OBSIDIAN . SYSTEMS
Cutting edge software solutions

2093 Philadelphia Pike #8128
Claymont, DE 19703
info@obsidian.systems
(888) 611-7759

VyFinance DEX Initial Audit Report

Summary

The scope of this audit includes the on-chain ‘smart contracts’ comprising the following source modules:

- Bar.Business
- Bar.OnChain
- Bar.Types
- Bar.Validator
- LiquidityPool.Business
- LiquidityPool.OnChain
- LiquidityPool.Types
- LiquidityPool.Validator
- Utils.CredentialScriptContext
- Utils.WalletAddress

Commit hash: **fa5e24e095c6559a179b27c1e55a2118560ad189**

After completing this initial audit, no critical issues were discovered in the VyFinance DEX ‘smart contracts’. Findings were limited to lower severity items.

Methodology

Each component underwent a manual code review. We ran and utilized the wider system to evaluate the various user journeys and workflows. Where possible, we ran automated tests or created test data.

Contracts were exercised using the VyFinance frontend to map out the transactions performed. Once this behavior was understood, auditors moved to manual transaction construction and submission to analyze possible on-ledger states in fine detail.

While analyzing the contracts on-chain, the code was reviewed line-by-line according to Haskell and Cardano best practices. On-chain state transitions were correlated with lines of code executed. Particular attention was paid to authorization logic and the handling of user data.

Aside from applying Haskell and Cardano industry standards, specific focus was devoted to Plutus versioning and its known distinctions from Haskell. Threat modeling included all common known DEX exploits.

Disclaimer

This initial report is not considered final - an upcoming audit phase will conclude with a final report. This report is presented for informational purposes and great care was taken to communicate clearly about limitations.

This initial report does not constitute legal or financial advice. Obsidian Systems can assume no liability for the deployment, operation, or use of the associated 'smart contracts'.

Constraints & Limitations

Some critical DEX business logic is implemented off-chain.

This audit does not validate guarantees about individual actors serving as liquidity pool and bar operators.

Once an Order Contract allows users to lock funds and note which operation they'd like performed, operators can use the Main Contract to perform arbitrary state transitions.

Some risks associated with this model may be mitigated when multiple operators are later invited and trust is distributed across them for transaction processing.

Severity Ranking

- **P1** – Critical issue risking funds, user data, system stability
    - Privilege escalation, identity theft, MITM, et al
- **P2** – Severe disruption
    - DDOS risks, disruptive invalid inputs allowed/accepted
- **P3** – Inconvenient / Non-intuitive
    - Funds loss through complicated workflows, soft-locking users through unexpected system states
- **P4** – Code health and clarity
    - Slow burn risks from excessive complexity, non-deterministic build behavior (different users building the same component resulting in different dependencies being used), code organization
- **P5** – Documentation and cosmetic issues
    - Hard to read error messages, undocumented workflows, out of date / incorrect documentation

Findings

OS-VYFI-01 – PAB compatibility

Severity: ***P4***
Description:
The Bar contracts are written to support the PAB in addition to the Cardano ledger. This introduces some complexity in the code. In particular (but not exhaustively):

1. Preprocessor directives to run some code differently depending on the environment Namely,
    - Utils.CredentialScriptContext.checkMainNFTInScriptOutput
    - Utils.CredentialScriptContext.checkCredentialTokenInWalletOutput
2. Custom text processing to deal with PAB quirkiness, LiquidityPool.Types.fromJSONText

Beyond minor increases in code complexity, relying on the PAB makes it more difficult to upgrade scripts as Cardano evolves. The PAB is often incompatible for a long period of time when Cardano upgrades the live version of Plutus on-chain.

Recommendation:
Explicitly document these issues in the code so that people unfamiliar with the PAB's technicalities can follow along— this will also signal to developers when they have to be careful about PAB compatibility.

If possible, plan to eventually phase out use of PAB to mitigate future risks stemming from Cardano's active development.

OS-VYFI-02 – Custom Eq instances

Severity: ***P4***
Description:
There are several handwritten instances of the PlutusTx.Prelude.Eq class. Custom Eq instances are a perpetual risk because they need not necessarily validate the laws one would expect of equality. Even if an instance is verified to work correctly today, a change in the data type to which it belongs can silently invalidate this property. We found custom instances for the following types:

- Bar.Business.BarState
- Bar.Types.Bar
- Bar.Types.MainDatum
- Bar.Types.OrderDatum

- Bar.Types.OrderKind
- Utils.WalletAddress.WalletAddress

Recommendation:
Unfortunately there is no official automatic way to derive Plutus.Tx.Prelude.Eq, neither the deriving mechanism nor template haskell. VyFinance could implement such a template haskell function, modeled after the one in deriving-compat so that correctness can be verified once and for all. A less desirable alternative would be simply to quickcheck Eq instances and flag all changes to data types for extra review.

OS-VYFI-03 – Unused Data and Redemeers

Severity: ***P4***
Description:
The *main* and *order* validators do not use all of their arguments. In particular,

- Bar.OnChain.mkMainValidator disregards its `MainDatum` and `MainRedeemer` arguments
- Bar.OnChain.mkOrderValidator disregards its `OrderDatum` argument
- LiquidityPool.OnChain.mkMainValidator disregards its `MainDatum` and `MainRedeemer` arguments
- LiquidityPool.OnChain.mkOrderValidator disregards its `OrderDatum` argument

This isn't an issue in itself— if the functionality using these arguments has not been implemented, then they can't help but be unused. The issue is that the arguments are silently discarded.

Recommendation:
To ensure that all data is accounted for, all records should be explicitly unpacked with meaningful variable names given to each field. If a field is not used, the warning given by the compiler will indicate this to the developers and CI. These warnings should be manually reviewed whenever they exist, and never ignored. Whenever it is possible, the Datum and Redeemer types should only contain the data required for the functionality implemented in the contracts.

OS-VYFI-04 – Custom Script Context

Severity: ***P4***
Description:
In order to necessarily save on script code size, the contracts use a custom script context that does not parse all of the provided TxInfo, namely Utils.CredentialScriptContext.CredentialScriptContext. This is not standard and increases code complexity.

Recommendation:
Instead of optimizing for script size as aggressively, consider a future upgrade to Babbage contracts which allow transactions to refer to scripts instead of being forced to include them.

OS-VYFI-05 – Credential tokens expected in certain output slots

Severity: ***P5***
Description:
The validators for the Bar and LiquidityPool contracts expect NFT and credential tokens to be in particular output slots of the transaction. This is not an issue as such, but does mean that users will be constrained in the sorts of transactions they construct which involve the VyFinance DEX.

Recommendation:
Document this for users and developers who may wish to interact with the DEX in more advanced ways than provided by the DEX frontend. If the recommendation for *OS-VYFI-04* is later implemented, then the script can be liberalized to analyze the entire transaction info to allow more flexible interoperability.

OS-VYFI-06 – Credential token minting policy and check mismatch

Severity: ***P4***
Description:
Utils.CredentialScriptContext.checkCredentialTokenInWalletOutput only checks the currency symbol when it verifies that a credential token is present in the transaction. The documentation of the function explicitly mentions the possibility of a vulnerability caused by a minting policy that allows multiple tokens, or multiple token names for the credential token. In particular, this function is used to check that a user is authorized as the operator of a liquidity pool or the Bar.

The minting policy used for the operator credential tokens was not included in the audit scope, so assessing this vulnerability further remains a limitation.

Recommendation:
checkCredentialTokenInWalletOutput (via valueHasCurrency) should check the token name even if the minting policy is set up to have a unique token with a unique name. This will guard against unanticipated changes to the DEX architecture involving authorization and control.

OS-VYFI-07 – Use of RecordWildcards

Severity: ***P5***
Description:
RecordWildcards is enabled throughout the audited code. The issue with RecordWildcards is that it brings variables into scope without explicitly mentioning them, making it harder to analyze code, as one will have to look up the record type definition to know which variables have been bound. This increases the chance of programmer error.

For example, a new variable may be introduced which conflicts with a record name, or vice versa. Resolving such shadowing incorrectly can lead to subtly broken code.

Recommendation:
Explicitly deconstruct records or explicitly use field accessors to make the code easier to reason about locally.

OS-VYFI-08 – Custom JSON instances

Severity: ***P5***
Description:
The FromJSON ToJSON instances for LiquidityPool.Types.LiquidityPool are handwritten, as well as those for LiquidityPool.Business.FeesSettings. The `aeson` library has facilities for automatically deriving these instances. Handwritten instances are more brittle and can lead to unexpected behavior as data types evolve. For example, correct round tripping of serialization/deserialization might not hold, which will lead to subtle RPC bugs.

Recommendation:
Use Data.Aeson.TH.deriveJSON to avoid bugs in JSON parsing. We recommend writing golden tests to ensure that wire formats do not change unexpectedly, thereby obtaining the main benefit of handwritten instances.

Frontend integration will have to be adapted. We recommend automating typescript definitions with a tool like the library *aeson-typescript* to ensure reliable interoperability.

OS-VYFI-09 – Type synonyms for Integer

Severity: ***P4***
Description:
There are several type synonyms for Integer in the code base, namely:

- Bar.Business.AdaAmount
- Bar.Business.VyFiAmount

- Bar.Business.XVyFiAmount
- LiquidityPool.Business.AAmount
- LiquidityPool.Business.BAmount
- LiquidityPool.Business.LPAmount

These type synonyms do not prevent errors where variables are mistakenly swapped or misused.

Recommendation:
Use newtypes to mark values with their semantic significance. Wrap them on initialization and use explicit conversion routines to destructure them back to Integers. Factor out operations involving such values into self-contained functions, so that the main body of the code does not need to handle conversion. While this will be slightly more cumbersome, it will help prevent potentially catastrophic situations where a bug is introduced because, for example, currency values are mistakenly swapped.

OS-VYFI-10 – Uses of bare Integers

Severity: ***P5***
Description:
Business logic is currently being described with bare integers:

- LiquidityPool.Business.FeeSettings
- LiquidityPool.Business.Liquidity
- LiquidityPool.Types.MainDatum

The types will not help catch errors where such values are accidentally misused.

Recommendation:
Use newtypes to help prevent potentially catastrophic situations where a bug is introduced because integer parameters are incorrectly used. The same guidelines recommended for *OS-VFYI-09* apply.