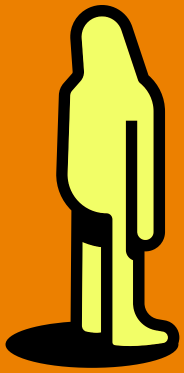
Tinyman 1.1 Update Audit

v220107

coinspect



Tinyman

Smart Contract Audit

V220107

Prepared for Tinyman • January 2022

1. Executive Summary

In **January 2022,** Tinyman engaged Coinspect to perform a source code review of a security update to their contracts. The objective of the project was to review the fixes for three vulnerabilities previously identified by the Tinyman team.

The following issues were analyzed during the assessment:

High Risk	Medium Risk	Low Risk
2	1	0
Fixed	Fixed	Fixed
2	1	0

2. Assessment and Scope

The audit started on January 5 and was performed on the `v1-1-updates` branch of the repository tinymanorg/tinyman-contracts-v1 with commit `8be3e7f8005bb131c51e10e82885a8e764d7a336`.

The scope of the audit was limited to the updates done by the Tinyman team due to three vulnerabilities which were already known to them and are described in this report. Coinspect did not review the Tinyman platform as a whole nor looked for other vulnerabilities besides those mentioned before.

The objective was to confirm that the updated code prevented the vulnerabilities and did not introduce side effects.

3. Summary of Findings

Id	Title	Total Risk	Fixed
TNY-1	Pools can be drained by burning liquidity for the same asset twice	High	✔
TNY-2	Liquidity assets can be minted using only one underlying asset	High	✔
TNY-3	Pools with high asset ratios get locked	Medium	✔

4. Detailed Findings

TNY-1 Pools can be drained by burning liquidity for the same asset twice

Total Risk **High**	Impact High	Location `contracts/pool_logicsig.teal.tmpl`
Fixed ✔	Likelihood High	

Description

An attacker can drain pools by burning their liquidity tokens and receiving the same asset twice.

When burning their liquidity assets, poolers should receive the underlying assets in proportion to the pool's holdings. Nevertheless, the pool contract template never checks which assets the sender requests. This allows a pooler to ask for the same asset to be transferred twice for their liquidity assets.

By extracting the most valuable asset twice, attackers can steal from the pool: they mint liquidity assets with a combination of the two underlying ASAs and then redeem them for the most valuable one.

Attackers can maximize their profit by abusing this vulnerability while using `TNY-2` to get liquidity assets by putting only the less valuable asset into the pool and then exchanging the gained liquidity for only the most valuable ASA.

Status

The vulnerability was addressed by adding an explicit check to the pool contract for the ID of the assets that are being exchanged when burning liquidity assets.

TNY-2 Liquidity assets can be minted using only one underlying asset

Total Risk	Impact	Location
High	High	`contracts/pool_logicsig.teal.tmpl`
Fixed	**Likelihood**	
✓	High	

Description

An attacker can mint liquidity assets for a pool by providing only one of the underlying assets of the pool instead of two.

When adding liquidity, poolers should provide an equivalent amount of each underlying asset being swapped in the pool. Nevertheless, the pool contract template never checks which assets the pooler is providing. This allows a pooler to mint liquidity by providing the same asset twice.

By providing the less valuable ASA twice, an attacker can get liquidity assets for a cheaper price than intended.

As described in `TNY-1`, the vulnerabilities are closely related and can be used together in a single exploit.

Status

The vulnerability was addressed by adding an explicit check to the pool contract for the ID of the assets that are being exchanged when minting liquidity assets.

The check is identical to the one performed by the `burn` operation.

TNY-3 Pools with high asset ratios get locked

Total Risk	Impact	Location
Medium	High	`contracts/validator_approval.teal`
Fixed ✓	Likelihood Low	

Description

When a pool has a big asset ratio and has not seen a recent trade, it will get locked due to an overflow in a calculation of the `cumulative price` field.

The calculation is `seconds_since_last_operation * (price * 10^6)` as described in Tinyman's own report of the problem. When this operation returns an integer that overflows an Algorand Virtual Machine word (64 bits), the transaction fails.

Since this calculation is performed at the beginning of all the swap, mint, and burn operations, the result is that the pool gets locked. There is no way to return the pool to a sane state, because it is impossible to perform any operation on it and the pools are permissionless.

Status

The vulnerability was addressed by performing the aforementioned multiplication with `mulw` in the TEAL code, which will perform a full precision multiplication. The result is then checked and updated only if it fits in a word.

An important caveat is that, even though the pool will not get locked, it will not record the correct `cumulative price` if it does not fit in a word. This may impact systems that rely on Tinyman as a price oracle. Tinyman has confirmed this is intended and will be clearly stated in the documentation of the price oracle feature.

5. Disclaimer

The information presented in this document is provided "as is" and without warranty. The present security audit does not cover any off-chain systems or frontends that communicate with the contracts, nor the general operational security of the organization that developed the code.