VANTAGEPOINT
security at the speed of development

Smart Contract Audit of Folks Finance V2

Final Report

Prepared for:

Folks Global LTD

8 Dec 2022

FOR GENERAL DISTRIBUTION

Table Of Content

1. Executive Summary 3
   1.1. Overview 3
2. Project Details 5
   2.1. Scope 5
   2.2. Status 5
3. Risk Assessment 7
   3.1. Summary of Vulnerabilities 7
   3.2. Vulnerabilities Statistics 8
4. Detailed Description of Vulnerabilities 9
   4.1. ARC-04 Smart Contract with Router Class without Pragma or pragma 9
   4.2. Lack of Validation for Compound Types 11
   4.3. Potential Manipulation of Application States using Flash Loans 13
   4.4. Inaccurate Comments 15
5. Appendix 17
   5.1. Disclaimer 17
   5.2. Risk Rating 17

1. Executive Summary

1.1. Overview

Vantage Point Security Pte Ltd was engaged by Folks Finance to conduct an Algorand smart contract audit of Folks Finance V2 protocol. Folks Finance V2 protocol is a capital markets protocol for borrowing and lending on Algorand blockchain with the changes for the following.

- Multi-collateral and multi-borrow loans
- Stable interest rate loans
- Repayment of borrows using existing collateral
- Swap collateral
- Flash loans
- Composability

The Algorand smart contract audit was conducted based on the following materials provided.
Supporting Documents

- Folks Finance V2 Lending Formulae
- Folks Finance Lending V2 Economic Model
- Folks Finance Lending V2 Technical Design

PyTeal Code Repo

- https://github.com/blockchain-italia/ff-vp-contracts

Vantage Point performed this audit by first understanding the Folks Finance V2 protocol's business logic based on the documents provided. We sought clarifications on potential issues, discrepancies, flaws and plans on how manual operations involved would be carried out through discussions with the Folks Finance team.
The smart contract audit was conducted on the provided PyTeal code to identify any weaknesses, vulnerabilities, and non-compliance to Algorand best practices. As part of the audit process, Vantage Point executed a set of baseline test cases tailored to smart contracts in scope. Out of 31 test cases generated at the commencement of the audit, 3 test cases noted below have failed but have been resolved based on additional commits made by the Folks Finance team.

- Inaccurate Comments
- Lack of Validation for Compound Types
- Lack of pragma or Pragma for Smart Contracts with Router Class

Above failed test cases have been noted documented as findings. Another low risk finding was identified based on the review of Folks Finance Lending V2 Economic Model document.
Total of 3 low and 1 observational findings were identified from the smart contract audit.
Potential Manipulation of Application States Using Flash Loans

- Loan V2 introduces stable interest rate loans which allow users to have fixed interest rate as long as the rebalance conditions are not met. In addition, Folks Finance also enforces the limit of how much of the remaining liquidity can be borrowed at a stable interest rate. However, with the use of flash loans, it may be possible to introduce large liquidity to the pool of an asset to temporarily increase the liquidity of the pool to execute a stable loan at a lower interest rate with a larger principal amount.

Lack of Validation for Compound types

- It is strongly recommended to validate inputs for compound types such as abi.Address as the Router class does not validate inputs for such. Specifically, abi.Address is not guaranteed to have length of 32 bytes and it is recommended to validate it before committing it as a state or use it for any state-changing logic.

ARC-04 Smart Contract with Router Class without Pragma or pragma

- PyTeal introduced Router class which adheres to ARC-04 ABI conventions and handles routing of base app calls and methods. However, as the Router class is still expecting backward-incompatible changes, it is recommended to use Pragma/pragma expression to pin the version of the PyTeal compiler in the source code.

Inaccurate Comments

- Based on the documentation provided about the protocol, instances were noted where the smart contract's code comments did not match the description of the formulae and the decimal points for expected input or output.

The outcome of this Algorand smart contract audit is provided as a detailed technical report that provides the project owners a full description of the vulnerabilities identified, the associated risk rating for each vulnerability, and detailed recommendation that will resolve the identified technical issue.

2. Project Details

2.1. Scope

App Name	Smart Contract Audit of Folks Finance V2
Testing Window	17 Oct 2022 to 25 Nov 2022
Target URL	https://github.com/blockchain-italia/ff-vp-contracts
Svn / Git Revision Number	4c542ab840a409d1f60aef6a2fb87c969292c773
Project Type	Smart Contract Audit
App Type	Algorand Smart Contract
Items Completed	contracts/loan/loan.py contracts/loan/loan_state.py contracts/loan/pool_inner_txns.py contracts/oracle/lp_token_oracle.py contracts/oracle/lp_token_oracle_state.py contracts/oracle/oracle_adapter.py contracts/oracle/oracle_adapter_state.py contracts/oracle/refresh_prices_inner_txn.py contracts/pool/deposits.py contracts/pool/deposits_state.py contracts/pool/pool.py contracts/pool/pool_manager.py contracts/pool/pool_manager_state.py contracts/pool/pool_state.py contracts/pool/withdraw_inner_txn.py
Issue Opening Date	19 Oct 2022 ● ARC-04 Smart Contract with Router Class without Pragma or pragma [Low] 9 Nov 2022 ● Lack of Validation for Compound Types [Low] 26 Nov 2022 ● Potential Manipulation of Application States using Flash Loans [Low] 22 Nov 2022 ● Inaccurate Comments [Observational]
Issue Closing Date	6 Dec 2022 ● ARC-04 Smart Contract with Router Class without Pragma or pragma [Low] ● Lack of Validation for Compound Types [Low] ● Inaccurate Comments [Observational]

2.2. Status

Component	Review Type	Status
Algorand Smart Contract	Smart Contract Audit	Completed

3. Risk Assessment

This chapter contains an overview of the vulnerabilities discovered during the project. The vulnerabilities are sorted based on the scoring categories CRITICAL, HIGH, MEDIUM and LOW. The category OBSERVATIONAL refers to vulnerabilities that have no risk score and therefore have no immediate impact on the system.

3.1. Summary of Vulnerabilities

Vulnerability Title	Risk Score	Closed
ARC-04 Smart Contract with Router Class without Pragma or pragma	Low	☑
Lack of Validation for Compound Types	Low	☑
Potential Manipulation of Application States using Flash Loans	Low	☐
Inaccurate Comments	Observational	☑

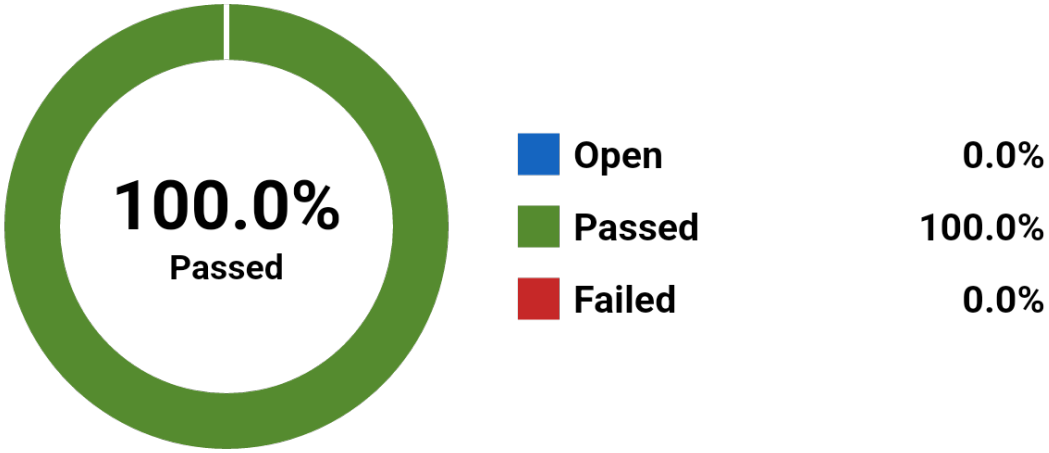
3.2. Vulnerabilities Statistics

Findings Overview

Total	Open	Closed
4	1	3

0	Critical	No findings
0	High	No findings
0	Medium	No findings
3	Low	1 Open, 2 Closed.
1	Observational	0 Open, 1 Closed.

Test Case Status



4. Detailed Description of Vulnerabilities

4.1. ARC-04 Smart Contract with Router Class without Pragma or pragma

Low

Closed

BACKGROUND

ARC-04 Algorand Smart Contract Transaction Calling Conventions is a standard for encoding smart contract's method calls. ARC-04 allows wallets and dapp frontends to properly encode call transactions based on the interface description and explorers to show details of invoked methods. Smart contracts complying to ARC-04 standards respond to both method calls and bare app calls. To make it easier for an application to route across many base app calls and methods, PyTeal introduced Router class which adheres to the ARC-4 ABI conventions with respect to when methods and base app calls. As Router class is still expecting backwards-incompatible changes, it is recommended use pragma/Pragma expression to pin the version of PyTeal in the source code.

DESCRIPTION

Affected Code/File

- https://github.com/blockchain-italia/ff-vp-contracts/blob/master/contracts/loan/loan.py
- https://github.com/blockchain-italia/ff-vp-contracts/blob/master/contracts/oracle/lp_token_oracle.py
- https://github.com/blockchain-italia/ff-vp-contracts/blob/master/contracts/oracle/oracle_adapter.py
- https://github.com/blockchain-italia/ff-vp-contracts/blob/master/contracts/pool/deposits.py
- https://github.com/blockchain-italia/ff-vp-contracts/blob/master/contracts/pool/pool.py
- https://github.com/blockchain-italia/ff-vp-contracts/blob/master/contracts/pool/pool_manager.py

The smart contract in scope is an ARC-04 compliant smart contract which makes use of PyTeal's Router class for routing contract's methods. About Router class, PyTeal Documentation states below.

PyTeal Documentation

- Warning - Router usage is still taking shape and is subject to backwards incompatible changes. Feel encouraged to use Router and expect a best-effort attempt to minimize backwards incompatible changes along with a migration path. For these reasons, we strongly recommend using pragma or the Pragma expression to pin the version of PyTeal in your source code. See Version Pragmas for more information.

As noted above, Router class is still expecting changes and therefore, it is strongly recommended to use Pragma or pragma to pin the version of PyTeal in the source code. However, in the affected smart contracts, pragma or Pragma was not observed.

IMPACT

As PyTeal's Router class may have backward-incompatble changes in the future, if the PyTeal compiler version is not pinned, this PyTeal smart contract may compile in an expected way which may result in an unwanted routing of application calls.

RECOMMENDATIONS

It is recommended to make use of Pragma or pragma to pin the version of PyTeal in the source code to cater for potential backward-incompatable changes to the Router class.

Sample Code

- pragma(compiler_version="0.17.0")

Do note that above is an example and correct range of compiler version should be determined and specified in the code.

COMMENT

Reviewed on 19 Oct 2022

Based on the commit ca6d859245a477088a9978d9de3f8043466d371a, the version of PyTeal compiler has been pinned to 0.18.1 for the following smart contracts. This issue is closed.

- contracts/loan/loan.py
- contracts/oracle/lp_token_oracle.py
- contracts/oracle/oracle_adapter.py
- contracts/pool/deposits.py
- contracts/pool/pool.py
- contracts/pool/pool_manager.py

REFERENCES

PyTeal - ABI Support
https://pyteal.readthedocs.io/en/stable/abi.html?highlight=Router#abi-support

PyTeal - Pragma
https://pyteal.readthedocs.io/en/stable/api.html?highlight=Router#pyteal.Pragma

4.2. Lack of Validation for Compound Types

Low

Closed

BACKGROUND

PyTeal's Router class does not validate inputs for compound types such as abi.StaticArray, abi.Address, abi.DynamicArray, abi.String, or abi.Tuple). It is strongly recommended to have the methods immediately access and validate compound type parameters before committing them for later transactions.

DESCRIPTION

It was noted that the affected smart contracts did not have sufficient length validation against abi.Address. As also highlighted in the PyTeal documentation, it is recommended to validate compound types such as abi.Address for its validity prior to persisting arguments for later transactions.

Affected Code/File

- https://github.com/blockchain-italia/ff-vp-contracts/blob/master/contracts/pool/pool_manager.py [create, update_admin]
- https://github.com/blockchain-italia/ff-vp-contracts/blob/master/contracts/oracle/oracle_adapter.py [create, update_admin]
- https://github.com/blockchain-italia/ff-vp-contracts/blob/master/contracts/oracle/lp_token_oracle.py [create, update_admin]
- https://github.com/blockchain-italia/ff-vp-contracts/blob/master/contracts/pool/pool.py [create, update_admin]
- https://github.com/blockchain-italia/ff-vp-contracts/blob/master/contracts/loan/loan.py [create, update_admin]

Instance 1 - pool_manager.py [create, update_admin]
It was noted that pool_manager smart contract's create and update_admin methods require a compound type parameter abi.Address. However, its validity is not validated before the address gets committed as the value for global state "admin_key".

Instance 2 - oracle_adapter.py [create, update_admin]
It was noted that oracle_adapter smart contract's create and update_admin methods require a compound type parameter abi.Address. However, its validity is not validated before the address gets committed as the value for global state "admin".

Instance 3 - lp_token_oracle.py [create, update_admin]
It was noted that the lp token oracle smart contract's create and update_admin methods require a compound type parameter abi.Address. However, its validity is not validated before the address gets committed as the value for global state "admin".

Instance 4 - pool.py [create, update_admin]
It was noted that the pool smart contract's create and update_admin methods require a compound type parameter abi.Address. However, its validity is not validated before the address gets committed as the value for global state pool_admin, params_admin, configs_admin and loans _admin.

Instance 5 - loan.py [create, update_admin]
It was noted that the loan smart contract's create and update_admin methods require a compound type parameter abi.Address. However, its validity is not validated before the address gets committed as the value for global state params.

IMPACT

As the Router class does not validate inputs for compound types, if not validated, compound types with incorrect format may cause an unwanted behavior. For example, abi.Address is not guaranteed to be exactly 32 bytes and such should be manually verified.

RECOMMENDATIONS

For each input which are compound types, ensure there is sufficient validation for each methods prior to persisting them for future transactions.

COMMENT

Reviewed on 20 Nov 2022

Based on the updates made in commit 16560ca2a9d1171652cf374ee0aed05d8fc88423, instance 1, 2, 3, 4 and 5 are closed.
New subroutine address_length_check was introduced within checks.py to validate if the length of abi.Address is exactly 32 bytes. Subroutine address_length_check has been added to affected methods for validation of abi.Address parameter before they get committed in the global state.

4.3. Potential Manipulation of Application States using Flash Loans

Low

Open

BACKGROUND

Flash loans allow anyone to borrow large amount of assets without collateral as long as the assets are paid back along with applicable flash loan fees. However, large amount of assets could be used to change specific parameters that may be used by the protocol to calculate business logic and validations in place, such as interest rate and borrow cap percentage.

DESCRIPTION

Affected File/Code

- https://github.com/blockchain-italia/ff-vp-contracts/blob/master/contracts/common/formulae.py
- https://github.com/blockchain-italia/ff-vp-contracts/blob/master/contracts/pool/pool.py
- https://github.com/blockchain-italia/ff-vp-contracts/blob/master/contracts/loan/loan.py

It was noted that the parameters involved in executing a stable interest rate loan can be manipulated with the use of large liquidity, which can be either taken from other protocols that support flash loans or Folks Finance Lending V2's flash loan.
Based on the provided documentation and code, the stable borrow interest rate depends on the **Utilisation** compared to its optimal value and total stable debt to **TotalDebt** ratio.
If an attacker takes a flash loan and deposits a large amount of assets to the pool to increase the value of **TotalDeposits**, **Utilisation** value decreases and results in a lower stable borrow interest rate.
In addition, there also exists a **check_borrow_cap** validation which checks for the following.

- new total debt amount after adding the new borrow amount is less or equal to **caps_state.BORROW_CAP** in dollar value
- if it is a stable interest rate loan, the loan amount is equal to or less than the **caps_state.STABLE_BORROW_PERCENTAGE_CAP** multiplied by the **availableLiquidity**

Where **availableLiquidity** is calculated as **TotalDeposit** - **TotalDebt**.
With a flash loan, it is possible for a user to deposit a large amount of assets to a pool to increase the **TotalDeposits** value and eventually have higher limit for the amount of stable interest rate loan that can be granted.
After creating a loan at condition that may be beneficial to the attacker, the attacker would just have to withdraw the liquidity supplied from the pool and pay back the flash loan amount with fees, within the same transaction group.

IMPACT

If the protocol relies on specific parameters for validation and calculation of interest rates or similar values, flash loans can be used to manipulate them within the atomic transaction group. In this scenario, the attacker may be able to get a loan position with significantly higher amount than allowed by the protocol or with lower interest rate.

RECOMMENDATIONS

Following can be considered when managing and implementing protocol logic and parameters

- Defensive approach for the slope of rate increase for both variable and interest rate loans
- Possibility of flash loans being used in the transaction groups for each operation
- External flash loans being made available for composable operations

In addition, for high value transactions, the project team may be able to implement following compensating measures

- Make the operation that updates the stable interest rate accessible so that any stable interest rate loans with very low interest rate can be updated to the correct rate easily
- Test, update and monitor key parameter values related to the slope for interest rates or similar values when transactions with large volume occur
- Flash loan fees should be kept to a level that any manipulative behavior to the protocol becomes economically illogical to execute

COMMENT

Reviewed on 30 Nov 2022

While the finding is kept open, followings were noted.

- Anyone is allowed to execute rebalance operations for stable interest rate loans to update the stable interest rate of existing loans as long as the required conditions are met.
- Folks Finance team's respective admin account can update the key parameters for the slope of the interest rate in case such is necessary
- Based on current conditions, stable interest rate loans tend to have higher interest rate than variable interest rates so there is less incentive for someone to incur flash loan fees to get a stable interest loan with higher interest rate, with a possibility of being rebelanced up by anyone as long as the conditions are met.

Similar findings have been identified from other Defi protocols across chains such as Aave V2 where large liquidity (possibly from flash loan) could be used to manipulate the interest rates and such problem is not an easy issue to handle. One recommended approach to mitigate possible risks to the protocol would be to be defensive in the implentation and operation of the protocol and make sure such abusive behavior damages neither the protocol nor non-abusive protocol participants. Key parameters of the protocols should be consistently monitored, reviewed and updated as an when necessary in the interest of the protocol and participants.

REFERENCES

Consensys Diligence - Aave Protocol V2 - 5.6 Potential manipulation of stable interest rates using flash loans
https://consensys.net/diligence/audits/2020/09/aave-protocol-v2/#potential-manipulation-of-stable-interest-rates-using-flash-loans

4.4. Inaccurate Comments

Observational

Closed

BACKGROUND

Code comments play a vital role in providing more human-readable context to the readers of the code so that technical specifications, transaction details and the expectations can be clearly conveyed to the users or future developer team members. Ensuring
Inaccurate comments could cause confusion to the developers and users who rely on the information provided in understanding the logic of smart contracts. Confusion in the business logic and technical specifications may not be critical in the short-term but for the continuity and maintainability of the smart contract projects, it is important to have the most updated details reflected in comments.

DESCRIPTION

Affected File/Code:

- https://github.com/blockchain-italia/ff-vp-contracts/blob/master/contracts/common/formulae.py [calc_stable_borrow_interest_rate, calc_borrow_balance]

It was noted that the affected contracts had incorrect comments which do not align with the documentation or code.
Instance 1 - formuale.py [calc_stable_borrow_interest_rate]
The calc_stable_borrow_interest_rate is used to calculate the stable borrow interest rate of a pool but the comments within the code described the details for the variable borrow interest rate. However, the logic for calculating the variable borrow interest rate was in line with the intended logic for stable borrow interest rate and therefore, only an update to the comments are required.
Instance 2 - formuale.py [calc_borrow_balance]
The calc_borrow_balance subroutine is used to calculate the borrow balance of the loan at time t and take borrow balance and interest indexes as arguments. However, the smart contract's code comments had incorrect comments on the decimal points of the bii_t argument. As all interest indexes within the protocol use 14 decimal points, such should be reflected in the code comments to avoid confusion.

IMPACT

Inaccurate comments could cause confusion to the developers and users who rely on the information provided in understanding the logic of smart contracts. Confusion in the business logic and technical specifications may not be critical in the short-term but could potentially end up with a wrong implementation which may have rooted from a wrong understanding of the existing smart contract logic.

RECOMMENDATIONS

Update comments according to the latest business logic and technical specification to ensure the understanding of the developers who read the code are coherent.

COMMENT

Reviewed on 5 Dec 2022

Based on the commit 15915e389b4d76e120d4d512792682f1347a3379, this issue is closed as both instances 1 and 2 have been updated with correct comments.

5. Appendix

5.1. Disclaimer

The material contained in this document is confidential and only for use by the company receiving this information from Vantage Point Security Pte. Ltd. (Vantage Point). The material will be held in the strictest confidence by the recipients and will not be used, in whole or in part, for any purpose other than the purpose for which it is provided without prior written consent by Vantage Point. The recipient assumes responsibility for further distribution of this document. In no event shall Vantage Point be liable to anyone for direct, special, incidental, collateral or consequential damages arising out of the use of this material, to the maximum extent permitted under law.

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. Furthermore, the security assessment applies to a snapshot of the current state at the examination time.

5.2. Risk Rating

All vulnerabilities found by Vantage Point will receive an individual risk rating based on the following four categories.

Critical

A CRITICAL finding requires immediate attention and should be given the highest priority by the business as it will impact business interest critically.

High

A HIGH finding requires immediate attention and should be given higher priority by the business.

Medium

A MEDIUM finding has the potential to present a serious risk to the business.

Low

A LOW finding contradicts security best practices and have minimal impact on the business.

Observational

An OBSERVATIONAL finding relates primarily to non-compliance issues,security best practices or are considered an additional security feature that would increase the security stance of the environment which could be considered in the future version of smart contract.