Three Sigma Labs

Code Audit

Ordiswap

Ordiswap Bitcoin AMM

Disclaimer

Code Audit

Ordiswap Bitcoin AMM

Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code's manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

Table of Contents

Code Audit

Ordiswap Bitcoin AMM

Table of Contents

Summary

Code Audit

Ordiswap Bitcoin AMM

Summary

Three Sigma audited Ordiswap in a 4 person week engagement. The audit was conducted from 26-12-2023 to 10-01-2024.

The codebase is still actively being developed and the Three Sigma team recommended the Ordiswap team to keep improving the protocol before launch. The Ordiswap team stated that the codebase will be released with a disclaimer and commentary that the audit will continue after the experimental mainnet launch, and this will all be accurately communicated to its users.

Protocol Description

Ordiswaps is a Bitcoin exchange which uses an off-chain server to manage the logic and parameters of the Automated Market Maker (AMM). This includes handling trades, liquidity provision, and setting dynamic AMM parameters.

Scope

Code Audit

Ordiswap Bitcoin AMM

Scope

The scope of the engagement includes the entire backend code for the Ordiswap server.

Assumptions

Since Ordiswap is a centralized custodial service, it is an assumption that the Ordiswap team is a trusted actor that will not act maliciously since they hold the custody of all the funds.

Methodology

Code Audit

Ordiswap Bitcoin AMM

Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

Taxonomy

In this audit we report our findings using as a guideline Immunefi's vulnerability taxonomy, which can be found at immunefi.com/severity-updated/. The final classification takes into account the severity, according to the previous link, and likelihood of the exploit. The following table summarizes the general expected classification according to severity and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

Severity / Likelihood	LOW	MEDIUM	HIGH
NONE	None		
LOW	Low		
MEDIUM	Low	Medium	Medium
HIGH	Medium	High	High
CRITICAL	High	Critical	Critical

Project Dashboard

Code Audit

Ordiswap Bitcoin AMM

Application Summary

Name	Ordiswap AMM
Commit	0c7e1081cb20a6cb69671e55028642cd4d6c444d
Language	Javascript
Platform	Bitcoin

Engagement Summary

Timeline	26-12-2023 to 10-01-2024
Nº of Auditors	2
Review Time	4 person weeks

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	3	2	1
High	3	3	0
Medium	3	1	2
Low	13	11	2
None	47	44	3

Category Breakdown

Suggestion	30
Documentation	1
Bug	24
Optimization	11
Good Code Practices	53

Code Maturity Evaluation

Code Audit

Ordiswap Bitcoin AMM

Code Maturity Evaluation

Code Maturity Evaluation Guidelines

Category	Evaluation
Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system.
Arithmetic	The proper use of mathematical operations and semantics.
Centralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Code Stability	The extent to which the code was altered during the audit.
Upgradeability	The presence of parameterizations of the system that allow modifications after deployment.
Function Composition	The functions are generally small and have clear purposes.
Front-Running	The system's resistance to front-running attacks.
Monitoring	All operations that change the state of the system emit events, making it simple to monitor the state of the system. These events need to be correctly emitted.
Specification	The presence of comprehensive and readable codebase documentation.
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage.

Code Maturity Evaluation Results

Category	Evaluation
Access Controls	Not in the scope of the audit. The codebase is a centralized closed-source server that can be updated at any time.
Arithmetic	Unknown. The team did not have enough time to thoroughly check all the arithmetics operations.
Centralization	Weak. The team has full control over the deposited funds and application functionality. The team signaled intention to minimize centralization as more open-source development on the bitcoin ecosystem becomes available.
Code Stability	Satisfactory. The code was stable during the audit.
Upgradeability	Not in the scope of the audit. The codebase is a centralized closed-source server that can be updated at any time.
Function Composition	Satisfactory. Functionalities are well split into different contracts and helpers.
Monitoring	Weak. The codebase relies on external indexing tools to properly function. The team signaled intention to improve monitoring as more open-source development on the bitcoin ecosystem becomes available.
Specification	Weak. The codebase does not have documentation or specification.
Testing and Verification	Weak. The codebase does not have testing or verification.

Findings

Code Audit

Ordiswap Bitcoin AMM

Findings

3S-ORDS-C01

src/core/orderbook Swap can happen without liquidity in pool

Id	3S-ORDS-C01
Classification	Critical
Severity	Critical
Likelihood	High
Category	Bug
Status	Acknowledged

Description

A swap of tokens can happen without existing any liquidity in the pool as there are no checks being done in the backend. This means the user will be able to send tokens, receive 0 tokens out in return, and disallow liquidity to be added to the pool since the checks on AddLiquidity do not contemplate a case where there's liquidity of just 1 of the tokens.

Recommendation

Check for liquidity of the tokens on the pool before processing a swap.

3S-ORDS-C02

src/server: It is possible to create 2 orders to create pools with the same tokens and lp token in the createPool endpoint

Id	3S-ORDS-C02
Classification	Critical
Severity	Critical
Likelihood	Medium
Category	Bug
Status	Addressed

Description

Related to the previous issue, because pools that are pending creation are not considered as existing, by calling the **`createPool`** endpoint with the same argument 2 times, it is possible to have 2 pending orders with the same tokens. In the end only 1 pool will be created, since there is a check in the moment of creation of the pool, but the correct behavior should be to not be possible to submit orders that create pools that are already pending creation. Currently, if the user does this he pays the fee 2 times, which is not the intended behavior

Recommendation

When creating a pool, make sure that there is no pool creation pending order with the same arguments to avoid the user spending money on an order that will fail

3S-ORDS-C03

src/server: After creating a pool using the endpoint createPool, pool doesn't appear in getPoolList endpoint

Id	3S-ORDS-C03
Classification	Critical
Severity	Critical
Likelihood	High
Category	Bug, Good Code Practices
Status	Addressed

Description

Whenever a pool is created using the **`createPool`** endpoint, an order is placed to create that pool, but whenever the **`getPoolList`** endpoint is called it doesn't returning pools that are still waiting for the transaction to be confirmed. This can lead to unwanted behavior in the frontend since the pool is already being created.

Recommendation.

Add pools that are pending creation to the response of **`getPoolList`** endpoint to signal that those pools already exist.

3S-ORDS-H01

src/core/orderbook: All the getTransactionFee calls don't verify if the return value is valid

Id	3S-ORDS-H01
Classification	High
Severity	Critical
Likelihood	Low
Category	Bug
Status	Addressed

Description

All the calls to the function **`getTransactionFee`** don't check to see if the return value of is valid or not. The subsequent sum will replace the value of **`order.spent_fee`** with **`NaN`**, which is not the intended behavior, and effectively losing critical information of the order.

Recommendation

Verify that the value returned by **`getTransactionFee`** is a valid before summing it to the spent fee of the order and try to fetch the value again if the function returns an error

3S-ORDS-H02

src/core/orderbook: All the getInscriptionSats calls don't verify if the return value is valid

Id	3S-ORDS-H02
Classification	High
Severity	Critical
Likelihood	Low
Category	Bug
Status	Addressed

Description

All the calls to the function **`getInscriptionSats`** don't check to see if the return value of is valid or not. The subsequent sum will replace the value of **`order.spent_fee`** with **`NaN`**, which is not the intended behavior, and effectively losing critical information of the order.

Recommendation

Verify that the value returned by **`getInscriptionSats`** is a valid before summing it to the spent fee of the order and try to fetch the value again if the function returns an error

3S-ORDS-H03

src/core/orderbook: If ord command to create a wallet address fails while creating a pool, the order is invalidated

Id	3S-ORDS-H03
Classification	High
Severity	Critical
Likelihood	Low
Category	Bug
Status	Addressed

Description

Whenever a pool is being created, if the ord command to create a wallet fails then the **`order_status`** variable is set to **`ORDER_STATUS_FAILED`** and is lost forever. This exception should be caught and the order status should be set accordingly so that we can identify the reason why the order failed. **Attention**: this error leads to the direct loss of funds as the user has spent money creating the pool but the other has been set as invalid and it is not possible to track the origin that invalidated the order.

Recommendation

Catch possible exceptions thrown by the **`createWalletAddress`** function and set the order status accordingly, so that these orders can be later identified in the database

3S-ORDS-M01

src/core/orderbook: The failProcess function should receive the order_status that explicitly states the order type

Id	3S-ORDS-M01
Classification	Medium
Severity	Medium
Likelihood	High
Category	Suggestion, Good Code Practices
Status	Addressed

Description

Currently every time an order fails, it has an **`order_status`** of **`ORDER_STATUS_FAILED`**. The reason of failure should be explicit so that orders can be queried per type. This order type can be an argument of the **`failProcess()`** function to facilitate modularity in the code

Recommendation

Explicitly state the type of order in the failure. We recommend the following names for the different orders failing:

`CREATE_POOL_ORDER_FAILED`, **`ADD_LIQUIDITY_ORDER_FAILED`**, **`REMOVE_LIQUIDITY_ORDER_FAILED`**, **`SWAP_ORDER_FAILED`**, **`UPDATE_ORDER_FAILED`**

3S-ORDS-M02

src/core/orderbook Unsafe calls to inscribe_transfer_brc20

Id	3S-ORDS-M02
Classification	Medium
Severity	High
Likelihood	Low
Category	Bug
Status	Acknowledged

Description

Every operation that leads to calling the function **`inscribe_transfer_brc20`** can fail silently without setting the appropriate error in the **`order.description`**.

Recommendation

Wrap the call to **`inscribe_transfer_brc20`** under a function which correctly logs root causes of the problem, or throws an error that can be caught by the handler of each operation.

3S-ORDS-M03

src/core/lib/brc20 inscribe_mint_brc20 directory access check

Id	3S-ORDS-M03
Classification	Medium
Severity	High
Likelihood	Low
Category	Bug
Status	Acknowledged

Description

The function writes to a file determined by **`TEMP_PATH`**. If **`TEMP_PATH`** is not securely configured, there could be security implications, such as directory traversal attacks.

A possible attach while hard to execute can lead to write of invalid inscriptions which can be used to send incorrect amounts of funds and possible drain funds.

Recommendation:

Ensure **`TEMP_PATH`** is securely configured and validate or sanitize the path. Consider using a more secure method to pass data to inscribeOrdinal.

3S-ORDS-L01

src/core/orderbook: If the state of an order is ORDER_STATUS_ORDERED but the transaction is never confirmed on-chain, the order gets stuck

Id	3S-ORDS-L01
Classification	Low
Severity	Low
Likelihood	Medium
Category	Bug
Status	Addressed

Description

Whenever an order is created and has **`ORDER_STATUS_ORDERED`**, but for some reason the transaction is never confirmed on-chain, this order will forever be stuck in a **`ORDER_STATUS_ORDERED`** state. There should be a check that tries to catch the case in which the transaction is not confirmed, and eventually the order is dropped.

Recommendation

Create a check that after a certain amount of time or tries, if the transaction is not confirmed on-chain, then the order is dropped

3S-ORDS-L02

src/server: The checkSignedRequest only checks if the variabless sender_address, fee_txid and fee_rate exist

Id	3S-ORDS-L02
Classification	Low
Severity	Low
Likelihood	Medium
Category	Bug, Good Code Practices
Status	Addressed

Description

In the **`checkSignedRequest`** function there is a check to make sure that **`sender_address`**, **`fee_txid`** and **`fee_rate`** exist, but there is no validation that makes sure that their format is correct. This means that an invalid address, fee transaction id or fee rate could be sent to the saved in the database leading to unwanted behaviours.

Recommendation

Add extra validation that makes sure that the variable **`sender_address`** is a valid bitcoin address, **`fee_txid`** is a valid transaction id and that **`fee_rate`** is a valid fee rate value.

3S-ORDS-L03

src/core/lib/brc20 inscribe_deploy_brc20 no error handling

Id	3S-ORDS-L03
Classification	Low
Severity	Low
Likelihood	Medium
Category	Good Code Practices
Status	Addressed

Description

The function lacks a try-catch block. If any part of the function throws an error (e.g., file write failure, issues in **`inscribeOrdinal`**), it will result in an unhandled promise rejection.

Recommendation:

Implement try-catch for error handling, especially since file operations and network calls are involved.

3S-ORDS-L04

src/core/orderbook: All calls to update order collection have no error handling

Id	3S-ORDS-L04
Classification	Low
Severity	Low
Likelihood	Medium
Category	Bug
Status	Addressed

Description

For example in the function **`processCreatePoolOrder`** the first call **`orderCollection.updateOne`** has no error handling. If there is no matching order in the list due to previous failures the functions will continue running.

Recommendation

Use a codeblock such as:

```
try {
    const updateResult = await orderCollection.updateOne({ _id: order._id },
{ $set: order });
    if (updateResult.matchedCount == 0) {
        console.error(`Failed to update order with id ${order._id}`);
        await failProcess(order);
        break;
    }
} catch (error) {
    console.error(`Error updating order with id ${order._id}: ${error}`);
    await failProcess(order);
    break;
}
```

In this updated code, the result of **`updateOne()`** is stored in **`updateResult`**. If no documents matched the filter **`(_id: order._id)`**, updateResult.matchedCount will be 0, indicating that the update failed. In this case, an error message is logged and the **`failProcess()`** function is called.

If **`updateOne()`** throws an error (for example, if there's a problem with the database connection), the error is caught, logged, and **`failProcess()`** is called.

3S-ORDS-L05

src/core/orderbook: failProcess should correctly handle re-initiating logic

Id	3S-ORDS-L05
Classification	Low
Severity	Low
Likelihood	Low
Category	Good Code Practices
Status	Acknowledged

Description

The issue #42 refer to a behavior that could be handled for correct restart by adding additional logic in the failProcess error handling to catch the right behaviors and update order status.

3S-ORDS-L06

src/core/orderbook: Order processing functions can fail in the middle of the process and need manual restart

Id	3S-ORDS-L06
Classification	Low
Severity	Low
Likelihood	Low
Category	Bug
Status	Acknowledged

Description

In the functions **`processCreatePoolOrder`** **`processAddLiquidityOrder`** **`processRemoveLiquidityOrder`** **`processSwapOrder`** **`processWithdrawBtcOrder`** if any of the steps fail until completion then the process is dropped midway and the corresponding data structures in the **`confirm...`** functions are not correctly initiated. Since there is no error handling process to resume then there would need to be a manual restart of the process.

Recommendation

Update the function to correctly resuming after failing any of the steps.

3S-ORDS-L07

src/server: /tokeninfo & /gettokenbalance endpoints checks if token ticker size is different than 4

Id	3S-ORDS-L07
Classification	Low
Severity	Low
Likelihood	High
Category	Bug
Status	Addressed

Description

In both the **`/tokeninfo`** and **`/gettokenbalance`** endpoints there is a check that returns an error if the token ticker size is different than 4, but in the database there are tokens have a size different than 4, like for example: 🚀. Maybe this is intended as we don't want to consider tokens such as 🚀 as valid, although it is an undocumented inconsistency that should be commented in the code

Recommendation

Remove the check for the ticker size being 4 or explicitly explain why tokens tickers with size different to 4 are invalid

3S-ORDS-L08

src/server: Server doesn't utilize the helmet package middleware.

Id	3S-ORDS-L08
Classification	Low
Severity	Low
Likelihood	
Category	Suggestion, Good Code Practices
Status	Addressed

Description

Helmet is a package that helps secure Express apps by setting HTTP response headers such as Content-Security-Policy, Cross-Origin-Opener-Policy, Cross-Origin-Resource-Policy and many others. Helmet allows to configure the different headers.

Recommendation

Add the Helmet package to the codebase as such:

```
server.use(helmet())
```

3S-ORDS-L09

src/server: Access Control Headers can be configured as middleware to avoid repetition and human errors

Id	3S-ORDS-L09
Classification	Low
Severity	Low
Likelihood	Medium
Category	Good Code Practices
Status	Addressed

Description

The Access-Control-Allow-Origin and Access-Control-Allow-Methods headers can be configured using middleware so that they don't have to be defined in every request. This makes the code cleaner and reduces possible mistakes

Recommendation

Configure a middleware function that sets the Access-Control-Allow-Origin and Access-Control-Allow-Methods for every request. This can be done using the cors library in the following manner:

1. Using the cors middleware library and configuring it with the necessary headers

```
const configCors = {
      origin: FRONT_SERVER,
      methods: ['GET,POST']
}
server.use(cors(configCors))
```

2. Manually setting up a middleware function that configures the headers

```
server.use(function (req, res, next) {
      res.header("Access-Control-Allow-Origin", FRONT_SERVER)
      res.header("Access-Control-Allow-Methods", "GET,POST")
```

```
        res.header("Content-Type", "application/json")
        next()
})
```

3S-ORDS-L10

src/core/brc20-indexer: Mongodb exceptions are never caught, which can lead to discrepancy of information

Id	3S-ORDS-L10
Classification	Low
Severity	Medium
Likelihood	Low
Category	Bug, Good Code Practices
Status	Addressed

Description

The errors thrown by Mongodb insert operations are never handled, which can lead to the discrepancy of information. It can happen that a new token is correctly inserted in the database, but the **`syncedTokenCount`** is not updated correctly. This will create a discrepancy between the number of tokens in the database and the value of **`syncedTokenCount`**.

Recommendation

Handle the mongodb exceptions to make sure that the values are correctly inserted into the database.

3S-ORDS-L11

src/core/brc20-indexer: whenever a request returns an error, the server doesn't retry to send the request

Id	3S-ORDS-L11
Classification	Low
Severity	Low
Likelihood	High
Category	Suggestion, Good Code Practices
Status	Addressed

Description

Whenever a request with axios is made, the server does not retry to send the request to get the correct response. This means that whenever the server makes a request and get's an error response, it returns the error to the client instead of trying to get the correct answer. Ideally, several retries are performed before giving up and returning an error to the client

Recommendation

Use the axios-retry library to configure axios to try again whenever a request fails - https://www.npmjs.com/package/axios-retry

We advise to use the **`axiosRetry.exponentialDelay`** since some requests are being rate limited.

3S-ORDS-L12

src/core/brc20-indexer: isSynced variable may block the sync progress of the server and is useless

Id	3S-ORDS-L12
Classification	Low
Severity	Medium
Likelihood	Low
Category	Bug
Status	Addressed

Description

In the **`brc20-index`** function there is a check to see if the variable **`isSynced`** is set to false, which makes the function return the value false. If it happens that the **`isSynced`** variable is set to false whenever the function is called, the **`brc20-index`** function will never run because it will just return. The consequence is that the server will never sync again.

Recommendation

Remove the check altogether, as well as the variable **`isSynced`**, since the variable does not add any functionality.

3S-ORDS-L13

src/core/brc20-indexer: Exception from axios.get call is never handled

Id	3S-ORDS-L13
Classification	Low
Severity	Low
Likelihood	Medium
Category	Bug
Status	Addressed

Description

Whenever the axios calls throw an exception, this exception is not handled, it's simply logged onto the console. The function then simply returns, meaning that the subsequent checks to see if the response returned an error is never reached, because an exception was thrown:

```
if (!response || !response.data || response.data.code !== 0) {
    return 0
}
```

The occurrences are in the following functions: **`getTokenBalance`**, **`getTokenList`**, **`getTokenInfo`**. In the **`brc20_index`** function there are 2 of this happening.

This may lead to unwanted behaviour such as returning an error back to the client instead of trying again to get a response.

Recommendation

Catching the exception the way it's recommended in the documentation in order to make sure that we get the expected behaviour depending on the function. - https://axios-http.com/docs/handling_errors

3S-ORDS-N01

src/core/orderbook: All the calls to isTransactionConfirmed don't check if the order is stuck

Id	3S-ORDS-N01
Classification	None
Category	Good Code Practices
Status	Addressed

Description

All the calls to **`isTransactionConfirmed`** check if the value is undefined and break the switch case in case they are. This means that the order status doesn't get updated. If the **`isTransactionConfirmed`** systematically returns undefined, the order gets stuck in this state and is never updated nor signaled to be stuck.

Recommendation

Identify if the order is stuck after a certain number of attempts and change the order state to accurately reflect that it is stuck

3S-ORDS-N02

src/core/orderbook: All the calls to getTokenTransferSpend don't check if the order is stuck

Id	3S-ORDS-N02
Classification	None
Category	Good Code Practices
Status	Addressed

Description

All the calls to **`getTokenTransferSpend`** check if the value is undefined and break the switch case in case they are. This means that the order status doesn't get updated. If the **`getTokenTransferSpend`** systematically returns undefined, the order gets stuck in this state and is never updated nor signaled to be stuck.

Recommendation

Identify if the order is stuck after a certain number of attempts and change the order state to accurately reflect that it is stuck

3S-ORDS-N03

src/util: There are several calls to console.error instead of console.error()

Id	3S-ORDS-N03
Classification	None
Category	Good Code Practices
Status	Addressed

Description

There are several calls to console.error, which returns a pointer to the console.error function but it does not print the error to the console, which is the expected behavior.

Recommendation

Change the console.error occurrences with console.error().

3S-ORDS-N04

src/core/orderbook add/removeLiquidity BTC cases

Id	3S-ORDS-N04
Classification	None
Category	Good Code Practices
Status	Acknowledged

Description

In the original UniswapV2 implementation there exist separate functions for when a user is adding liquidity in a pool which is token/token or token/ETH. This facilitates the processing of orders and ensures readability and minimization of possible bugs in the code. We recommend that the Ordiswap team uses a similar process for these functions.

Recommendation

Example can be found here.

3S-ORDS-N05

src/core/orderbook BTC/Token Pools

Id	3S-ORDS-N05
Classification	None
Category	Good Code Practices
Status	Acknowledged

Description

Throughout the code there are multiple points where edge cases are tested for the one of the token in the pool being BTC. The pools admit BTC as either token1 or token2 and always check for both edge cases. This leads to bloat in code which is undesirable and can lead to the introduction of potential bugs by not considering these edge cases in the future.

Recommendation

Sort tokens in a pool alphabetically and in the case of having a pool with BTC as pair then BTC should always be either token1 or token2. The requests can be sorted in the server.js handler and the backend logic always admits that if BTC is a pair then it is on just a single position.

3S-ORDS-N06

src/server: The endpoints /addLiquidity/getAmount and /removeLiquidity/getAmount are gets but should be post

Id	3S-ORDS-N06
Classification	None
Category	Bug
Status	Addressed

Description

Both the endpoints **`/addLiquidity/getAmount`** and **`/removeLiquidity/getAmount`** are gets, but they should be posts since both of them have a request body.

Recommendation

Change the method from get to post on both these endpoints

3S-ORDS-N07

src/server: The OPTIONS method is not modular across endpoints

Id	3S-ORDS-N07
Classification	None
Category	Suggestion, Good Code Practices
Status	Addressed

Description

In every endpoint there is a check to make sure the method being used is the **`OPTIONS`** one, in which the status 204 is then returned to the client. The code snippet goes as follows:

```
if (req.method.toUpperCase() === "OPTIONS") {
      return res.status(204).end();
}
```

This is redundant and can lead to mistakes when changing the codebase in the future.

Recommendation

Change the current custom header middleware to check for this use case and return 204 if the request is of the type **`OPTIONS`**, as follows:

```
server.use(async function (req, res, next) {
        res.header("Access-Control-Allow-Origin", "*");
        res.header("Access-Control-Allow-Methods", "*");
        res.header("Access-Control-Allow-Credentials", true);
        res.header("Content-Type", "application/json");
        res.header("Access-Control-Allow-Headers", "Origin,
X-Requested-With, Content-Type, Accept, Authorization")
        if (req.method === 'OPTIONS') {
          return res.sendStatus(204);
        }
        next();
```

```
})
```

3S-ORDS-N08

src/util: waitForSeconds function doesn't need to emit events

Id	3S-ORDS-N08
Classification	None
Category	Optimization
Status	Addressed

Description

The **`waitForSeconds`** function has an **`EventEmitter`** that is emitted inside a **`setTimeout`** function. Afterwards, the **`waitForEvent`** function is called that waits for the event to be emitted. For the intended functionality we don't require to emit events, since we only want to wait for n seconds

Recommendation

Implement a simpler version of the **`waitForSeconds`**, as suggested:

```
function waitForSeconds(seconds) {
  return new Promise(resolve => setTimeout(resolve, seconds * 1000));
}
```

3S-ORDS-N09

src/server: The /getorder returns success even when orders variable is set to undefined

Id	3S-ORDS-N09
Classification	None
Category	Suggestion, Good Code Practices
Status	Addressed

Description

Same as #51

Recommendation

Same as #51

3S-ORDS-N10

src/server: The getwallettransaction returns success even when walletTransaction is set to undefined

Id	3S-ORDS-N10
Classification	None
Category	Suggestion, Good Code Practices
Status	Addressed

Description

The server returns success even when the **`walletTransactions`** variable is undefined, which can lead to unwanted behavior in the client side.

Recommendation

Handle the use case where the walletTransactions variable is set to undefined and return that information to the client accordingly.

3S-ORDS-N11

src/server: The /getpool/:address and /getpool/:token1/:token2 endpoint names are counterintuitive

Id	3S-ORDS-N11
Classification	None
Category	Good Code Practices
Status	Addressed

Description

Both endpoints have a very similar name structure, but fundamentally they do different things: the first one returns the balances of the user in every pool, whereas the second returns the sum of the liquidity and swap amounts. This can be confusing and lead to errors in future changes to the code.

Recommendation

Change the name of the endpoints for something for explicit

3S-ORDS-N12

src/server: The /getpool/:address endpoint returns all the pools even if the address doesn't have balance for that pool

Id	3S-ORDS-N12
Classification	None
Category	Optimization, Suggestion, Good Code Practices
Status	Addressed

Description

The **`/getpool/:address`** endpoint returns all the pools with the balances that the user has for the tokens present in that pool. It also returns the pools where the user doesn't own any tokens, making the response much bigger than what it has to be, consequently making the code much less readable as well.

Recommendation

Only add to the response pools where the user owns at least 1 token in the pool.

3S-ORDS-N13

src/server: Naming of the different balances is not intuitive

Id	3S-ORDS-N13
Classification	None
Category	Suggestion, Good Code Practices
Status	Addressed

Description

The returned body is also not intuitive, since it's not clear what is the difference between **`lp_holder_balance`**, **`user_balance`** and **`withdraw_balance`**. Additionally, unnecessary balances shouldn't be returned in the response such as the **`admin_balance`** for non admin addresses

Recommendation

Rename the balances for a structure that more accurately describes what they mean and don't return balances that aren't relevant for given use cases.

3S-ORDS-N14

src/core/orderbook: const TRANSFERED typo

Id	3S-ORDS-N14
Classification	None
Category	Good Code Practices
Status	Addressed

Recommendation

Change to all constants with the suffix to **`TRANSFERED`** to **`TRANSFERRED`**

3S-ORDS-N15

src/server: The /getbtcbalance/platform/:address/list endpoint can make concurrent queries to the database improving performance and readability

Id	3S-ORDS-N15
Classification	None
Category	Optimization, Good Code Practices
Status	Addressed

Description

In the **`/getbtcbalance/platform/:address/list`** several queries are performed to the database. These queries can be performed concurrently making it more efficient and faster. By refactoring the code in this manner, we can separate the query logic from the rest of the logic of the function, which makes the code more readable

Recommendation

Instead of running a series of queries in this manner:

```
const withdraws = await balanceCollection.find({
      address,
      balance_type: BALANCE_TYPE_WITHDRAW,
}).sort({ _id: -1 }).toArray()
(...) // function logic
let lpHolderBalances = await balanceCollection.find({
      address,
      balance_type: BALANCE_TYPE_LP_HOLDER,
}).toArray()
(...) // more function logic
```

All of the queries can run concurrently in this manner:

```
const query1 = balanceCollection.find({
```

```
      address,
      balance_type: BALANCE_TYPE_WITHDRAW,
}).sort({ _id: -1 }).toArray()
let query2 = balanceCollection.find({
      address,
      balance_type: BALANCE_TYPE_LP_HOLDER,
}).toArray()
const [withdraws, lpHolderBalances] = await Promise.all([query1, query2]);
// All of the function logic
```

3S-ORDS-N16

src/core/orderbook: In the orderbookThread function the orders are being updated in the database one by one

Id	3S-ORDS-N16
Classification	None
Category	Optimization, Good Code Practices
Status	Addressed

Description

In the **`orderbookThread`** function orders get updated to the db one by one inside a for loop using the **`updateOne()`** function. This makes it that there are many accesses to the database, making the code slower and more resource intensive than batching all the updates at once using the **`updateMany()`** function.

Recommendation

Change the code to utilize the **`updateMany()`** function instead of **`updateOne()`** to optimize the accesses to the database.

3S-ORDS-N17

src/core/orderbook: In the orderbookThread function the if statement can be directly included in the quer

Id	3S-ORDS-N17
Classification	None
Category	Optimization, Suggestion, Good Code Practices
Status	Addressed

Description

In the **`orderbookThread`** function there is the following if to check the order status: **`if (updateOrder.order_status == ORDER_STATUS_ORDERED)`**. This if statement could be included directly in the query to the db, which would make both the query and code more efficient. Furthermore, the code would be cleaner and more readable.

Recommendation

Include the **`order_status`** condition directly in the query to the db.

3S-ORDS-N18

src/core/orderbook: In the orderbookThread function we are using the $lt condition to check enum values

Id	3S-ORDS-N18
Classification	None
Category	Optimization, Suggestion, Good Code Practices
Status	Addressed

Description

In the **`orderbookThread`** function the condition **`$lt: ORDER_STATUS_FAILED`** is being used to query the database. It is an anti-pattern to use the **`$lt`** condition with an enum value that looks to abstract away the value beneath it. This can lead to errors when adding or editing values in the enum, which can lead to unwanted behaviour.

Recomendation

Query the database using enum values that are the intended.

3S-ORDS-N19

src/server: /getbtcbalance/platform/:address/list endpoint naming is not clear

Id	3S-ORDS-N19
Classification	None
Category	Documentation, Suggestion, Good Code Practices
Status	Addressed

Description

The naming of the **`/getbtcbalance/platform/:address/list`** is not intuitive as it is not clear what the endpoint returns.

Recommendation

Change the endpoint name so it is clear that the endpoint is returning the different types of addresses for a given address

3S-ORDS-N20

src/server: /getbtcbalance/platform and /getbtcbalance/platform/:address/list endpoint doesn't verify if the provided address is valid

Id	3S-ORDS-N20
Classification	None
Category	Bug, Suggestion, Good Code Practices
Status	Addressed

Description

The endpoints don't verify if the provided address is valid or not. This means that it still returns **`success: true`** back to the client even when that is not the case.

Recommendation

Implement a check to make sure the provided address is valid and return an error back to the client if it isn't

3S-ORDS-N21

src/server: /getbtcbalance endpoint returns success even if the address is not valid

Id	3S-ORDS-N21
Classification	None
Category	Bug, Suggestion, Good Code Practices
Status	Addressed

Description

The **`/getbtcbalance`** endpoint returns **`success: true`** in the response even if the address provided is not valid. In this case, because the value of **`btcBalance`** is undefined, the data field is not added to the request.

Recommendation

Check if the value of **`btcBalance`** is undefined and properly handle that error when it happens

3S-ORDS-N22

src/server: The gettokenbalance returns status has success even if the address is not a valid one

Id	3S-ORDS-N22
Classification	None
Category	Bug, Good Code Practices
Status	Addressed

Description

The **`/gettokenbalance`** endpoint returns the status has successful even when the address is not valid.

Recommendation

Check the response of the indexer server for **`response.data.msg=== 'address invalid'`** and return that message back to the client.

3S-ORDS-N23

src/server: /tokeninfo & /gettokenbalance endpoints check for BTC ticker is redundant

Id	3S-ORDS-N23
Classification	None
Category	Suggestion, Good Code Practices
Status	Addressed

Description

In the **`/tokeninfo`** & **`/gettokenbalance`** there is a check that ensures that the ticker is different to the BTC ticker and that the ticker can't have a size different to 4. The check that verifies is the token ticker is different to the BTC ticker is redundant, since the BTC ticker has a size of 3, therefore the second works catches all the examples that the first check does.

Recommendation

Remove the **`!compareStringCaseInsensitive(token, BTC_TOKEN)`** check because it is redundant.

3S-ORDS-N24

src/util: compareStringCaseInsensitive has a try catch clause for string comparison that isn't handed well

Id	3S-ORDS-N24
Classification	None
Category	Bug, Suggestion, Good Code Practices
Status	Addressed

Description

There is a try & catch clause in the **`compareStringCaseInsensitive`**, but the catch clause doesn't explicitly return the value **`false`**, which makes the function return the value **`undefined`**. This can lead to unwanted behaviour depending on how the return value is being handled. Furthermore, if the codebase was implemented using TypeScript, we would not need to have the try & catch clause since any possible type errors are caught by the compiler.

Recommendation

Explicitly return false in the catch clause. Ideally, migrate the codebase to TypeScript and remove the try & catch clause altogether. Also, the equals clause should use "===" instead of just "==" to make sure the types match and not just the values.

3S-ORDS-N25

src/core/pool: calculateTokenAmountForAddLiquidity should match UniswapV2 naming and code convention

Id	3S-ORDS-N25
Classification	None
Category	Good Code Practices
Status	Addressed

Description

The function **`calculateTokenAmountForAddLiquidity`** mimicks the behavior of the function **`_addLiquidity`** present in the Router contract of the Uniswap codebase and therefore having the same name and code structure greatly improves its auditability to ensure that the functions have identical behavior.

Recommendation

Change the name and add variables such as **`amountADesired'`** and **`amountBDesired`** as well as the code flow as it can be found here..

3S-ORDS-N26

src/core/pool: calculateTokenAmountForSwap should match UniswapV2 naming and code convention

Id	3S-ORDS-N26
Classification	None
Category	Good Code Practices
Status	Addressed

Description

The function **`calculateTokenAmountForSwap`** mimicks the behavior of the function **`getAmountOut`** present in the Library contracts of the Uniswap codebase and therefore having the same name and code structure greatly improves its auditability to ensure that the functions have identical behavior.

Recommendation

Change the name and add variables such as **`numerator`** and **`denominator`** as well as the code flow as it can be found here..

3S-ORDS-N27

src/server: Endpoints don't return status code 400 for bad requests

Id	3S-ORDS-N27
Classification	None
Category	Bug, Suggestion, Good Code Practices
Status	Addressed

Description

The server doesn't return status code 400 for bad requests. Whenever handling an error, the server returns a JSON with the following values:

```
{
    "status": "error",
    "description": {{error_message}}
}
```

Although the JSON reflects that there was an error, the request status code is still 200, tricking the client into thinking that the response was successful when it was not.

Recommendation

Make sure all bad requests return the status code 400 so the client can accurately deal with errors coming from the server. Also make sure to have a descriptive error message according to the type of error, instead of just returning "unknown error" message.

3S-ORDS-N28

src/server: Endpoints that create resources don't return the correct 201 status code

Id	3S-ORDS-N28
Classification	None
Category	Bug, Suggestion, Good Code Practices
Status	Addressed

Description

The following endpoints create or edit resources and should return the status code 201 to reflect the action on the backend:

```
/createpool
/addliquidity
/removeliquidity
/swap
/withdrawbtc
/updateorder
```

The endpoints return the status code 200 instead.

Recommendation

Add the correct 201 status code to all the successful responses coming from those endpoints.

3S-ORDS-N29

src/server: A custom 404 route is not defined

Id	3S-ORDS-N29
Classification	None
Category	Suggestion, Good Code Practices
Status	Addressed

Description

Currently the server returns the default express html for 404 errors as shown below

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Error</title>
</head>
<body>
    <pre>Cannot GET /pooltokenlista</pre>
</body>
</html>
```

Since the server returns **`application/json`** content types, it makes sense to keep the consistency when returning the 404 errors, so that it can easily be handled by the client.

Recommendation

Implement middleware that returns a JSON like structure for requests that return the 404 status code.

3S-ORDS-N30

src/server: /pooltokenlist:address endpoint can heavily simplified

Id	3S-ORDS-N30
Classification	None
Category	Optimization, Suggestion, Good Code Practices
Status	Addressed

Description

Same issue as #23 but in a different endpoint

Recommendation

Same issue as #23 but in a different endpoint

3S-ORDS-N31

src/server: Endpoints don't return status code 500 for internal server errors

Id	3S-ORDS-N31
Classification	None
Category	Bug, Suggestion, Good Code Practices
Status	Addressed

Description

The server doesn't return status code 500's for internal server errors. Whenever handling an error, the server returns a JSON with the following values:

```
{
    "status": "error",
    "description": "Unknown error"
}
```

Although the JSON reflects that there was an error, the request status code is still 200, tricking the client into thinking that the response was successful when it was not.

Recommendation

Make sure all internal errors return the status code 500 so the client can accurately deal with errors coming from the server

3S-ORDS-N32

src/server: /pooltokenlist endpoint can heavily simplified

Id	3S-ORDS-N32
Classification	None
Category	Optimization, Suggestion, Good Code Practices
Status	Addressed

Description

In the /pooltokenlist endpoint there are a set of operations that push all the tokens from all the pools into an array with no duplicates. The logic implemented can be heavily refactored to be cleaner and more readable.

Recommendation

Change the following code block:

```
const poolInfo = await getPoolInfo0()
const poolTokenList1 = []
for (const pool of poolInfo) {
      poolTokenList1.push(pool.token1)
      poolTokenList1.push(pool.token2)
}
const poolTokenList2 = {}
for (const poolToken of poolTokenList1) {
      poolTokenList2[poolToken] = true
}
const poolTokenList3 = []
for (const poolToken in poolTokenList2) {
      poolTokenList3.push(poolToken)
}
```

To a simpler version utilizing the array's method filter and with better named variables or equivalent:

```
      const pools = await getPoolInfo0()
      const poolTokens = []
      for (const pool of pools) {
            poolTokens.push(pool.token1, pool.token2)
      }

         poolTokens = poolTokens.filter((value, index, self) =>
self.indexOf(value) === index)
```

3S-ORDS-N33

src/core/factory: Bad naming of the function getPoolInfo2

Id	3S-ORDS-N33
Classification	None
Category	Suggestion, Good Code Practices
Status	Addressed

Description

The **`getPoolInfo2`** function doesn't have an explicit name. This function returns the pool that contains both the specified tokens.

Recommendation

Change the function name to **`getPoolByPair()`** or to an equivalent name.

3S-ORDS-N34

src/core/factory: Bad naming of the function getPoolInfo1

Id	3S-ORDS-N34
Classification	None
Category	Suggestion, Good Code Practices
Status	Addressed

Description

The **`getPoolInfo1`** function doesn't have an explicit name. This function returns all the available pools that contain the specific token in it.

Recommendation

Change the function name to **`getTokenPools()`** or to an equivalent name.

3S-ORDS-N35

src/core/factory: Bad naming of the function getPoolInfo0

Id	3S-ORDS-N35
Classification	None
Category	Suggestion, Good Code Practices
Status	Addressed

Description

The **`getPoolInfo0`** function doesn't have an explicit name. This function returns all the available pools, hence it should have a name that clearly states that functionality

Recommendation

Change the function name to **`getAllPools()`** or to an equivalent name.

3S-ORDS-N36

src/core/factory: The getPoolInfo0 calls the find({}) function

Id	3S-ORDS-N36
Classification	None
Category	Good Code Practices
Status	Addressed

Description

The getPoolInfo0 queries the pool collection in the following manner: **`poolCollection.find({}).toArray()`**. Although this is correct, it is advised to remove the empty filter represented by **`{}`**, since it is redundant in order to make the code cleaner.

Recommendation

Remove the **`{}`** from the **`poolCollection.find({}).toArray()`** function call.

3S-ORDS-N37

src/server: The /tokenlist/:address endpoint can be made more readable

Id	3S-ORDS-N37
Classification	None
Category	Suggestion, Good Code Practices
Status	Addressed

Description

The endpoint **`/tokenlist/:address`** has 2 nested for loops, which makes the code less readable and harder to understand. By utilizing the **`find()`** method we are able to reduce nesting, have a more explicit intent of what the code is supposed to do while keeping the code more concise.

Recommendation

Our recommendation is that the code utilizes the find() method instead:

```
for (const token of tokenList) {
  token.balance = 0;
  const matchingTokenBalance = tokenBalances.find(
    (tokenBalance) => tokenBalance.ticker.toLowerCase() ===
token.tick.toLowerCase()
  );
  if (matchingTokenBalance) {
    token.balance = parseFloat(matchingTokenBalance.availableBalance);
  }
}
```

3S-ORDS-N38

src/server & src/core/brc20-indexer: Bearer token is manually sent in every request

Id	3S-ORDS-N38
Classification	None
Category	Suggestion, Good Code Practices
Status	Addressed

Description

The Authorization Bearer Token is manually sent in every request, which can lead to human errors when changes are performed in the code base and it also improves code readability. It is advised to configure this token globally so that every request has this setting has a default.

Recommended

Configure Axios in both src/server & src/core/brc20-indexer to utilize the Bearer token.

3S-ORDS-N39

src/server & src/core/brc20-indexer: Requests that utilize address parameter don't return 404 when that address doesn't exist

Id	3S-ORDS-N39
Classification	None
Category	Suggestion, Good Code Practices
Status	Addressed

Description

There are several endpoints that expect to receive an address that is then utilized to make a request to the indexer backend. There is not check to verify if the response status is 404. This means that the server returns **`ERROR_UNKNOWN`** when it could return 404 with a more specific error message.

Recommendation

Add an error handler that takes care of this particular use case and returns 404 along with an error message.

3S-ORDS-N40

src/server: Content-type is being returned as text/html when it is application/json

Id	3S-ORDS-N40
Classification	None
Category	Suggestion, Good Code Practices
Status	Addressed

Description

Currently the responses have the **`Content-Type`** header with the value of **`text/html`**, when in reality they should be **`application/json`**. This can cause issues to clients that expect the format to be html and not json, including testing frameworks.

Recommendation

Change the response **`Content-Type`** to application-json as middleware. This can be done in the following way:

```
server.use(function (req, res, next) {
      res.header("Access-Control-Allow-Origin", FRONT_SERVER)
      res.header("Access-Control-Allow-Methods", "GET,POST")
      res.header("Content-Type", "application/json")
      next()
})
```

3S-ORDS-N41

src/server: The /tokenlist endpoint returns balances of 0 to every token

Id	3S-ORDS-N41
Classification	None
Category	Question
Status	Addressed

Description

The **`/tokenlist`** adds the balance to 0 to every endpoint. It is unclear if this behaviour is intended or not and what is the utility of it.

Recommendation

If the behaviour is not intended, remove the for loop.

3S-ORDS-N42

src/core/brc20-indexer: The getTokenList function can be simplified

Id	3S-ORDS-N42
Classification	None
Category	Optimization, Good Code Practices
Status	Addressed

Description

The **`getTokenList`** function is inefficient and performs useless operations. There is no need to create a **`tokens2`** array and the assignment to the token.mint variable is not being utilized anywhere

Recommendation

The function can be replaced with the following code, making it more readable and more efficient.

```
const tokens = await brc20Collection.find().toArray();
return tokens.map(token => ({
   tick: token.ticker,
}));
```

3S-ORDS-N43

The codebase doesn't use Typescript.

Id	3S-ORDS-N43
Classification	None
Category	Suggestion, Good Code Practices
Status	Acknowledged

Description

The codebase is written using Javascript, which is not recommended. By using TypeScript we add static types to the language, reducing the amount of bugs and issues that can arise from the codebase. We highly recommend changing the codebase to TypeScript before adding more complexity to the server.

Recommendation

Change the language to TypeScript to ensure static types.

3S-ORDS-N44

src/core/brc20-indexer: The exceeds rate limit response should be handled with a specific behaviour

Id	3S-ORDS-N44
Classification	None
Category	Good Code Practices
Status	Addressed

Description

The server is making requests to an Indexer URL that is sometimes returning rate limit exceeded. This error returns a **`response.data.msg`** value of exceeds rate limit that should be handled specifically by backing off for a couple of seconds and then trying again. Instead, the server is checking for any **`response.data.code`** that is different than 0, which is not ideal if we want to catch errors different than rate limit exceeded.

Recommendation

Handle the **`response.data.msg`** with value exceeds rate limit specifically to make isolate this error from other possible error codes being returned by the Indexer URL.

3S-ORDS-N45

src/core/brc20-indexer: New brc20 tokens are being added to mongo one by one and not in batch

Id	3S-ORDS-N45
Classification	None
Category	Optimization, Good Code Practices
Status	Addressed

Description

In the **`brc20_index`** function, new brc20-tokens are added one by one to the mongodb database. This increases the number of accesses to the database as well as making the syncing progress much slower than what it has to be.

Recommendation

Save the tokens in an array and use the **`brc20Collection.insertMany(tokens)`** only at the end of the function to batch submit all tokens at once. This reduced the accesses to the database to one making the syncing progress much lighter and faster

3S-ORDS-N46

src/core/brc20-indexer: The syncedTokenCount variable is updated in the database for each request

Id	3S-ORDS-N46
Classification	None
Category	Optimization, Good Code Practices
Status	Addressed

Description

The variable **`syncedTokenCount`** is updated in the mongodb backend everytime a request is being made. this leads to an unnecessary amount of accesses to the database, since this operation can be performed at the end of the loop only once

Recommendation

Call **`await configCollection.updateOne({}, { $set: { synced_token_count: syncedTokenCount } })`** only after the loop as ended.

3S-ORDS-N47

src/core/brc20-indexer: The server waits for n seconds whenever a response returns an error

Id	3S-ORDS-N47
Classification	None
Category	Suggestion, Good Code Practices
Status	Addressed

Description

Even though the block of code that handles response errors in unreachable, instead of waiting for n seconds, axios should be configured to not go beyond the rate limit imposed by the indexer. (5 calls/second in free tier, 10 calls/second in standard tier - link)

```
if (!response || !response.data || response.data.code !== 0 ) {
      await waitForSeconds(1)
      continue
}
```

Recommendation

This can be done utilizing the **`axios-rate-limit`** library that configures axios to not exceed a certain threshold of requests - link.

Here is an example

```
const rateLimitedAxios = axiosRateLimit(axios, {
  maxRequests: 5, // Maximum number of requests per interval
  perMilliseconds: 1100, // Interval duration in milliseconds
});
```