Tinyman Governance Audit

Appendix: Summary of Findings

Below we present a deduplicated list of the findings from the review and our response and remediation where applicable.

Inconsistent use of data types
Some functions expected arguments of type `bytes` and later asserted their length while other functions expected arguments of the e.g `bytes[32]` and used the builtin type assertions of the Tealish language. This was inconsistent and it was suggested using the more specific Tealish types was more useful.

This issue arose due to refactorings during the development of the contracts and the introduction of new Tealish features after the contract development had begun. Some code was using new functionality while others had been written in an older style.

This issue was addressed in the following commit:
https://github.com/tinymanorg/tinyman-governance/commit/cdef5159371a6c116e26982379a76d60182124d1

Code duplication
A number of functions were copied between contracts with little or no modification. No issues were found but it would have been easier to understand if each function had a single implementation. It was suggested that the code could be refactored to allow reuse.

Unfortunately code reuse/imports is not a feature supported by Tealish and not trivial to implement.

No code changes were made.

Discrepancies between documentation & code
A number of minor discrepancies between the documentation and code were identified, particularly around app call arguments and box sizes.

The documentation was updated.
No code changes were made.

Deleting boxes & account state leads to loss of rewards, voting power

The reviewers pointed out that the contracts allow the user to delete boxes that they have created and in some cases this could lead to a loss of rewards or voting power if boxes are deleted too soon.

This issue can arise because of the decoupling between the Vault application and the Voting/Rewards applications. This makes it difficult to implement the safeguards necessary to prevent accidental deletion of currently active boxes.
Ultimately we see the delete box methods as power user functionality for reclaiming Algo paid for MBR after they reach significant levels but we do not expect to expose this functionality in normal user interfaces. The docs have been updated to carry a warning about the method and any future UI providing this functionality should include safeguards.

No code changes were made.

Redundant slope computation
In the Vault application the slope is recalculated in various places where it could be read from state. This doesn't affect the correctness or security but it would be easier to understand if the existing value was reused.

We agreed with the finding and recommendation of the reviewer but chose not to make the necessary changes to these maths heavy areas of code given that there was no issue of correctness.

No code changes were made.

Multi-purpose transfer function
Multiple contracts made use of a generic transfer function that worked for both Algo & ASAs. This was copied from the Tinyman AMM 2 contracts. The reviewers pointed out that in some contracts there was no use case for transferring Algo and in the other uses it would have been clearer to explicitly use separate methods for Algo and ASAs as they have different purposes in these contracts compared to the AMM where they are both used in the same context as liquidity in a pool.

We agreed with the finding and updated the code in the following commit:
https://github.com/tinymanorg/tinyman-governance/commit/7cd1161bae73a36322c24e759ee0fb7c8579224f

TINY Funding of Rewards
The Rewards application will fail to pay out rewards if it does not have sufficient TINY funds to pay the total outstanding rewards.

The rewards manager is responsible for setting the rewards amount per reward period. It is their responsibility to ensure the contract is sufficiently funded.

Initially this will be the Tinyman Team and they will fund it with relatively small amounts for short periods. After some time managerial control will be transferred to Tinyman governance and the full allocation of TINY rewards from the TINY tokenomics will be transferred to the application.

No code changes were made.

Voting on invalid asset IDs
The cast_vote method of the Staking application expects an array of asset ids along with an array of vote amounts. The reviewers pointed out that there is no check in the contract to ensure that the asset ids included are actual Tinyman AMM pool tokens. Including an asset that is not a pool token results in a wasted share of the vote because there is no way that TINY rewards will be allocated to a non-pool asset.

We acknowledge that there is an issue of a user's vote being wasted if they pass incorrect information to this method. In theory we could completely prevent this by checking the asset parameters for each asset in the array to ensure that it is a true Tinyman pool token. In practice this would add significant extra complexity and cost to the method due to the resource access constraints and opcode budget constraints of application calls. In this instance we decided it is not worth it because the only negative effect of passing non-pool token asset ids is a waste of a share of a user's vote.

No code changes were made.

Rounding when casting a vote
The cast_vote method of the Staking app includes a calculation to convert vote percentage into a share of voting power. The way the calculation is written causes a loss of precision that results in a slightly lower amount of voting power than would be expected. The reviewer analyzed the extent of this and determined that in the worst case scenario the loss of voting power is less than 1%.

We acknowledge that this issue exists and agreed that the calculation is written incorrectly according to the best practices for multiplication and division in the AVM. Despite the very low effect of the loss of precision we chose to correct the calculation as it was an easy localized fix.

The code was updated to fix this issue in the following commit:
https://github.com/tinymanorg/tinyman-governance/commit/ac45c588388512acac8b1fa6ed53ffdd542f0948

Risk of running out of storage when upgrading
The applications make use of Global state. In the AVM the number of Global state slots must be set when the application is created and cannot be modified by future updates, even if the application is updatable. The reviewers pointed out that the current scripts for creating the

application for tests only allocate as much state as is currently required, but they also note that the Staking, Proposal and Rewards contracts are updatable so a future update may require additional storage.

We acknowledge the issue and agree it makes sense to deploy the applications with a maximum allocation of Global state given that there is minimal additional costs associated with this.

No code changes were made to the in-scope files as these values will be configured elsewhere for production deployment.

Loss of admin keys
Some of the applications include manager or admin functionality. The manager can be updated using the set_manager method which takes an address as an argument and updates a global state variable in the application. The reviewers note that there are no checks for the correctness of this address or mechanism to revert an accidental change. The effect of setting the wrong address could be the loss of all managerial functionality or putting managerial functionality into the hands of a malicious actor.

We acknowledge this is a potential issue but we do not see a reliable way of making this safer without significantly increasing the complexity and limiting the future use of this method. It is intended that this functionality would initially only be used by the Tinyman team multisig and later by an executable governance proposal (a smart contract application). In both of these cases it is expected that multiple people would review the action before it is executed to ensure correctness. We see this operation as having a similar level of risk as transferring assets or rekeying an account using standard Algorand transactions.

No code changes were made.

Box MBR claiming
Creating Boxes in the AVM increases the Algo minimum balance requirement (MBR) of an application. The applications all make extensive use of boxes for state storage. The applications don’t explicitly check for an Algo transfer to the application every time a user causes the application to create a box. Instead the logic relies on the fact that the AVM will cause any box creation operation to fail if the MBR is not met so this balance increase check is made at the protocol level.

The Vault contract allows users to delete boxes (related to their own state) that are no longer needed and reclaim the MBR Algo that they paid. The reviewers noted that a user may be able to claim Algo they didn’t pay themselves IF the application had extra Algo beyond its MBR at the time of creating their box. By creating boxes without paying in Algo and then deleting them they could gain Algo.

This would be a serious issue if it was expected that the applications would hold an Algo balance for purposes other than the MBR as it would allow an attacker to gradually drain that Algo. However there is no expectation that the Vault application would hold Algo in addition to the MBR. If it does hold additional Algo, those funds could be considered ownerless or a donation so future users don't have to pay their own MBR.

Developers reading the code for inspiration or guidance while creating new contracts should note this carefully and understand that this implementation is only safe given the design assumptions and specific use case of this Vault.

No code changes were made.

Un-deployability of Staking Application
The Staking application contained an error in the create_application method which would be called when the application is first deployed. This error would have caused the application creation to fail.

We were surprised by this error given our extensive test suite but realized that the create_application method is not called during application setup in our test framework. We reviewed the tests again and implemented a test specifically for the application creation scenario and fixed the error.

The code was updated to fix this issue in the following commit:
https://github.com/tinymanorg/tinyman-governance/commit/bc50a36bc5f7dad912697b49490c3900016cf39a

Log Failure in The Get Box Method
Application Boxes are inaccessible to other applications in the AVM. In the interest of future composability with other Tinyman and third party applications we added a method to each application to log the contents of any box of the application. The reviewers pointed out that there was one instance of a box that was too large to be logged by the get_log method because a log can be a maximum of 1024 bytes and the size of the box plus the additional metadata was larger than this.

We acknowledge that this issue existed for one box and while it caused no current problem it defeated the purpose of future proofing the application if it didn't work in all cases. Rather than changing the box size and the associated calculations we realized we could reduce the size of the extra metadata without losing any functionality.

The code was updated to fix this issue in the following commit:
https://github.com/tinymanorg/tinyman-governance/commit/47baf3a04b3cc9134f44add74bd155be72daf325

Manager can re-enable approval requirement
The Proposal Voting application initially requires a Proposal Manager to flag proposals as 'approved' before voting can start. It is intended that this requirement will be lifted after the governance system has been observed to be functioning properly. Lifting this requirement is done using the disable_approval_requirement method.
The documentation states of this approval requirement "Once disabled, it's important to note that the manager cannot re-enable it."
The reviewers pointed out that this statement is not strictly true as the application itself is updateable by the manager account and any future update could re-enable this.

We acknowledge that the documentation is not strictly correct here and have updated it with a caveat.

No code changes were made.

Cannot execute proposed group of 16 arbitrary transactions
The documentation for the Arbitrary Executor describes it as a mechanism for executing 16 arbitrary transactions as part of a group based on the result of Proposal. The reviewers point out that the maximum group size is 16 transactions and one additional app call to the executor application must be required so the maximum number of arbitrary transactions should be 15.

We acknowledge the documentation issue and have fixed it.
It should be noted that the code for the executors is out-of-scope for this audit but the documentation does describe them.

No code changes were made.

Insufficient validation of governance asset parameters
The Vault application accepts the tiny_asset_id as one its parameters at creation time. This is used to optin the application to the TINY asset and set the id of the asset that should be accepted for locking in the Vault. Setting an incorrect id would block users from transferring TINY to the Vault and potentially setup the vault to work with a different asset.
The reviewers point out that there is no validation of the asset here. However they note that the remediation step would be to recreate the application once the error is noticed. They recommend confirming the details manually upon deployment.

We agree with the recommendation and intend that all details will be carefully checked by multiple people at deployment time.

No code changes were made.

Intcblock unused except for one instance
The reviewers pointed out that the generated Teal code does not use the intcblock at all except for in one case. While not causing any issues this stood out as an inconsistency.

We reviewed the issue and checked the Tealish compiler to understand the behaviour. The intention was to avoid the use of the intcblock generally in Teal generated from Tealish to ensure templated programs work as expected. The inclusion of an `int 0` operation in one code path and the resulting intcblock was an accidental oversight. The Tealish compiler has been updated.

This issue was solved together with the one below.

Unused _itxn_group_begin
The reviewers noted that the generated Teal code in each contract contains a function which is never called. This goes against best practices and can lead to confusion when reading the code.

We investigated the issue and understood this is a boilerplate that Tealish adds to deal with some intricacies of inner group transactions. However there are no inner groups in any of these contracts so it is unnecessary to include this code. We created a patch for Tealish that only conditionally adds this code to contracts if actually required.

The code was updated to fix this issue in the following commit:
https://github.com/tinymanorg/tinyman-governance/commit/e1d63a8581d8390fca7664bf213cbaf4a7d2492b

Centralization and Permission Concerns
The reviewers commented on the centralisation and permissioned features of the current iteration of the governance system. None of the comments highlight any permissioned functionality that was not mentioned in the documentation. The reviewers do note that there are plans to fully decentralize all permissed aspects and the current contracts to allow and plan for that.

Upgradable Contracts
The reviewers note that 3 of the 4 contracts in scope are upgradable by the managers.

We do not intend for this functionality to be used before transferring managerial control to the governance system. Instead this exists to allow governance to vote through upgrades to the system in future. However this functionality may be used earlier by the Tinyman Team if an implementation issue comes to light.

The Vault app is designed to be immutable because we do not think the team should have control of user's locked funds at any time. We also believe the fundamental rules of the vault's behavior should not be changeable, even by governance vote, due to the long lock times.

Approval Required for Proposals

The reviewers note that approval from the proposal manager (Tinyman Team) is required to activate a governance proposal for voting.

Off-Chain execution of proposals & funding of staking programs according to the on-chain vote

The reviewers note that the current setup relies on the Tinyman Team to take the actions to execute the governance proposals because there will not initially be a system in place to directly allow actions/transactions from project accounts.

One reviewer highlights that the proposals and votes are recorded on chain and voting power is determined by the immutable Vault app so this does represent an open voting system and the “recorded outcomes should be trustworthy”.

Funding of the TINY rewards in the Rewards application

The reviewers note that the Rewards application must be funded with TINY from the treasury.