Tinyman Governance Protocol

Security Assessment

April 9, 2024

Prepared for:
Tinyman

Prepared by:
Gidon Katten
Folks Finance

Contents

Project Goals and Scope

The purpose of the audit was to:

- Verify the protocol contracts implement the system as described in the documentation.
- Analyse the protocol's adherence to the ethos of DeFi.
- Analyse the protocol logic to detect potential avenues for abuse by bad actors.
- Analyse the protocol logic to detect potential errors, failures, miscalculations or other unexpected outcomes.
- Analyse the higher level Tealish code to ensure it correctly implements the documented logic.

Note that the generated TEAL code was not checked for errors introduced in the transpilation process.

The repository and commit which were reviewed was https://github.com/tinymanorg/tinyman-governance/tree/4a287f63cbd86462a2de7839ebf29c0d81c779e2.

The smart contracts reviewed were:

- *proposal_voting/proposal_voting_approval.tl*
- *proposal_voting/proposal_voting_clear.tl*
- *rewards/rewards_approval.tl*
- *rewards/rewards_clear.tl*
- *staking_voting/staking_voting_approval.tl*
- *staking_voting/staking_voting_clear.tl*
- *vault/vault_approval.tl*
- *vault/vault_clear.tl*

In addition, the *arbitrary_executor*, *fee_management_executor* and *treasury_management_executor* contracts were reviewed according to how they are described in the documentation provided. No code was reviewed for them.

Project Coverage

This section provides an overview of the analysis coverage of the review:

- **Vault.** Reviewed the TINY power calculations and compared the implementation to Curve's Voting Escrow CRV (veCRV) model. Looked at whether you could manipulate your own and others' TINY power. Looked for ways locked TINY could become stuck in the Vault and if it could be ever withdrawn when supposedly locked. Reviewed access control checks.

- **Rewards.** Reviewed calculations for reward amount. Looked at whether you could claim rewards multiple times for the same period. Looked at whether you could claim another account's rewards. Reviewed access control checks.

- **Staking Voting.** Reviewed calculations for voting power. Looked at whether you could vote multiple times on the same proposal. Looked at ways you could manipulate others' votes and their ability to vote. Reviewed access control checks.

- **Proposal Voting.** Reviewed how proposals are created and approved before they can be voted on. Reviewed calculations for voting power. Looked at whether you could vote multiple times on the same proposal. Looked for ways you could manipulate others' votes and their ability to vote. Reviewed access control checks.

- **Arbitrary Executor.** Reviewed whether it was possible to implement the application as described in the documentation. Looked for ways to execute transactions without having the corresponding proposal. Looked for potential design flaws.

- **Fee Management Executor.** Reviewed whether it was possible to implement the application as described in the documentation. Looked for ways to execute transactions without having the corresponding proposal. Looked for potential design flaws.

- **Treasury Management Executor.** Reviewed whether it was possible to implement the application as described in the documentation. Looked for ways to execute transactions without having the corresponding proposal. Looked for potential design flaws.

Centralisation Risks

As part of the project scope, the protocol's adherence to the ethos of DeFi was analysed. A large part of that is centralisation risk which is what this section covers.

The protocol contains privileged accounts with varying levels of ability. The Tinyman team has commented that the most important of these will be controlled by a multisig account of ledger devices with a signing requirement of 3/5.

There are a number of centralisation risks which are reliant on the Tinyman team:

- Until the *manager* accounts are transferred to the governance system, the accounts have the ability to execute any desired behaviour in the *proposal_voting, rewards* and *staking_voting* applications. There are *update_application* methods available which occur immediately without any delay.
- The *rewards* application is sufficiently funded with TINY.
- The actions on farms happen off-chain according to the results of the *staking_voting* application. The Tinyman team has commented that in the future the farms will be moved to an on-chain, trustless system.
- Until the *arbitray_executor* application is deployed, any actions voted on in the *proposal_voting* application are carried out.
- Until the *fee_management_executor* application is deployed, any actions voted on concerning fees in the Tinyman AMM are carried out.
- Until the *treasury_management_executor* application is deployed, any actions voted on concerning the treasury of the Tinyman AMM are carried out.
- The *proposal_manager* can block proposals after disabling the approval requirement.

The Tinyman team has communicated their commitment to transition to a more trustless approach over time with the vision being that all managerial functionality will be controlled entirely by governance.

Summary of Findings

Overall the code written was of a high-level. The code was clear and well documented. Applications and methods were divided by their own responsibilities. The accompanying documentation supported the protocol review by providing additional context when needed. There were instances where there were discrepancies between the code and documentation but these were minor in nature.

The test coverage was good and covered the main protocol interactions. One thing lacking were extensive tests to check for expected failing behaviour.

The protocol adheres to the ethos of DeFi. The governance system is based on Curve's governance which has proven to be robust and effective. There is a considerable amount of centralisation but there is a clear plan to reduce this in the near future.

The table below summarises the findings of the review:

ID	Title	Severity	Difficulty
1	Log Failure in The Get Box Method	Low	Low
2	Risk of unavailable storage when upgrading applications	Informational	High
3	Manager can re-enable approval requirement	Informational	High
4	Cannot execute proposed group of 16 arbitrary transactions	Low	Low
5	Lack of a two-step process for admin role transfers	Informational	Low

Detailed Findings

1. Log Failure in The Get Box Method	
Severity: Low	Difficulty: Low
Target: *rewards/rewards_approval.tl*	

A common method across the applications is the *get_box* method. It returns the data within a box to other applications through logging.

Documentation - Applications (Contract) - Common Solutions - Get Box

> ”To provide flexibility for future requirements, the “get_box_method” shares the raw box data with other applications. This allows them to process the data according to their specific needs.”

The *log* opcode in V9 (the version used by Tinyman) fails if the sum of the logged bytes exceeds 1024 bytes. Therefore it is important that the box sizes you allocate do not exceed this threshold (considering also any additional logs you make) if you want the box to be available to be shared with other applications.

Excluding the box content, the *get_box* method concatenates 14 bytes:

- arc4 return log 0x151f7c75: 4 bytes,
- exists: 8 bytes,
- length of content: 2 bytes.

To ensure the *get_box* method won’t fail, the box content must not exceed the remaining 1010 bytes available.

Code Snippet - File *rewards/rewards_approval.tl* - Line 204-211

```
@public()
func get_box(box_name: bytes) int, bytes, bytes:
    int exists
    bytes data
    exists, data = box_get(box_name)

    return exists, extract(6, 2, itob(len(data))), data
end
```

The *rewards_approval* contract has a *REWARD_CLAIM_SHEET* box of size greater than 1010 bytes. Therefore the *get_box* method would fail when passed a *REWARD_CLAIM_SHEET* box.

Code Snippet - File *rewards/rewards_approval.tl* - Line 32

```
const int  REWARD_CLAIM_SHEET_BOX_SIZE = 1012
```

In addition the documentation states the justification for the box size.

Documentation - Applications (Contract) - Common Solutions - Array Implementation

> ”We chose to set box sizes less than or equal to 1KB, so they work with 1 reference.”

Whilst the sentiment described is true, there is a more pressing and restrictive reason to limit them further. Namely the existence of the *get_box* method and the limitation surrounding the maximum logged bytes.

Recommendation is the following:

1. Either:
    a. Reduce the allocated box size to be 1010 bytes.
    b. Reduce the size of “exists” to be 1 byte.
    c. Remove “exists” and use the length of content to determine whether the box exists. A length of 0 indicates the box does not exist.
2. Clarify the maximum allocated box size and the reasoning behind the box size restrictions in the documentation.

Resolved. Commit [47baf3a04b3cc9134f44add74bd155be72daf325] removes the “exists” flag.

2. Risk of unavailable storage when upgrading applications	
Severity: Informational	Difficulty: High
Target: *staking_voting/staking_voting_approval.tl, staking_voting/staking_voting_clear.tl, rewards/rewards_approval.tl, rewards/rewards_clear.tl*	

The *staking_voting* and *rewards* applications are upgradeable. When creating a smart contract the amount of storage can never be changed once the contract is created. This applies to the global/local state schema and number of pages allocated.

To ensure future upgrades are not limited by the initial values set, it is recommended that you specify the maximum values when creating the applications.

Recommendation is to change the deployment scripts to allocate the maximum storage allocation possible. For example:

- Global state:
    - 32 uints
    - 32 byte slices
- Local state:
    - 8 uints
    - 8 byte slices
- Extra pages: 3.

A different split between units and byte slices can be chosen according to preference. Although byte slices can always be interpreted (or contain) a uint whereas a uint cannot do the same for a byte slice of size greater than 8 bytes

---

The Tinyman team confirmed they will deploy the application with the maximum storage allocation possible.

3. Manager can re-enable approval requirement	
Severity: Informational	Difficulty: High
Target: *proposal_votingl/proposal_voting_approval.tl*	

In the *proposal_voting_approval* contract, there is the *disable_approval_requirement* method.

Code Snippet - File *rewards/rewards_approval.tl* - Line 204-211

```
@public()
func disable_approval_requirement():
    bytes user_address = Txn.Sender
    assert(user_address == app_global_get(PROPOSAL_MANAGER_KEY))
```

The documentation states the intended behaviour of the method.

Documentation - Proposal Voting App - Settings - Disable Approval Requirement

> “This is a permissioned method, and only the manager has the authority to call it for the purpose of disabling the approval requirement. Once disabled, it’s important to note that the manager cannot re-enable it.”

The method is only callable by the *proposal_manager* but since the *manager* can set the *proposal_manager*, we can say that in actuality it is callable by both the *proposal_manager* and *manager*. Since the *manager* can call *update_application*, and the update can introduce new logic, it’s possible that the manager can re-enable the approval requirement. Therefore we can say the statement “it’s important to note that the manager cannot re-enable it” is false.

Recommendation is to clarify the intended behaviour in the documentation.

The Tinyman team has updated the documentation with the caveat mentioned above.

4. Cannot execute proposed group of 16 arbitrary transactions	
Severity: Low	Difficulty: Low
Target: *proposal_voting/proposal_voting_approval.tl, TBD*	

The *proposal_voting* application has the method *create_proposal* which contains *execution_hash* as one of its parameters. The idea is that this can be used in the future to execute a transaction or transaction group automatically based on the results of a proposal, as opposed to relying on a trusted manager to do so.

The *arbitrary_executor* application will carry out this automatic execution once it is deployed. The documentation describes a method *validate_group* which allows governance to execute an arbitrary transaction group.

Documentation - Executors (Contract) - Arbitrary Executor App - Methods - Validate Group

> Proposal's `execution_hash` field should be in following format:
> Lpad(base32decode(<arbitrary_transaction_id>), 128)
>
> Transaction Group:
> 1. App Call:
> Sender: User Address
> Index: Arbitrary Executor
> App ID
> OnComplete: NoOp
> App Args: ["validate_group", proposal_id]
> Foreign Apps: [Proposal Voting App ID]
> 2. 16. Arbitrary Transactions: Can be any type of transactions.

Algorand supports a maximum of 16 outer transactions in a single group so it is not possible to execute 16 arbitrary transactions when the *validate_group* application call is also present. Furthermore, the *execution_hash* must be defined using the *group_id* which includes the application call *validate_group* as part of the group.

Recommendation is to update the documentation for validate group to specify 15 arbitrary transactions and an *execution_hash* of *Lpad(base32decode(<arbitrary_group_id>), 128)*.

The Tinyman team has updated the documentation to specify 15 arbitrary transactions. The execution hash also now references a group.

5. Lack of a two-step process for admin role transfers	
Severity: Informational	Difficulty: High
Target: *proposal_voting/proposal_voting_approval.tl, rewards/rewards_approval.tl, staking_voting/staking_voting.tl*	

The protocol methods used to transfer the admin role from one address to another perform those transfers in a single step, immediately updating the admin address. Making such a critical change in a single step is error-prone and can lead to irrevocable mistakes.

These methods are:

- *proposal_voting/proposal_voting_approval.tl*
    - *set_manager*
    - *set_proposal_manager*
- *rewards/rewards_approval.tl*
    - *set_manager*
    - *set_rewards_manager*
- *staking_voting/staking_voting.tl*
    - *set_manager*
    - *set_proposal_manager*

Recommendation is to implement a two-step process for admin role transfers. One way to do this would be splitting each *set_manager* method into two methods: a *propose_manager* method that saves the address of the proposed new manager to the global state and an *accept_manager* method that finalises the transfer of the role (and must be called by the address of the new manager).

The Tinyman team has indicated they will not follow the recommendation as care will be taken when submitting and reviewing the admin transfer transactions before signing.

Notices and Disclaimers

The security testing team made every effort to cover the systems in the test scope as effectively and completely as possible given the time budget available. There is however no guarantee that all existing vulnerabilities have been discovered. The findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

It is important to note that this audit is informative and does not serve as an endorsement of the protocol under review. Despite our thorough examination, inherent limitations exist in any auditing process. As such, the audit should not be considered a guarantee against the presence of bugs, vulnerabilities, or unexpected behaviours within the code. Furthermore, the security assessment applies to a snapshot of the current state at the examination time.

Clients and stakeholders are encouraged to consider this audit as part of a broader risk management strategy. It is recommended that ongoing reviews and monitoring of the protocol be undertaken to manage potential risks effectively.

Appendix

The findings reference a severity and difficulty classification which has been defined by Trail of Bits:

For severity:

Severity Classifications	
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.

For difficulty:

Difficulty Classifications	
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.