HashCloak

Code Review and Security Assessment

For

Spark Fi

Initial Delivery: October 18, 2024

Prepared For:

Vitali Dervoed | *Composability Labs*

Alex Nagorny | *Composability Labs*

Prepared by:

Manish Kumar | *HashCloak Inc.*

Hernan Vanegas | *HashCloak Inc.*

Table Of Contents

Executive Summary

Composability Labs engaged *HashCloak Inc.* to audit Spark, a decentralized order book-based decentralized exchange implemented in Sway for the Fuel blockchain. The contract is designed to facilitate secure and transparent trading.

In the initial phase of the audit, our team familiarized ourselves with the relevant components in scope and sought to understand the overall Spark architecture by reviewing the code and the provided documentation. Also, we identified the areas of concern and possible attack vectors by conducting research and communicating with the team. We assessed the tests done using the Fuel Rust SDK, which allowed us to develop a more profound understanding of the orderbook contracts. Additionally, we ran the sway-analyzer, a static tool, to detect potential vulnerabilities in the codebase. We also tested the functionality of the spark-market contract by deploying the contract instance on the testnet and using the spark-cli for testing the abi's of spark-market and performing some vulnerability checks. These include edge cases such as zero-inputs in deposit, withdrawals, and order creations.

We found the source code to be of good quality, accurately representing the entities outlined in the specification and supported by sufficient testing. The code demonstrates idiomatic and fluent interaction between all system components, making it straightforward to follow. However, we identified a lack of documentation throughout the codebase. Further, we think that the interactions between the fee mechanisms should be more detailed to be able to understand the quirks of the implementations. While much of the formulas presented in the code were assessed based on the auditors' understanding, we believe that comprehensive documentation establishing clear rules for the fee structures is necessary. This would serve as a reliable reference to confirm whether certain behaviors in the code are intentional or not.

Overall, the issues that we identified range from medium to informational severity:

Severity	Number of Findings
Critical	0
High	0
Medium	2
Low	2
Informational	4

Scope

For the audit, we reviewed the repository located at https://github.com/compolabs/orderbook-contract, specifically at commit 4d146702952e72c1f4348c7ed117a59b0b8e6f86.

Components in scope:

- spark-market:
  - spark-market/src/data_structures/account.sw
  - spark-market/src/data_structures/asset_type.sw
  - spark-market/src/data_structures/balance.sw
  - spark-market/src/data_structures/limit_type.sw
  - spark-market/src/data_structures/match_result.sw
  - spark-market/src/data_structures/math.sw
  - spark-market/src/data_structures/order_change.sw
  - spark-market/src/data_structures/order.sw
  - spark-market/src/data_structures/order_type.sw
  - spark-market/src/data_structures/protocol_fee.sw
  - spark-market/src/data_structures/user_volume.sw
  - spark-market/src/data_structures.sw
  - spark-market/src/errors.sw
  - spark-market/src/events.sw
  - spark-market/src/interface.sw

    - spark-market/src/main.sw
- spark-registry:
    - spark-registry/src/errors.sw
    - spark-registry/src/events.sw
    - spark-registry/src/main.sw

Overview

Spark is a next-generation decentralized exchange (DEX) protocol designed for seamless spot trading, offering fast execution and minimal fees. It provides access to significant market liquidity on the Fuel network while ensuring self-custody of funds, giving users full control and security over their assets. Spark emphasizes community involvement with governance features through a DAO treasury, aiming for community-driven decisions. It offers low transaction costs, personalized trading tools, and caters to market makers with its Market Makers Incentivization Program. Spark's mission is to empower Ethereum DeFi developers and traders with a transparent, efficient, and secure platform.

Methodology

The audit was conducted through a combination of manual verification, writing test cases, and group audit sessions. Our primary focus was to ensure that the actual implementation aligned with the available specifications. Additionally, we assessed the code for various types of vulnerabilities, including:

- Single, cross-function, and cross-contract reentrancy
- Any undefined behavior
- Which function can be invoked by anyone and which should be invoked by only_owner based on the doc.
- Visibility of each function
- Check user inputs (e.g., prices, amounts, asset IDs) are validated properly.
- Unsafe arithmetic errors such as overflows, underflows, or rounding errors
- Precision Handling
- Fee Withdrawal can be done only by authorized person
- Any kind of fees Exploitations
- Denial of Service (DoS)
- Transaction-Ordering Dependence (TOD) / Front Running
- Timestamp Dependence

- Look for any dead code
- Replay protection
- Data privacy, data leaking, and information integrity
- Input validation

Additionally, we ensured that:

- The design and architecture match the docs and the flow.
- Correct Pricing & Matching Logic
- Every significant action like order creation, cancellation, trade execution, and deposits emits the correct event
- Correct account Balance Updates
- All storage updates occur atomically and correctly. If a function fails midway, no partial state updates should persist
- Fee calculations are done correctly based on the doc.
- Verify that funds are locked appropriately when orders are placed, and unlocked when orders are canceled or fulfilled.

Overview of Evaluated Components

- spark-market
    - Single, cross-function, and cross-contract reentrancy
    - Any undefined behavior
    - Which function can be invoked by anyone and which should be invoked by only_owner based on the doc.
    - visibility of each function
    - check user inputs (e.g., prices, amounts, asset IDs) are validated properly.
    - Unsafe arithmetic errors such as overflows, underflows, or rounding errors
    - Precision Handling
    - Fee Withdrawal can be done only by authorized person
    - Any kind of fees Exploitations
    - Denial of Service (DoS)
    - Transaction-Ordering Dependence (TOD) / Front Running
    - Timestamp Dependence
    - Look for any dead code
    - Replay protection

- Data privacy, data leaking, and information integrity
- The design and architecture match the docs and the flow.
- Correct Pricing & Matching Logic
- Every significant action like order creation, cancellation, trade execution, and deposits emits the correct event
- Correct account Balance Updates
- All storage updates occur atomically and correctly. If a function fails midway, no partial state updates should persist
- Fees calculations are done correctly based on the doc
- Verify that funds are locked appropriately when orders are placed, and unlocked when orders are canceled or fulfilled
- Verify the correctness of the system by deploying a testnet instance and using the spark-cli to interact with the contract. In particular we checked:
    i. Impossibility to match orders of the same type
    ii. Opening an already closed order
    iii. Closing an already closed order
    iv. Matching multiple orders
    v. Verify that the amounts were locked correctly when opening orders
    vi. Verify the correct calculation of prices when opening an order

- spark-registry
    - Correct registration and deregistration of the market
    - The registration and deregistration should be done by only the owner
    - The storage is updated correctly
    - Look for any dead code
    - Replay protection
    - Which function can be invoked by anyone and which should be invoked by only_owner based on the doc
    - Data privacy, data leaking, and information integrity
    - Timestamp Dependence
    - Replay protection
    - Input validation

Findings

SPF-1: `increase_user_volume` does not update the storage

Type: Medium

Files affected:

- spark-market/src/main.sw

Description: In the current version, the `increase_user_volume` function does not update the storage, and therefore, the change in the user volume will not be persistent. As a consequence, the protocol fees will not be computed correctly, given that they are calculated based on the current user volume. The function only updates the `UserVolume` object, as shown in the following code:

```
#[storage(read, write)]
fn increase_user_volume(user: Identity, volume: u64) {
    extend_epoch();
    let _ =
storage.user_volumes.get(user).try_read().unwrap_or(UserVolume::new()
).update(storage.epoch.read(), volume);
}
```

Impact: If the user volume is not increased, the protocol fees will not be computed correctly given that the protocol fees are calculated based on the user volume.

Suggestion: The function should update the storage by inserting the new user volume under the user ID.

Status: This issue has been corrected in https://github.com/compolabs/orderbook-contract/commit/4b6cd1905b913fe512b9a802399b0a38e8a129b4

SPF-2: Possible front-running during order matching

Type: Medium

Files affected:

- spark-market/src/main.sw

Description:

The functions `match_order_many`, `fulfill_order_many`, and `match_order_pair` may be susceptible to front-running. Specifically, front-running might be possible because, since anyone can match orders, someone could front-run a matching order transaction by simply copying the transaction from the *mempool* and offering a higher gas fee. Hence, this transaction will be more likely to be accepted in the block-building process instead of the original one. This means that the adversarial matcher could publish a match and receive the matcher fee without even trying to match the orders, which makes the competition unfair.

Impact: By exploiting this front-running vulnerability, an adversarial matcher can earn the matcher fee without performing any actual matching work (i.e. simply by copying an already published transaction), which undermines the fairness of the matcher competition.

Suggestion: One possible solution is to tie the matcher who works on the matching transaction to the transaction itself.

Status: This issue has been acknowledged by the Spark team.

SPF-3: Add restriction for IOC orders to the function `fullfill_order_many`

Type: Low

Files affected:

- spark-market/src/main.sw

Description: At the end of the `fulfill_order_many` function, orders should only be canceled if the limit type is IOC. However, in its current state, the function cancels any order, including the ones with a GTC limit type. GTC type allows the order to be opened until it is

completely fulfilled. The origin of this issue is that the conditional shown below does not check that the current transaction is IOC.

```
if matched == MatchResult::PartialMatch {
    cancel_order_internal(id0);
}
```

Impact: GCD orders may be deleted if they are partially fulfilled, which is not an intended behavior.

Suggestion: We recommend adding the condition `limit_type == LimitType::IOC` to the conditional in charge of canceling the order. The final version of the conditional should be:

```
if matched == MatchResult::PartialMatch && limit_type ==
LimitType::IOC {
    cancel_order_internal(id0);
}
```

Status: This issue has been corrected in https://github.com/compolabs/orderbook-contract/commit/72b866d81c755364f4edffa7b475d35fb7045502

SPF-4: `match_order_many` and `fulfill_order_many` do not have any restriction on the length of the orders given as inputs

Type: Low

Files affected:

- spark-market/src/main.sw

Description: The `match_order_many` and `fulfill_order_many` methods in the ABI `SparkMarket` do not restrict the length of orders given as input vectors. Specifically, both functions receive a vector of orders (`orders: Vec<b256>`) without limiting the size of this

vector. Hence, an adversary may submit a transaction to match multiple orders with an excessively long vector, which could potentially exhaust all available gas.

Impact: Without restrictions on the length of input vectors, this could potentially lead to Block Gas Limit Exhaustion.

Suggestion: We recommend establishing a limit on the length of the arrays provided as input to the mentioned ABI methods.

Status: This issue has been informed to the team.

SPF-5: Lack of documentation in source code

Type: Informational

Files affected:

- spark-market/src/data_structures/*
- spark-market/src/errors.sw
- spark-market/src/events.sw
- spark-registry/src/errors.sw
- spark-registry/src/events.sw

Description: There are no documentation comments inside the source code and the structures in the mentioned files are not explained. Much of the interpretation about the purpose of the structures can be obtained from the expertise in the field, but it is important to clearly state the design rationale and the interaction between the components of the system. Additionally, detailed documentation on the system's main functionalities and the fee computation process is essential. For instance, the code section that computes the fee amount when the seller and the buyer are the same identity lacks clarity, and it took considerable time to verify the correctness of the equation. Stating explanations for why certain equations are made and why some conditionals are included would significantly improve the code's readability and auditability.

Impact: The lack of documentation makes the code difficult to audit and maintain.

Suggestion: We recommend adding thorough documentation within the source code, describing the structures and explaining the purpose of each function. Furthermore, more detailed external documentation should be provided for auditors, users, and maintainers.

Status: Not yet informed.

SPF-6: `log_order_change_info` does not emit any logging

Type: Informational

Files affected:

- spark-market/src/main.sw

Description: The `log_order_change_info` function is not logging any event. The function only modifies the `storage.order_change_info` but it does not emit any logging. It would be important to make clear whether this function is used off-chain for debugging by adding documentation or providing a name that reflects this behavior. At first glance, the function name suggests that it should emit an event. This issue was also identified by the sway-analyzer under the vulnerability flag of `missing_log`. For more details, see the sway-analyzer report in the appendix here (last line).

Suggestion: Either add a log event at the end of the function or rename the function to better reflect its intended behavior.

Status: The Spark team has acknowledged this issue and plans to rename the function to better represent its intended meaning.

SPF-7: Dead code in `mul_div_rounding_up` function

Type: Informational

Files affected:

- spark-market/src/data_structures/math.sw

Description: The function `mul_div_rounding_up` is not used in any part of the codebase, making it dead code.

Suggestion: We recommend removing this unused function from the source code.

Status: This issue has been corrected in https://github.com/compolabs/orderbook-contract/commit/72b866d81c755364f4edffa7b475d35fb7045502

SPF-8: The `require` in the `order_id` function has no effect

Type: Informational

Description: The `order_id` function contains a `require` statement that has no effect. The current version of the `order_id` function is shown below:

```
fn order_id(
    order_type: OrderType,
    owner: Identity,
    price: u64,
    block_height: u32,
    order_height: u64,
) -> b256 {
    let asset_type = AssetType::Base;
    require(
        asset_type == AssetType::Base || asset_type ==
AssetType::Quote,
        AssetError::InvalidAsset,
    );
    Order::new(
        1,
        asset_type,
        order_type,
        owner,
        price,
        block_height,
```

```
        order_height,
        0,
        0,
        0,
    ).id()
}
```

The `asset_type` variable is equal to `AssetType::Base`, hence the condition in the next `require` is always true, and that `require` has no effect.

Suggestion: Since the `asset_type` is consistently `AssetType::Base`, the `require` should be removed.

Status: Not yet informed.

References

- https://www.investopedia.com/articles/active-trading/042414/what-makertaker-fees-mean-you.asp
- https://compo-labs.notion.site/Orderbook-cf1a7841650d42ec8bfaec084cb7a77b
- https://github.com/compolabs/orderbook-contract/blob/master/docs/scheme.drawio.png
- https://fuellabs.github.io/sway/v0.64.0/book/index.html
- https://docs.fuel.network/docs/intro/what-is-fuel/
- https://scsfg.io/hackers/
- https://github.com/ourovoros-io/sway-analyzer
- https://docs.sprk.fi/
- https://github.com/Raiders0786/web3-security-resources/blob/main/README.md#-sway-language-security--audit-resources

Appendix

Sway-analyzer report

Spark-market

Detectors flag	Description	Result	Details
arbitrary_asset_transfer	Checks for functions that transfer native assets to an arbitrary address without access restriction.	No Detection	N/A
arbitrary_code_execution	Checks for functions that make use of the `LDC` assembly instruction without access restriction.	No Detection	N/A
boolean_comparison	Checks if an expression contains a comparison with a boolean literal, which is unnecessary.	No Detection	N/A
discarded_assignment	Checks for variables that are assigned to without being utilized.	No relevant detection	All the outputs are either redundant or false positive
division_before_multiplication	Checks for division operations before multiplications, which can result in value truncation	inside the function u64::mul_div_rounding_up, but that is a dead code	N/A
explicit_return_statement	Checks for functions that end with explicit	No Detection	N/A

	`return` statements, which is unnecessary		
external_call_in_loop	Checks if any functions contain any loops which performs calls to external functions.	No Detection	N/A
inline_assembly_usage	Checks functions for inline assembly usage.	No Detection	N/A
large_literal	Checks for expressions that contain large literal values, which may be difficult to read or interpreted incorrectly.	L78: Storage contains a large literal: `2629800`. Consider refactoring it to be more readable: `2_629_800`	N/A
locked_native_asset	Checks if a contract can withdraw potential incoming native assets.	No Detection	N/A
magic_number	Checks for expressions that contain irregular numerical constants that can be introduced as named constants.	No relevant detection	Redundant results
manipulatable_balance_usage	Checks if any functions contain balance usage which	No Detections	N/A

	can potentially be manipulated.		
missing_logs	Checks for publicly-accessible functions that make changes to storage variables without emitting logs.	`L1113: The` `` `log_order_change_info` `` `function writes to` `` `storage.order_change_info` `` `without being logged.` The other results shown by the sway-analyzer are not relevant.	see SPF-6
msg_amount_in_loop	Checks for calls to `std::context::msg_amount()` or `std::registers::balance()` inside a while loop. In most cases, the result of the call should be stored in a local variable and decremented over each loop iteration.	No Detections	N/A
non_zero_identity_validation	Checks to see if functions containing `Identity`, `Address` and `ContractId` parameters are	No relevant detections	Redundant

	checked for a zero value.		
potential_infinite_loop	Checks for potentially infinite loops.	No Detections	N/A
redundant_comparison	Checks for functions that make redundant comparisons.	No Detections	N/A
redundant_storage_access	Checks for redundant calls to `storage.x.read()` and `storage.x.write(x)`.	No relevant detections	False Positive
storage_field_mutability	Checks for any storage fields that can be refactored into constants or configurable fields.	No relevant detections	False positive
storage_not_updated	Checks for local variables that are read from storage, then modified without being written back to storage.	`L851: The` `execute_trade` ` function has storage bound to local variable` `s_account` `which is not written back to` `storage.account`.	False positive
storage_read_in_loop_condition	Checks for loops that contain a storage read in their condition,	No Detections	N/A

	which can increase gas costs for each iteration.		
strict_equality	Checks for the use of strict equalities, which can be manipulated by an attacker.	No Detections	N/A
unchecked_call_payload	Checks for functions that supply a `raw_ptr` argument to the `CALL` assembly instruction, or a `Bytes` argument without checking its length.	No Detections	N/A
unprotected_initialization	Checks for initializer functions that can be called without requirements.	No Detections	N/A
unprotected_storage_variable	Checks for functions that make changes to storage variables without access restriction.	No relevant detections.	Redundant
unsafe_timestamp_usage	Checks for dependence on `std::block::timestamp` or `std::block::timestamp_of_block`, which can	No Detections	N/A

	be manipulated by an attacker.		

unused_import	Checks for imported symbols that are not used.	`spark-market/src/interface.sw: L11: Found unused import: `UserVolume`. Consider removing any unused imports.`
		
		`spark-market/src/events.sw: L4: Found unused import: `AssetType`. Consider removing any unused imports.`
			1. UserVolume is imported but not used in spark-market/interface.sw
			2. AssetType is imported in spark-market/event.sw but not used.

weak_prng	Checks for weak PRNG due to a modulo operation on a block timestamp.	No Detections	N/A

Severity classification

Type	Description
Informational	The issue does not pose any vulnerability risk but is relevant

	to best practices. This includes lack of documentation, dead code, unused imports, etc.
Low	The risk is small or has low relevance for the customer.
Medium	The issue could expose a vulnerability to the system, but the conditions to exploit the vulnerability are very specific or high to reach.
High	The issue exposes a vulnerability that affects the end-user drastically, and the conditions to exploit the issue are relatively easy to achieve.
Critical	A vulnerability that poses an immediate and severe threat, potentially allowing full system compromise or complete bypass of security controls.
Undetermined	The effects, severity, or specific exploits for the issue have not been determined during the engagement.