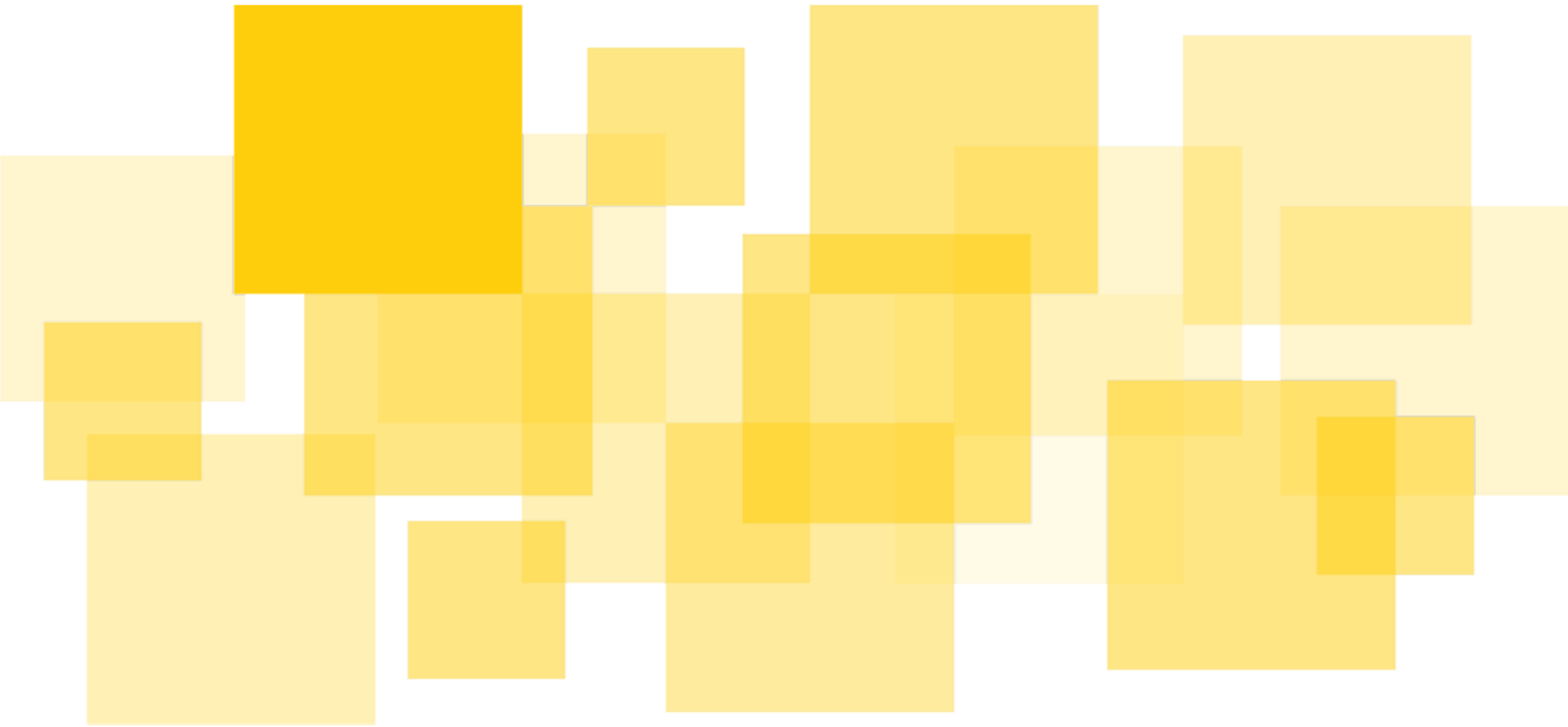
Security Audit Report

Pact.fi Router

Delivered: 10 May 2022



Prepared for Pact Finance by

runtime verification

Contents

Summary

Pact Finance engaged Runtime Verification Inc to conduct a security audit of their smart contract implementing a router interface for executing swaps across multiple Pact pools.

The objective was to review the contract's business logic and implementation in PyTeal and identify any issues that could potentially cause the system to malfunction or be exploited.

Timeline

The audit has been conducted over a period of 6 weeks, from March 7, 2022 to March 31, 2022 and from April 11, 2022 to April 22, 2022.

Findings

The audit has identified two low-severity issues, and one informative finding.

The low-severity issues are related to withdrawing (finding A1) and depositing (finding A2) maintenance funds and do not affect user funds. The informative finding B1 highlights an inherent feature of Algorand smart contract composability that the users of the Pact router, or any other composable Algorand smart contract, need to be aware of.

Both low-severity issues have been appropriately addressed. The informative finding is related to the nature of the Algorand blockchain and cannot be mitigated without breaking composability. Instead, the users must be educated about it through dApp documentation and the wallets must provide clear interface for transaction group examination.

Disclaimer

This report does not constitute legal or investment advice. The preparers of this report present it as an informational exercise documenting the due diligence involved in the secure development of the target contract only, and make no material claims or guarantees concerning the contract's operation post-deployment. The preparers of this report assume no liability for any and all potential consequences of the deployment or use of the contract.

Smart contracts are still a nascent software arena, and their deployment and public offering carries substantial risk. This report makes no claims that its analysis is fully comprehensive, and recommends always seeking multiple opinions and audits.

This report is also not comprehensive in scope, excluding a number of components critical to the correct operation of this system.

The possibility of human error in the manual review process is very real, and we recommend seeking multiple independent opinions on any claims which impact a large quantity of funds.

Goals

The goals of the audit were:

- Review the architecture of the router smart contract based on the provided documentation;
- Review the PyTeal implementation of the contract to identify any programming errors;
- Build an executable model of the contract and conduct extensive testing of the contract's life cycle leveraging pattern-based fuzzing in a simulated blockchain environment.

The audit focuses on trying to identify issues in the system's logic and its implementation that could potentially render the system vulnerable to attacks or cause it to malfunction.

Scope

The audit has been conducted on the commit `594279180d7ee7ffc0987423e520c50312e72d48` of Pact Finance's private GitHub Repository.

The audit scope includes the following files:

- The Router smart contract source code in PyTeal: ./assets/router.py
- Transaction validation code: ./assets/helpers/validation.py

Additionally, we have informally reviewed the `pytealext` library code referenced in `exchange.py`:

- Inner transaction preparation helpers in ./pytealext/inner_transactions.py

In a previous audit (public report available), we have reviewed the implementation of the Pact Automated Market Maker (AMM) liquidity pool. Since that previous audit completion, the pool contract's code was slightly refactored. However, there are no changes in functionality or on-chain interface. We use the updated implementation in our simulations as a black box, focusing on its interactions with the router contract.

We have not audited any off-chain components of the system. Further patches and developments following the aforementioned commit were not formally audited. We, however, have informally reviewed the patch for the findings A1 and A2 (commit hash c19ae0634d0dbaf226c7a49a4c847ca3488366f9).

Methodology

Audit process

The audit process has included the following stages:

1. Manual code review of the router contract
2. Integration of the router and pool contracts' PyTeal source code into the simulation environment
3. Testing of every privileged and non-privileged endpoint of the router contract
4. Pattern-based fuzzing of the router contract and several exchange contracts

Stages 1 and 2 create the foundation for the rest of the audit: the auditors gain deep understanding of the contracts' inner structure and obtain an executable model of the contract. The low-severity findings A1 and A2 have been discovered at stage 1 and confirmed at stage 3. Stage 4, although being the most comprehensive, has only led to the informative finding B1.

Findings

A1. Withdrawing maintenance funds from the contract's account is not possible (Severity: Low)

Description

The router contract holds a small balance of Algos to perform ASA opt-ins, and the balance is fully controlled by the contract's administrator account. For the administrator to withdraw Algos, the contract provides the `WITDRAW` ABI method. However, this method is not handled in the contracts method dispatch logic, hence the withdrawal is not possible.

Since the contract's code is permitted to be upgraded by a privileged account, the issue would be easy to resolve even post-deployment.

Recommendation

Dispatch the `WITDRAW` ABI method to be handled by the `on_withdraw` function.

Status

Acknowledged by Pact. The issue has been addressed in a subsequent commit c19ae0634d0dbaf226c7a49a4c847ca3488366f9. The changes were not formally audited.

A2. Depositing maintenance funds into the contract's account overrides the balance tracking variable (Severity: Low)

Description

The router contract holds a small balance of Algos to perform ASA opt-ins, and the balance is fully controlled by the contract's administrator account. When the administrator deposits Algos, the contract updates the `TOTAL_DEPOSIT` global state variable to track the available balance. However, instead of increasing the amount by the administrator's payment value, the contract overrides the amount on every deposit, therefore the tracked amount diverges from the actual contract's account Algo balance.

Recommendation

In `on_deposit`, the `TOTAL_DEPOSIT` global state variable must be increased, rather than overridden.

Status

Acknowledged by Pact. The issue has been addressed in a subsequent commit c19ae0634d0dbaf226c7a49a4c847ca3488366f9. The changes were not formally audited.

B1. (Severity: Informative)

Description

Pact's router smart contracts is *composable*, meaning that it allows the transaction group to include additional transactions which are not related to any interaction with the router. While composability is certainly a good thing, it is a double-edged sword.

Attack scenario

The composability can only be maliciously exploited under two conditions:

- The front-end of the Pact's dApp is compromised, allowing the attacker to serve users with a modified transaction group;
- The user does not examine the transaction group before signing and submitting it, allowing the malicious transaction to slip through.

If these conditions are met, the attacker can pad a valid transaction group with a payment transaction that drains all Algos from the users account. The malicious transaction **cannot** be interleaved with the router-related transactions and must be include either at the start or at the end of the group.

Recommendation

We do not think that Pact, or any other dApp, should (or even can) force their users to go through the raw transaction group before signing and submitting them. We, however, think that the **wallet** vendors must provide clear UI/UX for the user to easily verify which transactions they are signing.

We include this information for purely informative purposes. We advocate that the users of composable dApps on Algorand must be made aware of this peculiarity of the platform by the documentation page of every such dApp.