Tinyman Governance v1 Audit

bit@d13.co

May 31st, 2024

Contents

Summary

Audit of the Tealish source code of the Tinyman Governance v1 contracts. The contracts are well written, adhering to the specifications and requirements set forth in the documentation. Three of the four contracts are upgradable, which is in line with the Tinyman team decentralization roadmap. No significant security findings were found.

Disclaimer

TBD

Introduction

These smart contracts represent the first milestone towards a decentralized governance system of the Tinyman platform. They provide an immutable vault for locking up the TINY governance token, as well as a three upgradable smart contracts used for proposals and governor rewards. In this iteration the tinyman team holds significant administrative and operational power over the protocol, but the design allows for a progressive improvements towards a fully decentralized and autonomous system.

Methodology

The Tinyman team assembled an ad-hoc committee comprised of multiple developers and teams with deep domain expertise in the AVM:

- Folks Finance
- BlockShake
- Vestige
- Steve Ferrigno
- D13

Each member chose their preferred approach and methodology and we met frequently to discuss findings.

Approach

While the Tinyman team provided extensive documentation for the contracts, I chose to initially ignore it, so as to familiarize myself with the code without being influenced by the developers' intentions. After this first pass, I read the documentation to compare against my mental model and notes before continuing to review the contracts. Additionally, I ran static analysis on the generated TEAL using tealer, the results of which can be found in Appendix B.

Scope

The Tealish sources of the following contracts are in scope:

1. `vault_approval.tl`
2. `proposal_voting_approval.tl`
3. `staking_voting_approval.tl`
4. `rewards_approval.tl`

The git commit of the audit snapshot is: `9d116ff75f3bfcd54d59fecaa287b5f63c2f4b77`

Common Findings

This sections lists findings common among multiple of the contracts in scope.

Permissioned nature

Severity: Medium - Difficulty: High

Scope: Proposal Voting, Staking Voting, Rewards

This iteration of the governance protocol relies heavily on the Tinyman team for administrative and operational support. The upgradability of some of the contracts, as well as lack of autonomy (e.g. in executing voting proposal or staking outcomes) makes the current iteration of the protocol "centralized, with a roadmap towards decentralization and autonomy".

The choice to make the vault contract immutable is a good faith measure to ensure user value (governance tokens) cannot be directly extracted by anyone with administrative access.

The two staking and proposal voting contracts currently represent an open voting system. They record the proposals and votes on-chain, as well as any possible interference from the operations team. In the current state of the code, voting power is calculated in the immutable Vault contract, so the recorded outcomes should be trustworthy.

The roadmap includes autonomous execution of proposal outcomes from these contracts, which will introduce the "A" (autonomy) from the acronym DAO.

After the protocol has transitioned to full on-chain autonomy, true decentralization ("D" in "DAO") will rely on the full and transparent distribution of the $TINY governance asset. In a scenario where the protocol was fully decentralized and autonomous from the first deployment, it would still be centralized so long as the team held undistributed tokens allocated to other parties, as they would be able to control their voting power.

Recommendation: Progressively decentralize the protocol and distribute the governance token.

Multi-purpose transfer function

Severity: Informational

Scope: Vault, Rewards

The transfer function present in the Vault and Rewards contracts serves a dual purpose: it can transfer either ALGO or ASAs from addresses controlled by the smart contracts.

```
func transfer(asset_id: int, amount: int, sender: bytes, receiver: bytes):
    # This function is copied from Tinyman AMM Contracts V2.
    # "asset_id == 0" is updated as "!asset_id" for budget optimization.
```

```
    # https://github.com/tinymanorg/tinyman-amm-contracts-v2/blob/main/
        contracts/amm_approval.tl#L1146

    if !asset_id:
        inner_txn:
            TypeEnum: Pay
            Sender: sender
            Receiver: receiver
            Amount: amount
            Fee: 0
        end
    else:
        inner_txn:
            TypeEnum: Axfer
            Sender: sender
            AssetReceiver: receiver
            AssetAmount: amount
            XferAsset: asset_id
            Fee: 0
        end
    end
    return
end
```

Per the code comment, this function was copied from the Tinyman AMM contracts, where by nature of swapping between ALGO and ASAs, the same code paths would sometimes require sending either type of asset.

In this set of contracts, the code paths to pay ALGO (e.g. refunding storage costs from vault contract) and the code paths to pay TINY (e.g. rewards from the rewards contract) should not overlap. While the specific implementation as examined is correct, it would be better practice to split these functionalities into distinct functions.

Recommendation: Remove the ALGO functionality from the transfer function and implement it separately.

Ensure storage flexibility in upgradable contracts

Severity: Informational

Scope: Proposal Voting, Staking Voting, Rewards

The voting proposal, staking proposal and rewards contracts are currently upgradable. In the AVM, contracts can not mutate their non-box storage schema after being deployed, so the contracts should be provisioned with unforeseen future global storage or code storage needs in mind.

It is recommended to allocate the maximum available in the following:

Extra pages It is strongly recommended to deploy the upgradable contracts with 3 `extra_pages`. This will ensure significant extra space to grow the code on-chain in the future.

Global storage It is strongly recommended to deploy the upgradable contracts with the maximum allocation of global storage (64 key/values in total).

The typed schema needs to be decided when the contracts are first deployed, i.e. the number of uint64 and byte slice storage space. The team should accommodate for any expected storage growth needs with a bias towards byte slices, as uint64 can be easily stored in a byte slice (but not vice-versa.)

Local storage should not be necessary given the box storage utilization in these contracts.

Vault Contract Findings

Insufficient MBR payment enforcement can lead to ALGO extraction under non-standard conditions

Severity: Low - Difficulty: Medium

Each box created during the operation of the contract requires small amounts of ALGO to be locked up in the contract's escrow address.

These minimum balance requirement (MBR) payments are not strictly enforced during operations that create boxes, but when deleting boxes, the "freed" minimum balance requirement is returned to the user.

Under normal operating conditions, the escrow address of the smart contract should have exactly the ALGO balance that is required to cover the MBR.

In the non-standard scenario where there is an escrow balance surplus beyond the required, an attacker could create or manipulate their state without making the MBR payments, thus becoming eligible to receive ALGO when they delete their account state or account power boxes.

As an example, a new governor creating a lock for the minimum 10 $TINY without paying for MBR would "claim" 0.421 ALGO (2500 + (400 * 1008) + 2500 + (400 * 32)) from the escrow address' unlocked balance, at a cost of 0.003 ALGO for the transaction fees and a temporary lockup of the 10 $TINY tokens.

This "claimed" amount would be due for withdrawal when the lock ends, but if the user increased his locked amount to create 22 account power boxes in total (0.066 ALGO transaction fees, 220 $TINY locked total) the first account power box's capacity (21) would be exceeded and a second one would be created, thus making the first account box eligible for deletion, which would return the account power box MBR immediately (0.4057 ALGO)

Since the contract will not normally hold more ALGO balance than the MBR, this is a low impact vulnerability.

Recommendation: enforce minimum balance requirement payments during operations that create boxes.

Lack of timestamp validation when deleting account power boxes

Severity: Informational

The `delete_account_power_boxes` method is documented as intended to enable deletion of obsolete boxes, however the implementation does not enforce a minimum power struct "age" before deleting boxes.

As such, it would be possible for users to delete account power boxes that would cost them voting power in specific situations. As an example:

- A user had 21 power structs at time `t0` when a specific proposal `p` was created
- They increased their lock amount at time `t1`, thus creating a 22nd power struct which would create a second account power box
- They call `delete_account_power_boxes` with argument `1` at time `t2` to delete the first account power box
- Their voting power at time `t-` as applicable for proposal `p` is lost
- They are unable to vote in proposal `p`

Additionally, implementing this check would also slow down potential surplus MBR drains (see previous section), as account power boxes would not be immediately deletable.

Recommendation: assert that all power structs being deleted are older than a certain value (e.g. 4 weeks.)

Insufficient validation of governance asset parameters (Informational)

The application creation handler accepts `tiny_asset_id` to be stored as the governance asset ID, but does not validate that this is an existing asset. Furthermore, the documentation specifies that the asset is intended to have 6 decimal points, which is also not validated.

```
@public(OnCompletion=CreateApplication)
func create_application(tiny_asset_id: int):
    app_global_put(TINY_ASSET_ID_KEY, tiny_asset_id)
```

This is presented as an informational finding. In case of an error, the administrators of the contract can redeploy it with the correct parameters.

Recommendation: Confirm the asset configuration manually upon deployment.

Proposal Voting Contract Findings

TBD

Staking Voting Contract Findings

Unused create_application variable

Severity: Low - Difficulty: Low

The `create_application` method of the Staking Voting app specifies `vault_app_id` as an accepted argument, but instead it uses `btoi(Txn.ApplicationArgs[0])`.

```
@public(OnCompletion=CreateApplication)
func create_application(vault_app_id: int):
    app_global_put(VAULT_APP_ID_KEY, btoi(Txn.ApplicationArgs[0]))
    ...
```

Depending on how the application creation transaction group is formed, this could lead to an incorrectly deployed contract.

Recommendation: Replace `btoi(Txn.ApplicationArgs[0])` with `vault_app_id`

Unnecessary rounding down of voting power

Severity: Low - Difficulty: Low

The `cast_vote` method performs this calculation when allocating voting power to individual assets:

```
option_vote_amount = tmp_vote_percentage * (account_voting_power / 100)
```

The precedence of operations (division first, which does an implicit round-down) causes an unnecessary loss of precision.

This loss should be very small even in the worst case scenarios: for a user with the minimum locked amount (10 TINY) at a 1 week remaining lock duration, the loss was calculated to be $\sim 0.09\%$:

```
slope = 10 * 1e6 / 126144000 = 0.07927447995941146
power = slope * 604800 = 47945.20547945205
rounded = 47900
loss = 45
loss_pct = 100 * 45 / 47945 = 0.09385754510376473
```

Recommendation: Replace with `tmp_vote_percentage * account_voting_power / 100`

Rewards Contract Findings

No findings exclusive to this contract. See also: Common Findings section.

Appendix A: Classification

The committee agreed to use the finding classification criteria defined by Trail of Bits:

Difficulty	Description
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	"An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue."

Severity	Description
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	"User information is at risk; exploitation could pose reputational, legal, or moderate financial risks."
High	"The flaw could affect numerous users and have serious reputational, legal, or financial implications."

Appendix B: Static Analysis

The code was analyzed with `algokit task analyze` (v 1.12.3) which wraps the tealer library to perform static analysis of TEAL code to find some common vulnerabilities. The provided TEAL bytecode was recompiled (decompile/compile) to strip comments in order to make it compatible with tealer.

Beyond some warnings related to logic sig/stateless applications, the analysis did not yield any meaningful findings. The summary of findings for each contract is presented below.

Vault Contract

Detector	missing-fee-check
Impact	High
Details	https://github.com/crytic/tealer/wiki/Detector-Documentation#missing-fee-field-validation
Execution Paths (#Lines)	1-23->25-32->267-281->33-34 ...
Evaluation	False positive - irrelevant for stateful applications.

Detector	rekey-to
Impact	High
Details	https://github.com/crytic/tealer/wiki/Detector-Documentation#rekeyable-logicsig
Execution Paths (#Lines)	1-23->25-32->267-281->33-34 ...
Evaluation	False positive - irrelevant for stateful applications.

Proposal Voting Contract

Detector	unprotected-updatable
Impact	High
Details	https://github.com/crytic/tealer/wiki/Detector-Documentation#unprotected-updatable-application
Execution Paths (#Lines)	1-23->35-40->270-278->41-42
Evaluation	**Incorrect**. The application update code path is guarded by a manager check `Txn.sender == global_storage("manager")`

Detector	is-updatable
Impact	High
Details	https://github.com/crytic/tealer/wiki/Detector-Documentation#upgradable-application
Execution Paths (#Lines)	1-23->35-40->270-278->41-42
Evaluation	Correct, (documented.)

Detector	missing-fee-check
Impact	High
Details	https://github.com/crytic/tealer/wiki/Detector-Documentation#missing-fee-field-validation
Execution Paths (#Lines)	1-23->25-32->237-269->33-34
Evaluation	False positive - irrelevant for stateful applications.

Detector	rekey-to
Impact	High
Details	https://github.com/crytic/tealer/wiki/Detector-Documentation#rekeyable-logicsig
Execution Paths (#Lines)	1-23->25-32->237-269->33-34 ...
Evaluation	False positive - irrelevant for stateful applications.

Staking Voting Contract

Detector	unprotected-updatable
Impact	High
Details	https://github.com/crytic/tealer/wiki/Detector-Documentation#unprotected-updatable-application
Execution Paths (#Lines)	1-15->27-32->168-176->33-34 ...
Evaluation	**Incorrect**. The application update code path is guarded by a manager check `Txn.sender == global_storage("manager")`

Detector	is-updatable
Impact	High
Details	https://github.com/crytic/tealer/wiki/Detector-Documentation#upgradable-application
Execution Paths (#Lines)	1-15->27-32->168-176->33-34 ...
Evaluation	Correct, (documented.)

Detector	missing-fee-check
Impact	High
Details	https://github.com/crytic/tealer/wiki/Detector-Documentation#missing-fee-field-validation
Execution Paths (#Lines)	1-15->17-24->146-167->25-26 ...
Evaluation	False positive - irrelevant for stateful applications.

Detector	rekey-to
Impact	High
Details	https://github.com/crytic/tealer/wiki/Detector-Documentation#rekeyable-logicsig
Execution Paths (#Lines)	1-15->17-24->146-167->25-26
Evaluation	False positive - irrelevant for stateful applications.

Rewards Contract

Detector	unprotected-updatable
Impact	High
Details	https://github.com/crytic/tealer/wiki/Detector-Documentation#unprotected-updatable-application
Execution Paths (#Lines)	1-13->27-32->151-159->33-34
Evaluation	**Incorrect**. The application update code path is guarded by a manager check `Txn.sender == global_storage("manager")`

Detector	is-updatable
Impact	High
Details	https://github.com/crytic/tealer/wiki/Detector-Documentation#upgradable-application
Execution Paths (#Lines)	1-13->27-32->151-159->33-34
Evaluation	Correct (documented.)

Detector	missing-fee-check
Impact	High
Details	https://github.com/crytic/tealer/wiki/Detector-Documentation#missing-fee-field-validation
Execution Paths (#Lines)	1-13->15-24->126-150->25-26 ...
Evaluation	False positive - irrelevant for stateful applications.

Detector	rekey-to
Impact	High
Details	https://github.com/crytic/tealer/wiki/Detector-Documentation#rekeyable-logicsig
Execution Paths (#Lines)	1-13->15-24->126-150->25-26 ...
Evaluation	False positive - irrelevant for stateful applications.