PACT Finance Security Assessment

Findings and Recommendations Report Presented to:

Algorand Foundation and PACT Finance

July 14, 2022

Version: 2.0

Presented by:

Kudelski Security, Inc.
5090 North 40th Street, Suite 450
Phoenix, Arizona 85018

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

Overview

Algorand Foundation and PACT Finance engaged Kudelski Security to perform a PACT Finance Security Assessment.

The assessment was conducted remotely by the Kudelski Security Team. Testing took place on May 24 - June 06, 2022, and focused on the following objectives:

- Provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the result of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the Kudelski Security Teams took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key Findings

The following are the major themes and issues identified during the testing period. These, along with other items, within the findings section, should be prioritized for remediation to reduce the risk they pose:

- KS-PACT-01 – Deposit of ALGO is not checked against the source asset of the first swap

During the test, the following positive observations were noted regarding the scope of the engagement:

- The team was very supportive and open to discussing the design choices made

Based on formal verification we conclude that the reviewed code implements the documented functionality.

Scope and Rules of Engagement

Kudelski performed a PACT Finance Security Assessment. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through a private repository at https://github.com/pactfi/pact-contracts/ with the commit hash 3249cf4da3eb956c9658d3133b77550b8118df45.

A re-review of the recommended fixes was performed on July 1, 2022, with the commit hash 88e5ea256e07e2541658fbf824f3aacbb3f0ffaf.

Files included in the code review

```
pact-fi/
├── abi/
│   └── router_interface.json
├── assets/
│   ├── helpers/
│   │   ├── __init__.py
│   │   ├── abi.py
│   │   └── validation.py
│   ├── __init__.py
│   ├── contract.py
│   ├── exchange.py
│   ├── router.py
│   ├── stable_exchange.py
│   └── stake.py
├── devnet/
│   ├── primary/
│   │   ├── exchange-v1/
│   │   │   ├── Wallet1.0.3000000.partkey
│   │   │   └── Wallet1.rootkey
│   │   ├── goal.cache/
│   │   │   └── walletHandles.json
│   │   ├── kmd-v0.5/
│   │   │   ├── kmd.token
│   │   │   └── kmd_config.json.example
│   │   ├── agreement.cdv
│   │   ├── algod.admin.token
│   │   ├── algod.token
│   │   ├── config.json
│   │   └── genesis.json
│   ├── Dockerfile
│   ├── Wallet1.0.3000000.partkey
│   ├── Wallet1.rootkey
│   ├── devnet.json
│   ├── genesis.json
│   ├── network.json
│   └── run.sh
```

```
├── docs/
│   ├── README.md
│   ├── amm.md
│   └── router.md
├── scripts/
│   ├── __init__.py
│   ├── common.py
│   ├── compare_teal.sh
│   ├── compile.py
│   └── deploy.py
├── Dockerfile
├── Makefile
├── README.md
├── docker-compose.yml
├── lint.sh
├── poetry.lock
├── pyproject.toml
├── pytest.ini
└── setup.cfg
```

Table 1: Scope

TECHNICAL ANALYSIS & FINDINGS

During the PACT Finance Security Assessment, we discovered:

- 1 finding with HIGH severity rating.
- 1 finding with MEDIUM severity rating.
- 2 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

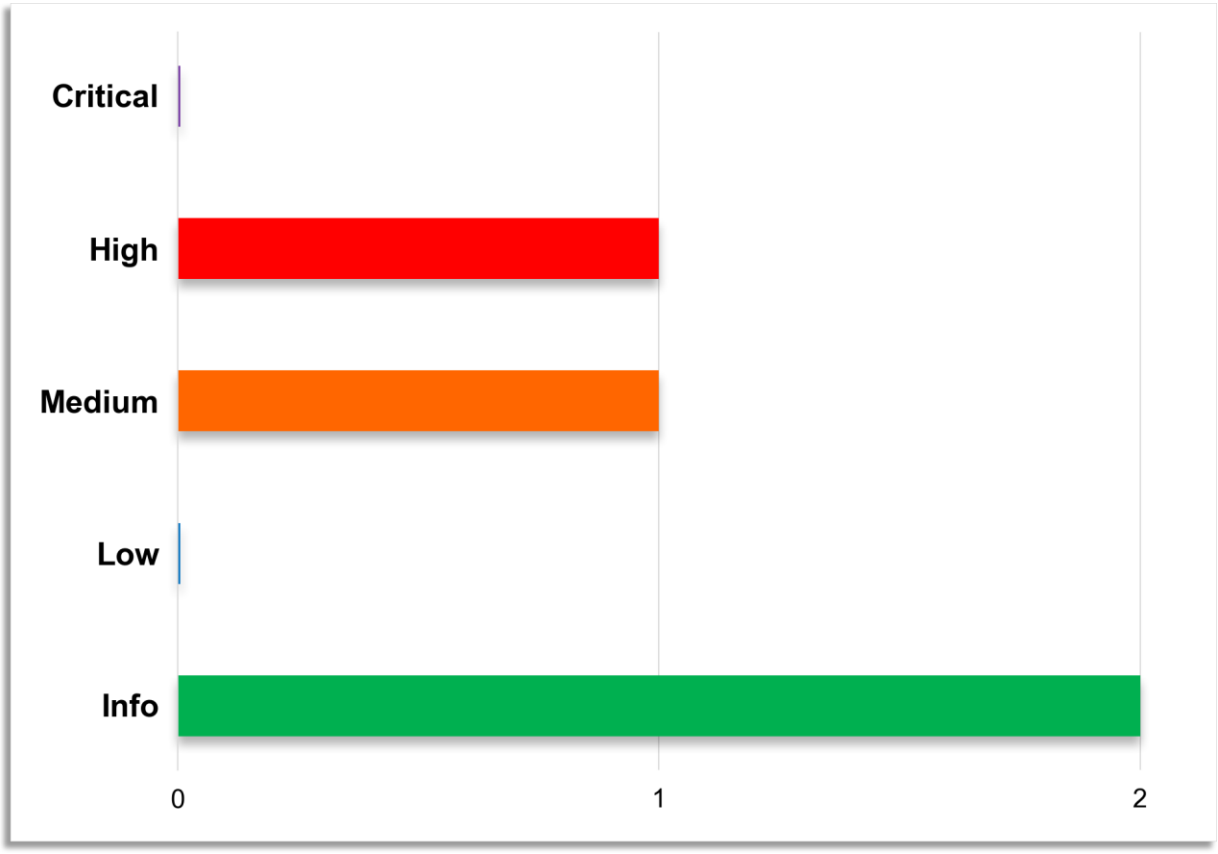

Figure 1: Findings by Severity

Findings

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

#	Severity	Description	Status
KS-PACT-01	**High**	Deposit of ALGO is not checked against the source asset of the first swap	**Remediated**
KS-PACT-02	**Medium**	Users cannot add liquidity if one of the tokens in the pool is drained	**Remediated**
KS-PACT-03	**Informational**	Non-Network tokens	**Open**
KS-PACT-04	**Informational**	Non-pythonic code	**Open**

Table 2: Findings Overview

Technical Analysis

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality.

Conclusion

Based on formal verification we conclude that the code implements the documented functionality to the extent of the reviewed code.

Technical Findings

General Observations

PACT Fi is a decentralized token exchange exchange between Algo and ASAs, or ASAs and other ASAs, written in Pyteal, built for the Algorand network.

Code reviewed

The Algorand smart contracts focused on in this review were the following:

- Exchange - ./assets/exchange.py
- Stable Exchange - ./assets/stable_exchange.py
- Router - ./assets/router.py

During the test, the following positive observations were noted regarding the scope of the engagement:

- Handles assertions correctly
- Checks for int overflows
- Checks transactions fields (AssetAmount, AssetReceiver, etc.)
- Makes onCompletion checks
- Uses Type and TypeEnum to check if something is a payment or asset
- Multiplies integers before division
- Uses scratch variables correctly
- Checks asset type and sender
- Checks for duplicate transactions, correct amounts, etc.

Summary

Code quality is good, operations carried out carefully.

Deposit of ALGO is not checked against the source asset of the first swap

Finding ID: KS-PACT-01
Severity: **High**
Status: **Remediated**

Description

A valid swap group transaction should look like this

1. opt-in to assets: $PLANET, $USDC
2. deposit of $PLANET to the router contract
3. swap (1 pool): $PLANET to $USDC
4. opt-out of assets: $PLANET, $USDC

The smart contract should enforce this by checking if the deposited asset is the source of the first swap. However, if users deposit ALGO, the contract does not check the first swap against the deposit. Thus, users can deposit ALGO, but swap it for another asset instead of ALGO. For example, this group transaction is valid:

1. opt-in to assets: $PLANET, $USDC
2. deposit of **ALGO** to the router contract
3. swap (1 pool): $**PLANET** to $USDC
4. opt-out of assets: $PLANET, $USDC

Even though the router does not own any assets or Algo. This group transaction should not be valid as it can cause users to lose their deposits.

Proof of Issue

File name: router.py
Line number: 710-734

```
    def _validate_swap_deposit_txn(self, txn: TxnObject) -> Expr:
        """
        Checks if deposit txn before SWAP is valid.

        Checks if asset id is same as first asset in current app call application
args and
        address of receiver is the same as contract's.
        """

        first_swap_arguments = extract_first_swap(Txn.application_args)
        return Assert(
            Or(
                LazyAnd(
                    txn.type_enum() == TxnType.AssetTransfer,
                    validate_asset_transfer(
                        txn,
                        asset_id=first_swap_arguments.source_asset,
                        receiver=Global.current_application_address(),
                    ),
                ),
                LazyAnd(
                    txn.type_enum() == TxnType.Payment,
                    validate_algos_transfer(txn,
receiver=Global.current_application_address()),
                ),
```

```
            )
        )
```

If the deposit is an asset, it is checked against the source asset of the first swap via the `validate_asset_transfer` function. However, it is not checked if the deposit is ALGO.

Severity and Impact Summary

This group transaction should not be valid as it can cause users to lose their deposits.

Recommendation

We recommend adding `Assert(first_swap_arguments.source_asset == 0)` before validating the algos transfer.

Users cannot add liquidity if one of the tokens in the pool is drained

Finding ID: KS-PACT-02
Severity: **Medium**
Status: **Remediated**

Description

In stable_exchange.py, it is possible for one of the tokens to be drained. If this happens, the total number of tokens `total_primary` or `total_secondary` is 0 but `total_liquidity` is not 0. Furthermore, with the current implementation of function `add_liquidity()`, there is no way to add more liquidity because of a division by 0 error.

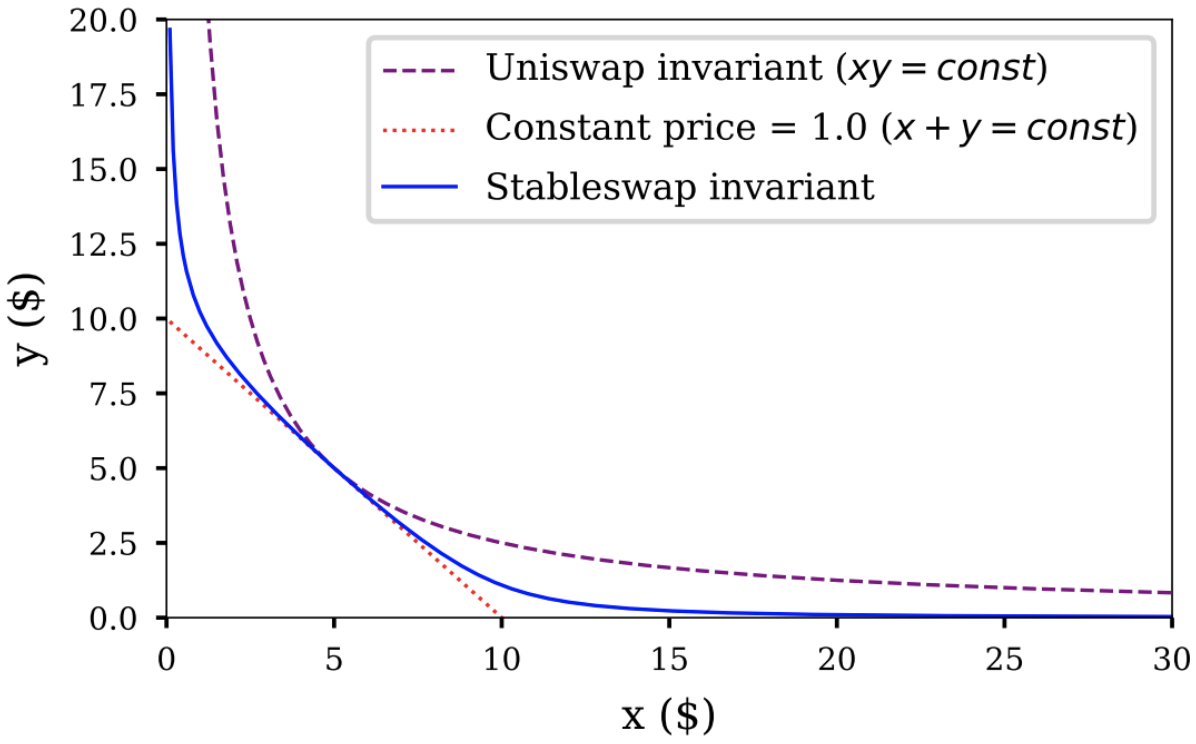

Figure 2: Comparison of StableSwap invariant with Uniswap (constant-product) and constant price invariants

Proof of Issue

File name: stable_exchange.py
Line number: 166-177

```
    D_P.store(
                    BytesDiv(
                        BytesMul(D_P.load(), D.load()),
                        BytesMul(Itob(total_primary), Itob(number_of_tokens)),
                    )
                ),
                D_P.store(
                    BytesDiv(
```

```
                        BytesMul(D_P.load(), D.load()),
                        BytesMul(Itob(total_secondary), Itob(number_of_tokens)),
                    )
                ),
```

Severity and Impact Summary

New users cannot add more liquidity. Moreover, to reset the pool, all users would have to remove their liquidity. It would only take one malicious user to not remove their liquidity and block the reset.

Recommendation

We recommend the team consider this case in the `add_liquidity()` function or ensure that `y>0` (the number of remaining tokens in the pool) so that the token can't be drained.

References

StableSwap - efficient mechanism for Stablecoin liquidity

Non-Network tokens

Finding ID: KS-PACT-03
Severity: **Informational**
Status: **Open**

Description

Non-Network tokens (NNTs) can have different behaviors from network tokens in a variety of ways.

Severity and Impact Summary

NNTs might have the option to freeze all future transfers. NNTs must also be pre-approved before transfer, meaning the receiver must also agree to the transfer beforehand. Tokens minted to Algorand Standard Assets (ASAs) can have extra functionality that can cause unexpected behavior (freezing, clawback, reserves, and separate managers). On Algorand, NNTs can also built into the network, and generally have predictable and stable behavior.

Recommendation

While this does not currently seem to be a problem, it is a difference to be mindful of.

References

- Algorand Developer Portal: Algorand Standard Assets (ASAs)

Non-pythonic code

Finding ID: KS-PACT-04
Severity: **Informational**
Status: **Open**

Description

Many parts of the code are written in a non-pythonic way. Python's style for code includes that "explicit is better than implicit" and "readability counts".

Proof of Issue

File name: stable_exchange.py

Example:

```
c = BytesDiv(BytesMul(BytesMul(D, BytesMul(D, D)), a_precision_bytes),
BytesMul(Itob(Int(4)), BytesMul(P, Ann)))
    b_q = ScratchVar()
    c_q = ScratchVar()
```

Severity and Impact Summary

While this is encouraged in languages that favor brevity, such as Golang, Python favors explicit variable names to make understanding it as easy as possible.

Recommendation

It is recommended to always follow pythonic code styles when writing in python.

Resources

Python Guide

METHODOLOGY

Kudelski Security uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

Figure 3: Methodology Flow

Kickoff

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

Ramp-up

Ramp-up consists of the activities necessary to gain proficiency on the particular project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project

3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general list and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

Reporting

Kudelski Security delivers a preliminary report in PDF format that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project. We may conclude that the overall risk is low but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We not only report security issues identified but also informational findings for improvement categorized into several buckets:

- Critical
- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps, that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

Verify

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes withing a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

Additional Note

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessment the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. These is a solid baseline for severity determination.

The Classification of identified problems and vulnerabilities

There are four severity levels of an identified security vulnerability.

Critical – vulnerability that will lead to loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets
- The complexity to exploit is low
- The probability of exploit is high

High - A vulnerability that can lead to loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that cannot be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material
- Weak encryption of keys
- Badly generated key materials
- Tx signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - a vulnerability that hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-revied crypto functions
- Program crashes leaves core dumps or write sensitive data to log files

Low - Problems that have a security impact but does not directly impact the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations