LinumLabs

Ruscet - Audit

Prepared by Linum Labs AG

2024-10-02

linumlabs.com

Executive Summary

This audit covers the Ruscet decentralized derivatives platform, which supports asset swaps, leveraged trading, and the minting/redeeming of the RUSD token. The audit reviewed core smart contracts, including Vault, RUSD, and price feed contracts.

The Ruscet team has addressed most issues; however, we recommend additional testing and focusing on security when building out their improvements.

Protocol Summary

Overview

Ruscet is a decentralized derivatives platform built for the FuelVM, supporting financial operations like asset swaps, leveraged trading, and interaction with the synthetic token RUSD. At the core of Ruscet's functionality is the Vault smart contract, which manages asset pools, user positions, and fees. Users can swap supported assets, open leveraged positions in both long and short directions, and mint or redeem RUSD tokens.

Audit Scope

```
../contracts/assets/rlp/src/errors.sw
../contracts/assets/rlp/src/main.sw

../contracts/assets/rusd/src/errors.sw
../contracts/assets/rusd/src/main.sw

../contracts/assets/time-distributor/src/errors.sw
../contracts/assets/time-distributor/src/events.sw
../contracts/assets/time-distributor/src/main.sw
../contracts/assets/yield-asset/src/errors.sw
../contracts/assets/yield-asset/src/main.sw

../contracts/assets/yield-tracker/src/errors.sw
../contracts/assets/yield-tracker/src/main.sw

../contracts/core/vault-pricefeed/src/constants.sw
../contracts/core/vault-pricefeed/src/errors.sw
../contracts/core/vault-pricefeed/src/main.sw

../contracts/core/vault-storage/src/constants.sw
../contracts/core/vault-storage/src/errors.sw
../contracts/core/vault-storage/src/events.sw
../contracts/core/vault-storage/src/internal.sw
../contracts/core/vault-storage/src/main.sw

../contracts/core/vault-utils/src/constants.sw
../contracts/core/vault-utils/src/errors.sw
../contracts/core/vault-utils/src/events.sw
../contracts/core/vault-utils/src/main.sw

../contracts/core/vault/src/constants.sw
../contracts/core/vault/src/errors.sw
../contracts/core/vault/src/events.sw
../contracts/core/vault/src/internals.sw
../contracts/core/vault/src/main.sw
../contracts/core/vault/src/utils.sw
```

```
../contracts/helpers/src/context.sw
../contracts/helpers/src/fixed_vec/sw
../contracts/helpers/src/lib.sw
../contracts/helpers/src/math.sw
../contracts/helpers/src/signed_256.sw
../contracts/helpers/src/time.sw
../contracts/helpers/src/transfer.sw
../contracts/helpers/src/utils.sw
../contracts/helpers/src/zero.sw

../contracts/pricefeed/src/errors.sw
../contracts/pricefeed/src/main.sw
```

Areas of Concern

- Code coverage is low and can lead to bugs being missed. We suggest implementing more tests to increase code coverage.
- Many of the issues found need to be fixed through migration to using Pyth's price feeds. We recommend caution when implementing the integration of Pyth's price feeds.
- Currently, the initialize functions can be called by anyone. Since there are no deploy scripts to verify, we urge building the deploy scripts with security and the possibility of front-running in mind.

Audit Results

Summary

Repository	Ruscet_Repo
Commit	93c58b12d72429d1...
Timeline	September 11th - October 02

Issues Found

Bug Severity	Count
Critical	2
Medium	2
Low	6
Informational	11
Total Findings	21

Summary of Issues

Description	Severity	Status
Missing access control in *set_latest_answer()* function	Critical	Acknowledged
Unauthorized access in vault-pricefeed *update_price()* function	Critical	Acknowledged
Balance drift between UTXO and contract	Medium	Resolved
Vault *buy_rusd()* function is not payable yet assumes asset transfer	Medium	Resolved
Use of non standard SRC20 implementation	Low	Resolved
Front running possibility with transactions calling *_transfer_in()*	Low	Resolved
Dead code in vault-pricefeed contract	Low	Resolved

write_buffer_amount() is not used, and duplicates *set_buffer_amount()*	Low	Resolved
write_max_rusd_amount() not used	Low	Resolved
AssetId::non_zero() always returns true	Low	Acknowledged
Centralized control in RLP asset minting and burning	Informational	Acknowledged
Unused parameter in *get_funding_fee()* function	Informational	Acknowledged
Update of type generation library	Informational	Acknowledged
Unused parameter in *_update_cumulative_funding_rate()* function	Informational	Acknowledged
Use the SRC-20 standard for inline documentation	Informational	Acknowledged
Redundant storage read in loop	Informational	Resolved
Duplicated write to storage in *_decrease_position()* function	Informational	Resolved
Centralization risk in RUSD contract	Informational	Acknowledged
Compiler warnings	Informational	Resolved
Unused imports	Informational	Resolved
Use the latest version of fuels-ts, forc and fuel core	Informational	Resolved

Issues

Critical Severity

1. Missing access control in *set_latest_answer()* function

Description: The bug in the *set_latest_answer()* function occurs because there is no access control enforcing that only authorized users can set the price. As a result, anyone can call this function and change the return value of the *latest_answer()* function.

```
Unset
contracts/pricefeed/src/main.sw
79: // require(get_sender() == storage.gov.read(), Error::PricefeedForbidden);
```

Source:
https://github.com/burralabs/ruscet-contracts/blob/93c58b12d72429d1a680429bcbf72c4246a70502/contracts/pricefeed/src/main.sw#L79

The commented-out line of code prevents unauthorized users from calling the function by checking if the caller value returned by *get_sender()* matches the stored governance address *storage.gov.read()*. Although this may have been commented out for testing purposes, it is best to fix this issue and update any tests that may be affected by this change.

Potential Risk: Since this line is commented out, anyone can call the function and set the price, which exposes the system to price manipulation attacks, which can lead to direct loss of user funds.

Suggested Mitigation: Add the *_only_gov()* modifier to the function or complete the integration to Pyths pricefeeds

Ruscet: This is a non-issue as we'll be switching to using Pyth's pricefeeds for low-latency prices.

Linum Labs: Acknowledged

2. Unauthorized access in the vault-pricefeed *update-price()* function

Description: The *update-price()* function in the vault-pricefeed contract allows any arbitrary caller to set the price of an asset.

Potential Risk: This function allows an attacker to arbitrarily set the oracle price from which the vault reads.
Suggested Mitigation: Add *_only_gov()* modifier to the function or complete the integration to Pyths pricefeeds.

Ruscet: Noted, thank you. Will be using Pyth pricefeeds for low-latency prices

Linum Labs: Acknowledged

Medium Severity

3. Balance drift between UTXO and contract

Description: In the RLP, RUSD, and Yield Asset contracts, there is a significant business logic flaw in how user balances are tracked. This bug can result in a situation where a user is unable to call the *transfer()* function for RLP, RUSD, or Yield Asset contracts. The issue stems from the *_transfer()* function. The *transfer()* function updates the caller's and recipient's balances in storage and also calls the *transfer_assets()* function, which creates a UTXO with the amount transferred to the

recipient. The problem arises because the user's asset balance in storage does not necessarily match the actual balance reflected via the *balance_of()* method.
Potential Risk: This bug can potentially lead to a situation where a user cannot call the *transfer()* function the RLP, RUSD, & Yield Asset.
Suggested Mitigation: The suggestion is to remove the storage writes in the *transfer()* function and the *get_balance()* function from the RLP, RUSD, and yield-tracker contracts. However, this may require additional refactoring to track how other functionality is handled in the contracts.

Ruscet: Fixed in PR

Linum Labs: Verified

4. Vault *buy_rusd()* function is not payable yet assumes asset transfer

Description: The *buy_rusd()* function is not payable, yet the function assumes the user will transfer an asset when calling the function. In the tests, a separate transaction transfers the asset to the contract, and then the *buy_rusd()* function is called.
Potential Risk: If the contract is used in this way in a production environment, there is a high risk of front running, which could result in a loss of user funds.
Suggested Mitigation: The *buy_rusd()* function should be made payable, and assets should be transferred to the contract in the function call using the *add_transfer()* method in the Fuels Typescript SDK.

Ruscet: Fixed in PR

Linum Labs: Verified

Low Severity

5. Use of non-standard SRC20 implementation

Description: The RLP, RUSD, and Yield Asset contracts are non-standard SRC20 implementations since they keep track of user balances within the contract itself. The contracts are also non-standard because they have a transfer function that conflicts with fuel's concept of native assets and the underlying UTXO model. This practice can lead to unintended side effects. The *_transfer()* function within the RLP, RUSD, and Yield Asset contracts updates the balance of sender and recipient within storage.

SRC20 example implementation:
https://docs.fuel.network/docs/sway-standards/src-20-native-asset/#example-implementation

Potential Risk: This will lead to unintended discrepancies between a user's true asset balance and the balance of the user written in the storage of the SRC20 contract and can potentially make the *_transfer()* uncallable for some users.
Suggested Mitigation: It is recommended to remove (or significantly modify) the transfer function from the RLP, RUSD, and Yield Asset contracts, as asset transfer can be performed via the native fuel UTXO coin transfer. In order to maintain the staking rewards functionality, it may be possible to expose the *_update_rewards()* function as a public method in the RUSD contract so that any user can call the function and receive staking rewards. It may also be possible to add a method that allows a user to "sync" their balance stored in the contract with the actual asset balance of the native asset of their EOA.

Ruscet: Fixed in PR

Linum Labs: Verified

6. Front running possibility with transactions calling *_transfer_in()*

Description: It is possible to front-run transactions that call the *_transfer_in()* function if the transaction call pattern in the /tests directory is used in production. In the tests, when calling payable functions, there are two transactions that occur:

1. Asset transfer to the contract
2. Function call to the contract

```
Unset
contracts/core/vault/src/internals.sw
pub fn _transfer_in(asset_id: AssetId, vault_storage_: ContractId) -> u64 {
    let vault_storage = abi(VaultStorage, vault_storage_.into());
    let prev_balance = vault_storage.get_asset_balance(asset_id);
    let next_balance = balance_of(ContractId::this(), asset_id);
    vault_storage.write_asset_balance(asset_id, next_balance);
    require(
        next_balance >= prev_balance,
        Error::VaultZeroAmountOfAssetForwarded
    );
    next_balance - prev_balance
}
```

Source:
https://github.com/burralabs/ruscet-contracts/blob/36668ffd579d0f666dcd8ab2530cc096d3bbb2f8/contracts/core/vault/src/internals.sw#L136-#L149

The _*tranfer_in()* function reads the previous asset balance value from storage and the current asset balance using the sway-lib method *balance_of()*. The function returns the difference between the current balance and the previous balance. The difference is considered to be the amount sent by the user to the contract. The issue arises in how the tests in the repository are set up. All of the tests in the repository that call functions that then call the _*transfer_in()* function do not use the fuels-ts method to transfer assets during the function call. In production, it is possible to front-run all transactions that follow this call pattern.
Potential Risk: This transaction call pattern if used in production, is possible to front run as a result of the logic in the _*transfer_in()* function.
Suggested Mitigation: Edit all of the tests that call payable functions to use the fuels-ts method to transfer assets during the function call to ensure the smart contracts are tested in a way that matches how they will be used in production. Another consideration is using the sway-libs *msg_amount()* method to get the exact amount of the asset transferred into the contract instead of computing the difference between the previously stored and current balances.

Ruscet: Fixed in PR

Linum Labs: Verified

7. Dead code in vault-pricefeed contract

Description: The _*get_amm_price()* function always returns zero, leading to dead code in parts of the vault-pricefeed contract. As a result, any conditional checks that rely on the AMM price being greater than zero never execute, effectively making the AMM price logic redundant.

Multiple examples of using amm_price:

```
Unset
contracts/core/vault-pricefeed/main.sw
if amm_price > 0 {
    if maximize && amm_price > price {
        price = amm_price;
    }
    if !maximize && amm_price < price {
        price = amm_price;
    }
}
```

```
Unset
contracts/core/vault-pricefeed/main.sw
let diff = if amm_price > primary_price {
    amm_price - primary_price
} else {
    primary_price - amm_price
};
if diff.mul(BASIS_POINTS_DIVISOR) <
primary_price.mul(storage.spread_threshold_basis_points.read().as_u256()) {
        if storage.favor_primary_price.read() {
            return primary_price;
        }
```

```
           return amm_price;
     }
     if maximize && amm_price > primary_price {
         return amm_price;
     }
     if !maximize && amm_price < primary_price {
         return amm_price;
     }
    primary_price
```

Get_amm_price() function returning zero

```
Unset
fn _get_amm_price(asset: AssetId) -> u256 {
    0
}
```

Source:

1) https://github.com/burralabs/ruscet-contracts/blob/36668ffd579d0f666dcd8ab2530cc096d3bbb2f8/contracts/core/vault-pricefeed/src/main.sw#L298-#L306
2) https://github.com/burralabs/ruscet-contracts/blob/36668ffd579d0f666dcd8ab2530cc096d3bbb2f8/contracts/core/vault-pricefeed/src/main.sw#L407-#L428

3) https://github.com/burralabs/ruscet-contracts/blob/36668ffd579d0f666dcd8ab2530cc096d3bbb2f8/contracts/core/vault-pricefeed/src/main.sw#L532-#L534

Suggested Mitigation: Remove unnecessary checks that will result in zero or finish migration to Pyths pricefeeds

Ruscet: This is likely a non-issue as we'll be transitioning to using Pyth's pricefeeds

Linum Labs: Acknowledged

8. *write_buffer_amount()* is not used, and duplicates *set_buffer_amount()*

Description: The *write_buffer_amount()* function is unused and duplicates the logic of the *set_buffer_amount()* function.
Suggested Mitigation: Consider removing the *write_buffer_amount()* as it duplicates the logic of the *set_buffer_amount()* function.

Ruscet: Fixed in PR

Linum Labs: Verified

9. *write_max_rusd_amount()* not used

Description: The *write_max_rusd_amount()* is unused
Suggested Mitigation: Remove the *write_max_rusd_amount()*

Ruscet: Fixed in PR

Linum Labs: Verified

10. *AssetId::non_zero()* always returns true

Description: The *non_zero()* method in the AssetId struct always returns true, regardless of whether the AssetId is zero or non-zero. This is due to the hardcoded true value in the method rather than performing the intended comparison logic.

Suggested Mitigation: Uncomment or implement the comparison logic to check if self is equal to the zero value of AssetId.

Ruscet: Great catch, but unfortunately like a couple of the other informational issues you found earlier, this breaks compiler inlining. In this particular scenario, the *AssetId.non_zero()* call is only used once in the *set_asset_config()* method and it's equivalent to removing this check entirely. imo, it's not a big deal to verify if the asset is ZERO because it's not public and the tx would just revert elsewhere in the fn. Explicitly removing this check however also breaks the inlining, so it's a delicate situation here that I'll label as a TODO when the compiler is more stable. Thanks!

Linum Labs: Acknowledged

Informational Severity

11. Centralized control in RLP asset minting and burning

Description: The contract initializer and any added minters have the ability to mint or burn unlimited amounts of the RLP asset. This introduces a centralization risk where these privileged accounts can arbitrarily affect the total supply of the asset, potentially undermining trust in the system and creating governance risks.
Suggested Mitigation: Consider implementing the following changes to mitigate centralization risk:

1. Remove the ability for any arbitrary amount to be minted or burned by a privileged account.
2. In the *initialize()* function, allow an initial minting of the asset to set the initial supply, and then lock minting/burning behind decentralized mechanisms such as multisig or governance votes.
3. Implement access control mechanisms, like a time delay or community voting, to limit the power of minters and reduce centralization risks.

Ruscet: This is by design. Minter roles are team multisigs, in the future, they will be delegated to governance/community voting

Linum Labs: Acknowledged

12. Unused parameter in ***get_funding_fee()*** function

Description: The function *get_funding_fee()* contains input parameters that are not used in the function body. Specifically, the parameters account, index_asset, and is_long are passed into the function but never utilized. This results in unnecessary complexity and could potentially lead to slight inefficiencies.

Potential Risk:

- Gas Optimization: Although the impact is minimal, reducing unnecessary input parameters can slightly improve gas usage.
- Code Readability: Removing unused parameters improves code clarity and maintainability.

Suggested Mitigation: Remove the unused input parameters account, index_asset, and is_long from the *get_funding_fee()* function signature. Update all instances where this function is called to reflect the updated parameter list. This will help reduce gas costs and improve code readability.

Ruscet: This breaks some compiler inlining optimizations that don't allow the contract to compile. Will address this in a future (more stable) release of the Sway compiler

Linum Labs: Acknowledged

13. Update of type generation library

Description: The project is currently using the [Caer library](), which is primarily a Python computer vision library, for file handling and path operations related to ABI formatting. While Caer provides the needed functionality, it is somewhat out of place in a smart contract library.

Suggested Mitigation: Consider using Python's standard libraries (e.g., os, pathlib) for file handling and path operations. This change would make the codebase more intuitive for contributors and align the implementation with best practices, ensuring the use of libraries that are appropriate for the domain of the project.

Ruscet: It is to be noted the Caer library is used only to format some files prior to testing. Granted that it may be out of place, so will keep this open as a future TODO.

Linum Labs: Acknowledged

14. Unused parameter in *_update_cumulative_funding_rate()* function

Description: The parameter *_index_asset* is unused in the *update_cumulative_funding_rate()* function.

Suggested Mitigation: Remove input _index_asset as an input parameter since it isn't used in the function *_update_cumulative_funding_rate()*. Update all functions that call *_update_cumulative_funding_rate()*.

Ruscet: Unfortunately addressing this breaks the inlining optimizations the Sway compiler applies. This is something we will definitely consider at a more stable version of the Sway compiler. Will leave this issue open intentionally

Linum Labs: Acknowledged

15. Use the SRC-2 standard for inline documentation and comments

Description: Consider adding comments to main public functions that adhere to the SRC-2 standard.

Suggested Mitigation: Consider adding SRC-2 compliant comments to improve the clarity of the codebase and to make it easier for other projects to integrate with the

Ruscet Contracts. The lack of documentation can make it more difficult to quickly understand the functionality of certain functions.

Ruscet: Acknowledged

Linum Labs: Acknowledged

16. Redundant storage read in loop

Description: In multiple parts of the codebase there are redundant reads to storage within a loop leading to increased gas usage. Examples include but are not limited too:

```
Unset
contracts/core/vault-pricefeed/src/main.sw
464: while i < storage.price_sample_space.read()


contracts/assets/rusd/src/main.sw
184: while i < storage.yield_trackers.len()
```

Potential Risk: Higher gas costs
Suggested Mitigation: Consider setting the storage read as a variable, and using the variable in the while loop to reduce gas across the codebase.

Ruscet: Fixed in PR

Linum Labs: Verified

17. Duplicated write to storage in *_decrease_position()* function

Description: There is a duplicate call to *vault_storage.write_position()* in the *_decrease_position()* function.
Suggested Mitigation: Consider removing duplicate calls to vault_storage on line 319 to reduce the gas cost when calling the *_decrease_position()* function.

Ruscet: Added label to indicate it will be a future fix

Linum Labs: Acknowledged

18. Centralization risk in RUSD contract

Description: The *add_vault()* and *remove_vault()* functions allow the contract initializer to change the vault contract address. The vault contract has the privilege of calling the *mint()* and *burn()* functions in the RUSD contract.
Suggested Mitigation: Consider limiting the ability to update the vault contract to a team multisig or voting contract.

Ruscet: Acknowledged. Gov will be a team multisig

Linum Labs: Acknowledged

19. Compiler warnings

Description: During the compilation of the Ruscet contracts, a significant number of compiler warnings are generated. Most of these warnings are related to the Sway compiler's detection of state changes in the caller contract following an external contract call. These warnings are inherent to the architectural design of the Ruscet contracts, where the application's logic is distributed across multiple contracts. This design is secure, as all invoked contracts are known, immutable by users, and intentionally structured in this manner. However, it is recommended that other compiler warnings are addressed:

```
Unset
contracts/core/vault/src/utils.sw
136 && 278:
vault_utils.validate_liquidation(
        account,
        collateral_asset,
        index_asset,
        is_long,
        true
 );

contracts/core/vault/src/utils.sw
544: _decrease_position(..
```

Suggested Mitigation: Consider catching the return values as such:

```
Unset
let (_liquidation_state, _margin_fees) = vault_utils.validate_liquidation(
        account,
        collateral_asset,
        index_asset,
        is_long,
        true
 );
```

```
let _amount_after_fees = _decrease_position(..
```

Ruscet: Fixed in PR

Linum Labs: Verified

20. Unused imports

Description: There are several uses of unused imports.
Suggested Mitigation: Consider removing the unused imports.

Ruscet: Fixed in PR

Linum Labs: Verified

21. Use latest version of fuels-ts, forc and fuel core

Description: Currently the versions of fuels-ts, forc and fuel core are outdated
Suggested Mitigation: Ensure that fuels-ts, forc, and fuel-core are updated to their latest versions at the time of deployment. Currently the most up to date versions are: fuels-ts v0.94.6, forc v0.64.0, and fuel-core v0.36.0

Ruscet: Bumped

Linum Labs: Verified

Disclaimer