Folks Finance Capital Market Protocol v2

Security Assessment

February 3, 2023

Prepared for:
Benedetto Biondi
Folks Finance

Prepared by: **Josselin Feist and Vara Prasad Bandaru**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Executive Summary

Engagement Overview

Folks Finance engaged Trail of Bits to review the security of version 2 of its capital market protocol. From October 31 to November 25, 2022, a team of two consultants conducted a security review of the client-provided source code, with six person-weeks of effort. Details of the project's timeline, test targets, and coverage are provided in subsequent sections of this report.

Project Scope

Our testing efforts were focused on the identification of flaws that could result in a compromise of confidentiality, integrity, or availability of the target system. We conducted this audit with full knowledge of the system, including access to the source code and documentation. We performed automated and manual static testing of the target system and its codebase.

Summary of Findings

The audit uncovered significant flaws that could impact system confidentiality, integrity, or availability, one of which could enable an attacker to drain the system of all funds. A summary of the findings and details on notable findings are provided below.

EXPOSURE ANALYSIS

Severity	*Count*
High	3
Medium	1
Low	5
Informational	1

CATEGORY BREAKDOWN

Category	*Count*
Data Validation	9
Undefined Behavior	1

Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

- **TOB-FOLKS-1**
  The lack of validation of the `flash_loan_end` transaction could enable an attacker to reuse the same repayment transaction for multiple loans, draining a pool of its funds.

- **TOB-FOLKS-5**
  Because of the insufficient validation of encoded byte arrays of compound type arguments, an attacker could force the `oracle_adapter` application to use outdated pricing information when calculating a liquidity provider (LP) token price.

- **TOB-FOLKS-7**
  The layout of the loan application's global state could cause a loan's `params` variable to collide with another loan variable.

- **TOB-FOLKS-9**
  The use of the `Btoi` instruction to decode the arguments of the other transactions in a group could lead to the use of incorrect values.

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager
dan@trailofbits.com

Jeff Braswell, Project Manager
jeff.braswell@trailofbits.com

The following engineers were associated with this project:

Josselin Feist, Consultant
josselin@trailofbits.com

Vara Prasad Bandaru, Consultant
vara.bandaru@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
October 12, 2022	Pre-project kickoff call
November 4, 2022	Status update meeting #1
November 18, 2022	Status update meeting #2
November 28, 2022	Delivery of report draft; report readout meeting
December 2, 2022	Delivery of fix review draft (appendix D)
December 30, 2022	Delivery of final report and fix review
February 3, 2023	Removal of source code at Folks Finance's request

Project Goals

The engagement was scoped to provide a security assessment of version 2 of Folks Finance's capital market protocol. Specifically, we sought to answer the following non-exhaustive list of questions:

- Is every transaction within a group properly validated?
- Are there any arithmetic-related flaws in the system?
- Could one pool impact the other pools?
- Could a user force the system to use stale prices?
- Could a user receive more f-assets than expected?
- Could an unprivileged actor execute privileged operations?
- Is the flash loan feature susceptible to abuse?
- Does the codebase avoid the most common Algorand vulnerabilities?
- Is it possible for an attacker to drain the system's funds?
- Is LP token data updated correctly?

Project Targets

The engagement involved a review and testing of the following target.

Folks Finance Capital Market Protocol v2

Repository	Private repository
Version	`54bbb774907416af223efae4dc2d64f6e918f69f`
Type	PyTeal
Platform	Algorand

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches and their results include the following:

- **Deposits.** We reviewed the conversion of assets to f-assets and checked for ways to generate more f-assets than expected. We checked that only the owner of an escrow account can perform the privileged operations of withdrawing funds from the account and opting in to an f-asset. We also looked for ways in which a user's funds could become stuck in escrow and checked whether input arguments are properly validated.

- **Pool.** We looked for ways in which a user could withdraw more funds than the user had deposited. We checked whether transaction groups use the expected parameters and reviewed the creation and use of internal transactions. We also reviewed the access controls enforced on the basis of loan application IDs as well as the access controls on privileged operations (e.g., those that prevent admins from making arbitrary global state updates). We checked for flaws in the flash loan functionality and looked for ways to evade repayment of a flash loan. We reviewed the calculation of the f-asset / asset exchange rate and the use of interest rates and interest indexes (as well as the process of updating them). Additionally, we checked whether pool updates are propagated to the `pool_manager` application, whether each method has proper access controls, and whether input arguments are validated correctly.

- **Pool manager.** We reviewed the methods' access controls and the validation of input arguments. We also looked for ways in which a pool could update another pool's information.

- **`lp_token_oracle` application.** We checked whether there are proper access controls on the application's privileged methods. We reviewed whether Tinyman and Pact Pool asset supplies are fetched correctly; however, we considered both Tinyman and Pact Pool to be trustworthy sources of information. We also reviewed the validation that the system performs before adding a Tinyman or Pact LP asset. Lastly, we checked whether pool updates are performed correctly.

- **`oracle_adapter` application.** We checked whether the `oracle_adapter` clears all available prices upon the invocation of the `refresh_prices` method. We assessed whether prices of assets passed to `refresh_prices` are available to the loan and pool contracts. We looked for ways to update the price of an LP asset without updating the information on that asset in the `lp_token_oracle` application. Finally, we checked whether there are proper access controls on privileged methods and verified that admins cannot arbitrarily update the global state.

- **Loan.** We reviewed the access controls on privileged operations and verified that only the owner of an escrow account can perform related privileged operations. We also reviewed the flow of assets throughout the system, analyzed the swap collateral functions, and looked for ways to bypass the check of whether a loan is overcollateralized. However, time constraints prevented us from reviewing all aspects of the loan functionality.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- The Folks Finance team identified the following system elements as out of scope:
    - The `oracle` smart contract (`contracts/oracle/oracle.py`)
    - The `get_safe_lp_asset_price` and `get_safe_non_lp_asset_price` subroutines of the `oracle_adapter`
    - The compiled Teal code (`contracts/loan/loan.teal`) and the changes made to the Teal compiler
    - The smart contracts in `contracts/testing`
- We assumed that the arithmetic in the whitepaper is correct and did not review the correctness of the system economics.
- We performed a partial review of the cap on the amount of funds that can be borrowed; that limit would benefit from further investigation.
- We partially covered aspects of the loan functionality including borrowing funds, switching the type of a borrowed asset, liquidating a loan, and repaying a loan.
- We partially reviewed the impacts of a system parameter update; we recommend that Folks Finance perform further modeling of the update process (see TOB-FOLKS-8).
- We did not review the issues raised in the pull requests that were made during the audit (e.g., the lack of `OnCompletion` verification and the incorrect use of the scale for arithmetic operations); nor did we review the issue found by the Folks Finance team during the audit (i.e., the ability to manipulate a stable borrow rate by rapidly depositing and then withdrawing funds).

- We did not review the impact that a black swan–like event would have on the system (i.e., the system lacks protections in the case of several shifts in the market that would lead most of the loan to become undercollateralized).

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	The system would benefit from further analysis of arithmetic-related edge cases (as well as from documentation on the expected rounding direction of every operation). Given the complexity of the arithmetic operations, analysis of those operations through fuzzing or another similar testing technique would greatly increase users' and developers' confidence in the implementation. Lastly, a systematic comparison of the values returned at runtime against a ground truth (using infinite-precision arithmetic) would help determine whether the bounds on the loss of precision are correct.	**Moderate**
Auditing	The system does not perform any logging. Without logging, monitoring of the applications is difficult, and a compromise of the system could go undetected.	**Missing**
Authentication / Access Controls	The applications have appropriate access controls, and those access controls are documented. However, additional documentation outlining the operations that privileged actors should not be able to perform would help clarify the expectations surrounding those actors' abilities.	**Satisfactory**
Complexity Management	The low-level nature of state manipulation in PyTeal makes PyTeal code inherently complex. The codebase would benefit from more thorough descriptions of the different operations (e.g., the fact that `update_borrow`'s `new_bor_bal` variable ignores the `amount` value, deviating from the implementation outlined in equation 19 of the whitepaper). Moreover, much of the system's	**Moderate**

	complexity comes from the applications' interactions with accounts, which would benefit from further documentation.	
Decentralization	The system includes multiple privilege levels, actors, and points of centralization (such as the oracle). However, certain risks associated with the privileged actors are not clearly documented. For example, the loan application admin could add a malicious application as a loan to a pool and then withdraw tokens from the pool.	Weak
Documentation	The documentation on the system's technical design and economic model provides a good introduction to the system. However, several aspects of the system lack documentation; these include the relationships between transactions and the invariants related to those relationships. Moreover, the documentation lacks a better mapping with the implementation.	Moderate
Front-Running Resistance	Documentation on the expectations surrounding arbitrage would help differentiate legitimate arbitrage opportunities from unintended ones. Additionally, the system lacks validation of the bounds on certain parameters; as a result, the success of the transactions executed in the applications is highly dependent on the order in which transactions are included in a block.	Moderate
Low-Level Manipulation	Because of the low-level nature of PyTeal, there are several instances of low-level manipulation in the codebase. These low-level operations require better testing and documentation and are the cause of several issues identified during the audit (TOB-FOLKS-5, TOB-FOLKS-7, TOB-FOLKS-9).	Moderate
Testing and Verification	The project contains only unit tests (albeit a large number of them). Fuzzing of the low-level manipulation and arithmetic used in the project would be beneficial.	Moderate

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Ability to drain a pool by reusing a flash_loan_end index	Data Validation	High
2	Lack of a two-step process for admin role transfers	Data Validation	High
3	Insufficient validation of application initialization arguments	Data Validation	Low
4	Ability to reuse swap indexes	Data Validation	Informational
5	oracle_adapter could be forced to use outdated LP token information in price calculations	Data Validation	Medium
6	Incorrect rounding directions in the calculation of borrowed asset amounts	Data Validation	Low
7	Risk of global state variable collision	Data Validation	High
8	Lack of documentation on strategies in case of system parameter update	Undefined Behavior	Low
9	Incorrect decoding of method arguments results in the use of invalid values	Data Validation	Low
10	Lack of minimum / maximum bounds on user operation parameters	Data Validation	Low

Detailed Findings

1. Ability to drain a pool by reusing a flash_loan_end index

Severity: **High**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-FOLKS-1
Target: `pool.py`	

Description

The lack of validation of the `flash_loan_end` transaction could enable an attacker to drain a pool of its funds by reusing the same repayment transaction for multiple loans.

Flash loan operations are split into two transactions included in the same group: a `flash_loan_begin` transaction (shown in figure 1.1) and a `flash_loan_end` transaction (shown in figure 1.2).

CODE REDACTED

Figure 1.1: REDACTED

CODE REDACTED

Figure 1.2: REDACTED

The `flash_loan_begin` method sends the assets to the user taking out the flash loan and checks that there is an associated `flash_loan_end` transaction later in the transaction group. The `flash_loan_end` method ensures that the associated repayment transaction (`send_asset_txn`) has the correct `amount` value.

CODE REDACTED

Figure 1.3: REDACTED

The `flash_loan_key` value serves as a mutex and is used to prevent a user from taking out a new flash loan before the user's previous flash loan is complete. The `flash_loan_begin` method checks that `flash_loan_key` is set to 0 and then sets it to 1; `flash_loan_end` checks that `flash_loan_key` is set to 1 and then sets it to 0.

However, there is no validation of whether the `flash_loan_end` transaction at the index passed to `flash_loan_begin` is the one that resets the `flash_loan_key` mutex. An attacker could reuse the same repayment in `flash_loan_end` transaction in multiple calls to `flash_loan_begin`, as long as he created additional calls to `flash_loan_end` (with any `amount` value) to reset the mutex. Thus, an attacker could drain a pool by taking out flash loans without repaying them.

Exploit Scenario
Eve creates a group of six transactions:

1. `flash_loan_begin(1000, 5, ..)`
2. An asset transfer with an `amount` of 0
3. `flash_loan_end(…)` (which serves only to reset the `flash_loan_begin` mutex)
4. `flash_loan_begin(1000, 5, ..)`
5. An asset transfer with an `amount` of 1000
6. `flash_loan_end(…)`

Transactions 1 and 4 credit Eve with 2,000 tokens (1,000 tokens per transaction). Transactions 2 and 3 serve only to reset the mutex. Transactions 5 and 6 repay one of the flash loans by transferring 1,000 tokens. Thus, Eve receives 1,000 tokens for free. (Note that for the sake of simplicity, this exploit scenario ignores the fees that would normally be paid.)

Recommendations
Short term, store the amount to be repaid in `flash_loan_key`, and ensure that the correct amount is repaid.

Long term, create schemas highlighting the relationships between the transactions, and document the invariants related to those relationships.

2. Lack of a two-step process for admin role transfers

Severity: **High**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-FOLKS-2
Target: `pool_manager.py`, `pool.py`, `loan.py`, `lp_token_oracle.py`, `oracle_adapter.py`	

Description

The Folks Finance methods used to transfer the admin role from one address to another perform those transfers in a single step, immediately updating the admin address. Making such a critical change in a single step is error-prone and can lead to irrevocable mistakes.

These methods include the `update_admin` method (figure 2.1), which is used to update the address of the `pool_manager` application's admin. If the `update_admin` method were called with an incorrect address, it would no longer be possible to execute administrative actions such as the addition of a pool.

```
CODE REDACTED
```

Figure 2.1: REDACTED

The `update_admin` methods of the `pool`, `loan`, `lp_token_oracle`, and `oracle_adapter` applications also perform admin role transfers in a single step.

Exploit Scenario

Alice, the admin of the `pool_manager` application, calls the `update_admin` method with an incorrect address. As a result, she permanently loses access to the admin role, and new pools cannot be added to the `pool_manager` application.

Recommendations

Short term, implement a two-step process for admin role transfers. One way to do this would be splitting each `update_admin` method into two methods: a `propose_admin` method that saves the address of the proposed new admin to the global state and an `accept_admin` method that finalizes the transfer of the role (and must be called by the address of the new admin).

Long term, identify and document all possible actions that can be taken by privileged accounts and their associated risks. This will facilitate reviews of the codebase and help prevent future mistakes.

3. Insufficient validation of application initialization arguments	
Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-FOLKS-3
Target: `pool_manager.py`, `pool.py`, `loan.py`, `lp_token_oracle.py`, `oracle_adapter.py`	

Description

Several of the methods involved in creating and setting up Folks Finance applications fail to validate incoming arguments. As a result, important state variables of an application can be set to invalid values, rendering the deployed application unusable.

```
CODE REDACTED
```

Figure 3.1: REDACTED

These methods include the pool application's `create` method, which does not properly validate the admin addresses before storing them in the global state. If one of those addresses were set to the zero address, it would not be possible to initialize (and therefore use) the pool application.

The following application initialization methods also fail to properly validate incoming arguments:

- The `setup` method of the `pool` application
- The `create` method of the `loan` application
- The `create` method of the `lp_token_oracle` application
- The `create` method of the `oracle_adapter` application
- The `create` method of the `pool_manager` application

Because the state variables set in the above methods cannot be updated after initialization, the only way to fix an incorrectly set state variable is to redeploy the affected application(s).

Exploit Scenario

Alice deploys the pool application with the `pool_admin` address set to the zero address. As a result, the pool application cannot be initialized or used.

Recommendations

Short term, implement proper validation of all arguments to ensure that users cannot set them to incorrect values.

Long term, implement comprehensive unit testing to ensure that the initialization methods properly validate their incoming arguments.

4. Ability to reuse swap indexes

Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-FOLKS-4
Target: `loan.py`	

Description

The lack of validation of the `swap_collateral_end` transaction enables reuse of the same `swap_collateral_end` transaction for multiple swap operations.

Swaps are split into two transactions included in the same group: a `swap_collateral_begin` transaction (shown in figure 4.1) and a `swap_collateral_end` transaction (shown in figure 4.2).

CODE REDACTED

Figure 4.1: REDACTED

CODE REDACTED

Figure 4.2: REDACTED

The `swap_collateral_begin` method sends the assets to the user executing the swap and checks that there is an associated `swap_collateral_end` transaction later in the transaction group. The `swap_collateral_end` method ensures that the loan is overcollateralized:

CODE REDACTED

Figure 4.3: REDACTED

The methods use a mutex to prevent a user from starting a new swap loan before the user's previous one is complete. The `swap_collateral_begin` method uses `loan_not_blocked_check` to check that the loan is not blocked; `swap_collateral_end` uses `loan_blocked_check` to check that the loan is blocked.

However, there is no validation of whether the `swap_collateral_end` transaction at the index passed to `swap_collateral_begin` is the one that resets the loan block mutex. An attacker could reuse the same `swap_collateral_end` transaction in multiple calls to `swap_collateral_begin`, as long as he created additional calls to `swap_collateral_end` to reset the mutex.

We set the severity of this finding to informational because it does not pose a direct threat to the system: despite this issue, a user must execute a call to `swap_collateral_end` between two calls to `swap_collateral_begin`, and the related loan must still be overcollateralized. However, the fact that the `swap_collateral_begin` transaction is not correlated to the `swap_collateral_end` transaction could lead to additional issues if the code is refactored. (See TOB-FOLKS-1 for details on a similar issue.)

Recommendations

Short term, consider storing the IDs of blocked operations, and ensure that the `swap_collateral_begin` and `swap_collateral_end` transactions are properly correlated with each other.

Long term, create schemas highlighting the relationships between the transactions, and document the invariants related to those relationships.

5. oracle_adapter could be forced to use outdated LP token information in price calculations

Severity: **Medium**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-FOLKS-5
Target: `oracle_adapter.py`	

Description

Because of the insufficient validation of encoded byte arrays of compound type arguments, an attacker could force the `oracle_adapter` application to use outdated information when calculating a liquidity provider (LP) token price.

The computation of an LP token price involves two transactions: `update_lp_tokens(asset_ids)` and `refresh_prices(lp_assets, ..)`.

The `update_lp_tokens` method updates the supply of the LP token, while `refresh_prices` computes the LP token's price. The `refresh_prices` method calls `check_lps_updated`, which checks that `update_lp_tokens` has been called and that the `update_lp_tokens.asset_ids` array is equal to the `update_lp_tokens.lp_asset_ids` array.

```
CODE REDACTED
```

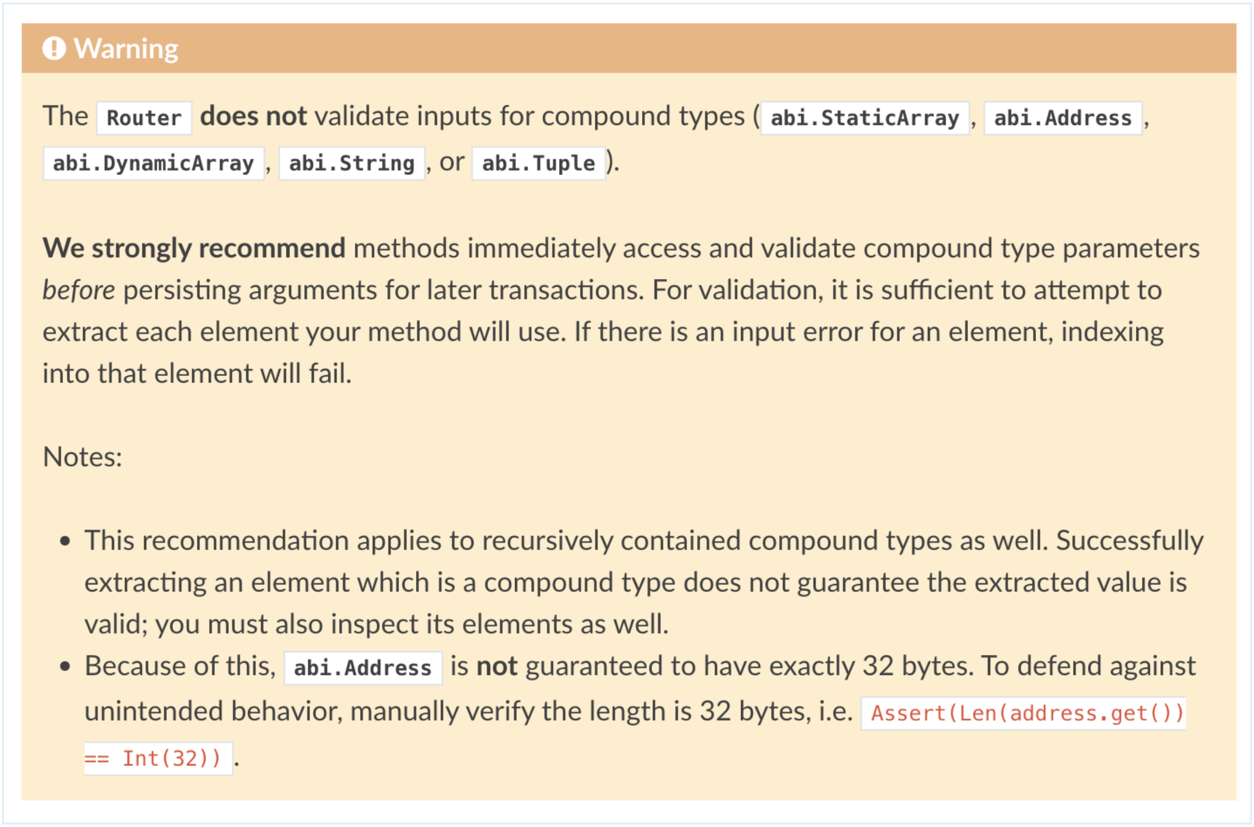
Figure 5.1: REDACTED

Instead of directly comparing `asset_ids` to `lp_asset_ids`, `check_lps_updated` calls `convert_uint64_abi_array_to_uint64_bytes_array` to convert the arrays into byte arrays; `convert_uint64_abi_array_to_uint64_bytes_array` removes the first two bytes of the array (which indicate the length of the array) and returns the remaining bytes:

```
CODE REDACTED
```

Figure 5.2: REDACTED

PyTeal does not provide any guarantees about the structure of compound type arguments. The PyTeal documentation includes the following warning:


Warning

The `Router` **does not** validate inputs for compound types (`abi.StaticArray`, `abi.Address`, `abi.DynamicArray`, `abi.String`, or `abi.Tuple`).

We strongly recommend methods immediately access and validate compound type parameters *before* persisting arguments for later transactions. For validation, it is sufficient to attempt to extract each element your method will use. If there is an input error for an element, indexing into that element will fail.

Notes:

- This recommendation applies to recursively contained compound types as well. Successfully extracting an element which is a compound type does not guarantee the extracted value is valid; you must also inspect its elements as well.
- Because of this, `abi.Address` is **not** guaranteed to have exactly 32 bytes. To defend against unintended behavior, manually verify the length is 32 bytes, i.e. `Assert(Len(address.get()) == Int(32))`.

Figure 5.3: `pyteal.readthedocs.io/en/stable/abi.html#registering-methods`

When data of the `uint64` type is converted into a byte array, the bytes' length may not match the `uint64` value's length. If that data is passed to a function that takes a `uint64[]` parameter, it may be a byte longer than the function expects.

Even if the data extracted as the bytes of `asset_ids` and `lp_asset_ids` is equivalent, the length of the original arrays might not be. Thus, `update_lp_tokens` could be called with an `lp_asset_ids` array that is shorter than the `refresh_prices.lp_asset_ids` array. In that case, the LP information would not be updated, and the price of the LP token would be based on outdated information.

Exploit Scenario

Bob holds a position that is eligible for liquidation. However, the AMM pool state changes, causing the price of the LP token that Bob is using as collateral to increase; thus, the loan is safe again, and Bob does not add more collateral. Eve notices that the oracle is still using old information on the LP token. Eve then creates a group of three transactions:

- `lp_token_oracle.update_lp_tokens("0x0000" + "0xdeadbeefdeadbeef")`
- `oracle_adapter.refresh_prices("0x0001" + "0xdeadbeefdeadbeef", ..)`

- `liquidate(...)`

The `check_lps_updated` method verifies that `lp_token_oracle` and `oracle_adapter` are using the same bytes (`0xdeadbeefdeadbeef`); however, `update_lp_tokens` interprets its parameter as an array with a length of zero (“`0x0000`”). As a result, the LP token information is not updated, and the old price is used, enabling Eve to liquidate Bob’s position.

Recommendations

Short term, have the `oracle_adapter` use the two dynamic arrays (`updated_lp_assets` and `lp_asset_ids`) directly and extract individual elements to perform an element-wise comparison.

Long term, avoid relying on internal structures used by the compiler. Review the PyTeal documentation and test edge cases more broadly.

6. Incorrect rounding directions in the calculation of borrowed asset amounts	
Severity: **Low**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-FOLKS-6
Target: `formulae.py`, `loan.py`	

Description

Multiple incorrect rounding directions are used in the computation of the amount borrowed in a loan. Thus, the result of the calculation may be too low, causing the system to underestimate the amount of assets borrowed in a loan.

To determine whether a loan is overcollateralized, `is_loan_over_collateralized` iterates over an array of all collateral assets and sums the underlying values of those assets:

```
CODE REDACTED
```

Figure 6.1: REDACTED

As part of this process, it calls `get_stable_borrow_balance` and `get_var_borrow_balance`, both of which call `calc_borrow_balance` to calculate the borrow balance of the loan at time `t`:

```
CODE REDACTED
```

Figure 6.2: REDACTED

The operation performed by the `calc_borrow_balance` function is equivalent to that shown in figure 6.3:

```
CODE REDACTED
```

Figure 6.3: REDACTED

The function adds 1 to the result of the equation to round it up. However, the $\frac{b_{ii_t} * 10^{14}}{bii_{tn_1}}$ portion of the equation rounds down, which can cause the overall rounding error to be greater than 1.

Similar issues are present in other functions involved in the computation, including the following:

- `calc_asset_loan_value`, which rounds down the results of its two calls to `mul_scale`
- `calc_borrow_interest_index`, which rounds down the result of the `mul_scale` call
- `exp_by_squaring`, which also rounds down the result of the `mul_scale` call

The cumulative loss of precision can cause the system to underestimate the amount of assets borrowed in a loan, preventing the loan's liquidation.

We set the severity of this issue to low because the loss of precision is limited. However, there may be other rounding issues present in the codebase.

Exploit Scenario

Eve's loan has become undercollateralized. However, because the loan contract rounds down when calculating the amount borrowed in a loan, it does not identify Eve's loan as undercollateralized, and the position cannot be liquidated. By contrast, if the contract performed precise accounting, Eve's loan would be eligible for liquidation.

Recommendations

Short term, ensure all arithmetic operations in `is_loan_over_collateralized` use a conservative rounding direction—that is, ensure that the loss of precision causes the system to interpret a loan as less collateralized than it actually is. Additionally, document those operations.

Long term, document the expected rounding direction of every arithmetic operation, and create rounding-specific functions (e.g., `mul_scale_down` and `mul_scale_down_up`) to facilitate reviews of the arithmetic rounding.

7. Risk of global state variable collision	
Severity: **High**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-FOLKS-7
Target: `loan/loan_state.py`, `loan/loan.py`	

Description

The layout of the loan application's global state could cause a loan's `params` variable to collide with a pool variable.

A loan has two types of variables:

- `params`, which is set by the owner and contains the loan parameters
- `pools`, which contains pool information

```
CODE REDACTED
```

Figure 7.1: REDACTED

```
DOCUMENTATION REDACTED
```

Figure 7.2: REDACTED

The `params` variable is stored at offset 112 `(Bytes("p"))`. The `pools` variable contains an array in which every slot contains 3 loans, and only slots 0–62 are assumed to be used for loans.

When the `pool`'s slots are used, there is no guarantee that the global state is being accessed through slots 0–62. This means that slot 112 can be used to store a loan.

We set the difficulty rating of this issue to high because exploitation of the issue would likely require exploitation of another bug. This is because if slot 112 were used for a loan, its underlying values would likely not be directly usable, particularly because of the following:

- The first element of `params` is an admin address.
- The first element of a loan variable is the pool application ID.

- If `params` collides with a pool variable, its admin address must collide with an application ID.

Exploit Scenario
Eve finds a lack of validation in the loan flow that allows her to trick the loan application into believing that there is a loan at slot 112. Eve uses the variable collision to change the system's parameters and update the `oracle_adapter` ID. As a result, the system stops working.

Recommendations
Short term, store the `params` variable at the offset `(Bytes("params"))`. Because loan indexes are `uint8` values, using a key with a value greater than 255 will prevent a collision.

Long term, create documentation on the management of the global state, and use unit and fuzz testing to check for potential collisions.

8. Lack of documentation on strategies in case of system parameter update	
Severity: **Low**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-FOLKS-8
Target: All contracts	

Description

Malicious users of the Folks Finance capital market protocol could try to game the system and earn more than honest users.

When users deposit assets into the protocol, they receive interest-bearing assets known as f-assets. To withdraw their original assets, users must return those f-assets. The amount of f-assets provided to a user upon a deposit (as well as the amount of the original asset collected during a withdrawal) depends on the deposit interest index, I_{d_t}.

```
DOCUMENTATION REDACTED
```

Figure 8.1: REDACTED

The value of I_{d_t} slowly increases over time, and certain actions can increase or decrease its rate of increase. For example, when someone borrows assets, the value of I_{d_t} will increase at a faster rate. By contrast, a deposit of additional assets or the repayment of a loan will cause I_{d_t} to increase at a slower rate.

Active lenders with knowledge of this behavior can prioritize strategies that will maximize their profits, giving them an advantage over passive lenders.

It is possible that updates to the system's parameters could also affect the way that the value of I_{d_t} changes; however, determining whether that is the case would require further investigation.

Exploit Scenario

Eve learns that Bob is going to borrow assets worth USD 10 million. Eve provides liquidity just before the execution of Bob's transaction and withdraws it right after. In this way, she earns fees from the protocol without participating in the protocol.

Recommendations
Short term, document the expected behavior of lenders and borrowers, and consider implementing a deposit lockup period.

Long term, model and document the strategies that users are expected to leverage. Additionally, evaluate the impact of system parameter updates on the protocol.

References
- https://uniswap.org/blog/jit-liquidity
- https://medium.com/@peter_4205/curve-vulnerability-report-a1d7630140ec

9. Incorrect decoding of method arguments results in the use of invalid values

Severity: **Low**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-FOLKS-9
Target: `loan.py`	

Description

Certain methods use the `Btoi` instruction to decode the arguments of the other transactions in their group, resulting in the use of incorrect values.

Most of the protocol operations involve a group of multiple transactions. In some cases, one method uses or validates the other transactions in the group and must decode their arguments. One such method is `loan.add_pool` (figure 9.1), which uses an argument of a pool application transaction.

```
CODE REDACTED
```

Figure 9.1: REDACTED

The `add_pool` method decodes the first argument of the `pool.add_loan_application` method, which is an `index` argument of type `uint8`. The pool application decodes this argument by using the `get_byte` operation to extract the first byte of the argument. However, `add_pool` decodes the argument by using the `Btoi` instruction, which is not the equivalent of extracting the first byte. Specifically, `Btoi(0x00..0X)` would return `X`, and `get_byte(0, 0x00..0X)` would return `0`.

This results in the use of different values in the two methods and causes the system to enter an invalid state.

This issue also affects `loan.swap_collateral_begin` (figure 9.2), which decodes an argument of `loan.swap_collateral_end` for validation.

```
CODE REDACTED
```

Figure 9.2: REDACTED

The `swap_collateral_begin` method also uses `Btoi` for decoding, while correctly decoding the argument would require extraction of the first byte. However, the issue has a limited impact on the collateral swap operation: a value of `0x00..0X` would cause `swap_collateral_end` to use the account at index 0 of the transaction's `accounts` array, which cannot be a valid escrow account.

Exploit Scenario

Alice, the admin of the loan application, creates a group of two transactions:

1. `pool.add_loan_application("0x0000000000000001", ...)`
2. `loan.add_pool(...)`

The loan application decodes the `add_loan_application` method's index as 1, whereas the pool application uses 0 as the index. The discrepancy causes the system to enter an invalid state.

Recommendations

Short term, use the compiler-provided `decode()` method (from the `abi.{type}` object) to decode application arguments.

Long term, avoid relying on compiler internals. Review the PyTeal documentation and test edge cases more broadly.

10. Lack of minimum / maximum bounds on user operation parameters	
Severity: **Low**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-FOLKS-10
Target: Pool and loan applications	

Description

The outcomes of several pool and loan operations are dependent on the system state. This means that users have no on-chain guarantees that their transactions will produce the outcomes they expect.

Examples of this issue include the following:

- The caller of `pool.deposit` may receive less f-assets than expected (e.g., zero f-assets in exchange for a small deposit).
- If the amount value passed to `pool.withdraw` is variable, the user may receive less assets than expected.
- An update to the retention rate would affect the outcomes of all loan operations that use the retention rate.

Note that `loan.borrow` and `loan.switch_borrow_type` do have a `max_stable_rate` parameter.

Exploit Scenario

Bob calls `pool.deposit` with a small amount of assets but does not receive any f-assets in return.

Recommendations

Short term, add minimum and maximum bounds on the parameters of all user operations.

Long term, document the front-running risks associated with each operation, and ensure that there are proper mitigations in place for those risks.

Summary of Recommendations

Trail of Bits recommends that Folks Finance address the findings detailed in this report and take the following additional steps prior to deployment:

- Create schemas highlighting the relationships between transactions, and document the invariants related to those relationships.
- Develop documentation outlining all other system invariants and explaining how they are checked (e.g., manually or through unit testing).
- Implement a logging mechanism in the contracts, and develop a monitoring system.
- Use fuzzing to test the low-level manipulation and arithmetic in the system.
- Perform a thorough analysis of the arithmetic rounding used in the system, and document the expected rounding direction of every operation.
- Expand the documentation regarding the privileged actors in the system. This documentation should outline the operations that they should not be able to perform as well as the use of a multisignature wallet for privileged accounts.
- Reduce the applications' reliance on compiler internals.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	Description
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories

Category	Description
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Decentralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Documentation	The presence of comprehensive and readable codebase documentation
Front-Running Resistance	The system's resistance to front-running attacks
Low-Level Manipulation	The justified use of inline assembly and low-level calls
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage

Rating Criteria

Rating	Description
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Code Quality Recommendations

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future.

General

- **Update the incorrect statement in the documentation regarding the number of pools that can be added to the system.** The technical design documentation on the loan application global state indicates that the system can support only 186 pools, while it can actually support 189 pools (as the ability to use slots 0–62 for keys means that 63 keys can be supported).

```
DOCUMENTATION REDACTED
```

Figure C.1: REDACTED

- **Standardize the approach to validation.** In some cases, checks such as `rekey_check` and `close_to_check` are performed at the beginning of the entry point method, while in other cases, they are performed in internal subroutines. This makes it difficult to determine whether all entry points are properly protected. A better approach would be to implement a generic subroutine that performs all validation and is called from all entry points.

```
CODE REDACTED
```

Figure C.2: REDACTED

```
CODE REDACTED
```

Figure C.3: REDACTED

- **Use `Txn.sender()` instead of `Int(0)` to access transaction sender information.** This will increase the readability of the code.

```
CODE REDACTED
```

Figure C.4: REDACTED

```
CODE REDACTED
```

Figure C.5: REDACTED

Loan

- **Remove the `pool_pos_in_indexes_array` scratch variable.** This variable is not used.

```
CODE REDACTED
```

Figure C.6: REDACTED

Deposit

- **Include an `assert` statement at the beginning of the `opt_escrow_into_asset` method to ensure that the escrow account has not yet opted in to the f-asset.** The sole purpose of `opt_escrow_into_asset` is to enable escrow accounts to opt in to the f-asset. If an escrow account has already opted in to the f-asset, the method does not need to perform any operations.

```
CODE REDACTED
```

Figure C.7: REDACTED

- **Set the `Receiver` and `AssetCloseTo` values of an asset transfer transaction to the escrow account's owner.** Funds held in an escrow account belong to the escrow account's owner. The `close_out_escrow_from_asset` method currently checks that the f-asset balance of an account is zero before closing out the account; however, that could change in future versions of the codebase, and setting `AssetCloseTo` to the escrow account's owner would reduce the impact of such a change.

```
CODE REDACTED
```

Figure C.8: REDACTED

- **Add a new method to the deposits application to enable users to withdraw f-assets directly from an escrow account.** Currently, the deposits application does not provide a way to withdraw f-assets directly. The only way to withdraw f-assets is to exchange the f-assets for assets and then exchange those assets for f-assets. Providing a way to withdraw f-assets directly will improve the application's usability.

Pool Manager

- **Use the pool application's `latest_update` value instead of `Global.latest_timestamp()` as the `latest_update` value of the `pool_manager`.** The loan application uses `pool.latest_update` as the `pool_manager`'s `latest_update` value, to indicate the "timestamp when the interest index was last updated". The use of `pool.latest_update` in the loan application ensures that the `pool_manager`'s `interest_indexes` and `latest_update` are in sync and that future updates will not cause any issues.

```
CODE REDACTED
```

Figure C.9: REDACTED

```
CODE REDACTED
```

Figure C.10: REDACTED

`math_lib`

- **Update the documentation on the `exp_by_squaring` subroutine to indicate that the return value will have "`hamming_weight(n) * (X - Z) + Z`" decimals.** The documentation incorrectly states that the value will have "`X - Z`" decimals.

```
CODE REDACTED
```

Figure C.11: REDACTED

D. Fix Review Results

From December 1 to December 2, 2022, Trail of Bits reviewed the fixes and mitigations implemented by the Folks Finance team for issues identified in this report.

We reviewed each of the fixes to ensure that the proposed remediation would be effective. For additional information, see the Detailed Fix Log.

ID	Title	Severity	Status
1	Ability to drain a pool by reusing a flash_loan_end index	High	Resolved (PR1)
2	Lack of a two-step process for admin role transfers	High	Partially Resolved
3	Insufficient validation of application initialization arguments	Low	Unresolved
4	Ability to reuse swap indexes	Informational	Resolved
5	oracle_adapter could be forced to use outdated LP token information in price calculations	Medium	Resolved (PR3)
6	Incorrect rounding directions in the calculation of borrowed asset amounts	Low	Resolved (PR4)
7	Risk of global state variable collision	High	Resolved (PR5)
8	Lack of documentation on strategies in case of system parameter update	Low	Resolved
9	Incorrect decoding of method arguments results in the use of invalid values	Low	Resolved (PR6)
10	Lack of minimum / maximum bounds on user operation parameters	Low	Unresolved

Detailed Fix Log

TOB-FOLKS-2: Lack of a two-step process for admin role transfers
Partially resolved. The Folks Finance team provided the following additional context:

> We already have a two step process. The admin account is a multisig account that already requires multiple signers to review the transaction and approve. The long term plan is also to move away from using admin accounts and replace with a DAO.

TOB-FOLKS-3: Insufficient validation of application initialization arguments
Unresolved. The Folks Finance team provided the following additional context:

> The outcome of incorrectly initialising a smart contract is that the creator will lose a few ALGOs. Therefore considering the low likelihood of the issue occurring and the fact that the consequences are insignificant, we do not consider it necessary to add these checks.

TOB-FOLKS-4: Ability to reuse swap indexes
Resolved. Folks Finance provided the following additional context:

> We have documented this behaviour. The protocol does not care how swap_colllateral_end is called, but rather that we can guarantee it is indeed called after swap_colllateral_begin.

TOB-FOLKS-5: oracle_adapter could be forced to use outdated LP token information in price calculations
Resolved. The original fix contained a vulnerability. Trail of Bits reported that vulnerability to the team on December 1 and reviewed the revised fix on December 2.

TOB-FOLKS-6: Incorrect rounding directions in the calculation of borrowed asset amounts
Resolved. While the rounding issues in `is_loan_over_collateralized` have been addressed, we recommend that Folks Finance create thorough documentation on the rounding direction of every operation, as suggested in the long-term recommendation of the finding.

TOB-FOLKS-8: Lack of documentation on strategies in case of system parameter update
Resolved. Folks Finance provided the following additional context:

> *This behaviour is expected in our lending protocol and is now documented.*

TOB-FOLKS-10: Lack of minimum / maximum bounds on user operation parameters
Unresolved. Folks Finance provided the following additional context:

The interest indexes change very slower so the expected output should be accurate to a very high precision. Also the withdrawal allows you to specify an exact asset received amount so such a parameter would be unnecessary in that case.