Technical Review of Booktoken

High Assurance Software Group
June 28th, 2023

Contents

Chapter 1

Executive Summary and Scope

This report is presented without warranty or guaranty of any type. This report lists the most salient concerns that have so far become apparent to Tweag after a partial inspection of the engineering work. Corrections, such as the cancellation of incorrectly reported issues, may arise. Therefore Tweag advises against making any business decision or other decision based on this report.

Tweag does not recommend for or against the use of any work or supplier referenced in this report. This report focuses on the technical implementation provided by the project's contractors and subcontractors, based on their information, and is not meant to assess the concept, mathematical validity, or business validity of Book.io's product. This report does not assess the implementation regarding financial viability nor suitability for any purpose.

Scope and Methodology

Tweag looks exclusively at the on-chain validation code provided by Book.io. This excludes all the frontend files and any problems contained therein. Tweag manually inspected the code contained in the respective files and attempted to locate potential problems in one of these categories:

a) Unclear or wrong specifications that might allow for fringe behavior.

b) Implementation that does not abide by its specification.

c) Vulnerabilities an attacker could exploit if the code were deployed as-is, including:

- race conditions or denial-of-service attacks blocking other users from using the contract,
- incorrect dust collection and arithmetic calculations (including due to overflow or underflow),
- incorrect minting, burning, locking, and allocation of tokens,
- authorization issues,

d) General code quality comments and minor issues that are not exploitable.

Where applicable, Tweag will provide a recommendation for addressing the relevant issue.

Chapter 2

Audit

2.1 Methodology

Tweag analyzed the validator scripts comprising Booktoken, contained in the repository[1] as of commit `4691f85`[2]. The names of the files considered in this audit and their `sha256sum` are listed in Table 2.1.

Our analysis is based on the documentation provided by Book.io, and on meetings with Book.io. The relevant documentation files are listed in Table 2.2 and their contents will be referred to as *the specification* of the product.

2.2 Findings

Table 2.3 lists our concerns with the current Booktoken implementation based on our partial exploration during a limited period of time. Throughout the rest of this section, we will detail each of our findings.

2.2.1 Design Choices

2.2.1.1 The reference token is not an NFT

Severity: *Medium*

The reference token that officializes the UTxO containing the `RefDatum` information (used as reference input in unlock and swap transactions) could benefit from an improved design.

a) The documentation mentions the token is intended to be burned and reminted every time the information is updated. This adds cost to the update transaction because it has to contain the minting policy of the reference token.

b) Here, security relies on the actions of the admin. They are expected to burn the previous token before minting the new one. If they fail to do so, or are compromised, there could be more than one reference token in the wild.

We suggest to use a one-shot minting policy that produces only one token once and for all. The reference UTxO could sit at a script that verifies the NFT is given back to the same script during each update. Both issues would be avoided. This design pattern is reliable and common among Plutus smart contracts. Relying on such design patterns contributes to the overall reliability and maintainability of a smart contract.

[1] `https://github.com/book-io/contracts`
[2] Full commit hash: `4691f8535e827f1a4c710e011058cad86ac662e6`

sha256sum	File Name
7de3893...255d34a	client-code/validators/book_time_lock.ak
cfa4ee9...e64a15f	client-code/validators/validators/book_token_swap.ak
a89997a...910587b	client-code/validators/validators/fail.ak
32b05d5...eda651e	lib/balance_helper.ak
fa7ca7a...b8eb364	lib/time_helper.ak
623f1e1...384a203	lib/types.ak
a319b63...875b9ea	lib/utils.ak

TABLE 2.1: *On-chain code source files and their* `sha256sum` *that were analyzed as part of the review*

sha256sum	File Name
5109036...3b358aa	client-doc/Book_Swap_and_Lock_Contract.pdf

TABLE 2.2: *Documentation files and their* `sha256sum` *that were used as the specification*

2.2.1.2 Obsolete tokens are not burned

Severity: Low

Obsolete old tokens are not burned in the swap transaction. Instead, they are sent to an always-failing script. There is no upside in doing this, neither in terms of security nor in terms of logging or accountability. We suggest to simplify the end-of-life process for new tokens if possible, by allowing them to be burned when applicable (e.g. by a future swap script, or by the Booktoken admin).

2.2.1.3 Custom failure scripts have no upside over the canonical failure script

Severity: Lowest

The always-failing scripts used to lock old tokens are parametrized. This does not seem to bring any upside compared to using the canonical (parameter-less) failure script. In particular, using parametrized scripts does not prevent third parties from polluting the addresses with junk UTxOs. Hence, looking for archived obsolete book tokens would require a filtering pass, much like when using the common canonical failure script. Moreover, the specification does not make the use of the parameter explicit.

2.2.2 Vulnerabilities

2.2.2.1 New tokens can be stolen by abusing an undesired rounding-up operation

Severity: Critical

In the swap transaction, a redemption rate (an integer describing a percentage) is applied as an incentive: if the redemption rate is greater than 100, users will get more than one new token for each old token. The

Severity	Section	Summary
Critical	2.2.2.1	New tokens can be stolen by abusing an undesired rounding-up operation
Medium	2.2.1.1	The reference token is not an NFT
Medium	2.2.3.1	Unlock transaction is underspecified
Medium	2.2.3.2	Swap transaction is underspecified
Low	2.2.1.2	Obsolete tokens are not burned
Low	2.2.4.1	Conditionals could be avoided using short-circuiting in boolean conjunctions
Lowest	2.2.1.3	Custom failure scripts have no upside over the canonical failure script
Lowest	2.2.4.2	Unnecessary named intermediates
Lowest	2.2.4.3	Some helper functions impact compiled script size
Lowest	2.2.4.4	Comments are lacking in the code

TABLE 2.3: *Table of findings*

amount of new tokens obtained for n old tokens at redemption rate r should be computed as $(nr \text{ div } 100)$, where "div" denotes integer division.

The integer division happens in the swap validator code (see `book_token_swap.ak`) at line 117. The problem is that the dividend `user_qty_burn_multiplied` is a negative number. This comes from the definition of `user_qty_burn_token_change` at line 106 as the variation of the total amount of old tokens between inputs and outputs. Since old tokens are supposed to be disposed of, this number is negative. This means that the result of the division is rounded up in absolute value, because integer division in Aiken rounds towards negative infinity.

For example, given a redemption rate of 101, 1 old token yields 2 new tokens, because $\frac{1\times101}{100}$ rounds up to 2. A user who has 50 old tokens would get 51 ($\frac{50\times101}{100}$ rounded up) tokens if they swapped in one go. However, if they swap each token individually, they would get $50 \times 2 = 100$ tokens.

We suggest to work using absolute values to avoid the issue.

2.2.3 Unclear Specification

2.2.3.1 Unlock transaction is underspecified

Severity: Medium

The lock validator's logic on the unlock transaction is implemented in such a way that

- it is never possible to redeem UTxOs belonging to different users in the same transaction,
- the new tokens can be paid to the recipient's wallet in as many or few UTxOs as you like, as long as the total is correct,
- the signer of the transaction is irrelevant.

These properties should be made explicit.

The current state of the documentation makes it difficult to assess the boundaries of the allowed behaviour, which makes auditing more difficult, increases the attack surface, and might hinder onboarding of new maintainers and maintainability in the longer term.

2.2.3.2 Swap transaction is underspecified

Severity: Medium

The swap validator's logic on the swap transaction is implemented in such a way that:

- It is not possible to scatter new tokens obtained during the swap into more than one lock UTxO. The specification only forbids that several addresses are involved in the swap but it does not explicitly forbid this.
- It is not possible to transfer new tokens freely, either to or from other users, or to and from oneself, in the swap transaction.

  For example, assume that the redemption rate is 150, and that Alice swaps 10 old tokens. The requirements of the swap transaction are then that 10 old tokens must go to the "burn" script, 15 new tokens must leave the swap script, and 15 new tokens must go to the "lock" script, with a datum indicating Alice as their recipient.

  - Alice cannot, in the same transaction, send/receive other new tokens to/from Bob/herself, even if the amounts all add up and the variation of new tokens in circulation fits the requirement of the swap transaction.
  - The validator forbids increasing the number of new tokens going to the lock script, either by adding to the required UTxO, or by an extra UTxO.
  - The validator allows freely to exchange other asset classes such as Ada in the transaction.
- It is possible to exchange any asset that is not a book token without restrictions. In particular, the swap script does not check conservation of value in any asset class other than new book tokens. The specification should make this explicit.
- It is possible to spend several swap UTxOs, and produce several swap UTxOs, during a swap operation:

  - The number of swap inputs is not necessarily 1, and the number of outputs does not necessarily follow the number of inputs from the swap script.
  - The returned new tokens to the swap script can be redistributed arbitrarily among the transaction outputs that go to the swap script.

  Consequently, one can produce as many swap UTxOs as there are new tokens left in the bank. This would be expensive, as the splitter would have to pay for all the minminum Ada per UTxO. So, there is no incentive to do this apart from slightly hindering the use of the swap operation. Indeed, if the number of required input UTxOs reaches Cardano transaction size limits, an operation which could normally be realized in only one transaction would require several. The minimum Ada carried by each input UTxO would exceed the combined transaction fees and the swapper could keep the difference.

As previously stated, the current state of the documentation makes it difficult to assess the boundaries of the allowed behavior, which makes auditing more difficult, increases the attack surface, and might hinder onboarding of new maintainers and maintainability in the longer term.

2.2.4 Code Quality

2.2.4.1 Conditionals could be avoided using short-circuiting in boolean conjunctions

Severity: Low

The swap validator make use of conditionals in their implementation following the given template:

```
if condition then ... else condition?
```

Making use of lazy evaluation in boolean conjunctions in Aiken, it is possible to rewrite it as follows:

```
condition? && ...
```

We think this significantly improves code readability and, ultimately, maintainability. Besides, in Aiken, `let` statements are only evaluated if they are used[3]. Thus, if `condition` is false, none of the let bindings in `...` will be evaluated so there is no risk of undesired behavior nor additional cost in execution time, hence no more fees.

2.2.4.2 Unnecessary named intermediates

Severity: Lowest

The source code contains a great amount of `let`-bound intermediate variables. We think that it sometimes hinders the overall design and clarity.

The main instance, in our opinion, concerns the many intermediate variables involved in computing the amounts of tokens in a swap (e.g. "multiplied", "divided"). We think that using more self-contained expressions rather than multiple chained name-bound intermediates might have prevented the issue described in concern 2.2.2.1.

2.2.4.3 Some helper functions impact compiled script size

Severity: Lowest

The modules under `lib` contain functions such as

- `to_ref_datum` and `to_lock_datum`: These could be replaced by pattern matching using the `expect` keyword of Aiken. This would also save an additional pattern match at the call site.
- `get_total_value_from_script(ctx)`: This could always be replaced by
  `get_total_value_from_address(ctx, get_validator_address(ctx))`

  Similarly, `get_total_value_from_script` and `get_all_script_inputs` can be replaced.

Our experiments show that removing these helper functions (inlining their definitions at the call site) yields smaller on-chain scripts and hence lower fees.

[3] This is indicated by the information messages given by the Aiken compiler. Our experiments confirm this.

2.2.4.4 ■ Comments are lacking in the code

Severity: Lowest

The source code of the validators would benefit from a few description comments. This would improve readability and maintainability.

2.3 Conclusion

This report outlines the 10 concerns that we have gathered while inspecting the design and code of Booktoken, pertaining to the code contained in the files listed in Table 2.1. As stated in Chapter 1, Tweag does not recommend for nor against the use of any work referenced in this report. Nevertheless, the existence of a *critical* severity concern is a warning sign.