller's payment logic, which follows immediately after the transfer, is never executed.
- Neither party can recover their assets through standard flows.

This is a critical liveness issue.

Recommendation: Consider addressing the edge case with one or more of the following design improvements:

- Wrap the `safeTransferFrom` in a `try/catch`, and on failure:
  - Mark the NFT as pending.
  - Enable a retry via a `claimNFT()` function.

- Use `transferFrom` instead of `safeTransferFrom`, though this skips important ERC721 compliance checks.
- Decouple the NFT delivery logic from `claimTlz`, similar to the seller's pull-based payment.
- Add a pre-bid check to ensure the bidder is a contract that implements `onERC721Received`.

Hyperware: Fixed in commits:

- 96a68a4c.
- b4287112.

Cantina Managed: Fix verified.

3.2.2 Incorrect calculation of `getUserUnlockStamp()`

Severity: Medium Risk

Context: TokenRegistry.sol#L371-L374, HyperGovernorToken.sol#L238-L243, HyperGovernorToken.sol#L267-L271

Description: Found by the project. Protocol delegation bug with the next scenario:

- User A locks 2 HYPR for 2 years.
- User B locks 3 HYPR for 1 year.
- User C locks 7 HYPR for 3 years.
- User A delegates to B.
- User B delegates to C.

When B delegates to C, when transferring voting power via `userUnlockStamp()` the unlock stamp for B is computed incorrectly. The function looks at the max of `unlock time` and `delegated unlock time`: the duration will incorrectly be returned as 2 years instead of 1 year.

Recommendation: Consider changing the code to:

```
  function getUserUnlockStamp(address _account) external view returns (uint256) {
      // ...
-     return _max($._userUnlockStamps[_account], $._delegatedUnlockStamps[_account]);
+     return $._userUnlockStamps[_account];
  }
```

Also adding a function `getUserOrDelegatedUnlockStamp()` that returns the max value of `unlock time` and `delegated unlock time`. This should be used in function `_getUserVotesAt()`.

Hyperware: Fixed in commit 885c384e.

Cantina Managed: Fix verified.

3.2.3 Detection of last month is not working

Severity: Medium Risk

Context: TokenRegistry.sol#L663-L679, TokenRegistry.sol#L756-L772

Description: The function `_processMonthlyVotes()` and `_processMonthlyVotesTransfer()` do a loop over `TOTAL_MONTHS`. This loop should end when `_unlockStamp` is reached. However the end isn't correctly detected due to `duration = _max(duration, MIN_REGISTRATION_DURATION)`. This means action are taken after the last month which is inefficient and make the administration incorrect.

Recommendation: Detect when `_unlockStamp` is reached.

Hyperware: Solved in commits:

- ccd67564.
- 1d474c48.

Cantina Managed: Fix verified.

3.2.4 Optimization in `_updateMultiplier()` not always correct

Severity: Medium Risk

Context: TokenRegistry.sol#L696-L720

Description: Function `_updateMultiplier()` has some optimization in case `(currentMultiplier == 0 && _votesBefore == 0)`. However this isn't correct when `_op == _subtract`, because the logic is different than in the `else` part. This could lead to errors in the calculations.

Recommendation: Consider removing the optimization:

```
- if (currentMultiplier == 0 && _votesBefore == 0) {
-     uint256 newMultiplier = _movedMonthlyVotes * MULTIPLIER_PRECISION / _votesAfter;
-     _setMultiplier(_account, _timepoint, newMultiplier);
- } else {
      uint256 monthlyVotes = currentMultiplier * _votesBefore / MULTIPLIER_PRECISION;
      uint256 newMultiplier = _op(monthlyVotes, _movedMonthlyVotes) * MULTIPLIER_PRECISION / _votesAfter;
      _setMultiplier(_account, _timepoint, newMultiplier);
- }
```

Hyperware: Fixed in commit f79b2ba6.

Cantina Managed: Fix verified.

3.2.5 Function `state()` doesn't cover all changes

Severity: Medium Risk

Context: TokenBoundMech.sol#L16-L18

Description: According to eip-6551, an account should have a function `state()` that is modified each time the account changes state.

The current implementation is only used in relation to account abstraction, in other situations it isn't updated.

The reason for this function is to detect last minute changes when selling an NFT, where these changes could make the NFT worth less.

Recommendation: Consider updating `state()` for every change on the token bound account:

- The number of tokens bound to the TBA as well as the duratation (might require additional administration).
- Additional childeren minted (or childeren of childeren).
- Updates of notes (or updates of notes of childeren or childeren of childeren).
- Upgrade of any TBA self or of the contracts of childeren.
- Usage of `_exec()`.
- Receival of funds in `receive()`.

Hyperware: Partly fixed. Unfortunately we cannot do a full-featured solution here as far as I can tell, but we can do better than nothing:

- Do not update `state()` on things that are not owner actions. Reasoning here is that it becomes a griefing vector (Eve the spammer can send a spam token to change `state()` and kill a sale). Receiving tokens is not something we're trying to prevent anyways;
- Do update state() on owner actions: this is the `_exec()` increment. So, e.g., if owner sends a token that belongs to the TBA, that can potentially kill sale.

Unfortunately it is still not foolproof: if Evan is the owner, he can have previously approved a `transferFrom()` and then, after the `state()` is recorded, transfer tokens out.

Documented the known weaknesses with this implementation. For a trustless marketplace we'll implement an offchain commitment scheme of some kind that doesn't rely on `state()`.

Hyperware: Fixed in commit f791d77b.

Cantina Managed: Fix verified.

3.2.6 Last minute changes to TBA before end of auction

Severity: Medium Risk

Context: TLZAuction.sol#L529

Description: Last minute changes could potentially be made to a TBA, just before an auction end. Then the buyer would buy a TBA that might be worth less.

EIP-6551 has defined an approach to mitigate that, via the function `state()` that is modified each time the account changes state.

Recommendation: Consider making use of the function `state()`. For example by storing the `state` at the start of the auction, checking it at all bids and checking it at `claimTlz()`. If the `state` has changed, then return the tokens to the bidder.

Hyperware: Fixed in commit c72363d6.

Cantina Managed: Fix verified.

3.2.7 Auctioned TBAs could still be managed

Severity: Medium Risk

Context: TLZAuction.sol#L310, TLZAuction.sol#L331

Description: Auctioned TBAs are transfered to the `TLZAuction` contract to prevent furter actions on the TBA. However, depending on the type of TBA some other parties might still be able to manage the TBA. For example when the `HyperAccountMultisig` is used. There is no guarantee that these parties will transfer their rights to the new owner.

Recommendation: Consider enforcing that additional rights are transferred. This might require additional functions in the TBA to verify this.

Hyperware: Acknowledged. Out-of-scope for this contract since it is scoped for use by us to auction brand-new TLZs.

Cantina Managed: Acknowledged.

3.2.8 Last member can be removed from multisig

Severity: Medium Risk

Context: HyperAccountMultisig.sol#L177-L181

Description: Function `removeMember()` allows removing the last member. This would make the multisig unsuable. Additionally `initialize()` also doesn't allow 0 members.

Recommendation: Consider enforcing that at least one member is present.

Hyperware: Fixed in commit e30074da.

Cantina Managed: Fix verified.

3.2.9 Rounding Error Prevented Voting Power Decay for Long Locks

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: Found by the project. Protocol delegation bug. The constant `C` used in the linear voting power calculation caused `duration / C` to round down early on, especially for `MAX_LOCK_DURATION`. This resulted in voting power **not decaying** for some months, breaking expected behavior.

Recommendation: Use a smaller `C` value to ensure monthly voting power decay occurs even for long-term locks, as implemented in the referenced fix.

Hyperware: Fixed on commit ea73e9fd.

Cantina Managed: Fix verified.

3.3 Low Risk

3.3.1 When is lock expired?

Severity: Low Risk

Context: TokenRegistry.sol#L103-L116, TokenRegistry.sol#L229-L233, TokenRegistry.sol#L259-L267, TokenRegistry.sol#L332-L345

Description: The situation where `$._userUnlockStamps[_account] == block.timestamp` is interpreted in different ways. According to `isLockExpired()`, the lock is not expired. However the other functions `manageLock()`, `calculateNewLockDuration()` and `transferRegistration()` do consider it expired. This is inconsistent.

Recommendation: Doublecheck the desired behaviour and adapt the code.

Hyperware: Fixed in commit e656d035.

Cantina Managed: Fix verified.

3.3.2 `updateDelegationMultipliers()` has no `nonReentrant` modifier

Severity: Low Risk

Context: TokenRegistry.sol#L103, TokenRegistry.sol#L174, TokenRegistry.sol#L332-L334, TokenRegistry.sol#L385

Description: Several functions in this contract have a `nonReentrant` modifier, for example `withdraw()`. Suppose there is a reentrancy possibility within `withdraw()`, then this could be used to call `HyperGovernorToken::delegate()` which calls `updateDelegationMultipliers()`. This could interfere with `withdraw()`.

Recommendation: Although we see no realistic reentrancy possibility, this is a low cost safety precaution. Consider adding a `nonReentrant` modifier to `updateDelegationMultipliers()`.

Hyperware: Fixed in commit 0a0386c9.

Cantina Managed: Fix verified.

3.3.3 Same value bids are possible

Severity: Low Risk

Context: TLZAuction.sol#L274-L285

Description: If the initial `highestBid` is (very)low, then `increaseAmount` can be rounded down to 0. This allows same value bids to be done, which is unwanted in an auction.

Additionally if `_bidIncreasePercentage` is set to 0, `increaseAmount` will also be 0, with the same result.

Recommendation: Consider doing the following:

- Require a minimum value for bids.
- Disallow `_bidIncreasePercentage` to be 0.
- For safety doublecheck `increaseAmount` isn't 0.

Hyperware: Fixed in commit 69c371a9.

Cantina Managed: Fix verified.

3.3.4 Bids of 0 tokens allowed

Severity: Low Risk

Context: TLZAuction.sol#L260-L267, TLZAuction.sol#L310, TLZAuction.sol#L323-L325, TLZAuction.sol#L410, TLZAuction.sol#L437, TLZAuction.sol#L461, TLZAuction.sol#L479-L480

Description: The `minPrice` for an auction could be 0, after a `resetAuction()`. Assuming `_validateBidAmount()` is updated to allow bids equal `minPrice`, then a `_newBidAmount` could be 0. Most tokens allow a transfer of 0 tokens, however some tokens might not allow this, see weird-erc20 Also allowing a bid of 0 is not logical in an auction.

Recommendation: Consider requiring a minimum amount for a bid. This should be checked in both `createAuction()` and `resetAuction()`.

Hyperware: Fixed in commit 846b7126.

Cantina Managed: Fix verified.

3.3.5 Auctions can be monopolized

Severity: Low Risk

Context: TLZAuction.sol#L492-L502

Description: Someone who is determined to win an auction can use two accounts and quickly bid with alternate accounts until `MAX_AUCTION_EXTENSIONS` is reached and thus monopolize the auction.

Note: depending on `bidIncreasePercentage` this might be economically infeasible.

Recommendation: Consider doing one of the following:

- Use blind auctions.
- Still allow bidding even after the time can't be extended any more.

Note: this leads to possibility of last-minute bidding, but this is more-or-less the same situation as last-minute bidding to be the final extension.

Hyperware: Fixed in commit 62e22d31.

Cantina Managed: Fix verified.

3.3.6 Two definitions for `auction has ended`

Severity: Low Risk

Context: TLZAuction.sol#L249-L254, TLZAuction.sol#L529, TLZAuction.sol#L542-L545

Description: Edge case: when `block.timestamp == auction.endAt`, then according to `_checkAuctionActive()` its `AuctionExpired`, however `claimTlz()` considers it `AuctionInProgress`.

Recommendation: Consider using the same defintion in all locations.

Hyperware: Fixed in commit 25c812d1.

Cantina Managed: Fix verified.

3.3.7 Missing Check to Restrict Multisig Members to EOAs

Severity: Low Risk

Context: HyperAccountMultisig.sol#L319-L325

Description: In `HyperAccountMultisig.sol` at line 319, the `_addMember` function adds a new signer to the multisig:

```
function _addMember(address _member, MainStorage storage _storage) internal {
```

It is implied that `_member` should be an Externally Owned Account (EOA), not a contract. However, there is currently no check enforcing that condition. This could allow contracts to be added as multisig members, which may lead to unexpected behavior or reentrancy risks depending on how those contracts behave during signature validation. Although Solidity permits contracts to appear as EOAs during construction (i.e., `code.length == 0`), this edge case is difficult to exploit in this context.

Recommendation: Consider enforcing an EOA-only policy with:

```
require(_member.code.length == 0, "Must be EOA");
```

This protects the multisig from being compromised by contracts with complex behaviors or upgradeable logic. If contract-based signers are intended in the future, this restriction can be revised or made configurable.

Hyperware: Fixed on commit 5c167266.

Cantina Managed: Fix verified.

3.3.8 `manageLock()` doesn't always check `MIN_LOCK_DURATION`

Severity: Low Risk

Context: TokenRegistry.sol#L102-L103, TokenRegistry.sol#L120-L125, TokenRegistry.sol#L596-L604

Description: In function `manageLock()`, in the situation where `_processLock()` is called, there isn't an explicit check that `_duration >MIN_LOCK_DURATION`. However the comment suggest this check should be done.

Note: `_processLockExtension()` does check this via `_validateExtensionDuration()`.

Recommendation: Consider enforcing the `MIN_LOCK_DURATION` or alternatively change the comment.

Hyperware: Fixed in commit 44557c86.

Cantina Managed: Fix verified.

3.3.9 Logic for `DEFAULT_REGISTRATION_NAMEHASH` is missing

Severity: Low Risk

Context: TokenRegistry.sol#L139

Description: A previous version of the code hade the following code, but that isn't present in the current version.

```
emit BindCreated(msg.sender, DEFAULT_REGISTRATION_NAMEHASH, _amount, 0);
// ...
emit BindAmountIncreased(msg.sender, DEFAULT_REGISTRATION_NAMEHASH, _amount, 0);
// ...
$._userBinds[msg.sender].add(DEFAULT_REGISTRATION_NAMEHASH);
// ...
$._userBinds[msg.sender].remove(DEFAULT_REGISTRATION_NAMEHASH);
```

Recommendation: Doublecheck the usefulness and the statements.

Hyperware: Fixed in commit 2e27f097.

Cantina Managed: Fix verified.

3.3.10 `initialize(address validator)` required for `mint`

Severity: Low Risk

Context: HyperGridNamespaceMinter.sol#L144-L162, HyperGridNamespaceMinter.sol#L347-L360, HyperGridNamespaceMinter.sol#L390, HyperGridNamespaceMinter.sol#L413-L414

Description: The function `HyperGridNamespaceMinter::_mint()` only works if the `implementation` contract (e.g. the TBA) has a function `initialize(address validator)`. However this currently isn't checked.

Recommendation: Consider checking the `implementation` contract is compatible in `initialize(address childImplementation, ...)` and `setChildImplementation()`. This can be done via `supportsInterface()`.

Note: `HyperGridNamespaceMinter()` currently doesn't have `supportsInterface()`, also see finding "`supportsInterface()` not implemented everywhere"

Hyperware: Fixed in commit e4a9de2a.

Cantina Managed: Fix verified.

3.3.11 `totalSupply()` doesn't use `Votes::_getTotalSupply()`

Severity: Low Risk

Context: HyperGovernorToken.sol#L177-L180

Description: `HyperGovernorToken::totalSupply()` uses `ERC20::totalSupply()`. However it more logical to use `Votes::_getTotalSupply()`. This is because `Votes::_getTotalSupply()` uses `_totalCheckpoints[]` similar to `_getPastTotalSupply()` /`getPastTotalSupply()`.

Also `ERC20::totalSupply()` returns all the tokens but this might be different from the number of votes. For example if tokens are delegated to `address(0)`, they don't count anymore as votes, althought this is prevented in the current code, there might be other edge cases.

Recommendation: Consider using `Votes::_getTotalSupply()`.

Hyperware: Fixed in commit dbe4e5fd.

Cantina Managed: Fix verified.

3.4 Gas Optimization

3.4.1 Unnecessary Inheritance from `ERC20Permit` in Non-Transferable Token

Severity: Gas Optimization

Context: HyperGovernorToken.sol#L23

Description: The `HyperGovernorToken` contract inherits from `ERC20Permit` (line 23), which provides off-chain signature-based approvals (EIP-2612) for spending allowances. However, if this token is not transferable, permit functionality has no practical effect. It also exposes unused functionality that could confuse integrators or increase the attack surface.

Recommendation: Consider removing the inheritance from `ERC20Permit` unless permit-based delegation or future transferability is explicitly planned and justified.

Hyperware: Fixed on commit 057459fc.

Cantina Managed: Fix verified.

3.4.2 Unused `AccesControl` Inheritance

Severity: Gas Optimization

Context: HyperGovernorToken.sol#L54

Description: In `HyperGovernorToken.sol` at line 54, the contract grants `DEFAULT_ADMIN_ROLE` to the `admin` address:

```
_grantRole(DEFAULT_ADMIN_ROLE, admin);
```

However, the contract does not appear to use `AccessControl` roles for any permission checks or role-restricted functionality. This role assignment is therefore unused and may be misleading to auditors or future contributors.

Recommendation: Consider removing the `AccessControl` inheritance and the unused `DEFAULT_ADMIN_ROLE` assignment unless you plan to enforce role-based access control in the future.

Hyperware: Fixed on commit f189f169.

Cantina Managed: Fix verified.

3.4.3 Redundant Check on `extensionTime > MAX_SINGLE_EXTENSION`

Severity: Gas Optimization

Context: TLZAuction.sol#L497-L498

Description: In `TLZAuction.sol` at line 497, the code includes a conditional cap:

```
if (extensionTime > MAX_SINGLE_EXTENSION) {
    extensionTime = MAX_SINGLE_EXTENSION;
}
```

However, this condition is already enforced in the `_setAuctionIncreaseTime()` function when the value is initially set. Assuming the implementation behind the proxy does not change, this check is effectively unreachable and redundant.

Recommendation: Consider removing the redundant check to simplify the code, reduce bytecode size, and eliminate confusion. If upgradability is a concern, document the assumption or enforce safety through external validation instead.

Hyperware: Fixed on commit 71d92869.

Cantina Managed: Fix verified.

3.4.4 `_revokeRole()` not necessary

Severity: Gas Optimization

Context: TLZAuction.sol#L57, TLZAuction.sol#L78-L79

Description: The function `TLZAuction::initialize()` calls `_revokeRole()` for `msg.sender`. However this isn't necessary because `msg.sender` hasn't been added to the admin role.

Recommendation: Consider removing the call to `_revokeRole()`.

Hyperware: Fixed in commit bdb2a765.

Cantina Managed: Fix verified.

3.4.5 `hypermap` could be made `immutable` in `TLZAuction`

Severity: Gas Optimization

Context: TLZAuction.sol#L57, TLZAuction.sol#L83, TLZAuctionStorage.sol#L22-L23

Description: Function `TLZAuction:initialize()` stores `_hypermap` in a storage variable. As the address of the Hypermap isn't expected to change (because it is upgradable) and assuming it is available before the deployment of `TLZAuction`, it can be made `immutable`.

Recommendation: Consider making `hypermap` an `immutable` variable and configure is in the constructor of `TLZAuction`.

Hyperware: Acknowledged.

Cantina Managed: Acknowledged.

3.4.6 Code can be simplified in `cancelAuction()` and `resetAuction()`

Severity: Gas Optimization

Context: TLZAuction.sol#L363-L379, TLZAuction.sol#L410-L432

Description: Both the function `cancelAuction()` and `resetAuction()` enforce `auction.highestBidder == address(0)`. If that is the case then `auction.endAt` is also 0, because both are set at the same time in `makeBid()`. So the code can be simplified to make it more readable and save some gas.

Recommendation: Consider changing the code to:

```
  function cancelAuction(bytes32 _nameHash) external nonReentrant {
      // ...
-     if (auction.endAt == 0 && block.timestamp < auction.startAt + RESET_COOLDOWN_PERIOD) {
+     if (block.timestamp < auction.startAt + RESET_COOLDOWN_PERIOD) {
          revert InCoolDownPeriod(_nameHash);
      }
      // ...
  }
```

```
  function resetAuction(bytes32 _nameHash, uint256 _minPrice) external {
      // ...
-     if (auction.endAt != 0 && block.timestamp < auction.endAt) {
-         revert AuctionInProgress(_nameHash);
-     }
-     uint256 cooldownStart = auction.endAt == 0 ? auction.startAt : auction.endAt;
-     if (block.timestamp < cooldownStart + RESET_COOLDOWN_PERIOD) {
+     if (block.timestamp < auction.startAt + RESET_COOLDOWN_PERIOD) {
          revert InCoolDownPeriod(_nameHash);
      }
      // ...
  }
```

Hyperware: Fixed in commit 086c86d3.

Cantina Managed: Fix verified.

3.4.7 Redundant check in `HyperAccountMultisig::initialize()`

Severity: Gas Optimization

Context: HyperAccountMultisig.sol#L130-L134, HyperAccountMultisig.sol#L130-L141, HyperAccountMultisig.sol#L346-L352

Description: Function `initialize()` does some checks on `_requiredSignatures`, however these checks are also done by `_changeRequiredSignatures()`, so the checks are redundant.

Recommendation: Consider removing the checks from `initialize()`.

Hyperware: Fixed in commit 0e52464c.

Cantina Managed: Fix verified.

3.4.8 `ECDSA.recoverCalldata()` can be used

Severity: Gas Optimization

Context: HyperAccountMultisig.sol#L218, HyperAccountMultisig.sol#L234-L242

Description: Function `extractSigners()` uses the `memory` version of `ECDSA.recover`. However `calldata` specific versions also exist which save some gas.

Recommendation: Consider changing the code to:

```
- bytes memory singleSig = _signature[offset:offset + SIGNATURE_LENGTH];
+ bytes calldata singleSig = _signature[offset:offset + SIGNATURE_LENGTH];
  // ...
- signers[...] = ECDSA.recover(..., singleSig);
+ signers[...] = ECDSA.recoverCalldata (..., singleSig);
```

Hyperware: Acknowledged.

Cantina Managed: Acknowledged.

3.4.9 `_getSender()` no longer necessary

Severity: Gas Optimization

Context: HyperAccountMinterUpgradable.sol#L87-L116

Description: Function `_getSender()` was implemented to support `_mintBySignature()`. However because `_mintBySignature()` is removed, it is no longer necessary.

Recommendation: Consider removing the function `_getSender()`.

Hyperware: Fixed in commit e2b938d8.

Cantina Managed: Fix verified.

3.4.10 `ReentrancyGuardTransientUpgradeable` could be used

Severity: Gas Optimization

Context: HyperAccountMultisig.sol#L7

Description: Contract `HyperAccountMultisig` imports `ReentrancyGuardUpgradeable`. However a more gas efficient version also exists.

Recommendation: Consider using `ReentrancyGuardTransientUpgradeable`.

Hyperware: Fixed in commit 00518c10.

Cantina Managed: Fix verified.

3.4.11 Extra inheritance level in `HyperGridNamespaceMinter`

Severity: Gas Optimization

Context: HyperAccountMinter.sol#L6-L8, HyperGridNamespaceMinter.sol#L26

Description: Contract `HyperGridNamespaceMinter` inherits from `HyperAccountMinter`,which inherits from `HyperAccountMinterUpgradable`. However most other contracts directly inherit from `HyperAccountMinterUpgradable`.

Recommendation: Consider directly inheriting from `HyperAccountMinterUpgradable`.

Hyperware: Fixed in commit 2f9c9b00.

Cantina Managed: Fix verified.

3.4.12 Unused `nonce` in `HyperAccountMinterUpgradable`

Severity: Gas Optimization

Context: HyperAccountMinterUpgradable.sol#L37

Description: In contract `HyperAccountMinterUpgradable` the `nonce` mapping is no longer used.

Recommendation: Consider removeing the `nonce` mapping.

Note: be aware of potential old values of the `nonce` mapping when new variables are added in `HyperAccountMinterUpgradableStorage` in the future in combination with upgrades.

Hyperware: Fixed in commit 0a2c3f39.

Cantina Managed: Fix verified.

3.4.13 `TimelockController` could be used

Severity: Gas Optimization

Context: 03-DeployDAO.s.sol#L25, TimelockControllerUpgradeableWithInit.sol#L7-L14

Description: The constructor of `TimelockControllerUpgradeableWithInit` effectely converts this into the nonupgradable version: TimelockController

Recommendation: Consider directly using `TimelockController`.

Hyperware: Fixed in commit ca0a7a88.

Cantina Managed: Fix verified.

3.4.14 `struct Auction` can be packed more efficient

Severity: Gas Optimization

Context: ITLZAuction.sol#L5-L15

Description: The `struct Auction` can be packed more efficient.

Recommendation: Consider moving the `bool completed` directly after an address.

Hyperware: Fixed in commit b53a778b.

Cantina Managed: Fix verified.

3.5 Informational

3.5.1 Redundant Declaration of `EIP1271_MAGIC_VALUE`

Severity: Informational

Context: HyperAccountMultisig.sol#L37

Description: The constant `EIP1271_MAGIC_VALUE` is redefined in `HyperAccountMultisig.sol` at line 37, despite already being declared as `EIP1271_MAGICVALUE` in the inherited `Mech` contract.

Inheritance path:

```
HyperAccountMultisig
  -> HyperAccountMinterUpgradable
    -> ERC721TokenBoundMech
      -> TokenBoundMech
        -> Mech
```

This duplication adds unnecessary code and increases the risk of inconsistency if one constant is modified independently in the future.

Recommendation: Consider removing the local declaration in `HyperAccountMultisig.sol` and relying on the inherited `EIP1271_MAGICVALUE` constant from `Mech.sol`.

Hyperware: Fixed on commit f13dab24.

Cantina Managed: Fix verified.

3.5.2 Hardcoded Minimum Auction Bid Period

Severity: Informational

Context: TLZAuction.sol#L587

Description: In `TLZAuction.sol` at line 587, the condition uses a hardcoded duration:

```
if (_defaultAuctionBidPeriod < 1 days) {
```

This check enforces a minimum bid period, but the `1 days` value is not declared as a named constant, making it less consistent with the rest of the codebase and harder to maintain or audit.

Recommendation: Consider defining `1 days` as a named constant `MIN_AUCTION_BID_PERIOD` to improve clarity and maintainability. This aligns with best practices for parameter configuration.

Hyperware: Fixed on commit f4a0b488.

Cantina Managed: Fix verified.

3.5.3 Misleading Error Name: `AuctionInProgress` Used After Finalization

Severity: Informational

Context: TLZAuction.sol#L372-L374

In `TLZAuction.sol` at line 372 (and similarly in `resetAuction`), the error `AuctionInProgress` is triggered when `auction.highestBidder != address(0)`:

```
if (auction.highestBidder != address(0)) {
    revert AuctionInProgress(_nameHash);
}
```

However, this condition may also be true after an auction has already been **finalized**, making the error message misleading. The name `AuctionInProgress` implies that bidding is currently allowed, which is not necessarily the case.

Recommendation: Consider renaming the error to better reflect its actual purpose (e.g., `AuctionAlreadyExists`, `AuctionAlreadyFinalized`, or `AuctionLocked`). Alternatively, differentiate between "active" and "finalized" states more explicitly in the error logic.

Hyperware: Fixed on commit 7107ee08.

Cantina Managed: Fix verified.

3.5.4 `getRawVotes()` can be called with `address(0)`

Severity: Informational

Context: TokenRegistry.sol#L103, TokenRegistry.sol#L130-L134

Description: Function `manageLock()` calls `getRawVotes()` with `address(0)`, if its the first call by a `msg.sender`. However its more logical to check the `getRawVotes()` of the `msg.sender`.

In practice this results in the same value because `address(0)` can't receive `gHyper` tokens and can't be delegated to. However its more clear to use the correct `delegatee`.

Recommendation: Consider changing the code to:

```
-  uint256 votesReceiverBefore = $.gHypr.getRawVotes(delegatee);
  if (delegatee == address(0)) {
      delegatee = msg.sender;
  }
+  uint256 votesReceiverBefore = $.gHypr.getRawVotes(delegatee);
```

Hyperware: Fixed in commit 23ad7fca.

Cantina Managed: Fix verified.

3.5.5 `updateDelegationMultipliers()` could use `_max()`

Severity: Informational

Context: TokenRegistry.sol#L385-L402, TokenRegistry.sol#L788-L792

Description: Some code in `updateDelegationMultipliers()` can be simplified.

Recommendation: Consider changing the code to:

```
  $._delegatedUnlockStamps[_dst] =
- $._delegatedUnlockStamps[_dst] > _unlockStamp ? $._delegatedUnlockStamps[_dst] : _unlockStamp;
+ _max($._delegatedUnlockStamps[_dst], _unlockStamp);
```

Hyperware: Fixed in commit fd514735.

Cantina Managed: Fix verified.

3.5.6 Sybils can circumvent sublinear approach

Severity: Informational

Context: TokenRegistry.sol#L418-L422

Description: An inherent issue of the sublinear approach is token owners can split their balance over multiple accounts (also known as sybils) and then lock multiple times. This reduces the penalty.

Recommendation: Just important to be aware of this.

Hyperware: Acknowledged.

Cantina Managed: Acknowledged.

3.5.7 `_timepoint` should always be month start

Severity: Informational

Context: TokenRegistry.sol#L663-L681, TokenRegistry.sol#L696-L698, TokenRegistry.sol#L756-L774

Description: The parameter `_timepoint` of `_updateMultiplier()` should always be a month start to be able to work with the rest of the code. This can made clear by renaming the variable.

Recommendation: Consider renaming `_timepoint` to `_monthStart`.

Hyperware: Fixed in commit 686df83f.

Cantina Managed: Fix verified.

3.5.8 `_updateMultiplier()` could use `getMultiplier()`

Severity: Informational

Context: TokenRegistry.sol#L362-L366, TokenRegistry.sol#L696-L709

Description: The calculation of `currentMultiplier` is equivalent to a part of the code of `getMultiplier()`. This part could be moved to a seperate function to hide implementation details and make it more in line with the rest of code.

Recommendation: Consider moving the code to a seperate function and also use that from `getMultiplier()`:

```
function _getMultiplier(address _account, uint256 _timepoint) internal view returns (uint256) {
    TokenRegistryStorageLayout storage $ = _getTokenRegistryStorage();
    return _account == address(0) ? $._totalMultipliers[_timepoint] : $._userMultipliers[_account][_timepoint];
}
```

Hyperware: Fixed in commit 1ac4d046.

Cantina Managed: Fix verified.

3.5.9 Function `_getRemainingDuration()` can be simplified

Severity: Informational

Context: TokenRegistry.sol#L726-L729, TokenRegistry.sol#L743-L745

Description: Function `_getRemainingDuration()` can be simplified.

Recommendation: Consider using `_subtract()`.

Hyperware: Fixed in commit 059ec707.

Cantina Managed: Fix verified.

3.5.10 Redundant check for `type(IAccessControl).interfaceId`

Severity: Informational

Context: TokenRegistry.sol#L799-L808, HyperGovernorToken.sol#L197-L209

Description: The check for `type(IAccessControl).interfaceId` in `supportsInterface()` is not necessary for contracts that inherit from [AccessControlUpgradeable](https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/access/AccessControlUpgradeable.sol#L89-L91) or AccessControl

Recommendation: Consider removing the check.

Hyperware: Fixed in commits:

- f189f169.
- bc7846ab.

Cantina Managed: Fix verified.

3.5.11 `tokenRegistry` can be `immutable`

Severity: Informational

Context: HyperGovernorToken.sol#L27

Description: `tokenRegistry` can be `immutable` because its never updated.

Recommendation: Consider changing `tokenRegistry` to `immutable`.

Hyperware: Fixed in commit e5f03d64.

Cantina Managed: Fix verified.

3.5.12 Function `_update()` not necessary

Severity: Informational

Context: HyperGovernorToken.sol#L288-L291

Description: Function `_update()` doesn't alter the functionality of `super._update()` so could be removed.

Recommendation: Consider removing `_update()` from `HyperGovernorToken`.

Hyperware: Fixed in commit 981a4769.

Cantina Managed: Fix verified.

3.5.13 Same `nameHash` can't be auctioned twice

Severity: Informational

Context: TLZAuction.sol#L304-L317, TLZAuction.sol#L529, TLZAuction.sol#L557

Description: The same `nameHash` can't be auctioned twice because `auction.completed` is set to `true` and this is checked in `createAuction()`. This is useful immediately after the auction to prevent accidental bid to a potentially new auction. However after soms time has passed it might be useful to be able to auction again.

Recommendation: Consider the usefulness of being able to auction again. If useful consider allowing this after some time has passed.

Hyperware: Fixed in commit c73f7aba.

Cantina Managed: Fix verified.

3.5.14 Bid of `minPrice` is not accepted

Severity: Informational

Context: TLZAuction.sol#L260-L267

Description: A bid of `minPrice` is not accepted which isn't logical based on the variable name.

Recommendation: Consider changing the code to:

```
- if (_auction.minPrice >= _bidAmount) {
+ if (_auction.minPrice > _bidAmount) {
```

Hyperware: Fixed in commit a77a645d.

Cantina Managed: Fix verified.

3.5.15 Old comment about `assembly`

Severity: Informational

Context: TLZAuction.sol#L260, TLZAuction.sol#L271-L274

Description: Function `_validateBidAmount()` contains a comment about assembly that isn't relevant (any more).

Recommendation: Consider removing the comment.

Hyperware: Fixed in commit ff507481.

Cantina Managed: Fix verified.

3.5.16 Repeated bids are inefficient

Severity: Informational

Context: TLZAuction.sol#L175-L194, TLZAuction.sol#L461, TLZAuction.sol#L480

Description: If someone want to do repeated bids, he has to pay for all the bids, or first `withdraw()` the payments for previous bids, which is inefficient.

Recommendation: Consider re-using the amount in `pendingWithdrawals[msg.sender]` and only `safeTransferFrom()` the extra amount.

Hyperware: Fixed in commit e34e7de6.

Cantina Managed: Fix verified.

3.5.17 Minting `gHYPR` Without Enforcing `HYPR` Max Supply Cap

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: In `HyperGovernorToken.sol` at line 69, the `mint()` function allows `TokenRegistry` to mint governance tokens (`gHYPR`) to an account:

```
function mint(address account, uint256 amount) external override onlyTokenRegistry notZeroAddress(account) {
    _ensureDelegateSet(account);
    _mint(account, amount);
}
```

However, this function does **not check** that the resulting `totalSupply()` of `gHYPR` stays within the `MAX_SUPPLY()` constraint of the underlying `HYPR` token. While `mint()` is restricted to be callable only by the `TokenRegistry`, the lack of a hard cap opens the door for accidental or inconsistent inflation if logic in `TokenRegistry` ever diverges from `HYPR`'s total supply tracking.

Recommendation: Consider adding a check to enforce that the amount of governance token can not exceed the total supply of `HYPR` token.

Hyperware: Fixed in commit c03c95cc.

Cantina Managed: Fix verified.

3.5.18 Unnecessary Exposure of `approve()` Function in Non-Transferable Token

Severity: Informational

Context: HyperGovernorToken.sol#L23

Description: In `HyperGovernorToken.sol` at line 23, the contract inherits from `ERC20`, `ERC20Permit`, and `ERC20Votes`, which expose ERC20 standard functions such as `approve()`:

```
contract HyperGovernorToken is IHyperGovernorToken, AccessControl, ERC20Permit, ERC20Votes
```

Although the token is explicitly non-transferable (with `transfer()` and `transferFrom()` disabled), the `approve()` function remains exposed. This can cause confusion for integrators or tooling that assumes approvals imply transferability.

Recommendation: Consider overriding and disabling the `approve()` function explicitly, for example:

```
function approve(address, uint256) public pure override returns (bool) {
    revert("gHYPR is non-transferable");
}
```

This will make the non-transferable nature of the token clearer and reduce the potential for integration bugs or false assumptions.

Hyperware: Fixed in commit 679dfbe5.

Cantina Managed: Fix verified.

3.5.19 `HyperAccountMinterUpgradable initialize()` function

Severity: Informational

Context: HyperAccountMinterUpgradable.sol#L17, HyperAccountMinterUpgradable.sol#L58-L61, HyperAccountPaidMinter.sol#L24-L29, HyperGridNamespaceMinter.sol#L116-L122, HyperGridNamespaceMinter.sol#L128, HyperGridNamespaceMinter.sol#L144

Description: `HyperAccountMinterUpgradable` is an abstract contract but has its own `initialize()` function.

However the contracts that inherit from `HyperAccountMinterUpgradable` have their own `initialize()` function, sometimes with a different number of parameters.

If the parameters are different then two instances of `initialize()` are present and of these should happen:

- Both `initialize()` are called, but this is prevented with the `initializer` modifier.
- The `initialize()` function of the main contract should do everything as the `initialize()` of the inherited contract. However this is not always the case (HyperAccountPaidMinter).

Recommendation: Consider creating a function `HyperAccountMinterUpgradable_init()` comparable to the OZ abstract contracts.

Call this function from all the main contracts:

- `HyperAccountAccessControlMinter`.
- `HyperAccountMinter`.
- `HyperAccountPermissionedMinter`.
- `HyperAccountPaidMinter`.
- `HyperAccount9CharCommitMinter`.
- `HyperAccountMultisig`.
- `HyperGridNamespaceMinter`.

According to the OZ pattern, the library functions are called from the `initialize()` on the main contract:

- `__UUPSUpgradeable_init();`
- `__EIP712_init("HyperAccountMinterUpgradable", "1");`

This means `HyperAccountMinterUpgradable_init()` will be an empty function.

Hyperware: Fixed in commit 716e7d93.

Cantina Managed: Fix verified.

3.5.20 `supportsInterface()` not implemented everywhere

Severity: Informational

Context: HyperAccountMinterUpgradable.sol#L67-L69, HyperAccountMultisig.sol#L24, HyperGridNamespaceMinter.sol#L26

Description: Function `supportsInterface()` isn't overridden in `HyperGridNamespaceMinter()` and `HyperAccountMultisig()`, unlike the other TBAs.

Recommendation: Consider adding `supportsInterface()` in `HyperGridNamespaceMinter()` and `HyperAccountMultisig()`.

Hyperware: Fixed in commit cd5995eb.

Cantina Managed: Fix verified.

3.5.21 `HyperGridNamespaceMinter` has two parts

Severity: Informational

Context: HyperGridNamespaceMinter.sol#L128-L136, HyperGridNamespaceMinter.sol#L144-L168

Description: The contract `HyperGridNamespaceMinter` has two different parts that can be deployed seperately and reference each other. This can also be seen by the two different `initialize()` functions. However this makes the code more difficult to understand.

Recommendation: Consider splitting the contract into two parts that can be deployed seperately.

Hyperware: Fixed in PR 107.

Cantina Managed: Fix verified.

3.5.22 Unused error `NotSupportedYet()`

Severity: Informational

Context: IMerkleDistributor.sol#L16

Description: The error `NotSupportedYet()` is not used.

Recommendation: Consider removing the error.

Hyperware: Fixed in commit 9e551a1a.

Cantina Managed: Fix verified.

3.5.23 Voting Power Decay Does Not Trigger Until Day 56 for Max Duration Locks

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: During testing, it was observed that when four users stake their full balance for `MAX_LOCK_DURATION`, their voting power remains constant through the first 55 days. Decay only begins on day 56.

This behavior is due to rounding in the duration term used in voting power calculation. Although the expectation was that decay would begin after the first month, the current precision and divisor cause the system to round the duration term such that the voting power remains unchanged for nearly two months.

This is not a critical bug but an important nuance in how vote decay is implemented.

Recommendation: No immediate fix is required. However, consider documenting this behavior or refining the duration divisor to ensure that decay begins closer to the expected one-month mark if stronger time resolution is desired.

Hyperware: Acknowledged. It is not ideal, but It is ok due to rounding issues.

Cantina Managed: Acknowledged.