# Weft Finance Security Analysis

## by Pessimistic

This report is public

December 8, 2023

# Abstract

In this report, we consider the security of the codebase of Weft Finance project. Our task is to find and describe security issues in the codebase of the platform.

# Disclaimer

The audit does not give any warranties on the security of the code. One audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of the code. Besides, a security audit is not investment advice.

# Summary

In this report, we considered the security of Weft Finance project smart contracts available on a public GitHub repository. We performed our audit according to the procedure described below. The initial audit showed one high-severity, five medium-severity, and several low-severity issues.

After the initial audit, the developers provided us with a new version of the code. The severity of the high-severity issue Test coverage was decreased to low. One of the medium issues was moved to notes Overpowered roles. Additional details on that can be found in the Codebase update #1 section.

# Project overview

## Project description

For the audit, we were provided with Weft Finance project on a public GitHub repository, commit 213fec6411324132cd199f5f336efd6dd6de97e7.

The scope of the audit included the following folders:

- **single_resource_pool/**,
- **lending_market/**,
- **internal_price_feed/**.

The documentation for the project included: https://docs.weft.finance/overview.

All 33 tests pass successfully.

## Codebase update #1

After the audit, we were provided with a commit f80961616249d35d710af2dbecdfa050f44b6e02. In that update, the developers either fixed or commented on all medium-severity issues. Moreover, most of the low-severity issues were fixed. The severity of the high-severity issue was decreased to low (L12). The number of tests increased.

All 49 tests pass successfully.

# Procedure

We perform the audit according to the following procedure:

- **Automated analysis**
  - We compile the contracts.
  - We run the provided tests and calculate code coverage using Cargo Tarpaulin.
  - We manually verify (reject or confirm) all issues reported by the following tools:
    - Cargo Geiger;
    - Cargo Audit;
    - Cargo clippy.

- **Manual audit**
  - We manually review the code and assess its quality.
  - We check the code for known vulnerabilities.
  - We check whether the code logic complies with the provided documentation.
  - We suggest possible gas and storage optimizations.

- **Report**
  - We reflect all the gathered information in the report.

# Issues

We are actively looking for:

- Access control issues (incorrect admin or user identification/authorization).
- Lost/stolen assets issues (assets being stuck on the contract or sent to nowhere or to a wrong account).
- DoS due to logical issues (deadlock, state machine error, etc).
- Contract interaction issues (reentrancy, insecure calls, caller assumptions).
- Arithmetic issues (overflow, underflow, rounding issues).
- Other issues.

# Automated analysis

Automated analysis shows the following:

- **Auto-tests**
  All 33 tests pass successfully.
- **Tests coverage**
  For code coverage calculation, we used Cargo Tarpaulin.
  The code coverage could not be calculated due to the nature of the wasm based test system in the Radix Scrypto engine. The manual review showed uncovered `view` and `write` external functions:
  - **single_resource_pool /src/lib.rs**:
    - protected_deposit
    - protected_withdraw
    - decrease_external_liquidity
    - increase_external_liquidity
    - contribute
    - redeem
    - get_pool_unit_ratio
    - get_pool_unit_supply
    - get_pooled_amount
  - **internal_price_feed /src/lib.rs**:
    - mint_updater_badge
    - update_updater_badge
    - admin_update_price
    - update_price
    - get_price

    - **lending**
        - supply_fee_reserve
        - set_price_feed
        - update_liquidation_threshold
        - set_interest_strategy
        - collect_reserve
        - create_delegatee_cdp
        - link_cdp
        - unlink_cdp
        - update_cdp
        - update_delegatee_cdp
        - take_batch_flashloan
        - repay_batch_flashloan
        - refinance
        - start_liquidation
        - end_liquidation

- **Check for unsafe Rust code**
  For unsafe Rust code test, we used Cargo Geiger. The tool terminates with an error. However, a manual check showed that unsafe blocks were not used.
- **Check for crates vulnerabilities**
  To test the codebase for crates vulnerabilities, we used Cargo Audit. The tool showed one vulnerability, but in a third-party crate (`scrypto-unit`). This could eliminated by change the version of the unit to the higher one. See Appendix A for more details.
- **Linter**
  For linter, we used Cargo Clippy. The tool showed 5 warnings. See Appendix A for more details.

# Manual analysis

## High-severity issues

High-severity issues seriously endanger project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

**The audit showed no issues of high severity.**

## Medium-severity issues

Medium-severity issues can influence project operation in the current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

### M01. Protocol cannot be paused (fixed)

The protocol currently lacks a pause mechanism for emergency situations. We strongly recommend incorporating the capability to pause the entire protocol. Furthermore, we suggest implementing a feature to independently pause a single pool in scenarios involving issues with a specific resource (e.g., depeg of stablecoin, malicious resource, etc.).

*The issue has been fixed and is not present in the latest version of the code.*

### M02. Problems with the price feed (addressed)

1. The price feed is centralized, and prices are updated by a bot written by the developers. Since the correctness of the prices is a crucial part of the lending protocol, an outage of the bot or compromise of the private key of the `updater` role can become a single point of failure. To enhance security, we recommend using a trusted oracle system, coupled with additional verification with the DEX prices.
2. The price update time is not checked (the corresponding code is commented out). The scenario when the oracle had an outage and prices are not updated for a long time can lead to potential problems. (**fixed**)

> *Comment from the developers: No decentralized price oracle is available to Radix components for now. As the used price feed is configurable, we will migrate to a decentralized price oracle once available. For now, the plan is to deploy at least 2 updater bots to mitigate outage risk.*

### M04. Deposit to delagatee CDP (fixed)

At line 1159 of the **lending_market.rs**, it is checked that delegatee CDP cannot deposit for consistency reasons. However, in the `link_cdp` function, there is no check that the delegatee does not have collateral.

*The issue has been fixed and is not present in the latest version of the code.*

### M05. Usage of IndexMap and IndexSet (fixed)

The project uses `IndexMap` and `IndexSet` structures. These structures do not have the laziness property like in the `KeyValueStore`, meaning that when retrieving the state of a component with these fields, all records will be retrieved at once. After some number of records (e.g., 1000), the contract will stop working. Try to avoid using these structures (or try to restrain the size with additional checks) where they can potentially store a lot of entries, as this can lead to DoS problems.

*The issue has been fixed and is not present in the latest version of the code.*

## Low-severity issues

Low-severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

### L01. Missing check (fixed)

The function `start_liquidation` in the **lending.rs** file does not check the amount of the `requested_collaterals` argument for negative values. Although it has been verified that providing a negative amount will eventually result in failure in the Radix part of the code during bucket transfer, we strongly recommend adding this check as a precautionary measure.

*The issue has been fixed and is not present in the latest version of the code.*

### L02. Rust syntax and language issues (fixed)

1. Unnecessary `let` bindings:
    - On lines 709 and 778 of the **lending.rs** file.
2. Issues mentioned in Appendix A

*The issue has been fixed and is not present in the latest version of the code.*

### L03. Usage of PreciseDecimal (fixed)

Consider using `PreciseDecimal` instead of `Decimal` for `minute_interest_rate` at line 268 in the **pool_state.rs** file for better accuracy.

*The issue has been fixed and is not present in the latest version of the code.*

### L04. Update of the interest rate (fixed)

Calling `take_batch_flashloan` before any functions that triggered `update_interest_and_price` may affect the `interest_rate`.

> *We recommend adding tests that handle borrowing to delegatee.*

*The issue has been fixed and is not present in the latest version of the code.*

### L05. Documentation

The protocol lacks a proper documentation of the calculations of the interest rates. We recommend explaining these calculations.

> *The developers added a new section to the docs. However, it lacks mathematical formulas that are used for calculation. We think that a detailed description of rate formulas is important for the transparency of the protocol. To get inspired, you may see an example of the Compound documentation.*

### L06. Recallable and Freezable tokens (fixed)

We recommend checking that `create_lending_pool` is not based on recallable or freezable tokens.

> *Comment from the developers: Right for recallable assets but a bit tricky to implement for freezable assets as most centralized stablecoin are freezable.*

*The issue has been fixed and is not present in the latest version of the code.*

### L07. Typos (fixed)

The codebase contains several typos in variable names and inside comments (`bactch_loan_term`, `necessery`, `wich`, `buket`, `revers`, etc.). We recommend fixing them by running a spell-checker linting tool.

*The issue has been fixed and is not present in the latest version of the code.*

### L08. Events

Consider emitting events on state changes.

> *The developers added events to most of the functions. However, `update_price` and `flashloan` functions do not emit events.*

### L09. Unused functions (fixed)

The `decrease_external_liquidity` and `increase_external_liquidity` functions of the **SingleResourcePool** component are not used. Note that these functions can be called only by the **LendingMarket** component. However, the **LendingMarket** does not contain any calls to these functions.

The `_lock_fee` function in the **lending_market.rs** is also not used.

*The issue has been fixed and is not present in the latest version of the code.*

### L10. Pool's resource owner (fixed)

Consider setting the pool's resource owner as `OwnerRole::None`, when calling `ResourceBuilder::new_fungible(owner_role.clone())`, at line 95 of the **single_resource_pool/src/lib.rs** file.

*The issue has been fixed and is not present in the latest version of the code.*

### L11. Usage of mutable reference

Consider using nonmutable reference in view functions (`get_pooled_amount`, `get_pool_unit_ratio`, and `_get_unit_to_asset_ratio`) in the **single_resource_pool/src/lib.rs**.

The same issue is present in the `get_loan_unit_ratio` function in the **src/modules/pool_state.rs**.

> *The developers fixed all functions except `_get_unit_to_asset_ratio`.*

### L12. Test coverage

The important parts of the protocol are not covered with tests (e.g., liquidation, flashloans, etc.).

Testing is crucial for detecting and resolving technical bugs. While manual audits primarily address logical aspects, they may not uncover all technical issues.

> *The developers increased the number of tests. However, we recommend testing more additional scenarios, including the following:*
>
> - *borrowing and repaying by delegatee,*
> - *handling max CDPs,*
> - *handling max positions,*
> - *handling the case when `add_liquidity`/`remove_collateral` is called between borrowing and repaying,*
> - *handling the case when `add_liquidity`/`remove_collateral` is called between taking flashloan and repaying,*
> - *tests for functions that are not tested individually (refinance, update_delegatee_cdp, update_cdp, link_cdp, unlink_cdp, create_delegatee_cdp).*

## Notes

### N01. Reproducible build using docker

We recommend using Docker for reproducible builds.

### N02. Update of the pool configuration

Pool and market configuration changes might affect current users' borrows.

### N03. Overpowered roles (addressed)

The system relies heavily on the roles that have centralized control over some aspects of the project (e.g., adding new pools, setting price feed, updating protocol config).

There are scenarios that can lead to undesirable consequences for the project and its users, e.g. if the private key for this role becomes compromised. We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig.

> *Comment from the developers: The admin role requires 4 admin badges, and we have introduced a moderator role that requires 2 admin badges. These badges will be spread over multiple accounts, and we will leverage the multi-signature feature of Radix once it is implemented in the wallet.*

> *Comment from Pessimistic: We recommend implementing a functionality which allows minting new badges (e.g., requires 4 admin badges to mint) in case one of the badges is lost.*

### N04. Usage of fold (new)

We recommend using `try fold` instead of `fold` iterator in line 475 of the **cdp_health_checker.rs** file.

# Appendix A

## Cargo clippy results

```
 Checking single_asset_pool v0.1.0
      (/home/gsg/Activities/Marchenko-
Blockchain/WeftFinance/single_resource_pool)
      warning: redundant clone
      --> src/lib.rs:95:81
      |
      95 | let pool_unit_res_manager =
ResourceBuilder::new_fungible(owner_role.clone())
      | ^^^^^^^^ help: remove this
      |
      note: this value is dropped without further use
      --> src/lib.rs:95:71
      |
      95 | let pool_unit_res_manager =
ResourceBuilder::new_fungible(owner_role.clone())
      | ^^^^^^^^^^
      = help: for further information visit https://rust-lang.github.io/rust-
clippy/master/index.html#redundant_clone
      = note: `#[warn(clippy::redundant_clone)]` on by default

      warning: redundant clone
      --> src/lib.rs:101:45
      |
      101 | burner => component_rule.clone();
      | ^^^^^^^^ help: remove this
      |
      note: this value is dropped without further use
      --> src/lib.rs:101:31
      |
      101 | burner => component_rule.clone();
      | ^^^^^^^^^^^^^^
      = help: for further information visit https://rust-lang.github.io/rust-
clippy/master/index.html#redundant_clone

      warning: returning the result of a `let` binding from a block
      --> src/lib.rs:181:13
      |
      179 | let pool_units = self.pool_unit_res_manager.mint(unit_amount);
      | ------------------------------------------------------------
unnecessary `let` binding
      180 |
      181 | pool_units
      | ^^^^^^^^^^
      |
```

```
      = help: for further information visit https://rust-lang.github.io/rust-
clippy/master/index.html#let_and_return
      = note: `#[warn(clippy::let_and_return)]` on by default
      help: return the expression directly
      |
      179 ~
      180 |
      181 ~ self.pool_unit_res_manager.mint(unit_amount)
      |

      warning: returning the result of a `let` binding from a block
      --> src/lib.rs:208:13
      |
      204 | / let assets = self
      205 | | .liquidity
      206 | | .take_advanced(amount,
WithdrawStrategy::Rounded(RoundingMode::ToZero));
      |-------------------------------------------------

      |- unnecessary `let` binding
      207 |
      208 | assets
      | ^^^^^^
      |
      = help: for further information visit https://rust-lang.github.io/rust-
clippy/master/index.html#let_and_return
      help: return the expression directly
      |
      204 ~
      205 |
      206 ~ self
      207 + .liquidity
      208 + .take_advanced(amount,
WithdrawStrategy::Rounded(RoundingMode::ToZero))
      |

      warning: returning the result of a `let` binding from a block
      --> src/lib.rs:285:13
      |
      279 | / let ratio = if total_liquidity_amount != 0.into() {
      280 | | PreciseDecimal::from(total_supply) /
PreciseDecimal::from(total_liquidity_amount)
      281 | | } else {
      282 | | 1.into()
      283 | | };
      | |_____________- unnecessary `let` binding
      284 |
      285 | ratio
      | ^^^^^
      |
      = help: for further information visit https://rust-lang.github.io/rust-
clippy/master/index.html#let_and_return
      help: return the expression directly
      |
```

```
      279 ~
      280 |
      281 ~ if total_liquidity_amount != 0.into() {
      282 + PreciseDecimal::from(total_supply) /
PreciseDecimal::from(total_liquidity_amount)
      283 + } else {
      284 + 1.into()
      285 + }
      |

      warning: `single_asset_pool` (lib) generated 5 warnings (run `cargo
clippy --fix --lib -p single_asset_pool` to
      apply 5 suggestions)
      Finished dev [unoptimized + debuginfo] target(s) in 35.59s
```

### Cargo audit results

```
      Fetching advisory database from https://github.com/RustSec/advisory-
db.git
      Loaded 576 security advisories (from /home/gsg/.cargo/advisory-db)
      Updating crates.io index
      Scanning Cargo.lock for vulnerabilities (176 crate dependencies)
      Crate: ed25519-dalek
      Version: 1.0.1
      Title: Double Public Key Signing Function Oracle Attack on `ed25519-
dalek`
      Date: 2022-06-11
      ID: RUSTSEC-2022-0093
      URL: https://rustsec.org/advisories/RUSTSEC-2022-0093
      Solution: Upgrade to >=2
      Dependency tree:
      ed25519-dalek 1.0.1
      └── transaction 1.0.1
      ├── single_asset_pool 0.1.0
      ├── scrypto-unit 1.0.1
      │ └── single_asset_pool 0.1.0
      ├── scrypto-test 1.0.1
      │ └── single_asset_pool 0.1.0
      ├── radix-engine-queries 1.0.1
      │ └── scrypto-unit 1.0.1
      └── radix-engine 1.0.1
      ├── single_asset_pool 0.1.0
      ├── scrypto-unit 1.0.1
      ├── scrypto-test 1.0.1
      └── radix-engine-queries 1.0.1

      error: 1 vulnerability found!
```

This analysis was performed by Pessimistic:

Nikita Kuznetsov, Security Engineer
Sergey Grigoriev, Security Engineer
Evgeny Marchenko, Senior Security Engineer
Yhtyyar Sahatov, Security Engineer
Irina Vikhareva, Project Manager
Konstantin Zherebtsov, Business Development Lead
Alexander Seleznev, Founder

December 8, 2023