CERTORA

Formal Verification of Pintu Token (PTU)

Summary

This document describes the specification and verification of the Pintu Token (PTU) using the Certora Prover. The work was undertaken August 12, 2021.

The scope of our verification was PTU, an ERC20 token.

The Certora Prover proved the implementation of PTU is correct with respect to the formal rules written by the Certora team. All issues were promptly corrected, and the fixes were verified to satisfy the specifications.

The next section formally defines high level specifications of PTU. The results of running the Certora Prover are available in:

- Pintu Token
- Pintu Proxy

List of Main Issues Discovered

Severity: Low (Fixed)

Issue:	**approve may be susceptible to front running**
Rules Broken:	Cannot approve non-zero amount when currently set to non-zero
Description:	Some tokens choose to limit the approval function to run only when the allowance is already set to 0, or when the desired spending amount is 0. PTU allows setting the approval freely.
Mitigation/Fix:	The Pintu team added requirements that render front running impossible. It is recommended for users to use increaseAllowance and decreaseAllowance if possible.

Disclaimer

The Certora Prover takes as input a contract and a specification and formally proves that the contract satisfies the specification in all scenarios. Importantly, the guarantees of the Certora Prover are scoped to the provided specification, and the Certora Prover does not check any cases not covered by the specification.

We hope that this information is useful, but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

Notations

✔️ indicates the rule is formally verified on the latest reviewed commit. We write ✔️* when the rule was verified on a simplified version of the code (or under some assumptions).

❌ indicates the rule was violated under one of the tested versions of the code.

✔️❌ indicates the rule was violated under a previously tested version of the code, but is now correct.

✍️ indicates the rule is not yet formally specified.

🔁 indicates the rule is postponed (<due to other issues, low priority>) .

We use Hoare triples of the form {p} C {q}, which means that if the execution of program C starts in any state satisfying p, it will end in a state satisfying q. In Solidity, p is similar to require, and q is similar to assert.

The syntax {p} [C1 ~ C2] {q} is a generalization of Hoare rules, called relational properties. {p} is a requirement on the states before C1 and C2, and {q} describes the states after their executions. Notice that C1 and C2 result in different states. As a special case, C1~op C2, where op is a getter, indicating that C1 and C2 result in states with the same value for op.

Verification of PTU

Properties

1. Integrity of transfer ✔️

- Succeeds if balance is sufficient, there is no overflow, the contract is not paused, the recipient is not the zero address, and the sender is not the zero address. Reverts otherwise.
- Moves exactly the specified amount between the sender and the recipient, unless the sender is the same as the recipient, in which case balances stay the same.
- No addresses other than the sender and the recipient are affected by the transfer.

2. Integrity of transferFrom ✔️

- Succeeds if balance is sufficient, there is no overflow, allowance is sufficient, the contract is not paused, the recipient is not the zero address, the sender is not the zero address, and the paying account is not the zero address. Reverts otherwise.
- Moves exactly the specified amount between the sender and the recipient, unless the sender is the same as the recipient, in which case balances stay the same. Also updates the allowances.
- Cannot transfer from the zero address or with a zero address spender
- No addresses other than the sender and the recipient are affected by the transfer.

3. Integrity of approve ✔️

- Succeeds if sender is not the zero address, spender is not the zero address, and the contract is not paused. Reverts otherwise.
- Zero address cannot approve, and cannot approve to zero address as the spender.
- No state other than the allowance of sender to the spender is affected.

4. Cannot approve non-zero amount when currently set to non-zero ✔️✖

5. Functions that may modify balances ✔️

- transfer
- transferFrom
- mint
- burn
- burnFrom
- initialize

6. Functions that may modify total supply ✔️

- mint
- burn
- burnFrom
- initialize

7. Functions that may modify allowances ✔️

- approve
- increaseAllowance
- decreaseAllowance
- burnFrom
- transferFrom

8. A user's balance cannot be reduced unless it allowed the sender ✔️

9. Approval must be authorized ✔️

The only way to increase an allowance is by the allowing address itself sending a transaction.

10. Standard getters for balance and allowance are not reverting ✔️

11. Privileged operations ✔️

There is a single owner, that can exclusively run the below functions.

There is no other privileged role.

- mint
- pause
- unpause
- renounceOwnership
- transferOwnership

12. Initialization can only be done once ✔️

- Initializer flags are false at construct time
- Once set, initializer flags cannot be turned off
- Only the initialize function can set the initializer flags

13. Proxy standard rules ✔️

- Implementation is immutable unless upgradeTo or upgradeToAndCall are called (assuming delegatee is not touching the implementation's storage slot)
- All external calls defined in the proxy other than the fallback are privileged.